Développement Go: Serveur et client

Stéphane Karraz
ESGI - cours électif de Go - 2022

Introduction

Le Go étant pensé pour le web, la librairie standard possède tout ce qu'il faut pour créer un serveur ou même un client gérant les requêtes HTTP.

Même s'il existe des frameworks de mieux en mieux pensés et complets, nous pouvons tout à fait créer un serveur "production ready" avec les fonctions et librairies natives.

Bootstrap d’un serveur

Ce dont on a besoin

Pour créer un serveur Go (et même un client) et mettre à disposition une API HTTP, nous n’avons besoin que:

- De la librairie standard “net/http”
- De déclarer une ou plusieurs fonctions et de les associer à nos différentes routes
- De lancer le serveur

main

Nous n’allons déclarer qu’une seule route pour l’exemple: “/hello”, puis nous démarrons le serveur:

```
func main() {
    http.HandleFunc("/hello", helloHandler)
    http.ListenAndServe(":9000", nil)
}
```

Handlers

Pour gérer les requêtes sur la route “/hello”, nous devons déclarer une fonction que nous avons déjà associé à cette route précédemment:

```
func helloHandler(w http.ResponseWriter, req *http.Request)
{
    // future gestion ici
}
```

Notre fonction prend deux paramètres:

- ResponseWriter permettant d’écrire une réponse à renvoyer au client
- Request contenant les informations envoyés par le client (headers, body etc)

Gestion des types de requêtes : GET

Nous allons commencer par gérer les requêtes GET sur notre route.

```
func helloHandler(w http.ResponseWriter, req *http.Request)
{
    switch req.Method {
    case http.MethodGet:
     fmt.Fprintf(w, "Hello world")
}
```

Dans le cas où nous faisons une requête GET sur la route “/hello”, le message “hello world” sera envoyé au client en réponse.

Gestion des types de requêtes: POST

```
case http.MethodPost:
     if err := req.ParseForm(); err != nil { // Parsing des paramètres envoyés
         fmt.Println("Something went bad")   // par le client et gestion
                                             // d'erreurs
         fmt.Fprintln(w, "Something went bad")
         return
     }
     for key, value := range req.PostForm{ // On print les clés et valeurs des
         fmt.Println(key, "=>", value)     // données envoyés par le clients
     }
     fmt.Fprintf(w, "Information received: %v\n", req.PostForm)
    // On termine en renvoyant au client ce qui nous a été envoyé
```

Type de formulaire

Dans cet exemple, pour récupérer les données envoyées par le client, nous ne récupérons que les données au format URL encoded.

Si vous le souhaitez, vous pouvez récupérer un payload au format JSON, mais dans ce cas, il faudra récupérer ce Body comme un array de bytes, et utiliser la fonction Unmarshall pour transformer cet array en structure attendu et l’exploiter sous ce type de donnée.

```
package main

import (
    "fmt"
    "net/http"
)

func helloHandler(w http.ResponseWriter, req *http.Request) {
    switch req.Method {
    case http.MethodGet:
       fmt.Fprintf(w, "Hello world")
    case http.MethodPost:
       if err := req.ParseForm(); err != nil {
             fmt.Println("Something went bad")
             fmt.Fprintln(w, "Something went bad")
             return
       }
       for key, value := range req.PostForm {
             fmt.Println(key, "=>", value)
       }
       fmt.Fprintf(w, "Information received: %v\n", req.PostForm)
    }
}

func main() {
    http.HandleFunc("/hello", helloHandler)
    http.ListenAndServe(":9000", nil)
}
```