Creació de Web API amb Go

Projecte Tecnologies d'Informació

Pau Coma Ninot

Àlex Aguilera Martínez
Grup 20

1. Objectius pràctica

L'objectiu d'aquesta pràctica és la creació d'una aplicació web senzilla de lloguer de cotxes implementant una web API senzilla fent ús del llenguatge de programació Go i JSON.

Les dues funcionalitats bàsiques de l'aplicació seran les mateixes de la pràctica 1:

- Fer una nova petició de lloguer de vehicles amb un formulari que especifica una sèrie de paràmetres (car maker, car model, number of days i number of units)
- Obtenir una llista amb els cotxes llogats anteriorment, però sense necessitat de contrasenya aquest cop, en format JSON.

Tant la nova petició com la obtención de la llista, es programa al fitxer webserver.go amb JSON responses i JSON requests. (majoria del codi proporcionat)

2. Implementació

1- Instal·lació i configuració

Per aquesta pràctica era necessari instal·lar "curl" i "git" amb les seves respectives comandes. I a més a més, les llibreries necessaries del llenguatge de programació utilitzat en aquesta pràctica, Go. Per fer-ho, s'han executat les següents comandes:

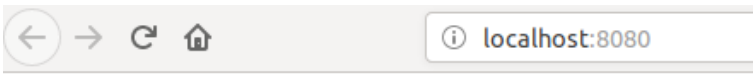
$wget https://gitlab.fib.upc.edu/pti/pti/raw/master/goREST/go1.7.1.linux-amd64.tar.gz (Obtenció)

$sudo tar -C /usr/local -xzf go1.7.1.linux-amd64.tar.gz (Extracció)

$export PATH=$PATH:/usr/local/go/bin (Afegim variable d'entorn nova per Go)

El següent pas a la guia de la pràctica era la creació d'un directori per els projectes de Go desde $HOME i veure amb un exemple "hello, world" (codi proporcionat pel professorat) com compilar i executar un fitxer .go.

A continuació, la creació d'un web server sentzill amb Go amb el fitxer "webserver.go" que seria després la base per fer la nostra aplicació de lloguer de cotxes. Fent ús del codi proporcionat i del browser, vam poder comprobar el seu correcte funcionament:




```
Hello, "/"
```


```
You are calling with param: 1234
```

Per finalitzar, la guia de la pràctica ens proporcionava un codi per la JSON response i la JSON request per veure el funcionament i fer la nostra implementació després. La JSON response es podia veure tant amb curl com amb el browser, deixem una captura del resultat:

```
alumne@pcrecanvib5:~/go/src/pti_golang/webserver$ curl -H "Content-Type: applica
tion/json" http://localhost:8080/endpoint/1234
{"Field1":"Text1","Field2":"1234"}
```

2- New rental

Per dur a terme la primera funcionalitat, vam crear un struct nou amb els paràmetres següents:

```
type Cotxe struct {
    Model string
    Venedor string
    Dies string
    Unitats string
}
```

Aprofitant la funció "endpointFunc2JSONInput" proporcionada i modificant segons les nostres necessitats, obtenim:

```
func endpointFunc2JSONInput(w http.ResponseWriter, r *http.Request) {
    //var requestMessage RequestMessage
    var cotxe Cotxe
    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, &cotxe); 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)
        }
    } else {
        llogarcotxe(w, []string{cotxe.Model, cotxe.Venedor, cotxe.Dies, cotxe.Unitats})
    }
}
```

On "llogarcotxe", és la següent:

```
func llogarcotxe (w http.ResponseWriter, values []string) {
    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)
    writer.Write(values)
    writer.Flush()
    file.Close()
}
```

Aquestes dues funcions, creen el .csv si no existeix i introduexen els paràmetres venedor, model, dies i unitats amb un JSON response utilitzant "curl" de la manera següent:

```
alumne@pcrecanvib5:~/go/src/pti_golang/webserver$ curl -H "Content-Type: applica
tion/json" -d '{"Venedor":"Seat", "Model":"Ibiza", "Dies":"10", "Unitats":"1"}'
http://localhost:8080/endpoint2/1234
```

I obteniem al fitxer "rentals.csv" el següent:

```
Ibiza,Seat,10,1
```

3- Obtenció llista

Per obtener la llista de cotxes definirem una nova funció al mateix arxiu. La funció s' anomena llistar i la definim de la següent manera:

```
func llistar(w http.ResponseWriter, r *http.Request) {
    var rentalsArray []Cotxe
    file, err := os.Open("rentals.csv", )
    if err != nil {
        log.Fatal(err)
    }
    reader := csv.NewReader(file)
    for {
        line, err := reader.Read()
        if err != nil {
            if err == io.EOF {
                break
            }
            log.Fatal(err)
        }
        rentalsArray = append(rentalsArray, Cotxe{Model: line[0], Venedor: line[1],
            Dies: line[2], Unitats: line[3]})
    }
    json.NewEncoder(w).Encode(rentalsArray)
}
```

Fem servir l' struct definit anteriorment.

A continuació, enllacem la funció amb el primer endpoint, tal que així:

```
router.HandleFunc("/endpoint/", llistar)
```

Ara tornem a fer el "go install..." i reiniciem el go.

Tenim un arxiu csv amb dos lloguers guardats i al fer la crida, obtenim els dos lloguers en format JSON. Ho veiem aquí:

```
root@pau-Lenovo-Z50-70:/home/pau/go# curl -H "Content-Type: application/json" http://localhost:8080/endpoint/
[{"Model":"SEAT","Venedor":"VENEDOR SA","Dies":"2","Unitats":"1"},{"Model":"NISSAN","Venedor":"VENEDOR2 SA","Dies":"2000","Unita
ts":"4"}]
```

4 – Extra

Per obtenir el punt extra faré un petit escrit comparant dues tecnologies utilitzades per la implementació d' APIs en les aplicacions web. Són Go i Rust.

Go es un llenguatge de programació concurrent i compilat orientat a objectes inspirat en la sintaxis de C. Va ser desenvolupat per Google, i es disponible tant en Windows, com en Linux i Mac.

Per l' altra banda, Rust és un llenguatge de programació compilat, de propòsit general i multi paradigma que està sent desenvolupat per Mozilla. També és un llenguatge orientat a objectes, disponible en Windows, Linux i Mac.

A primera vista, semblen dos llenguatges bastant similars. Així doncs, en què es diferencien? Quin és millor?

Com a primera comparació, tenim que Rust és més ràpid que Go. No hi ha cap argument, simplement les referències ens diuen que Rust s' executa més ràpid que Go. Tot i així, Go segueix sent més ràpid que Java, C# o Python.

Parlant de complexitat de codi, podem veure que son bastant semblants. En la meva opinió, Go és una mica més simple i més clar, i si no coneixes cap dels dos llenguatges, Go pot ser la teva primera opció ja que serà més fàcil d' aprendre.

A continuació, compararem com realitzen multi-threading aquests dos llenguatges. Quan fem un servidor HTTP, necessitem assegurar que les nostres dades estan guardades de manera segura, wue hi hagi una bona comunicació entre threads, que sigui fàcil treballar amb paral·lelisme... Els dos llenguatges són molt fiables a l' hora de fer totes aquestes coses, però Go requereix menys esforç. Rust és més potent, però Go és més simple. En aquest aspecte, la decisió és del programador.

Un altre aspecte import a l' hora de programar és la gestió d' errors: en aquest punt, Go i Rust són bastant similars. Rust permet al programador gestionar els errors a través de dos tipus de dades: Option o Result. La majoria de llibreries implementades fan servir aquestes dues variables i l' usuari només ha de gestionar-ho quan no es retorna res o quan es retorna un error. Go deixa la gestió d' errors mes lliure al programador. Per exemple, es poden ignorar els errors amb el símbol "_". Tot i així és recomanable gestionar-ho.

La meva conclusió és que són dos llenguatges molt, molt similars i amb les petites diferències que hem trobat, jo deixaria la decisió al programador.

Jo personalment faria servir Go, ja que sembla que té una sintaxis més fàcil i s' escriu de manera més ràpida. Tot i així, si necessités un llenguatge potent i que es comportes de manera molt ràpida potser triaria Rust.