INFORME

PRÁCTICA 4

PTI

Eduard González Moreno
Miguel Guerrero López
06/04/2018

Entorno y configuración

El entorno que hemos utilizado ha sido en un SO linux, en el cual hemos instalado un webserver compatible con Go. Para ello hay que crear los directorios donde estarán los archivos .go que será que el servidor procesa.

A la hora de ejecutar los archivos siempre hay que compilarlos y volver a ejecutar el servidor.

Resolución

Para la resolución de esta práctica hemos utilizado parte del archivo de webserver.go que nos daban. En este archivo hemos creado dos funciones, la primera de todas se encarga de recibir la petición mediante post en formato JSON, y escribirlo en un csv, y la otra se encarga mostrar todos los datos en formato JSON por pantalla.

Para las funciones hemos creado una struct para guardar los datos de renta:

```
type RequestMessage struct {
        Maker int
        Model int
        Nofdays int
        Nofunits int
}
```

Para la primera función lo que hacemos es tratar la petición y comprobamos que no hay ningún error:

```
var requestMessage RequestMessage
   body, err := ioutil.ReadAll(io.LimitReader(r.Body, 1048576))
   if err != nil {
      panic(err)
   }
   if err := r.Body.Close(); err != nil {
      panic(err)
   }
   if err := json.Unmarshal(body, &requestMessage); err != nil {
      w.Header().Set("Content-Type", "application/json; charset=UTF-8")
      w.WriteHeader(422) // unprocessable entity
      if err := json.NewEncoder(w).Encode(err); err != nil {
         panic(err)
      }
```

Si no ha habido ningún Error pasamos de formato JSON a una variable y lo asignamos a otras cuatro independientes que nos facilitarán más adelante el cálculo.

```
maker := requestMessage.Maker
model := requestMessage.Model
nofdays := requestMessage.Nofdays
nofunits := requestMessage.Nofunits
```

Una vez lo tenemos todo en diferentes variables realizamos el cálculo y mostramos por pantalla el desglose de la factura de su petición y el total.

```
total := maker * model * nofdays * nofunits
fmt.Println("Resumen de compra\n")
fmt.Print("Marca: ")
fmt.Println(maker)
fmt.Print("Model: ")
fmt.Println(model)
fmt.Print("Numero de dias: ")
fmt.Println(nofdays)
fmt.Print("Numero de unidades: ")
fmt.Println(nofunits)
fmt.Println("-------------------------------------------------")
fmt.Print("Total: ")
fmt.Println(total)
```

Se ha creado una función auxiliar que es la que se encarga de escribir el pedido en un archivo CSV para en un futuro poderlo leer.

```
writeToFile(w,maker,model,nofdays,nofunits)
```

En esta función se abre el archivo ya creado, y en el caso que no esté creado se crea uno nuevo. Después el método writer se encarga de escribir todas las variables que le pasemos en formato de CSV:

```
func writeToFile(w http.ResponseWriter, make int, model int, nofdays int, nofunits int) {
   file, err := os.OpenFile("rentals.csv", os.O_APPEND|os.O_WRONLY|os.O_CREATE,
0600)
   if err!=nil {
      json.NewEncoder(w).Encode(err)
      return
   }
   writer := csv.NewWriter(file)
   var data1 =
[]string{strconv.Itoa(make),strconv.Itoa(model),strconv.Itoa(nofdays),strconv.Itoa(nofunits)}
   writer.Write(data1)
   writer.Flush()
   file.Close()
}
```

Para resolver la segunda función lo primero tenemos que leer el CSV que hemos creado anteriormente. En el caso de no poder por alguna razón muestra el error por pantalla.

```
carrentalFuncread(w http.ResponseWriter, r *http.Request){
        var rental RequestMessage
        var rentals []RequestMessage
        file, err := os.Open("rentals.csv")
  if err!=nil {
                json.NewEncoder(w).Encode(err)
                return
  }
```

Si no ha habido ningún error tenemos que pasar el archivo a formato JSON para mostrarlo por pantalla para eso vamos recorriendo línea a línea y lo introducimos en la structs creada anteriormente y después en un Array de este mismo tipo de structs para tener todos los pedidos.

```
for {
    record, err := reader.Read()
    if err == io.EOF {
      break
    }
    rental.Maker , _ = strconv.Atoi(record[0])
    rental.Model, _ = strconv.Atoi(record[1])
    rental.Nofdays, _ = strconv.Atoi(record[2])
    rental.Nofunits, _ = strconv.Atoi(record[3])
    rentals = append(rentals, rental)
  }
```

Una vez tenemos todos los pedidos en la array con la siguiente función lo pasamos a formato JSON y lo imprimimos por pantalla.

```
jsonData, err := json.Marshal(rentals)
if err != nil {
        fmt.Println(err)
        os.Exit(1)
}
fmt.Fprintf(w, "%q", string(jsonData))
```

Evaluación de la solución

Para evaluar que la solución es correcta hemos realizado varias pruebas. Primeramente la orden que ejecutamos correctamente era :

curl -H "Content-Type: application/json" -d '{"Maker":2, "Model":3,"Nofdays":4,"Nofunits":6}' http://localhost:8080/carrentalnew

En la cual se ve claramente como le pasamos 4 datos de archivo int. Otras pruebas que hemos realizado ha sido poniendo datos erróneos, como por ejemplo con menos variables, tipo de variables diferentes, variables con nombre diferentes…

Ventajas tecnológicas

El lenguaje que hemos tratado es un lenguaje que es bastante nuevo, tiene la gran ventaja tecnológica que para hacer diferentes tareas como por ejemplo pasar de csv a JSON se hace en pocas líneas en cambio otros lenguajes como en JAVA es más complicado, pero al ser tan nuevo no hay tanto soporte para errores y todavía hay módulos que les falta por experimentar.

Además en esta tecnología para el ámbito de web es bastante engorroso el hecho que cada vez que quieras compilar tengas que parar el proceso del servidor y despues volverlo a ejecutar.