Alati za razvoj softvera

REST servisi, JSON, Context, Graceful shutdown

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

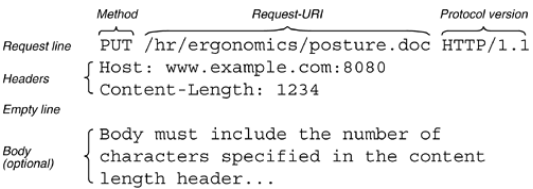
Uvod

- Fokus REST web service-a je na resursima, i kako omogućiti pristup tim resursima preko interneta
- Resurs moze biti predstavljen kao objekat u memoriji, file na disku, podaci iz aplikacije, baze podataka itd.
- Prilikom dizajniranja sistema prvo je potrebno da identifikujemo resurse, a zatim ustanoviti kako su oni povezani
- Ovaj postupaka je sličan modelovanju baze podataka

- Kada smo identifikovali resurse, sledeći korak je da uspostavimo način kako da te resurse reprezentujemo u našem sistemu
- Za te potrebe mozemo koristiti bilo koji format za reprezentaciju resursa (JSON, XML npr.).
- Da bi dobili sadržaj sa udaljene lokacije (od nekog server-a), client mora napraviti HTTP zahteh, i poslati ga web service-u
- Nakon svakog HTTP zahteva, sledi i HTTP odgovor od server-a ka client-u tj. onome ko je zahtev poslao
- Client može biti korisnik, ili može biti neka druga aplikacija (npr. drugi web service, mobilna aplikacija, ...)

HTTP zahtev

- Svaki HTTP zahtev se sastoji od nekoliko elemenata:
  - **[METOD]** GET, PUT, POST itd. odnosno koju operaciju želmo da uradimo nad podacima
  - **[URL/HOSTNAME]** Putanja do resursa nad kojim će operacija biti izvršena
  - **[HEADERS]** Dodatni podaci koji se šalju serveru
  - **[BODY]** Dodatni podaci koji treba da se pošalju serveru da bi opreracija (npr POST, PUT) bila uspešno izvršena



(HTTP Request)

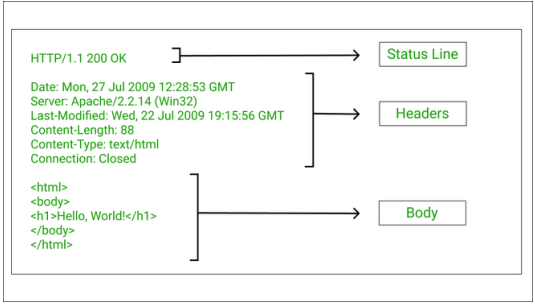
Slanje zahteva

- Svaki servis mora imati jedinstvenu adresu (URL) na koju šaljemo HTTP zahtev na primer:
  - http://MyService/Persons/1
- Ako zelimo da izvedemo nekakav upit nad web service-om, to možemo da uradimo koristeći HTTP METOD i parametrizaciom putanje web service-a
- Česta opcija je upotreba *?* simbol na kraj putanje (ova opcija više simbolizuje RPC nego REST zahtev, ali se može koristiti)
- Nakon specijanog simbola, slede parovi u obliku *key=value* spojeni & simbolom ako tih parametara ima vise od jednog na primer:
  - http://MyService/Persons/1?format=json&encoding=UTF8

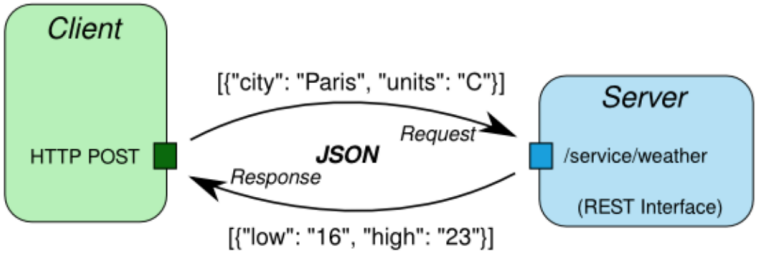
- Servisi koje pozivamo preko internet imaju unapred definisanu strukturu
  - tačnu putanju
  - metod kojim se pozivaju
  - podatke koje očekuju
  - kako se pretražuju
  - itd.
- Ako mi implementiramo web service-e, onda mi namečemo ova pravila
- Pojedinačna operacija nad resursom se često naziva *endpoint*
- Endpoint obično reprezentuje jedan resurs u sistemu

HTTP odgovor

- Nakon zahteva, klijent dobija HTTP odgovor nazad od servisa
- Svaki HTTP odgovor se sastoji od nekoliko elemenata:
  - **[STATUS]** nam govori da li je prethodno zahtevana operacija izvršena uspesno, ili ne. Status code je reprezentovan celim brojem I to:
    - Success 2xx sve je prošlo ok
    - Redirection 3xx desio se redirect na neki drugi servis
    - Error 4xx, 5xx javila se greska
  - **[HEADERS]** dodatni podaci poslati od servera
  - **[BODY]** konkretni podaci



(HTTP Response)


Client
HTTP POST
[{"city": "Paris", "units": "C"}]
Request
JSON
Response
[{"low": "16", "high": "23"}]
Server
/service/weather
(REST Interface)

Graceful shutdown

- Kada korisnik ili operativni sistem prekinu izvršavanje web service-a, mi ne moramo istog momenta da ugasimo naš web service, zato što možda postoje akcije koje se nisu završile, a trebalo bi
- Možda tim akcijama treba dodatno vreme da sačuvaju svoje stanje koje bi inače bilo izgubljeno
- Dobra je praksa, da se web service-u da neko dodatno vreme u kom on neće prihvatati nove zahteve, ali će ostali elementi moći da urade svoj posao
- Na primer, završe svi odgovori korisnicima, završi interkacija sa bazom, sašuvaju logovi itd.

- Za ovaj mehanizam možemo da iskoristimo sistemke pozive (i neke Go-ove pogodnosti)
- Kada dobijemo specifičan sistemski signal možemo da reagujemo na njega
- Ovo postaje bitno kada budemo radili sa kontejnerima, gde drugi sistemi mogu da ugase vašu aplikaciju u svakom momentu
- Mi možemo da zaustavimo web service da više ne prihvata zahteve, ali ostalim akcijama damo proizvoljno vreme da urade sve što treba pre nego što se program načisto zaustavi

Uvod

- JavaScript Object Notation (JSON) je format za laksu razmenu podataka, kao i XML nezavistan je od programskog jezika i tehnologije koja se koristi
- Podaci se zapisuju kao parovi **ključ:vrednost**
- Ključ se navodi kao tekst pod duplim navodnicima nakon cega sledi vrednost na primer:
  - "firstName":"John"
- JSON vrednosti su unapred definisane i možmeo izabrati iz konačnog skupa opcija
- JSON podseća na rad sa mapa u drugim programskim jezicima

Unapred definisane vrednosti

- JSON vrednosti mogu biti neke od unapred definisanih:
  - Broj (integer or floating point)
  - String (in double quotes)
  - Boolean (true or false)
  - Niz (koristi uglaste zagrade)
  - Objekat (koristi vitičaste zagrade)
  - Null

Konstrukcija

- JSON objekat se zapisuje u parovima **ključ:vrednost** koji se nalaze unutar vitičastih zagrada:
  - { "firstName":"John", "lastName":"Doe" }
- JSON niz sadrži ključ, nakon čega sledi niz elemenata u uglasim zagradama:

```
{
        "employees": [
                {"firstName": "John","lastName": "Doe"},
                {"firstName": "Anna","lastName": "Smith"},
                {"firstName":"Peter","lastName":"Jones"}
        ]
}
```

Napomena

- JSON može da kombinuje razne tipove podataka unutar jednog niza
- O tome treba voditi računa kada koristimo strogo tipizirane jezike da ne bi došlo do problema prilikom konverzije
- Mešanje tipova treba izbegavati, osim ako nemate baš jaku potrebu sa time
- U tom slučaju trebate naći način da rešite ovaj problem
- Ako koristite dinamičke jezike, ovo nije tako veliki problem

Uvod

- Golang u svojoj standardnoj biblioteci ima već podršku za implementaciju mrežnih aplikacija, samim tim i web service-a
- Standardna biblioteka je sasvim dovoljna za implementaciju
- Da bi olakšali sebi posao možemo koristiti neki od dostupnih bibliteka Gin, Gorilla, itd.
- U primerima biće korišćena biblioteka Gorilla
- Vi za projekat možete slobodno d koristite bilo koji framework koji želite

Context paket

- Context paket je jako bitan paket i nalazi se unutar go-ove standardne biblioteke
- Ovaj tip, nam pruža dosta stvari koje su jako bitne za mrežnu, procesnu komunikaciju unutar aplikacija
- Tip Context, se dosta koristi zato sto nam pruža jedinstvenu mogućnost da prenosimo podatke, signale za prekid izvršavanja svih povezanih učesnika u komunikaciji
- Kada zahtev stigne na web service i kada krenemo izvršavanje kod-a možemo da posaljemo i podatak tipa Context i na taj način stvaramo graf poziva drugih funkcija (npr logovanje aktivnosti, upiti ka bazi, util operacije itd.)

- Ako nekakav zahtev traje predugo, ili client prosto odustane od zahteva ili nešto slično, context nam pruža mogućnosti da prekinemo zahtev kao i upite nad bazom i sve ostale zavisne pozive koji su se desili (pod uslovom da je context iskorišćen)
- Ako na primer znamo da web service treba da odgovori u roku od 30s, a on to ne uradi...context može da emituje event kojim se svi ostali zavisni pozivi prekidaju (tj. bivaju obavešteni o prekidu izvršavanja) i korisniku možemo da vratimo odgovor da rezultata nema ili da server je zauzet ili nešto treće
- Ovaj paket možemo da koristimo implementiramo mehanizam našeg web service-a uz još par dodataka (u nastavku)

Kreiranje Context-a

- Postoji nekoliko verzija context-a sa kojima možemo da radimo
- Da bi napravili prazan context, to možemo uraditi pozivom *Background* funkcije ili *TODO*

```
context.Background() ctx
context.TODO() ctx
```

- Ova dva Context-a koristiti **samo** kao Context najvišeg nivao!

- Ostali Context-i se kreiraju od nekog postojećeg roditeljskog Context-a

```
context.WithDeadline(parent Context, d time.Time) (ctx Context, cancel CancelFunc)
context.WithTimeout(parent Context, timeout time.Duration) (ctx Context, cancel CancelFunc)
context.WithCancel(parent Context) (ctx Context, cancel CancelFunc)
context.WithValue(parent Context, key, val interface, key string) ctx
```

- Ove context elemente koristimo spram potreba, da li nešto treba sami da prekinemo (cancel), želimo da se zasutavi automatski nakon nekog vremena (deadline, timeout) ili samo želmo da prenesemo podatke kroz stablo context-a (value)
- Ove Context-e možemo da koristimo kao Context najvišeg nivao, i to je često i slučaj

Dodatni materijali

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

Kraj predavanja

Pitanja? :)