# Go HTTP

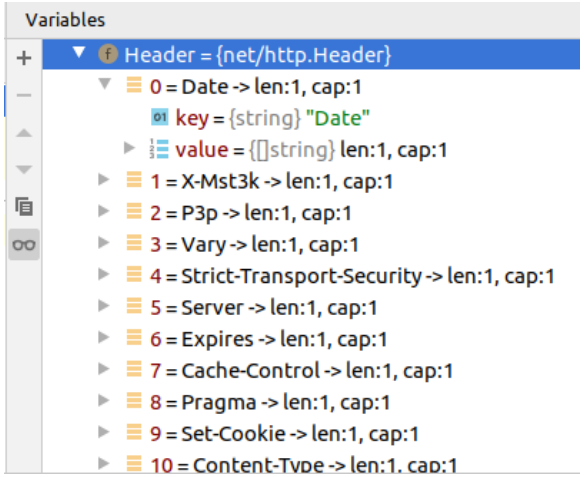
- Još jedna od fantastičnih prednosti Goa je njegov ugrađeni HTTP paket
- Paket sadrži klijentsku i serversku komponentu
- Po dizajnu, Go HTTP server:
  - se izvršava u gorutinama
  - nema blokirajući IO
  - trivijalan za korištenje, ne zahtijeva nikakav callback ili bilo što slično
- Izuzetno visoke out-of-the box performanse
- Intuitivan za korištenje

# Go HTTP

- **Klijent**
- *Get*, *Post* i *PostForm* rade jednostavne HTTP(S) requestove
- Odgovornost klijenta je zatvoriti body nakon što završi manipulaciju istim
- https://play.golang.org/p/HgfvnpBqc5P
- Za veću kontrolu HTTP klijenta (headeri, redirekt policy i slično), potrebno je kreirati I podesiti strukturu *Client*
- https://play.golang.org/p/X32a1c_w58F
- Dodatnu granulaciju klijenta ostvarujemo putem *Transport* i *Dialer* struktura
- Trivijalno je doci do headera odgovora klijentu (napomena – header je koncipiran kao *map[string][]string* pa su potrebne dvije petlje)

# Go HTTP

- https://play.golang.org/p/l14LDhDalFX



# Go HTTP

- Go omogućuje jednostavno korištenje HTTP glagola/metode (GET, POST, PUT, PATCH, DELETE...)
- Za to je potrebno eksplicitno definirati *Client* strukturu
- Poziv se definira *NewRequest* funkcijom
- Nakon eventualne dodatne konfiguracije, potrebno je pozvati *Do* na strukturu
- https://play.golang.org/p/4Ct63BfvQlh
- NewRequest prima tri parametra – naziv *metode*, *URL* i *body*, koji je tipa io.Reader!
- Tek koristeći NewRequeste možemo definirati headere
- https://play.golang.org/p/jvIFxjXUHNp

# Go HTTP

- **Server**
- Ključno - nonblocking IO server, nativno u gorutinama
- Učinkovit i lijep!

```
package main

import (
    "net/http"
)

func root(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("Hello World!"))
}

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

# Go HTTP

- "net/http" ime je paketa u kojem se nalazi HTTP(S) server
- Http.HandleFunc kazuje *http* paketu da obradi sve zahtijeve prema rootu ("/") putem funkcije *root*
- Nakon toga pozivamo *http.ListenAndServe* specificirajući port 8080 (":8080"). Drugi parametar koji je nil nije nam trenunto interesantan.
- Http.ListenAndServe uvijek vraća error, pa ga se može zamotati u log.Fatal() funkciju. No, da pokažemo jednostavnost, ~~*kako to često programeri vole raditi i u produkciji*~~, grešku ignoriramo.
- Funkcija *root* je tipa *http.HandleFunc*. Uzima *http.ResponseWriter* i *http.Request* kao argumente.
- Http.ResponseWriter vrijednost objedinjuje odgovor HTTP servera, pišući u nju vraćamo podatke natrag HTTP klijentu

# Go HTTP

- *http.Request* podatkovna je struktura koja prikazuje klijentski HTTP zahtjev
- **Nadogradimo server!**
- 1. želja - obavijestiti korisnika koji pristupi pathu kojeg nemamo s odgovarajućom greškom
- 2. želja - poslužiti datoteke iz vježbi koje smo odradili na ovom satu
- 3. želja - napraviti endpoint koji će obrađivati samo POST requestove a za druge će vraćati odgovarajuću grešku
- 4. želja - zaprimiti JSON, obraditi ga I vratiti obrađeni JSON natrag klijentu
- 4.5 želja – kontaktirati 3. server i vratiti podatak klijentu
- 5. želja - implementirati CORS