SADRŽAJ

Uvod

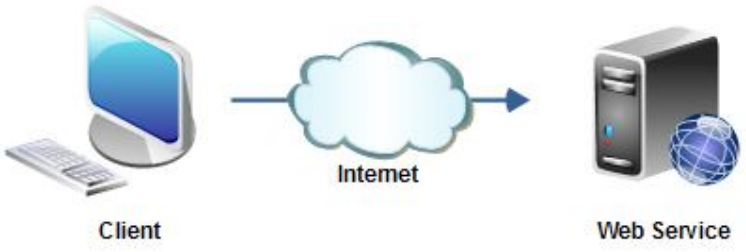
Ogromna zastupljenost Interneta u današnje vreme pruža mogućnost distribucije raznih vrsta podataka putem Web Servisa. Ovaj način distribucije informacija ima prednost da se one putem klijentskih aplikacija korisniku prezentuju na odgovarajući način. Ovakvi izvori informacija su sve zastupljeniji a kada se korisnici prilagode ovakvom načinu pribavljanja i distribucije informacija teško da će moći da se odviknu od njih.

Od momenta kad je nastao, ***World Wide Web*** je u stalnom usponu. Najveće zasluge leže u njegovoj softverskoj arhitekturi, projektovanoj da ispuni uslove za rad sa distribuiranim *hipermedija* aplikacijama. Osnovne karakteristike modernih Web arhitektura su interakcije između komponenata, skalabilnost, nezavisnost komponenti, uopštenost interfejsa, postojanje posrednih komponenata koje smanjuju kašnjenje pri interakciji i povećavaju sigurnost. U ovom radu biće objašnjena arhitektura koja počiva na ***RE****presentational* ***S****tate* ***T****ransfer-u*, kao i realizacija Web serisa korišćenjem ovog arhitekturnog modela. Representational State Transfer (REST) predstavlja koncept u kome jedinstveni URL-ovi domena predstavljaju, zapravo, objekte kojima upravljamo kroz HTTP metode. Kao i kod ostalih servisa i za REST važi pravilo platformske i jezičke nezavisnosti i otpornosti na Firewall iz prostog razloga, što je jedini protokol koji REST koristi, ustvari HTTP.

U prvom delu ovog rada biće objašnjeni osnovni principi Web servisa. Takođe, biće reči o arhitekturnim elementima, njihovoj ulozi i međusobnoj interakciji u okviru SOAP i REST modela. U drugom poglavlju, biće reči o Web servisima koji počivaju na REST-u. Osim definicije i osnovnih principa, biće objašnjen i primer implementacije REST servisa . U poslednjem poglavlju biće reči o prednostima REST tehnologije, kao i zaštiti RESTful Web servisa.

1. WEB Servisi

Web servis može da se definiše kao skup protokola i standarda koji se koriste za razmenu podataka između aplikacija ili sistema. Softverske aplikacije napisane na različitim programskim jezicima i pokrenute na različitim sistemskim platformama, mogu da koriste web servise za razmenu podataka putem računarskih mreža. Osnovni princip Web servisa možemo pokazati na sledeći način:



Klijent poziva web servis putem interneta

Web servisi, takođe, mogu da se koriste i u internoj mreži organizacije, kao način da se omogući interakcija između raznih aplikacija. Princip prikazujemo na sledeći način:

Nezavisne aplikacije na internoj mreži komuniciraju jedna sa drugom putem web servisa

Takođe, možemo reći da je svaki Web servis zasnovan na odnosu klijent – server. Model Web servisa bi mogao da se prikaže i na sledeći način:

Web servis je, u suštini, svaki servis koji:

- Je dostupan preko interneta ili privatne (interne) mreže
- Koristi standardizovan sistem XML poruka
- Nije vezan za određeni operativni sistem ili programski jezik
- Je prepoznatljiv od strane bilo kog mehanizma za pretragu

1.1 Osnovni pojmovi u Web servisima

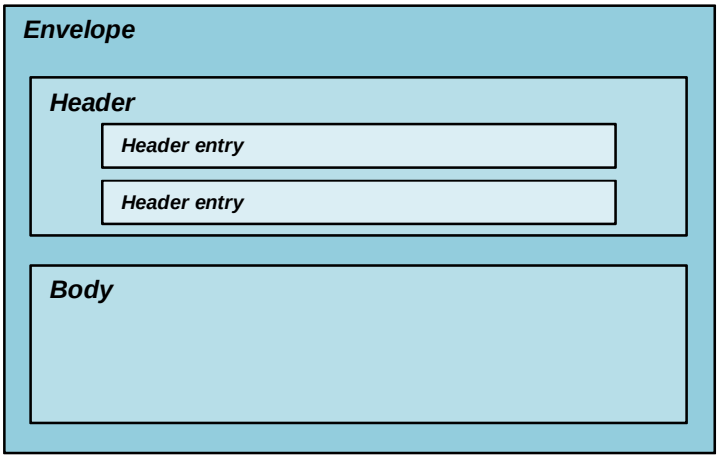
Osnovni pojmovi u Web servisima su:

SOAP (Simple Object Access Protocol)

Predstavlja okruženje za razmenu poruka u mreži zasnovanih na XML-u. Sam SOAP se služi aplikativnim HTTP protokolom koji koriste svi Web serveri, te se može reći sledeće: HTTP + XML = SOAP. Glavni deo SOAP specifikacije definiše skup pravila za upotrebu XML za reprezentaciju podataka. Drugi delovi SOAP specifikacije definišu proširivi format poruka, konvencije za udaljeno pozivanje metoda i povezivanje sa HTTP protokolom. SOAP je nezavisan od korišćenih platformi i tehnologija, odnosno programskih jezika. U kontekstu rada sa Web servisima, SOAP se koristi za specificiranje pozivnih parametara i rezultata rada servisa.
Postoji nekoliko različitih tipova poruka u SOAP-u. Najpoznatiji je poziv udaljenim procedurama (Remote Procedure Call – RPC), tip poruka u kojem jedan čvor u mreži(klijent) šalje zahtev drugom čvoru (server), a server odmah vraća odgovor na primljeni zahtev. Osnovni elementi SOAP poruke su:

- Envelope
- Header
- Body

Poznavanjem ovih elemenata znamo gde se smeštaju podaci, kako se poruka proširuje i kako se predstavlja greška.



Struktura SOAP poruke

Treba još navesti da je SOAP baziran na XML formatu poruka.

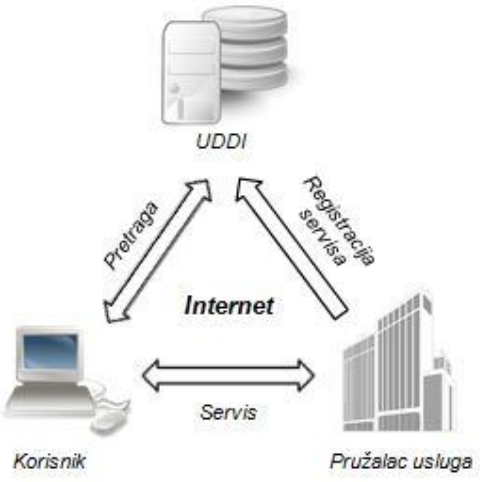
UDDI (Universal Description, Discovery and Integration)

je odgovor na pitanje: „Gde se može pronaći potreban servis?“. Web Servisi se objavljuju na jedinstvenoj lokaciji i nude se kao usluge. Kako bi usluga bila kompletna, nudi se i potpuna specifikacija interfejsa. UDDI predstavlja centralizoanu lokaciju koja obezbeđuje mehanizam za registrovanje i pronalaženje Web servisa. Koristi SOAP za komunikaciju i omogućava klijentima da pronađu servis, kao i serveru da ga objavi.

UDDI poslovna registracija sastoji se iz tri dela:

- Bijele stranice: adresa, kontakt i poznati identifikatori
- Žute stranice: industrijska kategorizacija
- Zelene stranice: tehničke informacije o usluzi

UDDI kreira standardnu interoperabilnu platformu koja omogućava kompanijama i aplikacijama da brzo, lako i jednostavno pronađu i koriste Web servise na Internetu. Osnovni koncept možemo da prikažemo na sledeći način:



WSDL (Web Services Description Language)

Jedna od najinteresantnijih osobina web servisa je ta da oni imaju mogućnost da opišu sebe. To znači da, kada lociramo web servis, možemo postaviti upit za opis, operacije koje podržavaju i kako ih prizvati. Za to služi WDSL, ili u prevodu Jezik za opisivanje Web servisa. Osnovne komponente su:

`<types>` pokazuje tip podataka u poruci.
`<message>` opisuje SOAP poruku
`<operation>` pokazuje direkciju (jednosmerna, notifikacije, zahtev/odgovor)

`<port type>` definiše web servis, operacije koje mogu da se izvrše i koje su poruke uključene.

`<binding>` opisuje format poruke i protokol za veb servise.

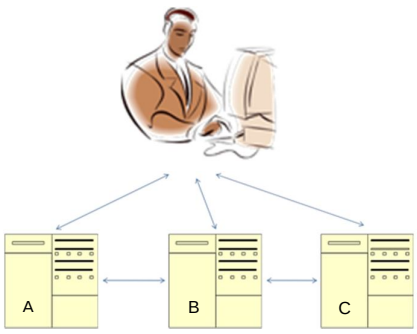
`<port>` Dokumentuje fizičku lokaciju (URI) Web servisa. Sadrži <binding> element.

REST

Prenos prikaza stanja (*Representational State Transfer* - REST) je model arhitekture za distribuirane hipermedija sisteme. Ovaj model se zasniva na prenosu prikaza stanja resursa, gde resurs može biti bilo koji smisleni, adresibilni koncept, a prikaz resursa je uglavnom dokument koji sadrži trenutno stanje resursa. Najveća REST aplikacija je sam Web, karakterističan po tome što koristi HTTP protokol za transport i URL mehanizam za adresiranje. REST podržava sve vrste medija i XML je najpopularniji metod koji se koristi za prenos i prezentaciju strukturnih informacija. Sistemi koji prate REST se nazivaju RESTful sistemima.

Da bi razjasnili pravilnu implementaciju REST-a, predstavićemo tri scenarija koji se odnose na mrežnu konfiguraciju mnogih sistema. Nakon toga ćemo prikazati dva slučaja korišćenja kako izgleda protok informacija između raznih mrežnih konfiguracija.

Scenario 1 – Sve se nalazi u istom domenu

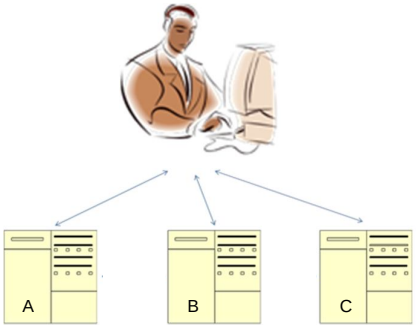


Klijent ima dozvoljen pristup svakom serveru. Serveri mogu da komuniciraju jedan sa drugim.

U ovom scenariju , serveri komuniciraju jedan sa drugim i klijent ima pristup svakom serveru posebno. Svi sistemi se nalaze na istom domenu. Agregacija podataka[1] može da bude i na klijentu i na svim serverima, tako da je autorizacija i zaštita podataka komplikovana – mora da postoji i na nivou klijenta i na nivou servera.

[1] Agregacija podataka je kompilacija različitih skupova podataka u konsolidovani skup podataka.

Scenario 2 – Pristup internim domenima



U ovom scenariju, klijent ima direktnu konekciju sa svakim serverom, ali serveri nemaju jedan sa drugim. U tom slučaju agregacija podataka se obavlja samo na klijentu. Ovim scenarijom se predstavljaju odvojeni serveri sa različitim namenama, što znači da svaki server ima odvojenu autorizaciju. Integritet podataka kao i pitanje poverljivosti izolovani su na klijentskom pretraživaču/interfejsu, kao i prenosni put između klijenta i servera.

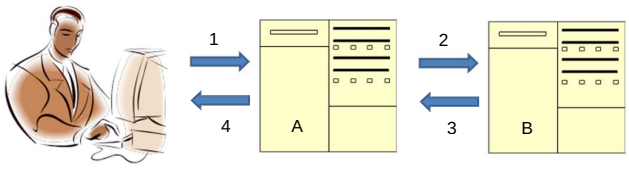
Scenario 3 – Federativna pretraga / pristup u ime korisnika

U ovom primeru, server A (jedini server kome klijent ima direktan pristup), ponaša se kao proksi Korisnik je autentikovan i autorizovan samo na serveru A koji vrši pretragu na serverima B i C u njegovo ime. Server A može da prosledi korisničke kredencijale (lančano poverenje) na servere B i C ili da server A filtrira rezultate pretrage servera B i C i izbaci ono sto je klijent ovlašten da vidi. U ovom slučaju, agregacija podataka je na Serveru A.

U situacijama gde server C treba da pošalje osetljive podatke klijentu (preko servera A i B), potrebno je koristiti XML enkripciju. Ovaj scenario ne opisuje tradicionalni REST model, pošto korisnik nema direktan pristup izvoru podataka. Najbolje bi bilo koristiti SOAP model u ovakvim situacijama.

Primeri korišćenja navedenih situacija:

Primer 1 – Federativna pretraga:



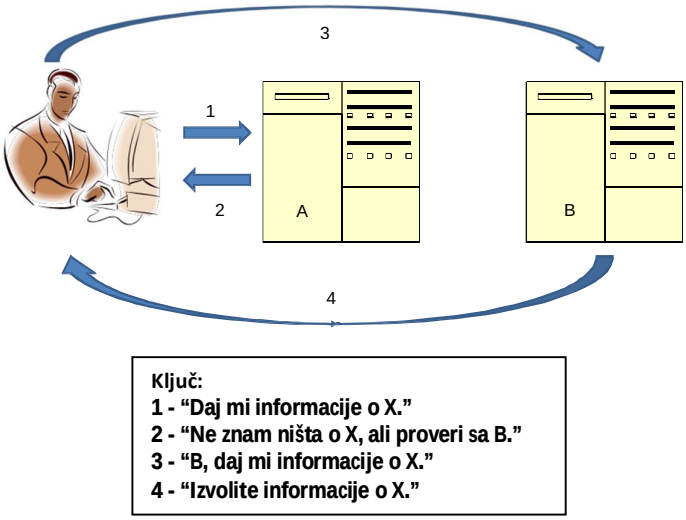
Ključ:
1 - "Daj mi informacije o X."
2 - "B daj A informacije o X za zahtev korisnika."
3 - "A, izvoli informacije o X."
4 - "Klijent, izvolite informacije o X."

Ovaj primer prikazuje kako svi zahtevi klijenta idu kroz server A jer klijent nema direktan pristup serveru B. Prednosti i mane ovog pristupa su:

Prednosti	Mane
1. Jednostavan klijentski interfejs	1. *Chained trust (AuthZ)*
2. Agregacija podataka na serveru A	2. Posrednici
3. Jednostavnija pretraga za klijenta	3. Server A je upoznat sa parametrima

U ovom slučaju, server A je posrednik. Pitanje je kako zaštititi osetljive informacije, zato što server A može da ima uvid u poverljivost i integritet podataka poslatih od strane servera B klijentu. U REST-u, najbolji metod je korišćenje enkripcije.

Primer 2 – Redirekcija klijenta



Prednosti	Mane
1. Direktna konekcija sa klijentom	1. Server A zna parametre pretrage
2. Klijent ima bolji pristup podacima	2. Nema agregacije podataka
3. Autorizacija direktno sa klijentom	3. Višestruki zahtevi klijenta (teža obrada)

Ovde vidimo da klijent ima direktan pristup podacima. Poverljivost i integritet je mnogo lakše održavati u ovoj situaciji.

1.2 Tipovi Web servisa

Postoje dve osnovne klasifikacije web servisa, i to:

„Big" web servisi

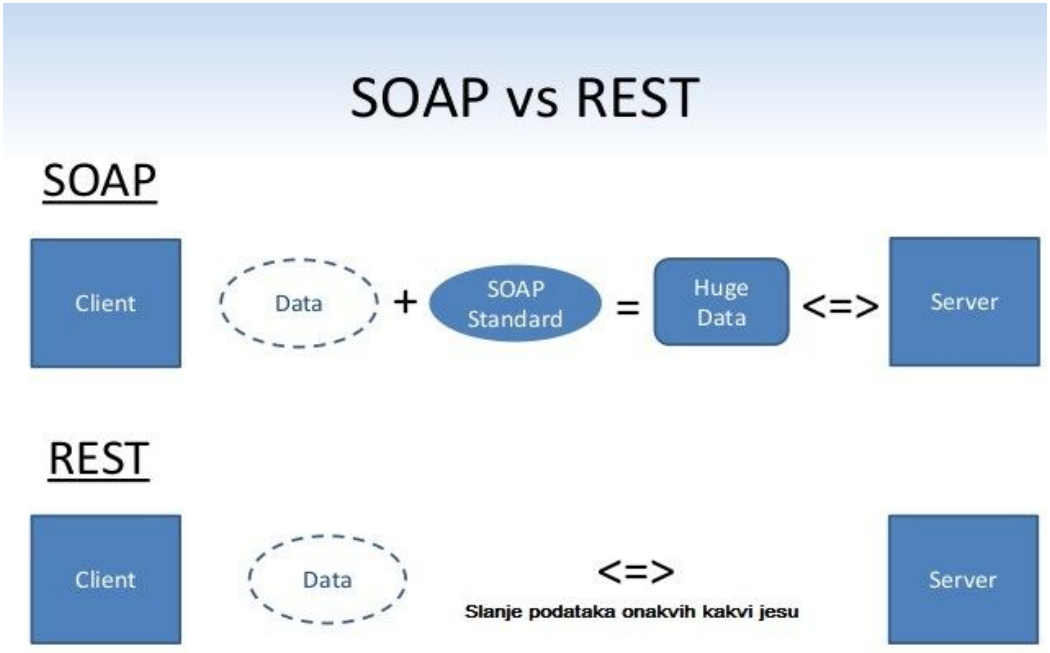
"Big" web servisi su bazirani na SOAP standardu, i često sadrže WSDL za opisivanje interfejsa koji web servis nudi. Detalji koje WDSL daje mogu da budu poruke, operacije, veze I lokacije web servisa.

Web servisi bazirani na SOAP platformi, nude dobra rešenja kada nam je potrebna:

- Asinhrona obrada
- Pouzdanost
- Formulisana (Stateful) operacija – Ako aplikacija traži dodatne informacije, SOAP nudi dodatnu specifikaciju u strukturi web servisa da bi podržao upit (bezbednost, transakcije, koordinaciju itd.)

„RESTful" web servisi

"RESTful" web servisi su bazirani na arhitekturalnom stilu koji se naziva REST. REST nije zavisan ni od jednog protokola, ali gotovo svaki RESTful servis koristi HTTP kao osnovni protokol. Ovi servisi su dosta bolje integrisani sa HTTP-om od SOAP servisa, pa kao takvi ne zahtevaju XML SOAP poruke ili WSDL definicije. Više o RESTful web servisima u sledećem poglavlju.



2. RESTful web servisi

Nakon više od decenije od predstavljanja, REST je postao jedna od najbitnijih tehnologija za Web aplikacije. Svaki veci programski jezik sadrzi okvire za izradu RESTful web servisa. Zbog toga što je dosta jednostavniji, REST je skoro u potpunosti zamenio SOAP i WDSL.

Postoje razni servisi koji se koriste u danasnje vreme, a jedan od najpoznatijih je WWW (World Wide Web). On predstavlja softversku osnovu za razvoj komercijalnih sadrzaja na internetu i sastoji se iz sledecih elemenata:

- HTML – Sluzi za formatiranje i povezivanje web stranica
- HTTP – Protokol koji sluzi za distribuciju sadržaja stranica, kao i komuniciranje između servera I klijenta
- CGI (Common Gateway Interface) – Sluzi za razmenu podataka sa drugim serverima ili CGI programima

Dobra stvar u vezi web-a je činjenica da klijenti i serveri mogu komunicirati na kompleksan način, bez da klijent unapred zna bilo šta o serveru i resursima koji se nalaze na njemu. Jedino ograničenje je da klijent i server moraju da se slože oko medija koji koriste, što je u ovom slučaju HTML.

Svaki sistem koristi resurse. Ovi resursi mogu biti slike, video fajlovi, web stranice, poslovne informacije, kao i bilo šta što može da se predstavi u kompjuterski baziranom sistemu. Svrha servisa je da klijentima obezbedi pristup ovim resursima. Svaki programer ili arhitekta želi da ovi servisi budu jednostavni za implementaciju, održavanje i nadogradnju. RESTful servisi upravo to i nude. Generalno gledano, RESTful servisi treba da imaju sledeće osobine i karakterisitike:

- Nepostojanje stanja (*Stateless*)
- Mogućnost keširanja (*Cacheable*)
- Uniformni interfejs (*Uniform interface URI*)
- Izričito korišćenje HTTP metoda
- Transfer XML i/ili JSON

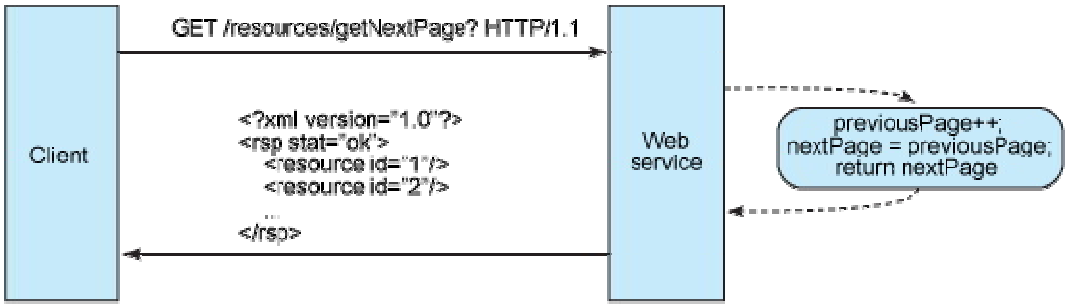
2.1 Nepostojanje stanja (Stateless)

„Sa stanjem (*Stateful*)“ i „bez stanja (*Stateless*)“ su pridevi koji opisuju da li je kompjuter ili kompjuterski program dizajniran da beleži i pamti jedan ili više prethodnih događaja u datom nizu interakcije sa korisnikom, drugim kompjuterom, programom, uređajem ili bilo kojim spoljim elementom. *Stateful* znači da kompjuter ili program vodi evidenciju o stanju interakcije, obično postavljanjem vrednosti u polje za skladištenje namenjeno za tu svrhu. *Stateless* znači da ne postoji evidencija prethodnih interakcija, i svaki zahtev za interakcijom mora biti baziran na informacijama koje dolaze uz njega. *Stateful* i *Stateless* su izvedenice od reči *State* (Stanje), što prikazuje skup stanja u nekom momentu.

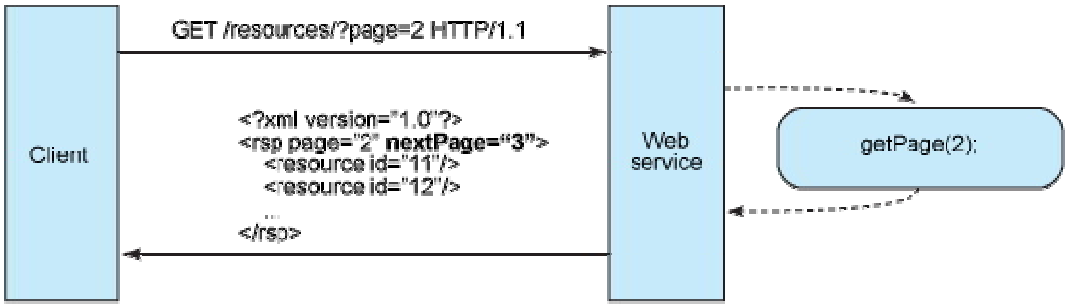
Komunikacija između klijenta i servera se obavlja bez postojanja stanja, tj. nijedan klijentski kontekst ne sme biti smešten na serveru. „Svaki zahtev od klijenta ka serveru, mora da sadrži sve potrebne informacije da bi bio uspešno procesuiran, ali ne sme da zloupotrebi sadržaje koji već postoje na serveru“[2].

Da bi u potpunosti objasnili razliku između „Stateful“ i „Stateless“ dizajna, prikazaćemo sledeće primere:

Primer 1. Stateful dizajn



Primer 2. Stateless dizajn



Na primerima možemo da vidimo koliko je Stateless jednostavniji za dizajniranje, zapis kao i distribuciju među serverima. Ne samo da se Stateless servisi ponašaju bolje, nego i prebacuju većinu odgovornosti održavanja na klijentske aplikacije.

2.2 Mogućnost keširanja

Pod keširanjem se podrazumeva čuvanje generisanih rezltata i njihovo ponovno korišćenje ukoliko se u bliskoj budućnosti ponovi isti zahtev. Keširanje može da se vrši na strani klijenta, servera ili bilo koje druge komponente između njih, kao što je recimo proksi server. Ovo je dobar način za poboljšanje performansi servisa, ali ako se ne održava kako

[2] 2000, **Architectural Styles and the Design of Network –based Software Architectures, R.T.Fielding**

treba, može doći do toga da klijent dobije ustajale informacije. Keširanje može da se kontroliše korišćenjem sledećih HTTP zaglavlja:

Zaglavlje	Aplikacija
Datum	Datum i vreme kada je prikaz generisan
Poslednja izmena	Datum i vreme kada je server poslednji put modifikovao prikaz
Keš kontrola	HTTP 1.1 zaglavlje kontroliše keširanje
Rok	Datum i vreme do kad traje prikaz. Podržava HTTP 1.0 klijente
Starost	Trajanje u sekundama od momenta učitavanja sa servera

Vrednosti ovih zaglavlja mogu da se koriste u kombinaciji sa direktivama iz Keš kontrolnog zaglavlja radi provere da li su rezultati važeći ili ne. Najčešće direktive iz keš kontrolnog zaglavlja su:

Direktiva	Aplikacija
Javna	Podrazumevana vrednost. Označava da sve komponente mogu keširati prikaz.
Privatna	Posredne komponente ne mogu keširati prikaz, samo klijent i server mogu.
no-cache/no-store	Keširanje isključeno
Maksimalna starost	Trajanje u sekundama nakon označenog trajanja prikaza u zaglavlju datuma
s-maksimalna starost	Slično „maksimalnoj starosti“, samo što se misli na posredno keširanje
Obavezno produženje	Označava da prikaz mora biti produžen od strane servera ako je istekla maksimalna starost
Proksi-potvrda	Slično „Obaveznom produženju“, samo što se misli na posredno keširanje

U zavisnosti od prirode resursa, servis može da odluči vrednosti ovih zaglavlja i direktiva. Server, klijent i bilo koja posredna komponenta između njih treba da prate ove direktive da bi izbegli serviranje zastarelih informacija.

2.3 Uniformni interfejs (URI)

Glavno svojstvo po kome se REST arhitekturni model razlikuje od ostalih mrežnih modela je upravo uniformni interfejs između komponenata. Uniformi interfejs potpuno odvaja zaduženja servera i klijenta i tako pojednostavljuje samu arhitekturu. HTTP 1.1 nudi komplet metoda koje nazivamo glagoli. Najpoznatiji su:

Metoda	Operacija na serveru	Kvalitet
GET	Čita resurs	Sigurno
PUT	Ažurira ili prepisuje postojeći resurs	*idempotent*
POST	Ubacuje novi resurs.	N/A
DELETE	Briše resurs	*idempotent*
OPTIONS	Izlistava dozvoljene operacije na resursu	Sigurno

Sigurna operacija je ta koja nema nikakav uticaj na originalnu vrednost resursa. Na primer, matematička operacija „podeljeno sa 1“ je sigurna operacija, jer koliko god puta da podelimo broj sa 1, originalna vrednost se neće promeniti.

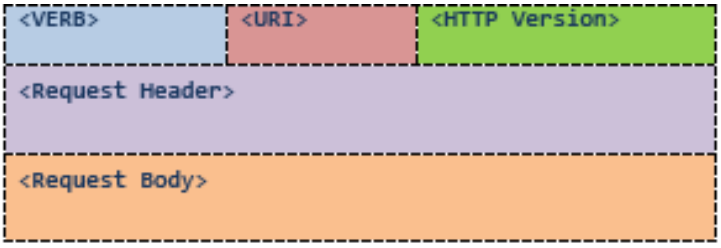
Idempotent operacija je operacija koja nam daje isti rezultat bez obzira koliko puta je izvršimo. Na primer, matematička operacija „pomnoži sa 0“ je idempotentna, jer koliko god puta pomnožimo broj sa 0, rezultat je uvek isti.

REST zahteva uniformni interfejs a HTTP ga obezbeđuje. Ipak, na programerima i arhitektama je da ga održe uniformnim.

2.4 Izričito korišćenje HTTP metoda

Jedna od ključnih karakteristika RESTful web servisa je izričito korišćenje HTTP metoda na način koji prati protokol definisan od strane RFC-a 2616. HTTP GET, na primer, je definisan kao metod koji je namenjen da bude korišćen kao klijentska aplikacija za pristup resursu, da povlači podatke sa web servera, ili da izvršava upit sa očekivanjem da će web server odgovoriti sa kompletom odgovarajućih resursa.

Klijent i servis komuniciraju međusobno putem poruka. Klijent šalje zahtev serveru, a server odgovara odazivom. Za dizajniranje RESTful servisa, jako je bitno imati pozadinu vezanu za HTTP 1.1 zahtev i odgovor. HTTP zahtev ima sledeći format:



Primer 1: Format HTTP zahteva

`<VERB>` je jedna od HTTP metoda, kao npr. GET, PUT, POST, DELETE...

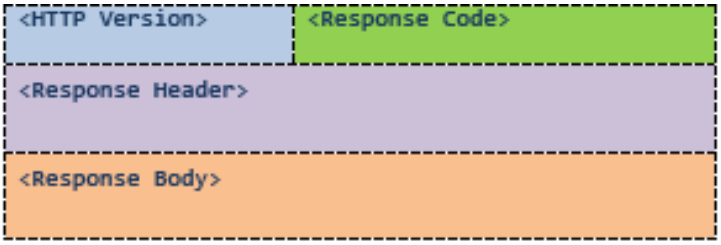
`<URI>` je uniformni interfejs resursa na kome će biti izvedena operacija

`<HTTP Version>` je verzija HTTP-a

`<Request Header>` sadrži meta podatke kao kolekciju ključnih vrednosti zaglavlja. Ovde su sadržane informacije o poruci i pošiljaocu, formati koje klijent podržava, format tela poruke, keš podešavanja za odgovor, i mnoge druge informacije.

`<Request Body>` je sadržaj poruke.

Odgovor bi izgledao ovako:



Primer 2: HTTP odgovor

`<HTTP Version>` je verzija HTTP-a.

`<Response Code>` sadrži status zahteva.

`<Response Header>` sadrži meta podatke i podešavanja vezana za odgovor.

`<Response Body>` sadrži prikaz ukoliko je zahtev bio uspešan.

Metode koje se koriste u HTTP/1.1 su:

1. ***OPTIONS***
   Metoda „Options“ predstavlja zahtev za informacijom o opcijama komunikacije u lancu zahtev/odgovor, prepoznatljivu od strane URI zahteva. Ovam metoda omogućava klijentu da utvrdi opcije i/ili potrebe vezane za resurse, kao i mogućnosti servera.

2. ***GET***
   Ovom metodom se zahteva prikaz određenog resursa. Koristi se samo za čitanje podataka, što znači da se smatra sigurnim.

3. ***HEAD***
   „Head“ metoda je identična kao i GET, s tim što server ne vraća telo poruke u odgovoru. Ova metoda može da se koristi za prikupljanje meta podataka zapisanih u zaglavlju odgovora, bez prebacivanja kompletnog sadržaja.

4. ***POST***
   Metod POST se koristi za kreiranje novih resursa. Podaci koji se kreiraju mogu biti npr, poruke za oglasnu tablu, mailing liste, komentari, predmet koji se dodaje u bazu podataka itd.

5. ***PUT***
   Ova metoda se najčešće koristi za ažuriranje, zamenu ili kreiranje novog prikaza resursa.

6. ***DELETE***
   Ova metoda se koristi za brisanje resursa.

7. ***TRACE***
   Vraća primljene zahteve tako da klijent može da vidi da li je napravljena neka izmena od strane posrednih servera.

8. ***CONNECT***
   Konvertuje konekciju u transparentni TCP/IP tunel. Obično se koristi da olakša SSL enkriptovanu komunikaciju kroz neenkriptovani HTTP proksi.

2.5 Transfer XML i/ili JSON

RESTful servisi su fokusirani na resurse i način kako da obezbede pristup istim. Prilikom dizajniranja sistema, prva stvar na koju obraćamo pažnju je identifikacija resursa i određivanje kako su resursi vezani jedan za drugi. Princip je sličan kao kod dizajniranja baze podataka: identifikacija entiteta i veza među njima.

Jednom kada identifikujemo naše resurse, sledeće šta moramo da uradimo je da te resurse prezentujemo u našem sistemu. REST dozvoljava korišćenje bilo kog formata za prezentaciju resursa.

Najbolji način da klijentskoj aplikaciji damo mogućnost da zahteva određeni tip sadržaja koji im odgovara, je da konstruišemo servis tako da koristi već ugrađeno HTTP Accept poglavlje čija vrednost je MIME(Multipurpose Internet Mail Extensions) tipa. Osnovni MIME tipovi koje koriste RESTful servisi su JSON i XML.

Na primer, u zavisnosti od naših potreba, možemo da odlučimo koji tip da koristimo. Ukoliko razvijamo Web servis koji će biti korišćen od strane Web stranica pozivanih od strane AJAX-a, onda je JSON bolja opcija. XML se koristi da prezentuje mnogo kompleksnije resurse. Na primer, resurs „osoba“ može biti prezentovan kao:

JSON

```
{
   "ID": "1",
    "Name": "M Vaqqas",
    "Email": "m.vaqqas@gmail.com",
    "Country": "India"
}
```

XML

```
<Person>

<ID>1</ID>

<Name>M Vaqqas</Name>

<Email>m.vaqqas@gmail.com</Email>

<Country>India</Country>
</Person>
```

U suštini, možemo da koristimo više od jednog formata i da odlučimo koji se koristi prilikom odgovora, a u zavisnosti od vrste klijenta ili nekog od parametara zahteva. Koji god format da koristimo, dobra prezentacija bi trebala da ima određene kvalitete:

- I klijent i server treba da budu upoznati sa formatom prezentacije
- Prezentacija bi trebalo da bude u mogućnosti da predstavi resurs u potpunosti. Ukoliko postoji potreba za parcijalnom prezentacijom resursa, onda možemo da podelimo resurs ma manje delove. Manje prezentacije automatski znače i manje potrebnog vremena za kreiranje i trensfer istih, što podrazumeva i bržu uslugu.
- Prezentacija treba da bude u mogućnosti da povezuje resurse jedan sa drugim.

Korišćenje MIME tipova i HTTP *Accept* zaglavlja je poznato kao pregovor između sadržaja(*content negotiation)*, što omogućava klijentima da biraju koji je format podataka pogodan za njih i smanjuje mogućnost dupliranja između servisa i aplikacije koja ga koristi.

3. Bezbednost web servisa

Pretnje Web servisima uključuju i pretnje samom sistemu, aplikacijama kao i kompletnoj mrežnoj infrastrukturi. Da bi zaštitili Web servise, potreban je čitav opseg zaštitnih mehanizama kako bi se rešio problem vezan za autentifikaciju, kontrolu pristupa i distribuiranje sigurnosnih polisa.

Implementacija Web servisa moze zahtevati *point-to-point* i/ili *end-to-end* sigurnosne mehanizme, u zavisnosti od nivoa pretnje ili rizika. Obično, *point-to-point* sigurnosni mehanizmi ne mogu da zadovolje *end-to-end* sigurnosne zahteve Web servisa. Ipak, sigurnost je balans procene rizika i cene kontramera.

3.1 Pretnje na sloju poruka

Tradicionalni sigurnosni mehanizmi na nivou mreže, kao što su SSL/TLS, VPN, IPSec i S/MIME, su point-to-point tehnologije. Iako su ovo tradicionalne sigurnosne tehnologije u Web servisima, nisu dovoljne da obezbede end-to-end mehanizme.

Web servisi se suočavaju sa tradicionalnim sigurnosnim izazovima. Poruka moze da putuje izmedju nekoliko različitih posrednika pre nego što stigne do odredišta. Zbog toga je sigurnost na sloju poruka jednako bitna kao i sigurnost na nivou transporta.

Pretnje koje se odnose na sigurnost poruka su:

1. ***Izmena poruka***
   Ove pretnje utiču na integritet poruka gde napadač može da izmeni delove ili celu poruku. Takođe, napadač može da manipuliše I prilozima (*attachments*) koji se nalaze u poruci.

2. ***Poverljivost***
   U ovoj pretnji, neovlašćena strana dobija pristup informacijama koje su namenjene krajnjem primaocu. Primer može da bude pristup informacijama sa kreditne kartice.

3. ***Man-in-the-middle (posrednik)***
   U ovoj vrsti napada, postoji mogućnost da napadač kompromituje SOAP posrednika I presretne poruku između pošiljaoca I krajnjeg primaoca. Obe strane će misliti da komuniciraju jedna sa drugom. Obostrana autentikacija može da se koristi da bi se ublažila ova vrsta pretnje.

4. ***Spoofing***
   Spoofing je kompleksan napad koji koristi sigurne veze. Napadač preuzima identitet jedne od strana da bi sabotirao sigurnosni mehanizam druge strane.

5. ***Denial of Service (DoS) napad***
   DoS napadi su fokusirani na sprečavanje korisnika da koriste određeni servis. Jednostavni su za izvođenje i mogu izazvati popriličnu štetu.

6. ***Replay napad***
   U ovom napadu, napadač presreće poruku i vraća je pošiljaocu. Najbolja vrsta zaštite je, u ovom slučaju, tehnika autentikacije uparena sa tehnikama kao što su vremenski pečat i brojanje sekvenci (*sequence numbering*).

Glavni zadatak sigurnosti je da primeni set mera bezbednosti i stvori okolinu gde transakcije na sloju poruka mogu da se sprovedu na siguran način, bilo da postoje posrednici ili ne.

Sigurnost web servisa uključuje nekoliko aspekata:

3.2 Autentikacija (*Authentication*)

Autentikacija utvrđuje da li je osoba ona kojom se predstavlja. Nekoliko metoda može biti korišćeno za autentikaciju: šifre, OTP(*One time pass*) i sertifikati. Postoji nekoliko metoda koje se koriste za podešavanje autentikacije:

- Osnovna autentikacija: je jedan od najlakših načina implementacije sigurnosti. Kao što i samo ime govori, jednostavna je po tome što šaljemo korisničko ime i šifru u *Base64* obliku šiforvanja, bez naprednih opcija. Uvek treba da se koristi sa SSL enkripcijom, da bi se sprečilo dekodiranje kredencijala.

- Oauth 1.0a: je protokol baziran na potpisima. Nalazi se u širokoj upotrebi, dobro je istestiran i veoma siguran. Protokol koristi kriptografske potpise, kao što su HMAC-SHA1, čija je prednost kombinovanje token koda za jednu upotrebu i drugih informacija koje su zahtevane.

- Oauth 2: Ovaj protokol u potpunosti izostavlja potpise, zbog čega je i mnogo jednostavniji. Zahteva TLS (*Transport layer security*), koji se brine o enkripciji. Oauth 2 se mnogo manje koristi i ima dosta slabiju sigurnost u poređenju sa Oauth 1.0a. Zbog toga se korist za manje osetljive podatke.

- Ustanovljeni sigurnosni protokol (*Custom security protocol*): Protokol koji treba izbegavati iz razloga što niko osim nas samih ne koristi naš ustanovljeni protokol. Ovaj protokol stavlja odgovornost na samog kreatora.

Zbog navedenih prednosti i mana, sigurnije i jednostavnije je koristiti Osnovnu autentikaciju sa SSL-om za vecinu REST servisa.

3.3 Autorizacija (*Authorization*)

Autorizacija određuje da li ste prava osoba za pristup resursima. Kad se autentifikujemo, autorizacioni mehanizmi kontrolišu pristup određenim sistemskim resursima. Prava korisnika su definisana od strane nekoliko atributa. Atribut je osobina ili karakteristika korisnika. Autorizacija može biti podešena I na resursima kontrolisanim od strane operativnog sistema koristeći grupe, ili na resursima u web aplikaciji koji su postavljeni na J2EE server (npr, WebSphere Aplication Server) koristeći LDAP korisničke grupe.

3.4 Integritet podataka i komunikacije

Tehnika integriteta podataka se brine o tome da informacija ne bude izmenjena u toku prenosa. Ova tehnika se takođe brine o tome da su podaci dostupni onima kojima su namenjeni. Što se tiče integriteta komunikacije, on je potreban da bi se utvrdilo da su proces i protok informacija izvršeni kako treba.

3.5 Neporečivost

Je sigurnosni servis koji se brine da poruka ostane nepromenjena prilikom prenosa, tako što zahteva da pošiljaoc digitalno potpiše poruku.

3.6 Poverljivost i privatnost (*confidentiality, privacy*)

Jednostvano, čuvanje poverljivih informacija. Poverljivost i privatnost mora biti osigurana čak i u prisustvu posrednika.

U dosta slučajeva, sigurnosni alati web servisa (kao što je Oracle WSM) oslanjaju se na PKI (*Public Key Infrastructure*) okruženje. PKI koristi kriptografske ključeve koji mogu biti privatni ili javni. Privatni ključ se koristi da kreira digitalni potpis tako što potpisuje poruku, a javni da proveri potpis.

4. Bezbednost RESTful web servisa

Bezbednost RESTful web servisa može da se obezbedi korišćenjem jedne od sledećih metoda za autentikaciju, autorizaciju i enkripciju:

- Ažuriranjem ***web.xml*** identifikatora za razvoj da bi se definisala sigurnosna konfiguracija
- Korišćenje ***javax.ws.rs.core.SecurityContext*** interfejsa za implementaciju programske sigurnosti
- Primena napomena za JAX-RS klase
- Korišćenje Jersey Oauth biblioteka za potpisivanje i proveru zahteva

4.1 Zaštita RESTful Web servisa korišćenjem web.xml

Da bi zaštitili RESTful web servis koristeći osnovnu autentifikaciju, potrebno je pratiti sledeće korake:

- Definisati `<security-constraint>` za svaki set RESTful resursa (URI) koji planiramo da zaštitimo
- Koristiti `<login-config>` element za definisanje tipa autentifikacije koji želimo da koristimo, kao i za oblast zaštite na koji ćemo primeniti sigurnosna ograničenja
- Definisati jednu ili više sigurnosnih rola koristeći `<security-role>` oznaku, i mapirati ih na sigurnosno ograničenje definisano u koraku 1.
- Da bi omogućili enkripciju, dodati `<user-data-constraint>` element, i postaviti `<transport-guarantee>` podelement na `CONFIDENTAL`.

```
<web-app>
    <servlet>
        <servlet-name>RestServlet</servlet-name>
        <servlet-
class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>RestServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
    <security-constraint>
         <web-resource-collection>
             <web-resource-name>Orders</web-resource-name>
             <url-pattern>/orders</url-pattern>
             <http-method>GET</http-method>
             <http-method>POST</http-method>
         </web-resource-collection>
         <auth-constraint>
             <role-name>admin</role-name>
         </auth-constraint>
    </security-constraint>
        <login-config>
            <auth-method>BASIC</auth-method>
```

```
            <realm-name>default</realm-name>
        </login-config>
    <security-role>
        <role-name>admin</role-name>
    </security-role>
</web-app>
```

Primer 1 – Zaštita RESTful Web servisa korišćenjem osnovne autentifikacije

4.2 Zaštita RESTful Web servisa korišćenjem SecurityContext

`Java.ws.rs.core.SecurityContext` interfejs nudi pristup sigurnosnim informacijama vezanim za zahtev. **`SecurityContext`** nudi funkcionalnosti slične kao **`javax.servlet.http.HttpServletRequest`**, pružajući pristup sledećim sigurnosnim informacijama:

- **`Java.security.Principal`** sadrži ime korisnika koji je poslao zahtev.
- Tipu autentifikacije koja štiti resurs, kao što je **`BASIC_AUTH`**, **`FORM_AUTH`** i **`CLIENT_CERT_AUTH`**.
- Da li je autentifikovani korisnik uključen u izvesnu rolu.
- Da li je zahtev napravljen koristeći sigurnosni kanal, kao što je **`HTTPS`**.

```
package samples.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.SecurityContext;
import javax.ws.rs.core.Context;
...

@Path("/stateless")
@Stateless(name = "JaxRSStatelessEJB")
public class StlsEJBApp {
...
        @GET
        @Produces("text/plain;charset=UTF-8")
        @Path("/hello")
        public String sayHello(@Context SecurityContext sc) {
                if (sc.isUserInRole("admin"))  return "Hello World!";
                throw new SecurityException("User is unauthorized.");
        }
```

Primer 2 – Zaštita RESTful Web servisa korišćenjem SecurityContext

4.3 Zaštita RESTful Web servisa korišćenjem napomena (*Annotations*)

`Javax.annotation.security` paket nudi napomene (definisane u listi ispod), koje možemo da koristimo prilikom zaštite RESTful Web servisa.

Anotacije (napomene) za zaštitu RESTful Web servisa su:

- **`DeclareRoles`** – Deklariše role.
- **`DenyAll`** – Precizira da ni jedna sigurnosna rola nije dozvoljena da pozove određene metode.
- **`PermitAll`** – Suprotno od DenyAll, svim rolama je dozvoljeno da pozovu određene metode.
- **`RolesAllowed`** – Precizira listu određenih rola, kojima je dozvoljeno da pozovu određene metode.
- **`RunAs`** – definiše identitet aplikacije tokom izvršavanjau J2EE.

Sledeći primer prikazuje kako se definiše sigurnosna rola kojoj je dozvoljen pristup metodama definisanim u **`helloWorld`** klasi. Pristup je obezbeđen samo korisnicima koji pripadaju ADMIN grupi:

```
package samples.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.annotation.Security.RolesAllowed;


@Path("/helloworld")
@RolesAllowed({"ADMIN", "ORG1"})
public class helloWorld {

   @GET
   @Path("sayHello")
   @Produces("text/plain")
   @RolesAllows("ADMIN")
   public String sayHello() {
      return "Hello World!";
   }
}
```

Primer 2 – Zaštita RESTful Web servisa korišćenjem Anotacija

Zaključak

Pojavom interneta počela je globalizacija sveta u svim segmentima ljudskog života, ponajviše računarskom. Tehnologije se rađaju I koriste zahvaljujući svojim pozitivnim aspektima I rasprostranjenosti, ili pak izumiru zbog svojih nedostataka ili nezastupljenosti. WWW(*World Wide Web*) je najveća distribuirana aplikacija na svetu. Razumevanje osnovnih arhitekturnih principa koji se nalaze u pozadini Web-a, može da pomogne u objašnjavanju njegovog uspeha i da vodi do pronalaženja novih poboljšanja, koja bi mogla da se integrišu u druge distribuirane aplikacije.

REST je najbolji način za razvijanje ultra lakih Web servisa koji su jednostavni za implementaciju, održavanje i otkrivanje. HTTP nudi odličan interfejs za implementaciju RESTful servisa, sa karakteristikama kao što su uniformni interfejs ili keširanje.

Osnovni cilj ovog rada bilo je upoznavanje sa REST tehnologijom i servisima baziranim na njoj. Takođe, opisana je i razlika između REST i SOAP tehnologije. Zatim je bilo reči o Web servisima zasnovanim na REST-u, da bi na kraju opisali princip zaštite i sigurnosti RESTful Web servisa.

Literatura

- Roy T. Fielding (2000): **Architectural Styles and the Design of Network-based Software Architectures.**
- John Cowan (2005): **RESTful Web Services.**
- R.T.Fielding, Richard N. Taylor (2002): **Principled Design of the Modern Web Architecture.**
- Singhal (2007): **Guide to Secure Web Services, NIST SP 800-95.**
- A. Njeguš, G. Grubor (2009): **Zaštita Web servisa, Sinergija 2009.**

Linkovi:

[1] http://theopentutorials.com/tutorials/web-services/types-of-web-services-big-and-restful/
[2] http://docs.oracle.com/cd/E50245_01/E50253/E50253.pdf
[3] http://docs.oracle.com/cd/E24329_01/web.1211/e24983/secure.htm#RESTF113
[4] http://www.networkedassets.com/configuring-spring-security-for-a-restful-web-services/
[5] http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069
[6] http://www.ibm.com/developerworks/library/ws-restful/
[7] http://www.tutorialspoint.com/webservices/what_are_web_services.htm