TD 2
Golang pour DevOps

Note préliminaire: en Go, la portée d'un champ, d'une fonction,ou d'une variable est déterminée par la première lettre de son nom: si c'est une majuscule, l'entité est "publique", autrement dit "exportée". Si la première lettre de son nom est minuscule, l'entité n'est pas exportée.

1 Les interfaces

1.1. Reprenez le code suivant:

```
package main

import "fmt"

type IPAddr  [4]byte

func main() {
    hosts := map[string]IPAddr{
        "loopback": {127, 0, 0, 1},
        "googleDNS": {8, 8, 8, 8},
    }
    for name, ip := range hosts {
        fmt.Printf("%v: %v\n", name, ip)
    }
}
```

Et implémentez fmt.Stringer pour IPAddr, de façon à afficher l'adresse IP comme une suite de nombres séparés de points.
Autrement dit, la slice IPAddr{1, 2, 3, 4} devrait s'afficher de la façon suivante: "1.2.3.4". Implémentez ceci de deux façons:

(1.a) En utilisant le package https://golang.org/pkg/fmt/ et fmt.Sprintf()

(1.b) En utilisant le package https://golang.org/pkg/strconv/

1.2. On va implémenter une erreur "custom". Pour cela, remarquez que les erreurs en Go sont simplement des entités qui implémentent l'interface:

```
https://golang.org/pkg/builtin/#error
```

(2.a) Définissez une struct ”MyError” qui contient deux champs:

- ”When” de type time.Time (voir https://golang.org/pkg/time/#Time)
- ”What” de type string

(2.b) Implémentez l'interface Error pour votre struct ”MyError”.

(2.c) Écrivez une fonction ”run()”, dont la signature est:

```
func run() error
```

qui retourne systématiquement une erreur comme celle que vous venez de créer.

- Pour donner une valeur au champ du temps, utilisez ”Now()”, du package time de la librairie standard.
- Pour donner une valeur au champ du ”What”, choisissez un texte d'erreur non vide qui vous plaît.

(2.d) Dans la fonction main de votre programme, exécutez la fonction ”run()” que vous venez de définir, et récupérez l'erreur en l'assignant à une variable, comme ceci par exemple:

```
err := run()
```

(2.e) Vérifiez si votre erreur est vide (elle ne devrait jamais l'être) en écrivant un ”if”, et si elle ne l'est pas, affichez-la dans la console. Pour tester si l'erreur est ”vide”, basez-vous sur le fait qu'une erreur est une interface, et comparez la valeur de votre erreur à la zero value d'une interface.

1.3. On va à présent étudier l'interface vide, et la détermination des types sous-jacents:

```
interface{}
```

(3.a) Écrivez une fonction qui a la signature suivante, et qui se contente d'exécuter fmt.Println() sur l'argument qui lui est passé:

```
func PrintIt(input interface{})
```

(3.b) Essayez d'exécuter cette fonction en passant un entier en argument. (c'est à dire: appelez cette fonction dans votre fonction main et lancez votre programme).

Que se passe-t-il ?

(3.c) Essayez à présent avec une chaîne de caractères. Même question.

(3.d) L'interface vide est un contrat qui n'a aucune méthode. Donc pour l'implémenter, n'avoir aucune méthode sur un type suffit. Tous les types en Go ont 0 ou plus méthodes, donc tous les types vérifient l'interface vide.
Modifiez à présent votre fonction PrintIt() pour qu'elle affiche le type de l'argument passé, et non plus sa valeur. Utilisez le switch vu en cours.

2 Le parsing JSON

2.1 Parsing pur

Soit la structure User suivante:

```
type User struct {
    Login    string
    Password string
}
```

1.1. Afficher dans le terminal le résultat de la sérialisation d'un utilisateur dont le login est "Paul" et le mot de passe "pass123".

1.2. Remplacer le nom du champ "Login" dans le JSON de sortie par "userName" en utilisant les annotations.

1.3. Soit users.json un fichier contenant une liste d'utilisateurs au format JSON:

```
[{"userName": "matm", "Password": "123456"}, {"userName":
    "fake44", "Password": "azerty"}]
```

Lire puis désérialiser le contenu de ce fichier dans une liste de User.

1.4. Que se passe-t-il si vous renommez "Password" en "password" dans votre structure et que vous lancez à nouveau le programme précédent ?

2.2 Serveur HTTP

On va écrire dans cette partie un petit serveur HTTP qui permet d'obtenir des information sur des utilisateurs, en passant des query. Voir:

https://en.wikipedia.org/wiki/Query_string

2.1. Ajoutez dans votre fichier users.json un champ "userID" aux utilisateurs de la liste, et donnez lui une valeur différente pour chaque utilisateur.

2.2. Ajoutez dans la struct User définie précédemment un champ permettant de correctement récupérer le user ID lorsque l'on désérialise le json du fichier.

2.3. Créez une map, sous forme de ”variable globale”, que vous remplirez dès que la fonction main sera exécutée en lisant votre fichier JSON. Le type de cette map est:

```
map[string]User
```

avec en clé le user ID et en valeur le user correspondant.
Pour définir une ”variable globale”, définissez-la en dehors de toute fonction. Elle est alors accessible dans toutes les fonctions de votre fichier main.go.

2.4. Définissez un handler pour votre serveur. Un handler est une fonction qui a cette signature:

```
func(http.ResponseWriter,  *http.Request)
```

avec l'interface:
https://golang.org/pkg/net/http/#ResponseWriter et la struct:
https://golang.org/pkg/net/http/#Request
comme arguments. Laissez le corps de la fonction vide pour l'instant.

2.5. Définissez un handler dans votre fonction main, en utilisant http.HandleFunc de:

```
https://golang.org/pkg/net/http/#HandleFunc
```

Utilisez le pattern ”/” dans votre appel à cette fonction, et passez lui comme second argument la fonction définie au point précédent.

2.6. Écrivez l'appel qui permettra de lancer votre serveur en utilisant:

```
https://golang.org/pkg/net/http/#ListenAndServe
```

2.7. On va maintenant écrire la logique de notre fonction de handler. Lorsque vous faites une requête http au serveur que nous sommes en train d'écrire, le serveur appelle notre fonction handler. On a accès à la requête envoyée par le client par l'objet *http.Request passé en argument de votre fonction de handler, et on peut écrire notre réponse sur le http.ResponseWriter.

(7.a) Écrivez du code qui vérifie que le user passé dans la requête est bien dans notre map, définie en variable globale. Pour cela, utilisez:

```
id := r.FormValue("id")
```

qui permet de récupérer le champ de query ”id” dans une requête HTTP de type Get à l'adresse:

```
http://localhost:8000/?id=id1
```

(7.b) Si l'id est trouvé, sérialisez le User correspondant en JSON. Autrement, ne faites rien.

(7.c) Écrivez le header de votre réponse avec:

```
w.Header().Set( "
    Content-Type",
    "application/json; charset=utf-8",
)
```

(7.d) Ajoutez ensuite le code de la réponse avec:

```
w.WriteHeader(http.StatusOK)
```

lorsque le user existe et:

```
w.WriteHeader(http.StatusNotFound)
```

lorsqu'il est introuvable.

(7.e) Enfin, écrivez le JSON du User (si votre code réponse n'est pas "Not- Found") dans le ResponseWriter avec:

```
Write([]byte) (int, error)
```

de:

```
https://golang.org/pkg/net/http/#ResponseWriter
```

(7.f) Lancez votre serveur, et testez-le avec des requêtes curl comme:

```
$ curl -i http://localhost:8000/?id="id1"
```