Go Academy

#2 Web server

www.3fs.si

Overview

- Web server
  - Handlers
  - Routing
  - Templates
- Best practices

Hello Web example

```go
package main

import "fmt"
import "net/http"

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

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

`http.ResponseWriter`

- Interface used by an HTTP handler to construct an HTTP response
- Its methods:
  - `Header() http.Header`
  - `Write([]byte) (int, error)`
  - `WriteHeader(int)`
- Which means it implements `io.Writer`

`http.Request`

- Represents an HTTP request received by a server
  - Or request to to be sent when using a client
- Some of the fields and methods:
  - `Method string`
  - `URL *url.URL`
  - `Header http.Header`
  - `Body io.ReadCloser`
  - `FormValue(key string) string`
  - `Cookies() []*http.Cookie`

Demo

Templates

- `"html/template"`
- Same as `"text/template"`, but outputs HTML safe against injection
- Understands HTML, CSS, JavaScript and URIs
  - `<a href="/search?q={{ .Query }}">{{ .Query }}</a>`

  -
    ```
    <a href="/search?
      q={{ .Query | urlescaper | attrescaper }}">
      {{ .Query | htmlescaper }}</a>
    ```

Templates

- `{{/* comment */}}`
- `{{p}} //{{.Name}} {{ . }}`
- `{{if p}} T1 {{end}}`
- `{{if p}} T1 {{else}} T2 {{end}}`
- `{{if p}} T1 {{else if p2}} T2 {{end}}`
- `{{range p}} T1 {{end}}`
- `{{range p}} T1 {{else}} T2 {{end}}`
- `{{with p}} T1 {{end}}`
- `{{ .Method argument }}`
- `{{ .Name | ToUpper }}`

Demo

Routing

- `http.ListenAndServe(":8080", nil)`
- `DefaultServeMux`
- Most specific (longest) pattern wins

-
  ```
  http.HandleFunc("/user/new", newUserHandlerFunc)
  http.Handle("/user", userHandler)
  ```
  - /user/new, /user/new/foo will run `newUserHandlerFunc`
  - /user, /user/foo/bar will run `userHandler`

Routing alternatives

- github.com/gorilla/mux
- github.com/husobee/vestigo
- github.com/julienschmidt/httprouter
- github.com/bmizerany/pat

Demo

Best practices

- No hardcoded values
- Code structure
- Makefile
- CI

Code structure

```
├── .gitlab-ci.yml
├── Dockerfile
├── Makefile
├── build
│   ├── cli
│   └── server
├── cmd
│   ├── cli
│   │   └── main.go
│   └── server
│       └── main.go
└── pkg
    └── greeter
        └── service.go
```

No dependency development environment*

- *almost no dependency
- Docker
- Make
  - `builder/%:` target

GitLab CI

- Runners
- `.gitlab-ci.yml`
  - Image
  - Stages
  - Jobs
  - Artefacts

Demo

Challenge to tackle at home

- Refactor to proper directory structure
- Web calculator
  - HTML form

www.3fs.si