Serverske veb tehnologije
- REST -

Dragan Ivanović

Katedra za informatiku, Fakultet Tehničkih Nauka, Novi Sad

2022.

REST vs RESTful

- REST je skup principa koji definišu kako da se koriste standardi (HTTP, URI, ...)
- Pretpostavka je da će aplikacija koju razvijamo dobro koristiti arhitekturu Weba ako se pridržavamo ovih principa
- RESTful je implementacija servisa poštujući REST principe
- REST je arhitektura, RESTful je pridev koji ide uz reč servis

REST principi

- Svaki resurs treba da ima svoj ID
- Resursi treba da budu povezani
- Treba koristiti standardne HTTP metode
- Resursi treba da imaju višestruke reprezentacije
- Komunikacija treba da bude stateless

ID resursa

- Na Webu, postoji unifikovan koncept ID-a: URI
- URI predstavlja globalni prostor imena i korišćenje URI-a za identifikovanje resursa u aplikaciji znači da resurs ima globalni jedinstveni ID
- Resursi ne moraju biti entiteti iz baze, najčešće su resursi apstraktniji od entiteta u bazi.
- Primeri identifikatora:

```
http://example.com/customers/1234
http://example.com/orders/2007/10/776654
http://example.com/products/4554
http://example.com/processes/salary-increase-234
```

- Identifikatori mogu da se dodele i kolekcijama resursa - formalno i kolekcija resursa je resurs

Povezanost resursa

- "**H**ypermedia **a**s **t**he **e**ngine **o**f **a**pplication **s**tate" (HATEOAS)
- Linkovi pružaju informaciju potrebnu da se dinamički navigira kroz REST interfejs
- Linkovi mogu biti ugrađeni u odgovore:

```
<order self='http://example.com/customers/1234' >
  <amount>23</amount>
  <product ref='http://example.com/products/4554' />
  <customer ref='http://example.com/customers/1234' />
</order>
```

Povezanost resursa

- Linkovi mogu da pokazuju na resurse koje nude različite aplikacije
- Umesto jedne monolitne aplikacije možemo da imamo međusobno nezavisne servise namenjene pojedinim tipovima resursa koji se razvijaju, deplojuju, održavaju, ... nezavisno
- Pošto server nudi skup linkova klijentu, klijent može da prelazi iz jednog stanja aplikacije u drugo prateći linkove - ne moramo da imamo stanje aplikacije na serveru!

Korišćenje standardnih HTTP metoda

- Standardne HTTP metode imaju jasno definisanu semantiku i treba ih koristiti u skladu sa tom semantikom (http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html):
  OPTIONS – zahtev za informacijama o dostupnim opcijama komunikacije u request/response lancu.
  GET – preuzimanje resursa identifikovanog URI-jem
  HEAD – isto kao i GET, samo što odgovor ne sme da sadrži sam resurs (telo odgovora)
  POST – postavljanje resursa dok još ne postoji identifikator samog resursa - kreiranje resursa pri čemu mu server dodeljuje identifikator.
  PUT – postavljanje resursa kada postoji identifikator - ukoliko resurs sa zadatim identifikatorom postoji izmeniće se, a ukoliko ne postoji, kreiraće se resurs i postaviti mu se identifikator.
  DELETE – brisanje resursa.

Korišćenje standardnih HTTP metoda

- Prednosti:
  - Resursi u našoj aplikaciji postaju dostupni kao i bilo koji drugi resursi na Webu
  - Bilo koja aplikacija koja „razume“ HTTP protokol može da koristi našu aplikaciju

Višestruka reprezentacija resursa

- Pored metoda i resursa, klijent u zahtevu specificira i format u kom resurs treba da bude predstavljen.
- Primer:

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: application/json
```

- Prednosti:
  - Klijentska aplikacija koja „zna“ HTTP protokol i skup formata podataka može da komunicira sa bilo kojom REST aplikacijom
  - Ako REST aplikacija nudi HTML reprezentacije resursa, bilo koji brauzer može da bude klijent za tu aplikaciju

Stateless komunikacija

- Ne znači da u REST aplikacijama ne postoji stanje!
- Inače bi REST aplikacije bile neupotrebljive
- Stanje može da bude:
  - stanje resursa
  - prebačeno na klijenta
- Server ne treba da čuva stanje komunikacije za klijenta
- Prednosti:
  - Skalabilnost – broj klijenata ima velik uticaj na footprint (deljenje resursa) servera ako se za svakog klijenta čuva stanje
  - Izolovanje klijenta od promena na serveru – pošto ne postoji stanje klijenta na serveru, klijentu nije važno kom konkretnom serveru se obraća.

Primeri iz loše prakse

- Zloupotreba GET metode
- Zloupotreba POST metode
- Ignorisanje kodova odgovora
- Zloupotreba cookie-a
- Zanemarivanje HATEOAS
- Ignorisanje MIME tipova
- Zanemarivanje samo-deskriptivnosti servisa

Zloupotreba GET metode

- Miskoncepcija: *REST znači koristiti HTTP da se ponude neke od funkcionalnosti aplikacije*
- Mnoge biblioteke za rad sa HTTP čine veoma lakim da se URI posmatra ne kao identifikator resursa, nego i kao zgodan način da se enkoduju parametri. Na primer: http://example.com/user?method=delete&id=1234
- Zašto je anti-patern raširen
  - Lako se implementira
  - Lako se testira (ne treba nam Postman ili Advanced REST Client)
- Ključni problemi
  - URI ne identifikuje resurs, nego enkodira operaciju i parametre
  - GET metoda se ne koristi u skladu sa njenom semantikom
  - Ovi linkovi tipično nisu namenjeni za bookmark
  - Crawler može da načini nepoporavljivu štetu aplikaciji (na primer da pobriše podatke iz aplikacije)

Zloupotreba POST metode

- Slično kao zloupotreba GET metode
- U tipičnom scenariju koristi se samo jedan URI za slanje poruka i različiti parametri za različite operacije

Ignorisanje kodova odgovora

- Status kodovi nisu jedino 200, 404 i 500
  http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- Na primer, 201 znači da je resurs kreiran i da je njegov URI u Location zaglavlju odgovora
- Specijalni slučaj ovog anti-paterna: zloupotreba status koda 200:
  - Sa svakim odgovor se šalje kod 200
  - U telu odgovora se nalazi poruka o grešci
- Ključni problemi:
  - smanjuje se re-use
  - interoperabilnost

Zloupotreba Cookie-a

- Miskoncepcija: *Pomoću cookie klijent može da se poveže sa stanjem aplikacije na serveru*
- Ne znači da se u REST aplikacijama ne koriste cookie.
- Sve dok postoje stanje resursa i stanje klijentske aplikacije, a ne postoji stanje aplikacije u redu je koristiti cookie
- Na primer, u redu je da cookie sadrži authentication token koji server koristi da utvrdi identitet klijenta, ali bez oslanjanja na stanje aplikacije

Zanemarivanje HATEOAS

- REST aplikacija često postane CRUD Web aplikacije
  - Koristi HTTP metode u skladu sa njihovom semantikom
  - Zanemarili smo HATEOAS
- Prvi indikator ovog antipaterna je odsustvo linkova iz reprezentacija resursa
- Srodan anti-patern je insistiranje na čoveku čitljivim URI-jima

Ignorisanje MIME tipova

- Prilikom razvoja REST aplikacije često se ignoriše mogućnost da će tu aplikaciju koristiti veći broj klijenata
- Fiksiramo se samo za format reprezentacije resursa koji koristi klijent kome je aplikacija primarno namenjena

Zanemarivanje samo-deskriptivnosti servisa

- Toliko čest anti-patern da je prisutan gotovo u svim REST aplikacijama
- Idealno: *svaki HTTP zahtev i odgovor (uključujući zaglavlja i telo) bi tebao da sadrži dovoljno informacija da bilo koji generički klijent ili server može da ga obradi*

Ričardsonov model zrelost

- Mehanizam kojim se ocenjuje API prema REST principima
- Što se API više pridržava REST principa, to ima višu ocenu
- Ocene od 0 do 3
- Dobar način da se razmišlja o REST aplikacijama
- Dobar korak-po-korak vodič za razumevanje REST aplikacija, kao i mehanizam da se stvari poprave u aplikacijama koje već postoje

JavaScript Object Notation

- Laki (lightweight) format za razmenu podataka
- Jednostavan
  - Za ljude koji ga pišu
  - Za mašine koje ga procesiraju
- JSON je tekstualni format koji je nezavistan od programskog jezika

Tipovi podataka

- Number
- String – sekvenca karaktera
- Boolean – true ili false
- Objekat – neuređen asocijativni niz (ključ/vrednost)
- Niz – uređena sekvenca nula ili više vrednosti
- null – “prazna vrednost”

JSON objekat

- Predstavlja neuređenu kolekciju parova naziv/vrednost
- JSON objekat počinje sa otvorenom vitičastom zagradom ({), a završava sa zatvorenom vitičastom zagradom (})
- Naziv i vrednost su razdvojeni dvotačkom (:), a parovi naziv/vrednost su razdvojeni zapetom (,)
- Primer:

```
{
  "ime": "Marko",
  "prezime": "Marković",
  "godinaRodjenja": 1982
}
```

JSON niz

- JSON niz predstavlja uređenu sekvencu JSON objekata
- Počinje simbolom [, a završava simbolom]
- Objekti su razdvojeni zapetom
- Primer:

```
[
  {
    "ime": "Marko",
    "prezime": "Marković",
    "godinaRodjenja": 1982
  },
  {
    "ime": "Petar",
    "prezime": "Petrović",
    "godinaRodjenja": 1986
  }
]
```

JSON Schema

- Format zasnovan na JSON-u za definisanje strukture i validaciju JSON podataka
- Primer:

```
{
  "$schema": "http://json-schema.org/draft-03/schema#",
    "name": "Osoba",
    "type": "object",
    "properties": {
      "ime": {
      "type": "string",
      "required": true
      },
   ...
}
```

MIME Type

- Zvanični MIME tip koji se koristi za JSON je application/json
- Razni servisi podržavaju i nezvanične MIME tipove text/json ili text/javascript

Java i JSON

- Biblioteke u Javi za manipulisanje JSON-om:
  - org.json
  - google-gson
  - **Jackson**
  - ...

RESTful servisi i Java

- Java API for RESTful Web Services: JAX-RS
- Implementacije:
  - Jersey
  - Apache CXF
  - RESTEasy
  - Restlet
  - Apache Wink
- Pisanje servisa pomoću anotiranih Java klasa

Resurs

- Resurs = anotirana POJO klasa

```
@Path("/users")
public class UserService {
...

  @Path("/products")
  public class ProductService {
  ...
```

Operacije

- Operacije = anotirane metode u resurs klasi

```
@Path("/users")
public class UserService {

  @GET
  public List<User> findAll() { ... }

  @POST
  public User create(User user) { ... }

  @PUT
  @Path("{id}")
  public User update(User user) { ... }

  @DELETE
  @Path("{id}")
  public String remove(@PathParam("id") String id) { ... }
  ...
```

URI promenljive

- Prijem parametara iz URI-ja
- Može i više parametara odjednom

```
@Path("/users")
public class UserService {

  @DELETE
  @Path("{id}")
  public String remove(@PathParam("id") String id) { ... }
  ...
```

Različiti formati podataka

- Ista operacija može primiti podatke u različitim formatima

```
@Path("/users")
public class UserService {

  @PUT
  @Path("{id}")
  @Consumes("application/xml")
  public User updateXML(User user) { ... }

  @PUT
  @Path("{id}")
  @Consumes("application/json")
  public User updateJSON(User user) { ... }
  ...
```

Različiti formati podataka

- Ista operacija može **vratiti** podatke u različitim formatima

```
@Path("/users")
public class UserService {

  @PUT
  @Path("{id}")
  @Consumes("application/xml")
  @Produces("application/xml")
  public User updateXML(User user) { ... }

  @PUT
  @Path("{id}")
  @Consumes("application/json")
  @Produces("application/json")
  public User updateJSON(User user) { ... } ...
```

Prijem podataka iz HTML formi

- Primer HTML forme:

```
<form action="users" method="POST">
  Username: <input type="text" name="username"/>
  Password: <input type="text" name="password"/>
  Firstname: <input type="text" name="firstname"/>
  ...
</form>
```

Prijem podataka iz HTML formi

```
@Path("/users")
public class UserService {

  @POST
  @Consumes("application/x-www-form-urlencoded")
  public User create(
  @FormParam("username") String username,
  @FormParam("password") String password,
  @FormParam("firstname") String firstname,
  ...
  ) { ... }

  ...
```

Primer REST API-ja

- osa.pr25.service
- info: osa/pr25/readme.txt

Spring Boot podrška za REST

- Servis se definiše kreiranjem klase anotirane kao *@RestController*
- Servis će biti javno dostupan putem URL definisanog anotacijom *@RequestMapping*
- Metode servisa se implementiraju kao metode klase
  - *@RequestMapping* anotacija definiše putanju do konkretne metode
  - Putanja može da sadrži dinamičke vrednosti
    - Metodi se automatski prosleđuju kao parametri označeni anotacijom *@PathVariable*

Spring Boot podrška za REST

- Povratna vrednost metode se serijalizuje u željeni format za prenos preko mreže
  - Povratna vrednost su objekti klase *ResponseEntity*
- Parametri HTTP zahteva se automatski parsiraju
  - Kreira se parametar metode
  - Parametar se označeni anotacijom *@RequestParam*
- Telo HTTP zahteva se automatski parsira
  - Kreira se parametar metode
  - Parametar se označi anotacijom *@RequestBody*
- Parametri i telo se automatski deserijalizuju u tip koji odgovara tipu parametra metode

Primer REST API-ja u Spring-u

- osa.spring.pr25.controller
- info: osa/spring/pr25/readme.txt

Java klijent za RESTful web servis

- Potrebni sastojci:
  - Rukovanje HTTP konekcijama
  - Parsiranje XML-a ili JSON-a

Twitter klijent

```
URL twitter = new URL(
    "http://twitter.com/statuses/public_timeline.xml");
URLConnection tc = twitter.openConnection();
BufferedReader in = new BufferedReader(
    new InputStreamReader(tc.getInputStream(), "UTF8"));
```

Odgovor Twittera

```
<?xml version="1.0" encoding="UTF-8"?>
<statuses type="array">
  <status>
    <created_at>Fri May 20 18:49:46 +0000 2011</created_at>
    <id>71648829237248000</id>
    <text>So high school is done. LET THE PARTY BEGIN</text>
    <truncated>false</truncated>
    <favorited>false</favorited>
    <in_reply_to_status_id></in_reply_to_status_id>
    <in_reply_to_user_id></in_reply_to_user_id>
    <in_reply_to_screen_name></in_reply_to_screen_name>
    <retweet_count>0</retweet_count>
    <retweeted>false</retweeted>

    <user>
      <id>250394499</id>
      <name>Taylor Stricklin</name>
      <screen_name>TaylorStricklin</screen_name>
  ...
```

Odgovor Twittera u raznim formatima

- XML:
  `http://twitter.com/statuses/public_timeline.xml`
- JSON:
  `http://twitter.com/statuses/public_timeline.json`
- RSS:
  `http://twitter.com/statuses/public_timeline.rss`
- ATOM:
  `http://twitter.com/statuses/public_timeline.atom`

Korisne biblioteke

- Apache HttpComponents: precizna i detaljna implementacija HTTP protokola sa klijentske strane
- Apache Commons Codec: konverzija različitih formata

Klijent sa Apache bibliotekama

```
HttpClient client = new HttpClient();
GetMethod get = new GetMethod(
  "http://twitter.com/statuses/public_timeline.json");
int statusCode = client.executeMethod(get);
if (statusCode == HttpStatus.SC_OK) {
   ... method.getResponseBody() ...
```

JQuery klijent

```
$.ajax({
  url: "http://localhost:8080/pr25/api/categories",
  dataType: "json",
  success: function(response) {
    for(var i=0; i<response.length; i++) {
       category = response[i];
       alert(category.id);
       ...
    }
  },
  error: function(request, options, error) {
    alert(error);
  }
});
```

Postman

- Klijent za pisanje HTTP zahteva i prijem HTTP odgovora
- Može da se instalira kao dodatak u GoogleChrome i Firefox

Advanced REST Client

- Klijent za pisanje HTTP zahteva i prijem HTTP odgovora
- Može da se instalira kao dodatak u GoogleChrome
- Ima slične funkcionalnosti kao i Postman
- Poredjenje *Postman*-a i *Advanced REST Client*-a - link