

RESTful Web servi

Šta je REST?

- REST = Representational State Transfer
- Stil softverske arhitekture namenjen distribuiranim hipermedijalnim sistemima, kao što je World Wide Web
- Termin „skovao“ Roy Fielding u svojoj disertaciji na University of California, Irvine.

Šta je REST?

- Representational State Transfer (REST) - stil arhitekture sistema koji specificira određena ograničenja:
 1. Način identifikacije resursa
 2. uniformisani interfejs
GET, PUT, DELETE, POST (HEAD, OPTIONS...)
 3. Samoopisive poruke
 4. Stanje aplikacije upravljano hipermedijom (linkovi predstavljaju endpointe kojim se manipuliše resursima, manipulacija resursima menja tekuće stanje aplikacije)
 5. Stateless interakcije
- Primena ovih ograničenja na web servise pojačava pozitivne osobine, kao što su performanse, skalabilnost, izmenjivost.

Šta je REST?

- U REST arhitekturi, podaci i funkcionalnost se posmatraju kao resursi i pristupa im se putem Uniform Resource Identifiers (URIs), tipično kao linkovima na web-u.
- Resursima se manipuliše primenom skupa jednostavnih, dobro definisanih operacija.
- REST arhitektura je klijent/server i dizajnirana je da koristi stateless komunikacioni protokol, tipično HTTP.
- klijent i server razmenjuju **reprezentacije** resursa koristeći pri tome standardizovan interfejs i protokol.

Dizajn REST aplikacije / metodologija

1. Identifikovanje resursa koji treba da su vidljivi kao servis (npr. godišnji izveštaji, katalozi knjiga, porudžbine...)
2. Modelovanje relacija između resursa kao hiperlinkova koje je moguće slediti kako bi se dobilo više detalja (ili kako bi se izvela promena stanja resursa)
3. Deifinisati „lepe“ URI-je za adresiranje resursa
4. Razumeti smisao izvršavanja GET, POST, PUT, DELETE zahteva na svaki od resursa (i sa li su svi i dozvoljeni za svaki resurs)
5. Dizajnirati i dokumentovati reprezentaciju resursa (može biti više)
6. Implementirati i postaviti na web server
7. Testirati (browser, PostMen...)

	GET	PUT	POST	DELETE
/loan	✓	✓	✓	✓
/balance	✓	✗	✗	✗
/client	✓	✓	✓	✗
/book	✓	✓	✓	✓
/order	✓	?	✓	✗
/soap	✗	✗	✓	✗

REST aplikacije / prostor modelovanja i razvoja

M Representations (Variable)

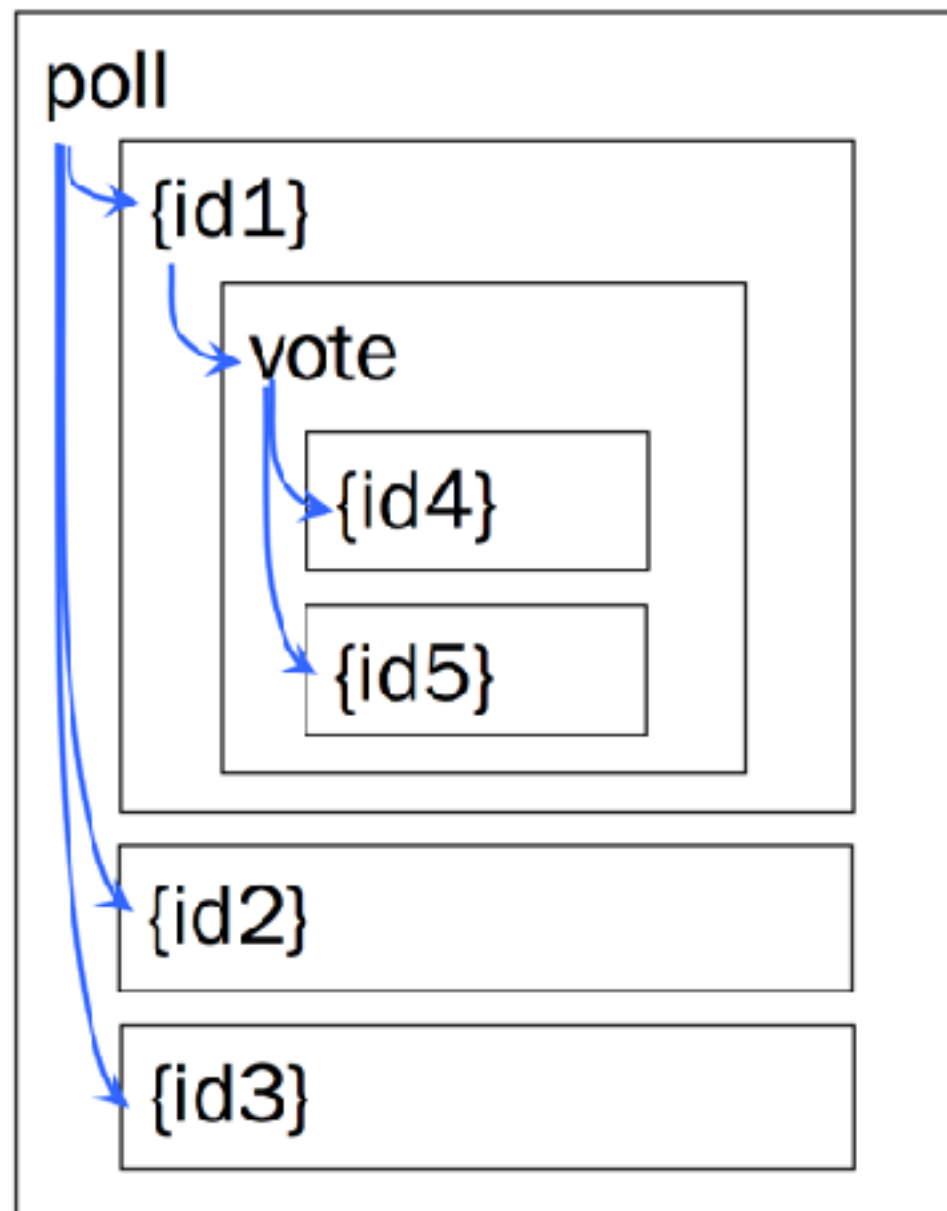
4 Methods (Fixed)

N Resources (Variable)

	GET	PUT	POST	DELETE
/loan	✓	✓	✓	✓
/balance	✓	✗	✗	✗
/client	✓	✓	✓	✗
/book	✓	✓	✓	✗
/order	✓	?	✓	✓
				✗
/soap	✗	✗	✓	✗

Primer simplifikovanog Doodle API-ja

1. Resursi su:
anketa i glasovi
2. Relacije između resursa:



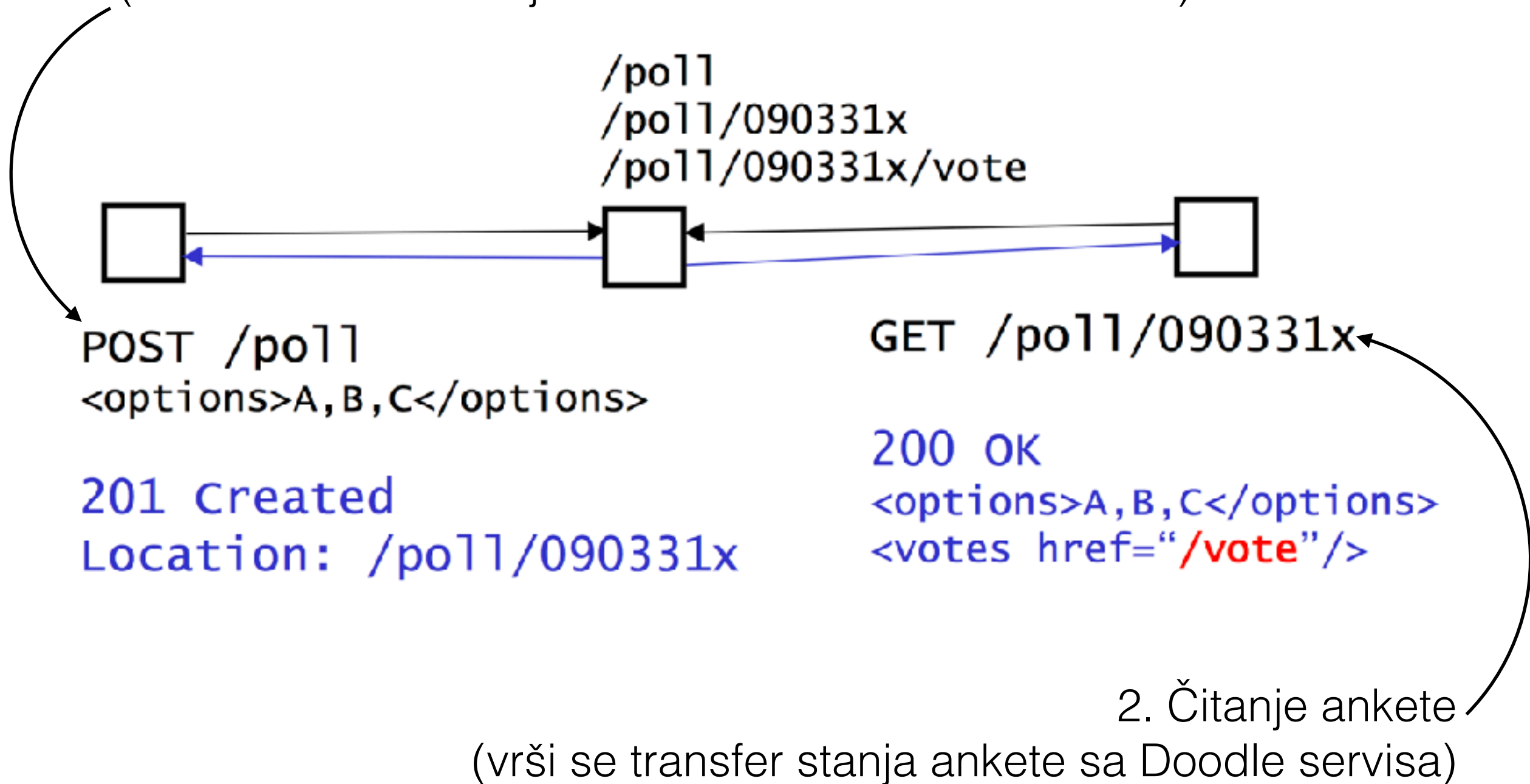
	GET	PUT	POST	DELETE
/poll	✓	✗	✓	✗
/poll/{id}	✓	✓	✗	✓
/poll/{id}/vote	✓	✗	✓	✗
/poll/{id}/vote/{id}	✓	✓	✗	?

3. URIs sadrže ID-eve resursa
4. POST na URI kontejnera se koristi da kreira novi resurs
5. PUT/DELETE se koristi za ažuriranje ili brisanje resursa **/id**

Primer simplifikovanog Doodle API-ja

1. Kreiranje ankete

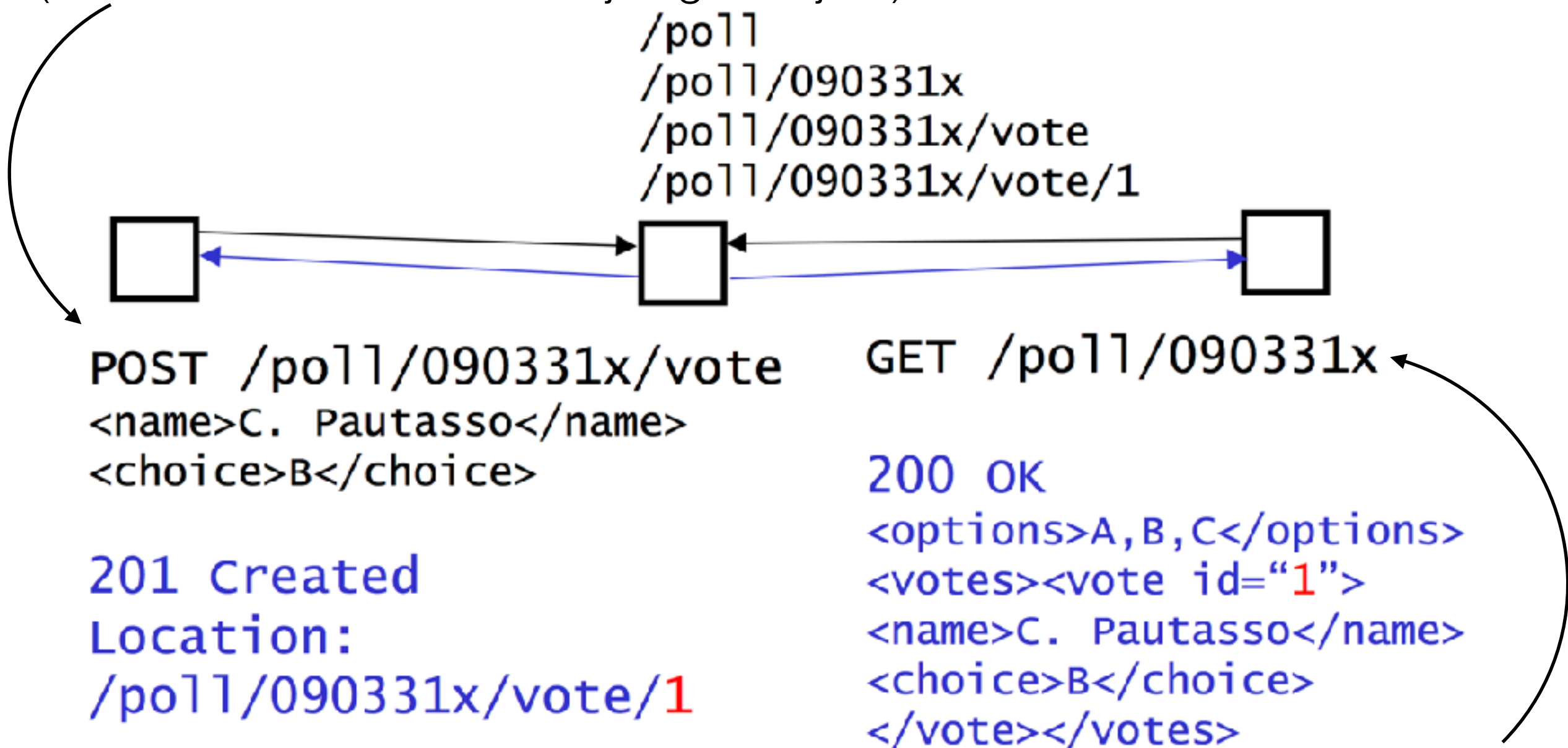
(vrši se transfer stanja nove ankete na Doodle servis)



Primer simplifikovanog Doodle API-ja

3. Učestvovanje u anketi

(kreira se novi resurs sa mojim glasanjem)

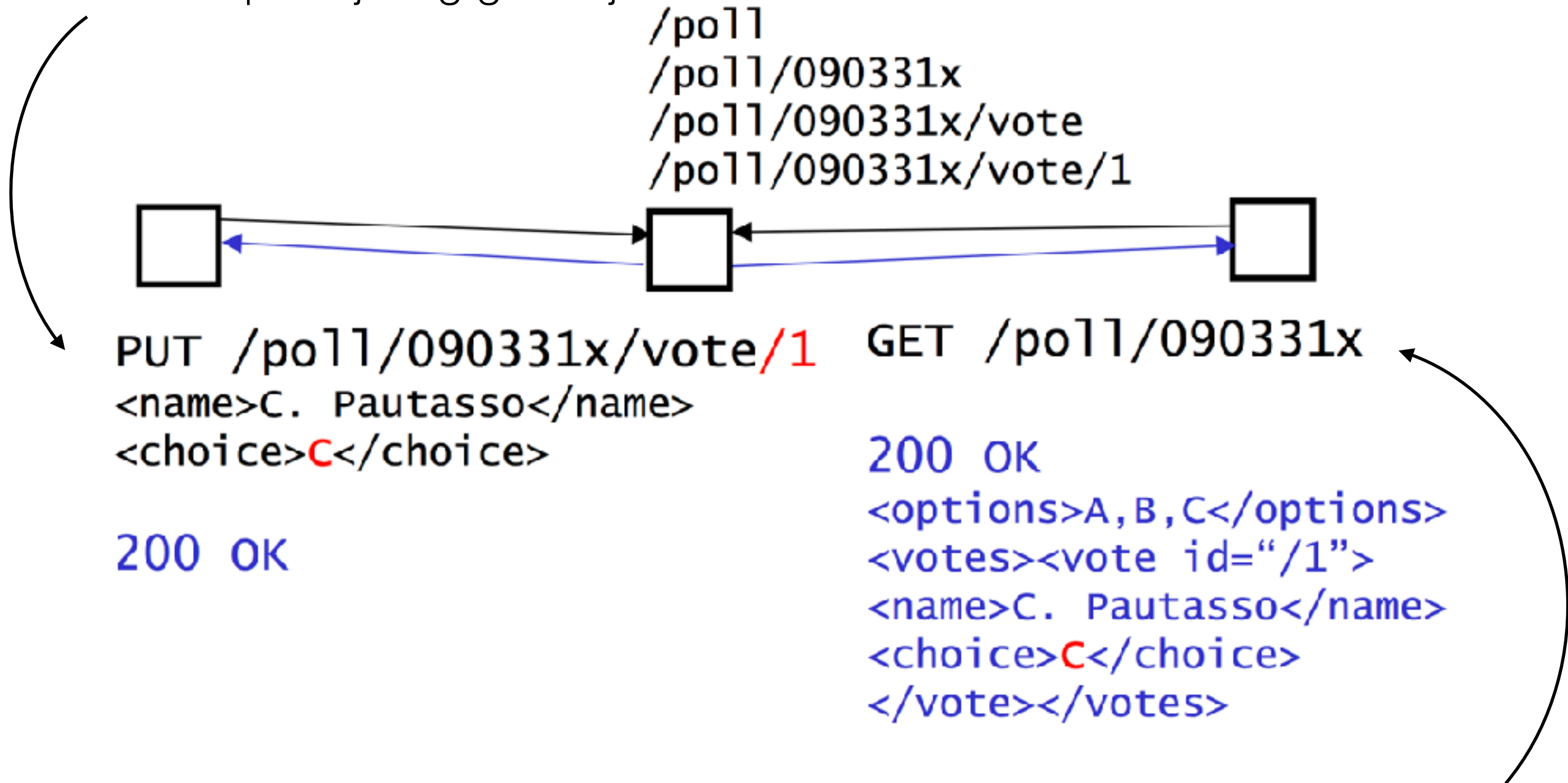


4. Čitanje ankete

(vrši se transfer stanja ankete sa Doodle servisa)

Primer simplifikovanog Doodle API-ja

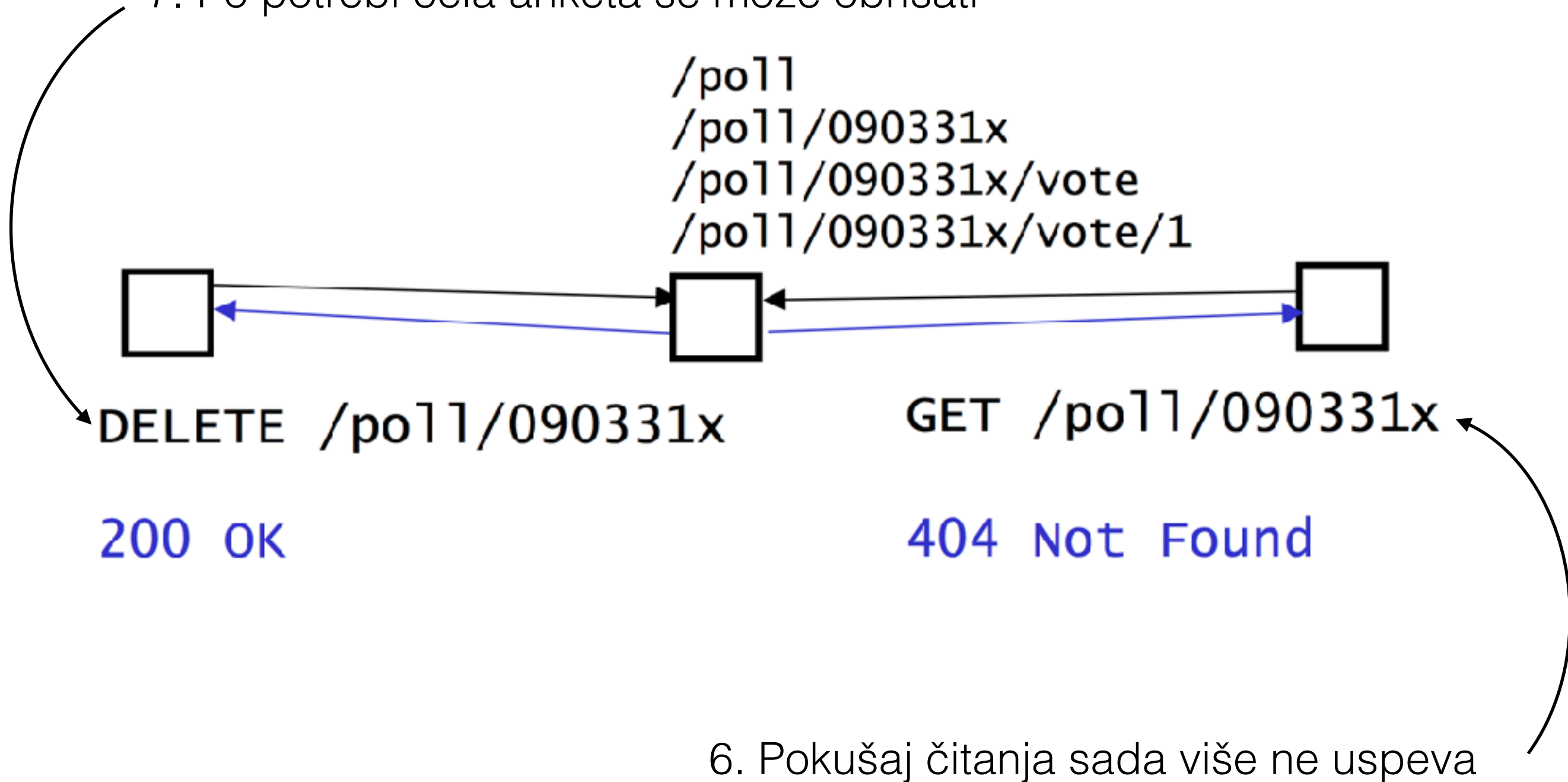
5. Izmena postojećeg glasanja



6. Opet čitanje ankete
(vrši se transfer stanja ankete sa Doodle servisa)

Primer simplifikovanog Doodle API-ja

7. Po potrebi cela anketa se može obrisati



6. Pokušaj čitanja sada više ne uspeva

URI

- Internet Standard za imenovanje i identifikaciju resursa (originalni iz 1994, revidiran od 2005)
- Primer:

`http://tools.ietf.org/html/rfc3986`

URI Scheme Authority Path

`https://www.google.ch/search?q=rest&start=10#1`

Query Fragment

Šta je „lep“ URI

- Poželjno je putanju do resursa pretvoriti u niz segmenata, a ne koristiti key=value parove u query-ju
- bolje
<http://www.mojaknjizara.com/knjige/beletristika>
- nego
[http://www.mojaknjizara.com?
katalog=knjige&kategorija=beletristika](http://www.mojaknjizara.com?katalog=knjige&kategorija=beletristika)

Smernice za osmišljavanje URI-ja

- Preferiraju se imenice a ne glagoli za imenovanje resursa
- Težite ka tome da URI bude kratak
- Ako je moguće koristite pozicionu šemu prosleđivanja parametara, a ne query string
- Ponekad se koriste URI postfixi da se specificira tip sadržaja
- Ne menjajte URI-je za resurse
 - Ako baš morate koristite redirekciju

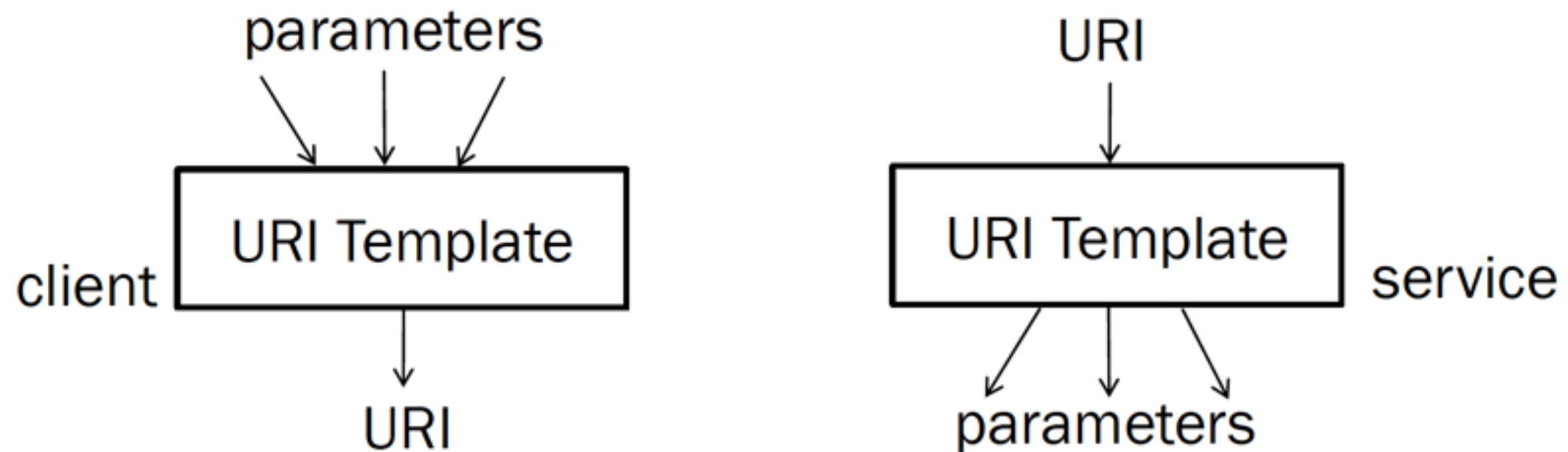
GET /book?isbn=24&action=delete
DELETE /book/24

Često se koristi URI templejting, ali to onda ostvaruje čvršće povezivanje klijenta sa serverskom implementacijom

URI šabloni

- URI Templates (šabloni) specificiraju kako konstruisati i parsirati parametrizovane URI-je.
- Na servisnoj strani obično se koriste “ruting pravila”
- Na klijentskoj strani ovi šabloni se koriste da se konsturiše URI do resursa na osnovu lokalnih parametara.

URI šabloni







- Izbegnite hardkodiranje URI-ja u klijentskoj aplikaciji
- Smanjite međuzavisnost od servera tako što ćete povući URI šablon sa servera i dinamički popunjavati na klijentu

URI šabloni

- Template:
`http://www.myservice.com/order/{oid}/item/{iid}`
- Example URI:
`http://www.myservice.com/order/XYZ/item/12345`
- Template:
`http://www.google.com/search?{-join|&|q,num}`
- Example URI:
`http://www.google.com/search?q=REST&num=10`

Uniforman interfejs

CRUD	REST	
CREATE	POST 	Krerira (pod)resurs
READ	GET 	Preuzima trenutno stanje resursa
UPDATE	PUT 	Ažurira stanje resursa na zadatom URI-ju
DELETE	DELETE 	Uklanja resurs. Nakon toga URI više nije validan.

HTML forme

- HTML4/XHTML

`<form method="GET|POST">`

- HTML5

`<form method="GET|POST|PUT|DELETE">`

GET / POST

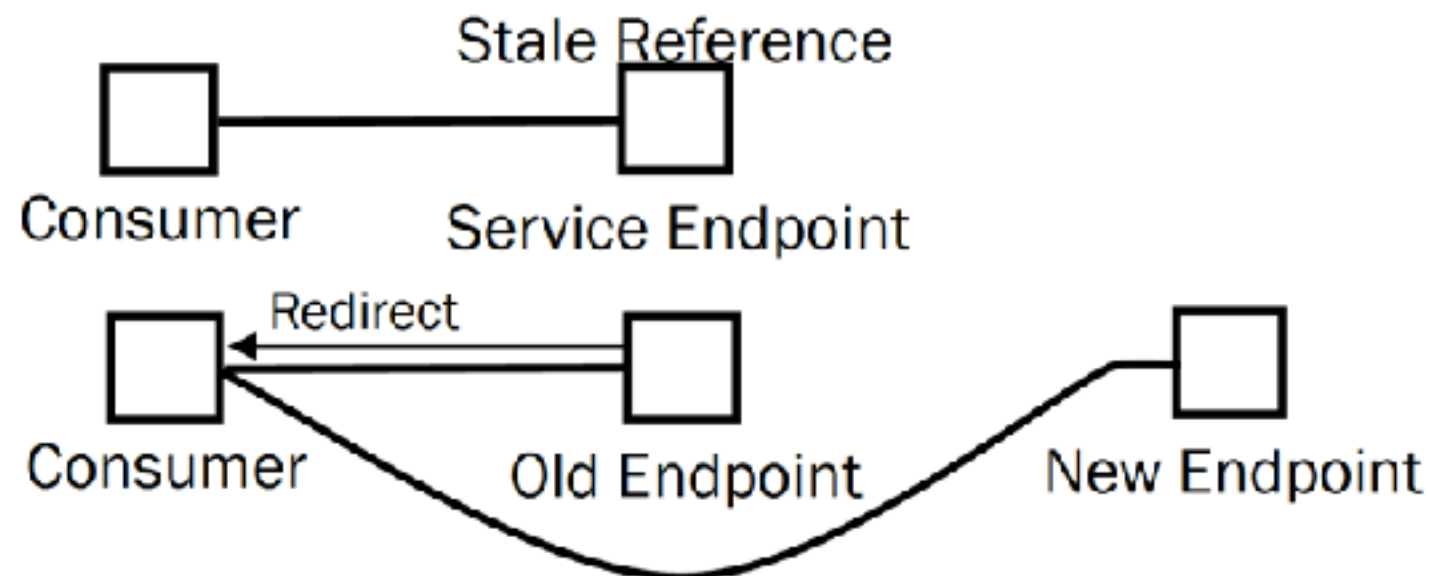
- GET je operacija čitanja - *read-only*.
Može se ponavljati proizvoljan broj puta, pri čemu se stanje resursa neće menjati (idempotentna operacija). Može se i keširati.
 - Napomena: Ovo ne znači da će uvek biti vraćena ista *reprezentacija* resursa.
- POST je operacija i pisanja i čitanja, i može promeniti stanje resursa. Može izazvati i bočne efekte na serveru.

POST / PUT

- Koji je dobar način za kreiranje resursa?
 1. Klijent kreira id resursa.
Problem: Kako obezbediti stvarnu jedinstvenost?
PUT /resource/{id} //po definiciji može da se koristi i za inicijalizaciju kreiranih resursa
 2. Servis kreira id resursa, i vraća ceo kreirani resurs klijentu. Problem: moguće je kreiranje višestrukih instanci ako se više puta pozove endpoint.
POST /resource
 3. Server bi za novi resurs trebao da odgovori sa **201 Created** statusom.

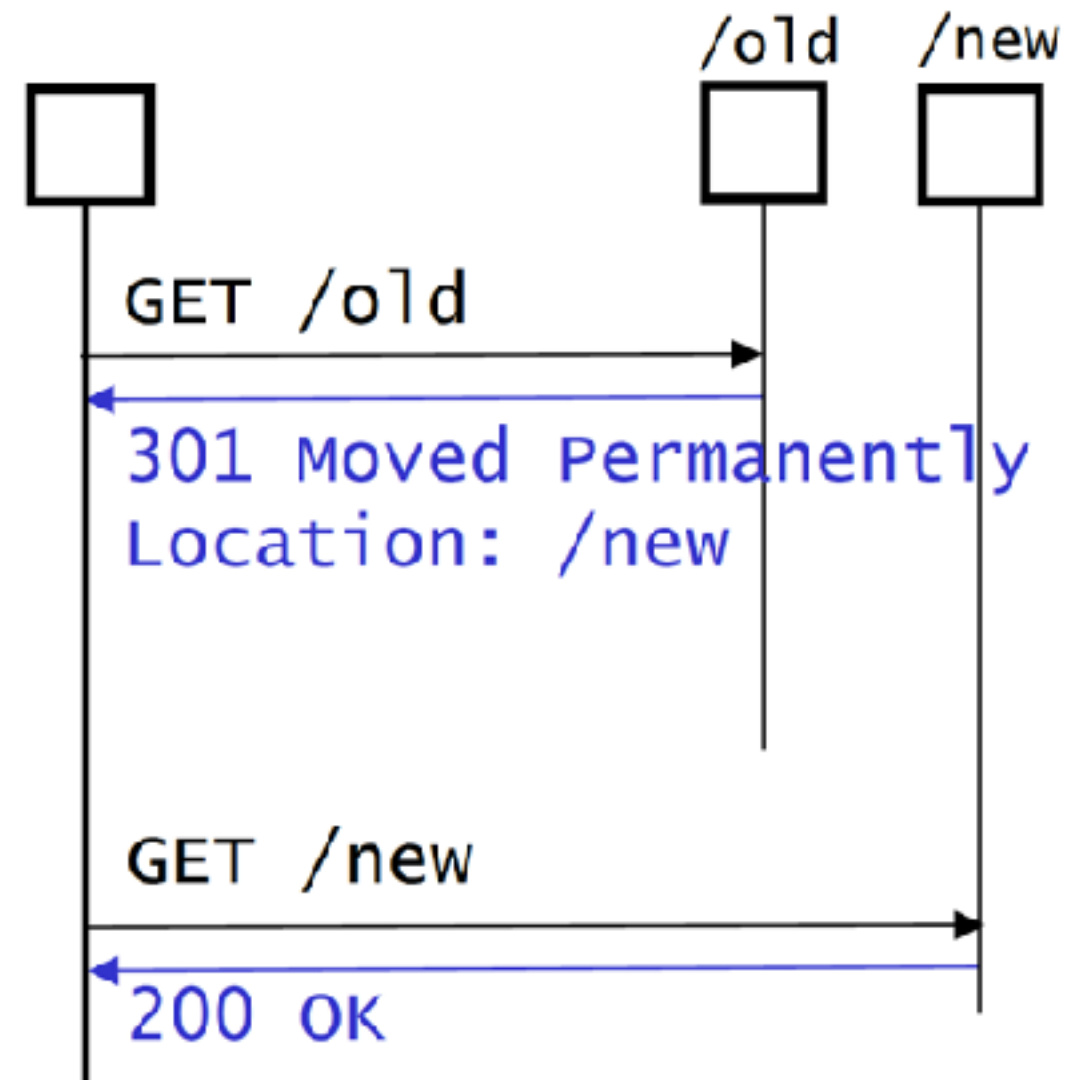
Kako se adaptirati na promene URI-ja

- Kako obezbediti da servis funkcioniše za krajnjeg korisnika čak i kada je stvarno došlo do promene endpoint(ova)?
 1. Do ove promene može doći kako iz poslovnih razloga tako i iz tehničkih
 2. Možda neće biti moguće odjednom promeniti sve linkove do servisnih endpointa, što dovodi do potencijalnih problema
- Koristiti automatsku redirekciju tako da sve korisnike koji se obrate starom endpointu redirektuje na novi endpoint



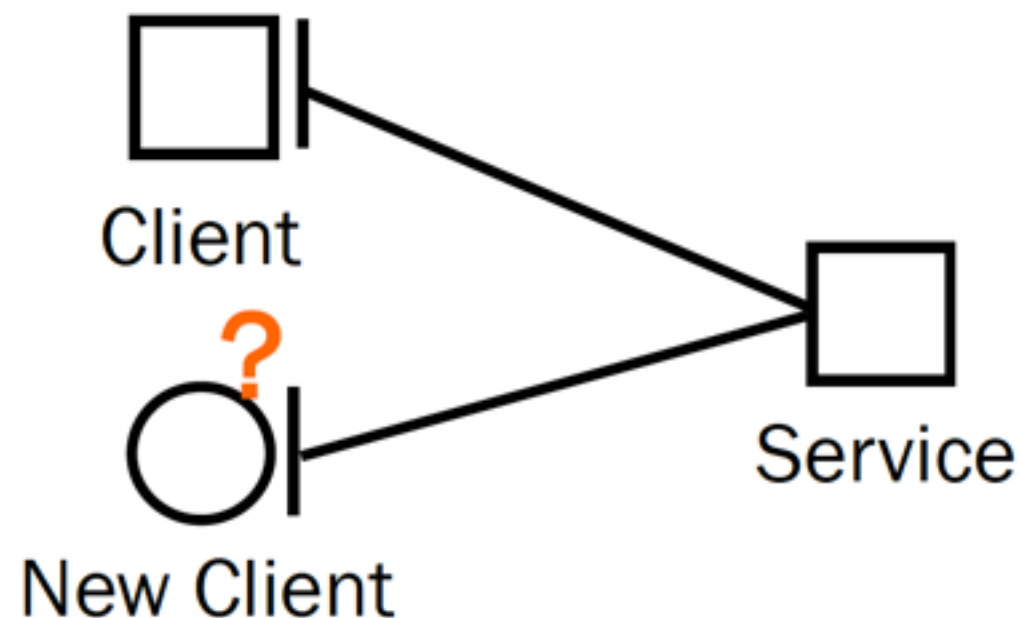
Redirekcija uz pomoć HTTP-a

- HTTP ima ugrađenu podršku za ovakve situacije
- Status kodovi 3xx
 - 301 Moved Permanently
 - 307 Moved Temporarily
 - Location: /newURI
- Redirekcije se mogu ulančavati, ali se pri tome mora paziti da se ne napravi cirkularna redirekcija



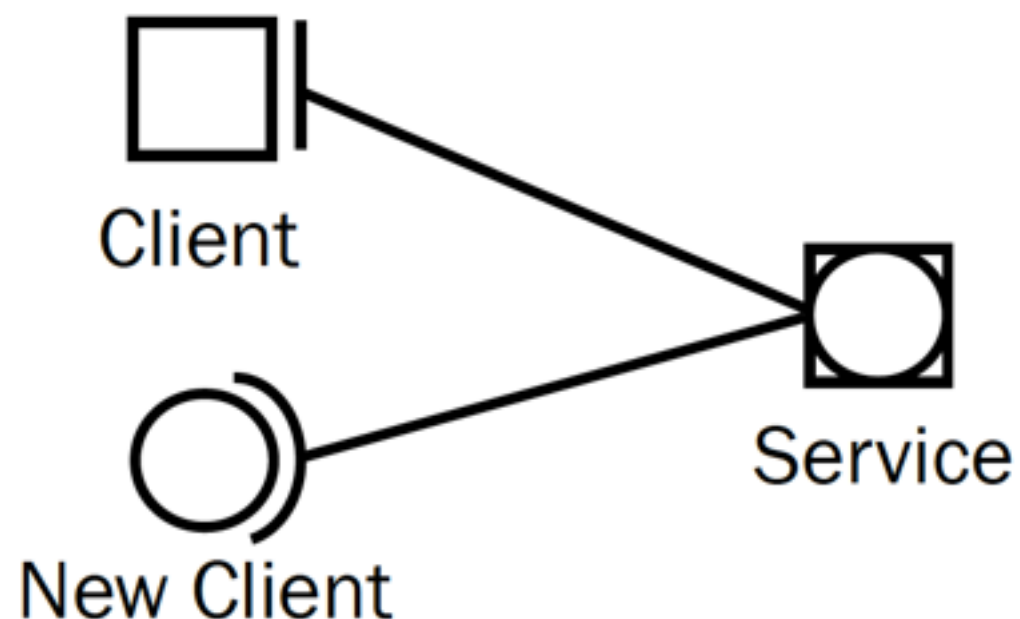
Da li se svi moraju dogovoriti o zajedničkom formatu

- Kako obezbediti da servis podržava korisnike koji koriste ili žele da koriste različite formate poruka (prenosa podataka)
- Ponekad klijenti mogu da promene svoje zahteve
- Servis bi možda trebalo da može da podrži različite korisnike bez potrebe da se pravi poseban interfejs za svakog od njih



Rešenje: *Content Negotiation*

- Specifičan sadržaj i formati reprezentacije podataka koji će biti prihvaćeni ili vraćeni od strane servisa može se dogovoriti u momentu izvršavanja servisa kao deo njegovog poziva.
- Dogovor o načinu korišćenja servisa se zasniva na tipovima sadržaja “*media types*”.
- Prednosti ovog pristupa: obezbeđuje slabu povezanost (uslovljenost), povećanu interoperabilnost, povećanu agilnost za adaptiranje



Content Negotiation u HTTP

- Dogovaranje o formatu razmene poruka je podržano u samom HTTP-u i ne zahteva slanje dodatnih poruka između klijenta i servera

GET /resource

Accept: text/html, application/xml, application/json

- Klijent saopštava koje formate može da razume (MIME types)

200 OK

Content-Type: application/json

- Server bira najpogodniji i njega vraća (status 406 ako nije u stanju da odgovori ni u jednom od traženih formata)

Content Negotiation u HTTP

- Dogovaranje o formatu razmene poruka je podržano u samom HTTP-u i ne zahteva slanje dodatnih poruka između klijenta i servera

GET /resource

Accept: text/html, application/xml, application/json

- Klijent saopštava koje formate može da razume (MIME types)

200 OK

Content-Type: application/json

- Server bira najpogodniji i njega vraća (status 406 ako nije u stanju da odgovori ni u jednom od traženih formata)

Content Negotiation - malo naprednije

- “Faktor kvaliteta” može da se koristi da sugeriše koliko koji format klijent preferira da dobije kao odgovor.

Media/Type; q=X

- Ako se za neki tip navede q=0, takav sadržaj nije prihvatljiv za klijenta.

Accept: text/html, text/*; q=0.1

- Klijent preferira HTML (ali prihvaćće i bilo koji drugi tekstualni format, ali sa nižim prioritetom)

Accept: application/xhtml+xml; q=0.9, text/html; q=0.5, text/plain; q=0.1

- Klijent preferira XHTML, ili HTML ako već ne može prvo, a kao *fallback* prihvata i običan tekst

Forsirani *Content Negotiation*

- Sam generički URI podržava ovaj koncept.

GET /resource

Accept: text/html, application/xml,application/json

- Napravi se poseban URI za svaki tip odgovora dodavanjem ekstenzije (postfixa) na URI

GET /resource.html

GET /resource.xml

GET /resource.json

- Napomena: Ovo je samo prihvaćena praksa nije standard.

Content Negotiation - mogu se podešavati razni aspekti

- Content Negotiation - veoma je fleksibilan i može se sprovesti po različitim aspektima (dimenzijama). Svaki od njih se podešava specifičnim parom HTTP zaglavlja

Request Header	Example Values	Response Header
Accept:	application/xml, application/json	Content-Type:
Accept-Language:	en, fr, de, es	Content-Language:
Accept-Charset:	iso-8859-5, unicode-1-1	Charset parameter fo the Content-Typeheader
Accept-Encoding:	compress, gzip	Content-Encoding:

Definisanje medijskih tipova za REST

- Kako naći najbolji medijski tip za reprezentaciju podataka?
- Da li koristiti generičke ili smisliti novi specifični medijski tip?
- Da li uvek standardizovati tipove?

Neke preporuke za medijske tipove za REST

- Kad je moguće koristite postojeće dobro poznate tipove
- Ali kada je potrebno, slobodno kreirajte svoj
 - ali ga onda standardizujte i koristite kad god je moguće
- Medijski tipovi zapravo prenose informaciju o reprezentaciji resursa predstavljenih našim modelom
- Ne postoji “najbolji tip” za neki servis, sve zavisi od potreba i očekivanja klijenata
- Klijenti ne moraju nužno da procesiraju određeni medijski tip onako kako mi očekujemo

Obrada grešaka

Learn to use HTTP Standard Status Codes

100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect

4xx Client's fault

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed

500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported

5xx Server's fault