INTRODUCTION AU DÉVELOPPEMENT WEB EN GO

Les serveurs HTTP en Go

- Pour créer des serveurs HTTP en Go, nous avons besoin du package **net/http** !

```go
import "net/http"

func main() {
    http.ListenAndServe(":80", nil)
}
```

Cette ligne permet ainsi de lancer un serveur local sur le port 8080.

Les serveurs HTTP en Go

Mais maintenant que nous avons réussi à afficher du contenu sur notre navigateur, la question est la suivante : **Comment créer un site en HTML, et le lier au Go ?**

Pour cela, nous aurons besoin des **templates HTML (html/template)**

```go
import "html/template"

var tmpl = template.Must(template.ParseFiles("index.html"))
```

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        <h1>Bienvenue !</h1>
    </body>
</html>
```

En utilisant template.Must, nous pouvons charger un fichier html avec template.ParseFiles("index.html"))

Les serveurs HTTP en Go

Voici un exemple concret d'une application web simple en Go :

```go
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello world !")
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Nous pouvons ainsi voir deux fonctions :

- La fonction « handler », renvoyant « Hello world ! »
- La fonction « main », contenant HandleFunc et ListenAndServe

HandleFunc permet ainsi d'associer une route (ou URL) à une fonction (appelée *handler*) qui va générer une réponse HTTP.

Cela signifie que quand un utilisateur accède à http://localhost:8080, Go exécute la fonction handler et c'est le contenu que la fonction écrit dans http.ResponseWriter qui est affiché dans le navigateur.

Les serveurs HTTP en Go

Pour le moment, notre html est statique, nous ne pouvons pas le modifier. C'est pour cela que nous allons légèrement le modifier :

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        <h1>{{.Titre}}</h1>
    </body>
</html>
```

Dorénavant, le <h1> de notre HTML sera « Page dynamique ».

```go
import (
    "net/http"
    "text/template"
)

var tmpl = template.Must(template.ParseFiles("indew.html"))

func handler(w http.ResponseWriter, r *http.Request) {
    data := struct{ Titre string }{Titre: "Nouveau Titre"}
    tmpl.Execute(w, data)
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Et dans notre main.go, on ajoute ces lignes et on modifie la ligne **tmpl.Execute(w, nil)** par **tmpl.Execute(w, data)**

Les serveurs HTTP en Go

```
<!-- check l'existence -->
    {{if .Message}}
        <p>{{.Message}}</p>
    {{end}}
    <!-- check l'existence 2-->
    {{with .Message}}
     <p>{{.}}</p>
    {{end}}
    <ul>
        <!-- parcours de liste-->
    {{range .List}}
        {{if isEven .}} <!-- utilisation de fonction -->
            <li>{{.}} est pair </li>
        {{else}}
            <li>{{.}} est impair </li>
        {{end}}
    {{end}}
    </ul>

```

If :permet de vérifier une condition (`nil`, chaîne vide, booléen, slice vide, etc.).

With : sert à tester la présence d'une valeur **et** à changer le contexte (" **.** " devient cette valeur).

Range :permet de parcourir des slices, arrays, maps

End : ferme l'action en cours (`if`, `with`, `range`).

Et maintenant... Passons à l’exercice pratique !

Liens Utiles

Documentation net/http

Documentation html/template