REST

Autori:
Goran Savić
Milan Segedinac

1. REST

Veb servis je softverski sistem dizajniran da podrži komunikaciju tehnološki raznorodnih programa putem mreže. Obzirom da sistemi koji komuniciraju mogu biti implementirani u različitim tehnologijama, protokol komunikacije ne sme biti specifičan za tehnologiju implementacije nekog od učesnika u komunikaciji. Veb servis možemo smatrati mrežno dostupnim API-jem sa kojim se komunicira putem dogovorenog protokola.

Postoji više načina realizacije veb servisa. Prvi široko popularan tip veb servisa bili su SOAP veb servisi, koji su predviđali XML bazirani protokol za komunikaciju. U poslednje vreme dominantno se koriste REST veb servisi.

REST (Representational State Transfer) predstavlja stil softverske arhitekture koji se zasniva na postojanju resursa i uniformnog upravljanja njima putem skupa predefinisanih operacija. Iako ova arhitektura nije ciljno dizajnirana za veb servise, najčešće se koristi za upravljanje resursima koje obezbeđuje serverski API veb aplikacije. Ovakav API zovemo REST API veb aplikacije. REST API specificira funkcionalnosti koje veb aplikacija pruža klijentima. Pozivom REST veb servisa klijenti koriste ovaj API.

Osnovni pojam u REST arhitekturi je resurs. Resurs predstavlja informaciju identifikovanu jedinstvenim identifikatorom (URI). Resurs može da se odnosi na entitet, kolekciju entiteta ili na neku funkcionalnost. Dakle, resurs može biti bilo šta što je potrebno referencirati putem jedinstvenog identifikatora. Resurs je odvojen od konkretnog formata reprezentacije. Isti resurs može biti predstavljen različitim formatima (npr. HTML, JSON, XML), pri čemu klijent treba da naznači koji format resursa očekuje. Ovo se naziva *content negotiation*. REST arhitektura predviđa ograničen broj predefinisanih operacija koje mogu biti izvršene nad resursima. Neke od operacija su *Create*, *Read*, *Update*, *Delete*. Važna osobina REST arhitekture je i da predviđa *stateless* komunikaciju sa servisom. To znači da svaki klijentski zahtev sadrži sve podatke potrebne za izvršenje zahteva, tako da za obradu zahteva nije potrebno koristiti podatke iz ranijih klijentovih zahteva. Vidimo da REST arhitektura uvodi određena pravila za dizajniranje serverskog API-ja. Ovim dobijamo uniforman API sa jasnije definisanim značenjem i pravilima korišćenja.

U praksi, kao protokol za komunikaciju sa REST API-jem veb aplikacije standardno se koristi HTTP protokol. Za identifikaciju resursa se koristi internet URL. REST operacije nad resursima se vrše kroz odgovarajuće HTTP metode. Npr. GET za *Read* ili PUT za *Update*. *Takođe,* ovaj protokol je *stateless* što se uklapa sa REST arhitekturom. Pomenuli smo da REST predviđa različite reprezentacije resursa. U HTTP zaglavlju klijent navodi koji tip resursa prihvata, a server je odgovoran da vrati reprezentaciju resursa u tom formatu.

Veoma je važno za svaku funkcionalnost koristiti odgovarajuću REST metodu u skladu sa predviđenim značenjem ovih metoda. S tim u vezi objasnićemo kako REST metode možemo klasifikovati. Prvi kriterijum je da li je metoda **sigurna** (eng. *safe*). Sigurne metode nikada ne menjaju resurs nad kojim se izvršavaju. Drugi kriterijum je kako ponovna izvršavanja metode utiču na krajnji rezultat. **Idempotentna** metoda (eng. *idempotent*) ima isti efekat kroz uzastopne pozive. Na primer, GET zahtev ponovljen više puta nad istim resursom vraća isti rezultat. Za metode koje nisu idempotentne ovo ne važi. Na primer, POST metoda postavlja novi resurs i ponovni pozivi metode će iznova postavljati nove resurse. Pogledajmo u tabeli pregled HTTP metoda koje se standardno koriste za REST API.

Primer URL-a	Metoda	Opis	Sigurna	Idempotentna
/api/countries	GET	Vraća kolekciju država	Da, jer ne menja sadržaj kolekcije	Da, jer uzastopni pozivi vraćaju isti rezultat
/api/countries/5	GET	Vraća državu sa identifikatorom 5	Da, jer ne menja sadržaj resursa	Da, jer uzastopni pozivi vraćaju isti rezultat
/api/countries	POST	Postavlja novu državu u kolekciju	Ne, jer će kolekcija biti izmenjena dodavanjem nove države	Ne, jer uzastopni pozivi iznova dodaju državu
/api/countries/5	PUT	U državu sa identifikatorom 5 ubacuje novi sadržaj	Ne, jer će država biti izmenjena	Da, jer će uzastopni pozivi ubacivati isti sadržaj
/api/countries/5	DELETE	Uklanja državu sa identifikatorom 5	Ne, jer država više neće postojati nakon izvršavanja metode	Da, jer uzastopni pozivi daju isti efekat

Iz prikazane tabele možemo da vidimo kako se u informacionim sistemima standardno realizuju CRUD operacije (Create, Read, Update, Delete) kroz REST API. Za Create se koristi POST metoda, Read se vrši kroz GET metodu, Update korišćenjem PUT metode, dok se za Delete operaciju koristi istoimena HTTP metoda.

Termin REST servis se često koristi za opis bilo kojeg servisa identifikovanog URL-om i dostupnog putem HTTP protokola. Da bismo naglasili stepen usaglašenosti sa REST arhitekturom, servisi koji striktno poštuju REST arhitekturu se nazivaju RESTful veb servisi, dok servise koji tu arhitekturu labavije slede nazivamo REST-Like servisi. U nastavku ćemo za veb orijentisane informacione sisteme analizirati osnovna pravila za definisanje REST API-ja tako da bude što približniji RESTful tipu.

Dizajniranje REST API-ja

Dobra je praksa da se kao identifikator resursa koriste imenice, a ne glagoli. Imenica ukazuje na resurs, a tip HTTP metode ukazuje na operaciju nad resursom. Sa druge strane, korišćenje glagola bi umesto identifikacije resursa, kroz URL identifikovalo akciju. Na primer, za preuzimanje svih država API bi trebao da predviđa pristup URL-u /api/countries kojem se pristupa putem GET metode. Slično, državi sa identifikatorom 5 bi se pristupalo GET metodom na URL-u /api/countries/5. Primer loše prakse bi bio pristup URL-u /api/getAllCountries.

URL-ovi resursa bi trebalo da budu dizajnirani tako da su povezani resursi uniformno predstavljeni. Na primer, gradovi koji se nalaze u državi 5 bi bili dostupni na URL-u /api/countries/5/cities. Slično, grad sa identifikatorom 7 u državi 5 bi bio na URL-u /api/countries/5/cities/7.

Kada je reč o filterisanju resursa, parametri URL-a bi trebali da označe koji resursi ispunjavaju kriterijum za preuzimanje. Na primer, svi gradovi koji se nalaze u Srbiji bi

mogli biti identifikovani URL-om /api/cities?country=Serbia. Na sličan način se mogu zadati i drugi kriterijumi preuzimanja, npr. sortiranje gradova u Srbiji na osnovu broja stanovnika bi bilo dostupno kroz URL /api/cities?country=Serbia&sort=population.

Tip HTTP metode treba da odgovara prirodi akcije. Ako akcija vrši dodavanje resursa, trebalo bi da se vrši kroz POST metodu, a ne npr. GET. Primer loše prakse bi bio API koji za dodavanje država očekuje GET zahtev na URL /api/addCountry. Generalno, GET metode se ne koriste za akcije koje će menjati stanje resursa.

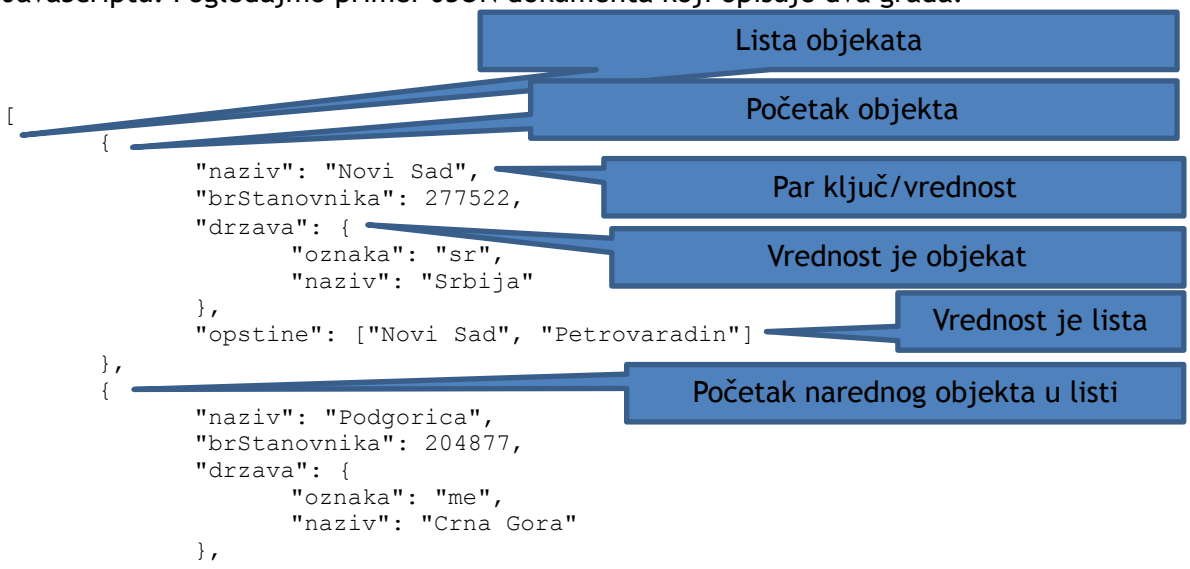
HTTP odgovor koji servis vraća ima svoj statusni kod, koji bi trebao da ukazuje na rezultat operacije. Nakon uspešno izvršenog dodavanja države, serverska metoda bi trebala da vrati HTTP statusni kod 201 – Created. Primer loše prakse je vraćanje koda 200 OK za svaku operaciju.

Kao što smo objasnili, URL identifikuje resurs, a sam resurs može da ima različite reprezentacije. Zato bi URL trebao da bude isti za metode koje vraćaju isti resurs u različitim reprezentacijama. U zaglavlju HTTP zahteva se u polju *Accept* navodi koji format podataka klijent očekuje. Server u zaglavlju HTTP odgovora u polju *Content-type* navodi format podataka u odgovoru (polje Content-type koristi i klijent da specificira format podataka koje šalje serveru).

2. JSON

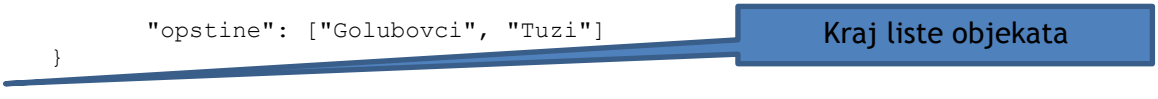
Za prenos informacija o resursu između klijenta i servera, podaci moraju biti serijalizovani u tekstualnu reprezentaciju. Ranije smo se upoznali sa XML jezikom kao jednim često korišćenim načinom reprezentacije podataka. Slično, ako su resursi veb stranice, oni su reprezentovani HTML jezikom. Trenutno najčešći format za serijalizaciju podataka koje klijent i server razmenjuju je JSON (JavaScript Object Notation). Format je čitljiv za čoveka, ali i pogodan za mašinsko procesiranje. Kao i XML, generički je dizajniran za opis bilo kojeg skupa informacija. U odnosu na XML, koristi manje karaktera koji ne nose korisne informacije (npr. nema zatvarajućih tagova), pa se podaci efikasnije prenose.

Notacija je napravljena po ugledu na JavaScript notaciju za definisanje objekata putem parova ključ-vrednost. Objekte je moguće hijerarhijski organizovati. Pored ovog konstrukta, objekte je moguće organizovati u listu, za šta se koristi slična sintaksa kao u JavaScriptu. Pogledajmo primer JSON dokumenta koji opisuje dva grada.




```
[
    {
        "naziv": "Novi Sad",
        "brStanovnika": 277522,
        "drzava": {
            "oznaka": "sr",
            "naziv": "Srbija"
        },
        "opstine": ["Novi Sad", "Petrovaradin"]
    },
    {
        "naziv": "Podgorica",
        "brStanovnika": 204877,
        "drzava": {
            "oznaka": "me",
            "naziv": "Crna Gora"
        },
```

```
            "opstine": ["Golubovci", "Tuzi"]
        }
]
```



3. REST u Spring aplikaciji

Već smo se ranije sretali sa Spring kontrolerima. REST API se u Spring veb aplikaciji realizuje kroz metode Spring kontrolera (klase anotirane anotacijom @RestController), s tim da se metode dizajniraju u skladu sa REST principima. Konkretno URL-ovi zahteva treba da identifikuju REST resurse. Resurs može biti isporučuje u formatu koji klijent zahteva. Podaci se šalju kroz parametre ili telo zahteva, a odgovor osim sadržaja sadrži i HTTP statusni kod, kao i zaglavlje sa dodatnim podacima.

Pogledajmo na primeru jednu metodu serverskog REST API-ja realizovanu kroz Spring podršku za veb aplikacije.

```
@RequestMapping(value = "api/countries", method = RequestMethod.GET,
produces = {MediaType.APPLICATION_JSON_VALUE,
MediaType.APPLICATION_XML_VALUE })
public ResponseEntity<List<Country>> getAllCountries() {
      List<Country> countries = countryService.findAll();

      return new ResponseEntity<>(countries, HttpStatus.OK);
}
```

Metoda vraća spisak država, što je resurs identifikovan sa /api/countries. Podaci se dobijaju GET metodom obzirom da akcija ne vrši izmenu nad pomenutim resursom. Metoda podatke isporučuje u JSON ili XML formatu u zavisnosti od toga šta klijent naznači u zaglavlju zahteva u polju Accept. Podatke koje metoda vrati će Spring automatski serijalizovati u traženi format, ukoliko su dostupne klase koje vrše serijalizaciju. Spring boot već sadrži Jackson biblioteku koja vrši konverziju Java objekata u JSON i obrnuto. Za konvertovanje iz/u XML format potrebno je dodatno uključiti biblioteku *jackson-dataformat-xml*.

Obzirom da osim podataka, HTTP odgovor treba da nosi i dodatne informacije, odgovor je predstavljen klasom ResponseEntity. U prikazanom primeru se pri instanciranju objekta ove klase navodi lista država kao sadržaj odgovora, ali i HTTP statusni kod postavlja na 200 – OK.

U prethodnom primeru smo videli kako Spring može automatski serijalizovati Java objekte u format definisan atributom *produces* u anotaciji @RequestMapping. Pored toga, omogućena je i automatska deserijalizacija podataka koji stižu sa klijenta u Java objekte. Naredni primer pokazuje kako server prima podatke o državi u REST metodi za unos države.

```
@RequestMapping(value = "api/countries", method = RequestMethod.POST,
          consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Country> create(@RequestBody Country country) {
      Country retVal = countryService.save(country);

      return new ResponseEntity<>(retVal, HttpStatus.CREATED);
}
```

Vidimo da je atributom *consumes* u anotaciji @RequestMapping navedeno da server očekuje podatke u telu zahteva u JSON formatu. Podaci će automatski biti prebačeni u Java objekat naveden kao parametar metode anotiran anotacijom @RequestBody. U

konkretnom slučaju, od podataka pristiglih sa klijenta će biti instanciran i inicijalizovan objekat klase Country. Metoda dalje taj objekat može da koristi na proizvoljan način. U prikazanom primeru objekat se dodaje u listu država.