



# Tips en Go pour le Web

## [Golang HTML/CSS](#)

[Servir des fichiers statiques \(HTML, CSS, Images...\)](#)

[Envoyer des variables](#)

[Requêtes GET et POST, logique de jeu](#)

[Conditions et boucles en HTML](#)

[Variables globales](#)

 [Hangman](#)

 [Hangman-Web](#)

 [Groupie Tracker](#)

 [Forum](#)

## [Golang HTML/CSS](#)

### Servir des fichiers statiques (HTML, CSS, Images...)

Vous vous en serez peut-être rendu compte, mais vos fichiers CSS ne fonctionnent pas dans votre serveur Go ! C'est normal, dans les frameworks et les serveurs backend, les fichiers statiques ne sont pas forcément chargés. Il faut dire à votre code d'aller chercher un dossier où se trouvent vos fichiers statiques. Pour se faire, rajoutez ces deux lignes en-dessous de vos routes (func) :

```
fs := http.FileServer(http.Dir("./static/"))
http.Handle("/static/", http.StripPrefix("/static/", fs))
```

Mais que font ces lignes ? La création de variable `fs` lui dit que créer un microserveur pour servir des fichiers, et ce serveur se trouve à l'emplacement que vous précisez en argument de `http.Dir()`.

Enfin, il y a une petite triche. Ici, on ne peut avoir deux Handler sur le répertoire racine ("/"), ou sur un quelconque chemin. Donc on lui dit avec le Handle "vas chercher tout ce qu'il y a sur "/static", mais avec `http.StripPrefix()`, enlève le string "/static", et sert les fichiers avec la variable `fs`.

### Envoyer des variables

Voici un petit code exemple pour montrer comment faire passer des variables au front-end. Après, à vous de gérer les variables.

```
// Je crée ma structure
type Test struct {
    MaVariable string
}

func main() {
    http.HandleFunc("/", Handler) // Ici, quand on arrive sur la racine, on appelle la fonction Handler
    //
```

```

fs := http.FileServer(http.Dir("./static"))
http.Handle("/static/", http.StripPrefix("/static/", fs))
//

http.HandleFunc("/hangman", Handler) // Ici, on redirige vers /hangman pour effectuer les fonctions POST
http.ListenAndServe(":8080", nil)    // On lance le serveur local sur le port 8080
}

func Handler(w http.ResponseWriter, r *http.Request) {
    // J'utilise la librairie tpl pour créer un template qui va chercher mon fichier index.html
    tpl := template.Must(template.ParseFiles("index.html"))

    // Je crée une variable qui définit ma structure
    data := Test{
        MaVariable: "Ouba ouba",
    }

    // J'exécute le template avec les données
    tpl.Execute(w, data)
}

```

```

<!DOCTYPE html>
<html lang="en">

<head>
    <title>Bonjour</title>
</head>

<h1>{{ .MaVariable }}</h1>

```

## Requêtes GET et POST, logique de jeu

Il existe en HTTP plusieurs types de requête, les deux principales sont les requêtes GET et POST.

Une requête en HTTP, ça ressemble à ça :

```

GET /test/demo.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: monsite.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

```

Donc en gros ça ressemble à pas grand chose. Ca n'envoie que des métadonnées sur votre client, plus les données en interne pour communiquer avec votre serveur.

Une requête GET est une requête “publique”, c'est-à-dire que les données qui transitent sont simples et à but de diriger votre navigateur vers une cible. Par exemple :

<http://monsite.com/test/demo.php?name1=value1&name2=value2>

dirige le navigateur vers monsite.com/test/demo, avec les paramètres publics name=value1, et name2=value2. Vous pouvez voir ces paramètres, c'est du GET, ils ne sont pas dans la requête mais dans l'URL.

Une requête en POST, ça ressemble à ça :

```

POST /test/demo.php HTTP/1.1
Host: monsite.com

```

```
name1=value1&name2=value2
```

Cette fois-ci, les données sont envoyées dans la requête même et sont donc invisibles de l'utilisateur. Elles ne sont pas non plus mis en cache.

Vous souhaitez récupérer la/les lettres envoyées par le joueur dans votre hangman. Pour cela, créez dans votre HTML un formulaire pour rentrer des données :

```
<form action="/hangman" method="post">
  <input id="inputBox" name="input" type="text" />
  <input type="submit" value="Entrer" />
</form>
```

- `action=""` permet de préciser l'URL de destination
- `name=""` sera le champs à récupérer dans le back pour recevoir la valeur entrée

Dans votre handler, gérez votre arrivée pour différencier le GET et le POST :

```
switch r.Method {
  case "GET":
    fmt.Println("GET")
  case "POST": // Gestion d'erreur
    if err := r.ParseForm(); err != nil {
      fmt.Fprintf(res, "ParseForm() err: %v", err)
      return
    }
}
// Récupérez votre valeur
variable := r.Form.Get("input")
```

Et boum : `variable` sera une string correspondant à la valeur que vous aurez récupéré dans la petite boîte pour rentrer une lettre. Vous pourrez en faire ce que vous voulez.

## Conditions et boucles en HTML

Avec le templating, vous avez la possibilité de faire une légère programmation dans votre HTML. Ce n'est pas obligatoire, mais ça peut vous être utile.

Toute la doc est ici : <https://pkg.go.dev/text/template>

Chaque instruction commence par des crochets `{{ }}` et finissent par `{{ end }}`

Voici les principales utilisations :

```
<!-- Afficher une variable si elle remplit une condition -->
<!-- Ici, afficher Bonjour si la variable Test est égale à 10 -->
<body>
  <h1>Bonjour</h1>

  {{ if eq .Test 10 }}
  <h2>Je suis égal à 10 !</h2>
```

```
{{ end }}  
</body>
```

```
<!-- Afficher tous les éléments d'un tableau de string Test -->  
  
<body>  
  <h1>Bonjour</h1>  
  
  {{ range .Test }}  
    <p>{{ . }}  
  {{ end }}  
</body>
```

## Variables globales

Vous pouvez passer la portée de vos variables à un niveau supplémentaire pour pouvoir les utiliser dans un fichier entier, ou dans tous vos fichiers dans vos projets. Pour ce faire, faites comme suit :

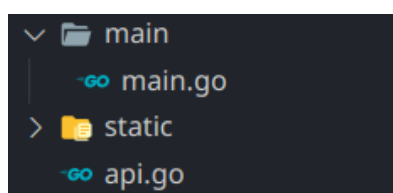
Si vous utilisez une variable de cette manière, elle ne sera disponible que dans la fonction :

```
func main() {  
    bonjour := 3  
    fmt.Println(bonjour)  
}  
  
// Vous ne pourrez pas l'utiliser ici, par exemple :  
func OtherFunc() {  
    bonjour = 2 // ERROR  
}
```

Par contre, vous pouvez déclarer une variable tout en haut de votre fichier hors des fonctions, et vous pourrez l'utiliser dans tout le fichier :

```
package main  
  
import "fmt"  
  
var aurevoir string // Ici  
var merci = 5 // Vous pouvez soit les initialiser à vide, soit avec une valeur  
                // par défaut  
  
func main() {  
    aurevoir = "test"  
    fmt.Println(aurevoir) // Ca fonctionne !  
}
```

Mettez les variables en majuscule pour les utiliser dans n'importe quel fichier. Par exemple, avec une arborescence comme suit :



Dans le fichier `api.go` qui est à la racine :

```
package groupie

var Biensur string // Déclaré avec une majuscule, utilisable partout

func APITest() {
    fmt.Println("fonction vide")
}
```


Et dans `main/main.go` :

```
package main

import (
    "fmt"
    g "groupie" // Import du package groupie, on met un alias pour pouvoir
                // utiliser plus rapidement les fonctions importées

    func main() {
        g.Biensur = "bonjour"
        fmt.Println(g.Biensur) // Ca marche !
    }
```

## Hangman

 [Hangman](#)

## Hangman-Web

 [Hangman-Web](#)

## Groupie Tracker

 [Groupie Tracker](#)

## Forum

 [Forum](#)