Internet Protocols — Laboratory 1

Juliusz Chroboczek

30 September 2025

Exercice 1.

1. Create a directory `~/go/src/hello/`. Within this directory, type `go mod init hello`. Which files were created? Examine their contents.
2. Using *Emacs*, crate a file called `hello.go` in the directory that you have created. Check that the buffer is in *Go* mode[1]. Type the following code:

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello, world!")
}
```

3. Format your code using `M-x gofmt RET` in Emacs then save. (If you prefer to work from the command line, you may type `gofmt -w hello.go` instead.)
4. Check your code with the command `go vet`.
5. Compile and execute with the command `go run hello.go` (an executable is created, executed, and deleted immediately).
6. Create an executable with the command `go build`, then execute it using `./hello`.
7. Examine the documentation of the function `fmt.Println` on `https://pkg.go.dev`. Also check `fmt.Print` and `fmt.Printf`.

Exercice 2. Write a Go program that displays the list of prime numbers less than 1000 using the Eratosthenes' sieve algorithm. You may create an array of size n (a *slice*, as it is called in Go) with the following syntax:

```
a := make([]int, n)
```

Do not forget to format your code with `gofmt` and check it with `go vet`.

1. If that is not the case, type `M-x install-package go-mode` and restart Emacs.

Exercice 3. Download the code available at

`https://www.irif.fr/~jch/enseignement/internet/tp1.tar.gz`

Unarchive it and execute it.

1. Using a web browser, connect to `http://localhost:8080/hello.text` and à `http://localhost:8080/hello.html`. Examine both URLs using the command `curl -i`.
2. Modify the code so that the greetings are displayed in your mother tongue.

Exercice 4.

1. Examine the page `http://localhost:8080/name-get`. What's the button for? Why does the URL have no extension?
2. Create a handler for the URL `http://localhost:8080/request-name` that :
   - checks that the method is either `HEAD` or `GET` and returns an error *Method not allowed* if that is not the case
   - calls the method `r.ParseForm()` on the request passed as a parameter to the handler;
   - returns an HTML page that contains the text "Your name is" followed with the name given in parameter `name` of the URL, which you can obtain using the method `r.Form.Get`.

   Test your handler first with the provided web page, then by using `curl -i`.
3. Examine now the page `http://localhost:8080/name-post`, then create a handler at the URL `http://localhost:8080/request-name-post` that:
   - verifies that the method is `POST` and returns an error *Method not allowed* if that is not the case ;
   - displays in the terminal the type of the body of the posted data using `r.Header.Get("Content-Type")`;
   - displays the body of the request using `io.Copy(os.Stdout, r.Body)`.

   Examine the body of the request created by the form.
4. Modify your handler so that the text displayed in the browser is Your name is followed with the name passed in the body. You can use `r.ParseForm()` and `r.Form.Get` as above.

Exercice 5. Write a web server that displays a form that requests an integer *n*, then displays a web page that contains the list of primes between 2 and *n*. You may use the function `strconv.ParseInt` in order to parse the integer passed to the form.

Which method should be used: `GET` ou `POST` ?

Exercice 6.

Modify your server so that it uses HTTPS instead of HTTP, and port 8443 instead of port 8080. You may use the module `github.com/jech/cert`.