Examen
Golang pour DevOps

On va réaliser un petit serveur HTTP de liste de tâches. Quand le programme sera terminé, vous pourrez ajouter des tâches, en faire la liste, et mar- quer des tâches comme terminées, en utilisant des requêtes HTTP.

1 La base du serveur

1.1. Créez un nouveau projet go.

1.2. Définissez une fonction main, qui sera la base de l'exécution de votre programme.

1.3. Définissez un type Task, qui est une struct avec deux champs:

- "Description", de type string
- "Done", de type bool

1.4. Définissez une variable globale "task" qui est une slice de Task.

1.5. Définissez trois HandleFunc (https://golang.org/pkg/net/http/#HandleFunc) et donnez-leur respectivement:

- Un path "/", et une fonction "list"
- Un path "/done", et une fonction "done"
- Un path "/add", et une fonction "add"

1.6. Appelez, dans le main, la fonction ListenAndServe (https://golang.org/pkg/net/http/#ListenAndServe) qui vous permettra de lancer votre serveur HTTP avec les handlers que vous venez de définir.

2 La fonction ”list”

2.1. Définissez une fonction ”list”, c’est celle que vous avez passé au handler avec le path ”/”, avec la signature suivante:

```
func list(rw http.ResponseWriter, _ *http.Request)
```

(notez qu’on ne se servira pas de la Request)

2.2. Générez une chaîne de caractère à partir de la liste de vos tâches, qui se trouve dans la variable globale ”tasks” que vous avez définie plus tôt. C’est à dire que vous devez fabriquer une variable de type string, qui représente votre liste de tâches. Pensez à inclure un ID pour votre tâche. Cet ID est l’indice de la tâche dans la slice tasks.
Exemple de string à générer:

```
ID: 0, task: "Faire les courses"
ID: 1, task: "Payer les factures"
```

si la slice est:

```
[{"Faire les courses" false},
{"Payer les factures" false}]
```

2.3. Ajoutez une condition au code précédent pour n’inclure que les tâches qui ne sont pas terminées (celles pour lesquelles ”Done” == false).

2.4. Écrivez le header de la réponse à envoyer au client:

```
rw.WriteHeader(http.StatusOK)
```

2.5. Puis écrivez votre réponse dans rw, avec:

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

Notez que vous avez actuellement une chaîne de caractères et que Write prend une slice de byte. Vous pouvez convertir la chaîne en slice de byte avec:

```
chaine:= "ma chaine"
enBytes:= []byte(chaine)
```

3 La fonction ”add”

La fonction add ne va accepter que des requêtes HTTP ”POST”. On n'enverra et on n'attendra dans ces requêtes qu'un petit descriptif sous forme de chaîne de caractères.

3.1. Commencez par vérifier que la méthode de r (la *http.Request), est bien http.MethodPost.

La http.MethodPost est une constante:
https://golang.org/pkg/net/http/#pkg-constants

Vous pouvez récupérer la méthode qui est dans le champ Method de la variable r.
Documentation du type de r:
https://golang.org/pkg/net/http/#pkg-constants

3.2. Si la méthode n'est pas POST, écrivez le status ”bad request” dans votre réponse, et terminez la fonction:

```
rw.WriteHeader(http.StatusBadRequest)
```

3.3. À la suite, lisez le corps de la requête avec:

```
body, err :=
ioutil.ReadAll(r.Body) if err !=
nil {
    fmt.Printf("Error reading body: %v", err)
    http.Error(
        rw,
        "can't read body", http.StatusBadRequest,
    )
    return
}
```

Le body ainsi récupéré est une slice de byte. Castez-le en string, puis placez la chaîne de caractères ainsi obtenue dans un nouvel objet Task, dans le champ ”Description”.

3.4. Ajoutez la tâche que vous venez de fabriquer avec ”append”:

---

https://tour.golang.org/moretypes/15

3.5. Écrivez le status ”OK” dans votre fonction, et terminez son exécution.

3.6. Pour tester un add, vous pouvez utiliser la commande curl suivante:

```
curl --request POST --data 'Description'
http://localhost:8080/add
```

4 La fonction ”done”

On va ajouter ici un moyen de marquer une tâche comme terminée, ainsi que de n’afficher que les tâches terminées.

4.1. Faites un switch sur la valeur de r.Method

4.2. Pour le cas où r.Method est égal à http.MethodGet, écrivez du code simi- laire à l’affichage des tâches dans list, mais filtrez les tâches pour n’afficher que celles qui sont à Done==true.

4.3. Pour le cas où r.Method est égal à http.MethodPost, lisez le body de r comme précédemment, et convertissez-le en entier.

4.4. Utilisez l’entier pour récupérer la tâche dans la variable globale tasks (on rappelle que cet entier est la position de la tâche dans la slice), et changez Done à ”true”.
(attention à bien vérifier que l’indice n’est pas plus grand que le tableau sinon le programme panique ! Vous pouvez utiliser len(tasks) pour avoir le nombre de vos tasks)

4.5. Pour le cas ”default” de votre switch, renvoyez un status ”BadRequest” comme précédemment.

4.6. Vous pouvez tester un marquage de task à ”Done == true” avec la requête curl:

```
curl --request POST --data '1' http://localhost:8080/done
```

4.7. Vous pouvez vérifier le fonctionnement de votre serveur en ouvrant dans votre navigateur les adresses:

- http://localhost:8080/done (le navigateur fait une requête HTTP GET)
- http://localhost:8080/