Alati za razvoj softvera

REST servisi u Go-u, go mod

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Setup

- Golang ima nekoliko alata za instalaciju paketa i zavisnosti, ovde će biti korišćen alat koji se zove **go mod** i dostupan je u standardnoj bibliotekci od verzije 1.11
- Go mod je jednostavan za korišćenje i ima nekoliko komandi:
  - go mod init, koristi se da inicijalizuje prazan go.mod file u direktorijumu projekta. Ovaj file sadrži spisak svih potrebnih biblioteka vačem go projektu
  - go get, dobaljna novi biblioteke ili instalira nove
  - go mod tidy, briše biblioteke koje se ne koriste
- Potrebno je instalirati gorilla mux biblioteku i ostale potrebne elemente preko sledeće komande

```
go get -u github.com/gorilla/mux
go get -u github.com/gorilla/handlers
go get github.com/google/uuid
```

Service

- Naš service će vremenom imati dosta elemenata koji mu pomažu
- Iz tog razloga, najbolje je da našervice definišemo kao struturu
- Kasnije možemo jednostavno da proširimo sa stvarima koje nam trebaju

```
type postServer struct {
        data map[string]*RequestPost // izigrava bazu podataka
}
```

Model podataka

- JSON podatke koje dobijamo od korisnika možemo da prebacimo u go strutkure za jednostavniji rad

```
type RequestPost struct {
        Id     int      `json:"id"`
        Title  string   `json:"title"`
        Text   string   `json:"text"`
        Tags   []string `json:"tags"`
}
```

Zahtevi klijenata

- Napravićemo funkciju koja dekodira dolazni *JSON* (text) u Go strukturu koja ga modeluje

```
func decodeBody(ctx context.Context, r io.Reader) (*RequestPost, error) {
        dec := json.NewDecoder(r)
        dec.DisallowUnknownFields()

        var rt RequestPost
        if err := dec.Decode(&rt); err != nil {
                return nil, err
        }
                return &rt, nil
}
```

Odgovor klijentu

- Napraviće funkciju koja Go strukture prebacuje u *JSON* (text) i vraća korisniku

```
func renderJSON(ctx context.Context, w http.ResponseWriter, v interface{}) {
        js, err := json.Marshal(v)

        if err != nil {
                http.Error(w, err.Error(), http.StatusInternalServerError)
                return
        }

        w.Header().Set("Content-Type", "application/json")
        w.Write(js)
}
```

Endpoints

- Nije redak slučaj da funkcije koje treba da izvršavaju operacije web service-a, budu *nakačene* na server strukturu
- To je lepa praksa i možemo funkcije spakovati u jedan paket *handlers*
- Da bi to uradili, naše funkcije treba da imaju odgovarajući potpis (treba da imaju odgovarajuće parametre)

```
func(w http.ResponseWriter, r *http.Request)
```

- I treba da definišemo endpoint koji će prozvati tu funkciju kada se pogodi

Endpoints — Create (POST)

```
func (ts *postServer) createPostHandler(w http.ResponseWriter, req *http.Request) {
        contentType := req.Header.Get("Content-Type")
        mediatype, _, err := mime.ParseMediaType(contentType)
        if err != nil {
                http.Error(w, err.Error(), http.StatusBadRequest)
                return
        }

        if mediatype != "application/json" {
                err := errors.New("Expect application/json Content-Type")
                http.Error(w, err.Error(), http.StatusUnsupportedMediaType)
                return
        }

        rt, err := decodeBody(req.Body)
        if err != nil {
                http.Error(w, err.Error(), http.StatusBadRequest)
                return
        }

        id := createId()
        rt.Id = id
        ts.data[id] = rt
        renderJSON(w, rt)
}
```

Endpoints — Get All (GET)

- Treba nam Endpoint koji vraća sve postove
- Pošto tražimo od servisa neke podatke pravimo GET zahtev

```
func (ts *postServer) getAllHandler(w http.ResponseWriter, req *http.Request) {
        allTasks := []*RequestPost{}
        for _, v := range ts.data {
                allTasks = append(allTasks, v)
        }
        renderJSON(w, allTasks)
}
```

Endpoints — Get Post (GET)

- Treba nam Endpoint koji vraća pojedinačan post preko ID-a
- Pošto tražimo od servisa neke podatke pravimo GET zahtev

```
func (ts *postServer) getPostHandler(w http.ResponseWriter, req *http.Request)
        id := mux.Vars(req)["id"]
        task, ok := ts.data[id]
        if !ok {
                err := errors.New("key not found")
                http.Error(w, err.Error(), http.StatusNotFound)
                return
        }
        renderJSON(w, task)
}
```

Endpoints — Delete Post (DELETE)

- Treba nam Endpoint koji briše post preko ID-a
- Pošto tražimo od servisa da obriše podatke pravimo DELETE zahtev

```
func (ts *postServer) delPostHandler(w http.ResponseWriter, req *http.Request)
        id := mux.Vars(req)["id"]
        if _, ok := ts.data[id]; ok {
                delete(ts.data, id)
                renderJSON(w, "Deleted")
        } else {
                err := errors.New("key not found")
                http.Error(w, err.Error(), http.StatusNotFound)
        }
}
```

Rutiranje

- Moramo napraviti rutire koji će rutirati odgovarajuće zahteve na odgovarajući handler — endpoint

```
func main() {
        router := mux.NewRouter()
        router.StrictSlash(true)

        server := postServer{
                data: map[string]*RequestPost{},
        }
        router.HandleFunc("/post/", server.createPostHandler).Methods("POST")
        router.HandleFunc("/posts/", server.getAllHandler).Methods("GET")
        router.HandleFunc("/post/{id}/", server.getPostHandler).Methods("GET")
        router.HandleFunc("/post/{id}/", server.delPostHandler).Methods("DELETE")

        loggedRouter := handlers.LoggingHandler(os.Stdout, router)
        http.ListenAndServe("0.0.0.0:8000", loggedRouter)
}
```

Graceful shutdown

```
func main(){
        quit := make(chan os.Signal)
        signal.Notify(quit, os.Interrupt, syscall.SIGTERM)

        router := mux.NewRouter()
        router.StrictSlash(true)

        server := postServer{
                data: map[string]*RequestPost{},
        }

        router.HandleFunc("/post/", server.createPostHandler).Methods("POST")
        router.HandleFunc("/posts/", server.getAllHandler).Methods("GET")
        router.HandleFunc("/post/{id}/", server.getPostHandler).Methods("GET")
        router.HandleFunc("/post/{id}/", server.delPostHandler).Methods("DELETE")

        // start server
        srv := &http.Server{Addr: "0.0.0.0:8000", Handler: router}
        go func() {
                log.Println("server starting")
                if err := srv.ListenAndServe(); err != nil {
                        if err != http.ErrServerClosed {
                                log.Fatal(err)
                        }
                }
        }()
```

Graceful shutdown nastavak

```
        <-quit

        log.Println("service shutting down ...")

        // gracefully stop server
        ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
        defer cancel()

        if err := srv.Shutdown(ctx); err != nil {
                log.Fatal(err)
        }
        log.Println("server stopped")
}
```

Pozivi

- Kada se je naš servis podignut, postoji nekoliko načina kako mu možemo pristupiti
- cURL, Iz browser-a, terminala, Postman
- Jednostavan alat koji nam omogućava da radimo (pozivamo) naše servise je Postman
- Postman je UI alat kojim vrlo jednostavno možemo pozivati endpoint-e naših servisa, slati parametre itd

Dodatni materijali

- Graceful shutdown
- Graceful shutdown in go
- REST services Red Hat
- JSON
- Context package
- HTTP Request And Response

Kraj predavanja

Pitanja? :)