A Simple Web Server

This lesson explains how to design a web server that provides services over the web.

WE'LL COVER THE FOLLOWING

Simple web server

HTTP is a higher-level protocol than TCP, and it describes how a web server communicates with client-browsers. Specifically for that purpose, Go has `net/http` package, which we will now explore.

We will start with some really simple concepts. First, let's write a **Hello world!** web server. We import `http`, and our web server is started with the function `http.ListenAndServe("0.0.0.0:3000", nil)`, which returns *nil* if everything is OK or an error otherwise (0.0.0.0 can be omitted from the address, 3000 is the chosen port number).

A web-address is represented by the type `http.URL`, which has a `Path` field that contains the `URL` as a string. Client-requests are described by the type `http.Request`, which has a `URL` field.

If the request `req` is a POST of an Html-form, and `var1` is the name of an Html input-field on that form, then the value entered by the user can be captured in Go-code with: `req.FormValue("var1")`. This also works when `var1` is a parameter given via the `url`: https://1dkne4jl5mmmm.educative.run/?var1=value (in your case URL will be different). An alternative is to first call `request.ParseForm()`, and the value can then be retrieved as the 1st return parameter of `request.Form["var1"]`, like in:

```
var1, found := request.Form["var1"]
```

> **Remark:** If you're running it locally, http://localhost:nnnn/?var1=value will work.

The 2nd parameter `found` is then true. If `var1` was not on the form, `found` becomes *false*. The `Form` field is in fact of type `map[string][]string`. The web server sends an `http.Response`, its output is sent on an `http.ResponseWriter` object. This object assembles the HTTP server's response; by writing to it, we send data to the HTTP client. Now we still have to program what the web server must do, how it handles a request. This is done through the function `http.HandleFunc`. In this example, it says that if the root "/" is requested (or any other address on that server) the function `HelloServer` is called. This function is of the type `http.HandlerFunc`, and they are most often named `Prefhandler` with some prefix `Pref`. The `http.HandleFunc` registers a handler function (here `HelloServer`) for incoming requests on /. The / can be replaced by more specific URL's like /**create**, /**edit**, and so on; for each specific URL, you can then define its corresponding handler-function.

This function has as 2nd parameter the request `req`. Its first parameter is the `ResponseWriter`, to which it writes a string composed of `Hello` and `r.URL.Path[1:]`. The trailing `[1:]` means *create a sub-slice of Path from the 1st character to the end*. It drops the leading / from the pathname. This writing is done with the function `fmt.Fprintf()`; another possibility is `io.WriteString(w, "hello, world!\n")`

The 1st parameter is a requested path and the 2nd parameter is a reference to a function to call when the path is requested.

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package main
import (
"fmt"
```

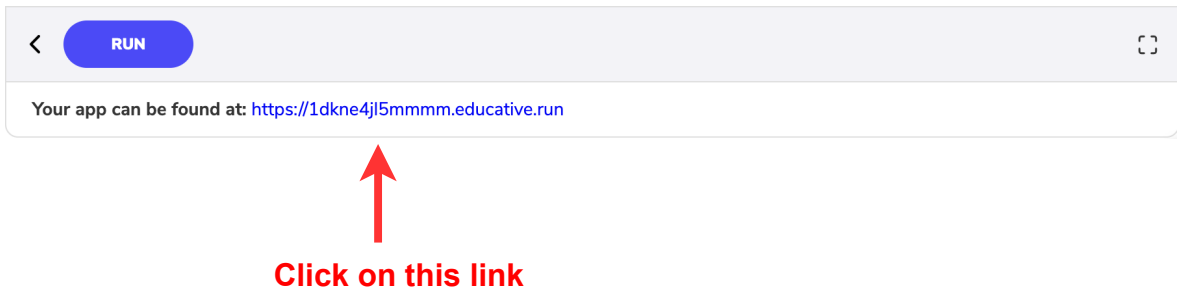
```
"net/http"
"log"
)


func HelloServer(w http.ResponseWriter, req *http.Request) {
  fmt.Println("Inside HelloServer handler")
  fmt.Fprint(w, "Hello, " + req.URL.Path[1:])
}

func main() {
  http.HandleFunc("/",HelloServer)
  err := http.ListenAndServe("0.0.0.0:3000", nil)
  if err != nil {
    log.Fatal("ListenAndServe: ", err.Error())
  }
}
```

> **Remark:** Change **line 15** as `err := http.ListenAndServe("localhost:8080", nil)`, if you're running the server locally.

Click the **Run** button, and wait for the terminal to start. Once it starts, perform the following steps:



Step 1

1 of 4

Broswer opens a page

Step 1

2 of 4

Add "/world" at the end of the URL and reload the page

Step 2

3 of 4

Server appends the variable after **"hello,"**

Step 2

4 of 4

> **Remark:** If you're running locally, open your browser with the address(URL): http://localhost:8080 and then append /**world** at the end of URL. After reloading, in the browser window the text: **Hello, world** appears.

The `fmt.Println` statement prints on the server console; logging what was requested inside every handler could be somewhat more useful.

Now that you're familiar with how to devise a web server, in the next lesson, you'll learn how to poll and read a website with Go.