REST ARHITEKTURA

Kuserbajn, Sara

VELEUČILIŠTE U ŠIBENIKU

ODJEL INFORMATIčKI MENADŽMENT

STRUČNI STUDIJ MENADŽMENT

Sara Kuserbajn

REST ARHITEKTURA

Završni rad

Šibenik, 2016

VELEUČILIŠTE U ŠIBENIKU

ODJEL INFORMATIČKI MENADŽMENT

STRUČNI STUDIJ MENADŽMENT

REST ARHITEKTURA

Završni rad

Kolegij: Baze podataka

Mentor: dr.sc. Frane Urem, v. pred.

Student: Sara Kuserbajn

Matični broj studenta: 14392131

Šibenik, kolovoz 2016

Sadržaj

1. Uvod 1/2
2.Definicija REST arhitekture 3
3. Poznati stilovi arhitekture 4
3.1 Stil protok podataka (Data-flow Styles) 4
3.2 Replikacijski stil (Replication Styles) 4
3.3 Hijerarhijski stil (Hierarchical Styles) 4
3.4 Stil pokretan kod (Mobile Code Styles) 5
3.5 Stil ravnopravan ravnopravnom (Peer-to-Peer Styles) 5
4. Proces nastajanja REST-a 6
5. Roy Thomas Fielding I predstavljanje REST arhitekture 7
4.1 Fieldingov opis REST-a 7
4.2 Fieldingov retrospektivan pogled na nastanaka REST-a 7
6. REST ograničenja 8
5.1 Službena REST ograničenja 8/9/10
7. REST elementi arhitekture 11
7.1 Elementi podataka 11
7.2 Resursi I identifikatori resursa 12
7.3 Reprezentacije 12
7.4 Konektori 12/13
7.5 REST komponente 13/14
8. REST pogled arhitekture 15
8.1 Pogled na proces 15
8.2 Pogled konektora 15/16
8.3 Pogled podataka 16
9. Svojstva REST arhitekture 17
10. API- Aplikacijsko programsko sučelje 18
11. Web usluge 19
12. RESTful web usluge 20
12.1 RESTful dizajn 20
12.2 Svojstva RESTful usluga 20/21
13. HTTP 22/23

14. REST primijenjen na HTTP 24
14.1 Nepodudaranje između REST-a I HTTP-a 24/25
15. REST primijenjen na URI 26
15.1 Redefiniranje resursa 26
15.2 REST nepodudarnosti s URI 26/27
16. Kršenje REST ograničenja 28
16.1 REST API apecifikacije 28
17. Zaključak 29/30
Literatura 31
Prilog 32/33

1. Uvod

REST danas predstavlja jedan od najvažnijih i najpopularnijih stilova arhitekture za opisivanje željene web arhitekture i kreiranje aplikacija i web usluga. Što je zapravo REST, kako je nastao, koja je njegova svrha, pravilan način interpretacije i implementacije, te koji je razlog za njegovo kreiranje kao hibrida više postojećih stilova arhitekture kompleksna su pitanja koja traže poznavanje cjelokupne računalne i informacijske tehnologije i terminologije (protokoli, standardi, stilovi, mreže, usluge, aplikacije) potrebne za razumijevanje REST-a i načina njegove implementacije.

Od izuma prvih računala pa do danas, ni jedna tehnologija nije toliko brzo napredovala ni doprinijela mnogobrojnim otkrićima kao ona računalna. Iako su prva računala imala vrlo limitirane sposobnosti te su bila samo za privatnu upotrebu njihove mogućnosti su se mijenjale velikom brzinom te su ubrzo postala dostupna široj populaciji. Najznačajnije postignuće unutar računalne tehnologije postao je internet osnovan 1969 godine od strane ministarstva obrane sa ciljem povezivanja određenog broja računala. Nakon toga, kako bi računala mogla razmjenjivati podatke preko interneta nastao je World Wide Web poznat i kao svjetska mreža.

World Wide Web je internacionalna mreža kompjuterskih baza podataka koja koristi internet i njegov poseban sistem za povezivanje informacija. Jedna je od najkorištenijih usluga interneta, pomoću koje se dohvaćaju hipertekstualni dokumenti. Ti dokumenti mogu sadržavati tekst, slike i multimedijalne sadržaje. Nakon nastanka web je postao ključan za ulazak u informacijsko doba te primarni alat korišten za interakcije na internetu.[1] Uz World Wide Web nastali su i drugi alati potrebni za njegov rad poput HTTP, HTML… Od komercijalizacije WWW-a stalno se radilo na njegovom razvitku te je on postao brzo razvijajuća industrija. Razvoj weba može uključivati jednostavne stvari poput razvitka jednostavnih statičnih stranica teksta, i najkompleksnije web temeljene internet aplikacije, mrežne usluge, elektroničko poslovanje. Razvijanje weba često podrazumijeva web inženjerstvo, web dizajn, razvijanje web sadržaja, web sisteme... Uz mnogobrojne usluge i stranice dostupne na webu pokazala se potreba za web arhitekturom. Ona se sastoji od zahtjeva, ograničenja, principa i izbora dizajna koji utječu na dizajn sistema, te ponašanje agenata unutar sistema. Rana web arhitektura temeljena je na solidnim principima; razdvajanje problema, jednostavnost, općenitost, no nedostajalo je opisa arhitekture i logičke podloge.

Stil arhitekture može se koristiti za definiranje principa iza web arhitekture na način da bude vidljivi budućim arhitektima. Stil ubraja set ograničenja na elmentima arhitekture koji obuhvaćaju set svojstva koja su tražena od arhitekture. Za stvaranje stila arhitekture potrebno je identificirati ograničenja unutar web arhitekture koja su odgovorna za željena značajke. Kako u postojećoj web arhitekturi nisu bili udovoljeni svi njeni zahtjevi te su postojali problemi u dizajnu i evaluaciji poboljšanja bilo je potrebno stvoriti metodu za dizajniranje poboljšanja arhitekture na način da se poboljšanja evaluiraju prije njihovog uvođenja. Iz te

[1] https://hr.wikipedia.org/wiki/World_Wide_Web (10. 9. 2016)

potrebe nastao je stil arhitekture REST. On predstavlja rezultat stvaranja stila arhitekture za definiranje i poboljšanje logike iza dizajna web arhitekture, te stil koji bi se koristio kao test za dokazivanje predloženih poboljšanja prije njihovog korištenja, te kako bi se koristila promijenjena i ispravljena arhitektura preko direktnog sudjelovanja u projektima razvijanja softvera koji su kreirali infrastrukturu weba. [2]

REST predstavlja stil arhitekure za distribuirane hipermedijske sustave te je stvoren za predstavljanje modela kako bi web trebao funkcionirati. REST pruža set ograničenja arhitekture koja ako se koriste zajedno naglašavaju važnost skalabilnosti interakcije komponenta , općenitost sučelja, neovisnost korištenja komponenta i posredničke komponente za smanjenje vremena čekanja interakcije, te nameću sigurnost i enkapsuliraju postojeće starije sisteme. REST je nastao kao hibrid drugih, mrežno baziranih stilova arhitekture, te je kombiniran sa dodatnim ograničenjima koja definiraju jedinstveno sučelje konektora.[3] REST je specificiran svojim ograničenjima i elementima arhitekture, te pristupu stvaranja web usluga što ga čini drugačijim od drugih stilova arhitekture.

REST arhitektura generalno se sastoji od klijenta, poslužitelja, resursa te HTTP operacija poznatijih kao metode zahtjeva koje REST koristi za svoj dizajn. Klijent šalje zahtjev poslužitelju, a poslužitelj odgovara. U REST-u ti zahtjevi su neovisni jedan o drugome. Te interakcije su centrirane oko interakcije resursa poput dokumenata ili URI-a. Reprezentacije se izmjenjuju uz pomoć standardiziranog sučelja HTTP. Aplikacije temeljene na REST-u podržavaju njegove principe i ograničenja. Ona dolaze u obliku jedinstvenog sučelja koje uključuje identifikaciju resursa, u obliku manipulacije resursa, samo opisnih poruka te se smatraju osnovnim pravilima REST-a. Web usluge koje su dizajnirane u skladu sa tim principima i ograničenjima nazivaju se RESTful usluge.

[2] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 75.

[3] Ibid, str. 76.

2. Definicija REST arhitekture

REST- Reprezentacijsko stanje prijenosa

REST (Representational State Transfer) predstavlja stil arhitekture aplikacijskog programskog sučelja koji se sastoji od koordiniranog skupa komponenti, konektora i podatkovnih elemenata unutar distribuiranog hipermedijskog sustava.

REST označava način komunikacije između klijenta i poslužitelja prilikom korištenja mrežnih resursa pomoću HTTP protokola.

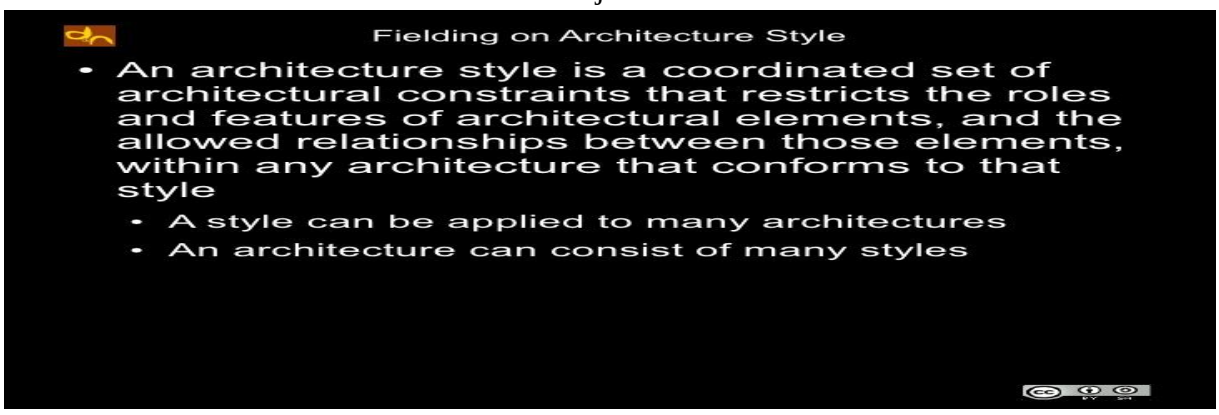
REST je softverski stil arhitekture weba.

REST je set principa koji definira kako bi web standardi poput HTTP i URI-a trebali biti korišteni.

REST je korišten kako bi opisao željenu web arhitekturu, identificirao postojeće probleme, usporedio alternativna rješenja i osigurao da proširenje protokola neće kršiti temeljna ograničenja koja čine web uspješnim.[4] Ideja REST-a jest da se umjesto kompleksnih mehanizama poput CORBE, RPC ili SOAP-a koristi jednostavan HTTP za uspostavljanje veze između uređaja. Sistemi koji udovoljavaju REST ograničenjima nazivaju se RESTful sistemi, a oni obično komuniciraju preko HTTP-a putem HTTP metoda. Na mnogo načina, sam World Wide Web temeljen na HTTP-u može se gledati kao REST temeljena arhitektura.

Samo ime Reprezentacijsko stanje prijenosa namijenjeno je za dočaravanje slike ponašanja pravilno dizajniranih web aplikacija. Mreža web stranica gdje se korisnik kreće kroz aplikaciju koristeći veze, što rezultira prijenosom sljedeće stranice do korisnika za njihovu upotrebu.[5]

Slika 1: Definicija stila arhitekture



Preuzeto iz [1]

[4] https://en.wikipedia.org/wiki/Representational_state_transfer (8. 9. 2016)
[5] Ibid

3. Poznati stilovi arhitekture

REST je nastao kao hibrid (spoj) nekoliko mrežno temeljenih stilova arhitekture. Ti stilovi su česti stilovi arhitekture za mrežno temeljeni aplikacijski softver koji su klasificirani na način da se svaki stil može procijeniti prema svojstvima arhitekture koje bi izazvao ako bi se primijenio na arhitekturu za prototip mrežno temeljenog hipermedijskog sistema. Neki od poznatih stilova su: [6]

3.1 Stil protok podataka (Data - flow Styles)

Cijev i filter je stil gdje svaka komponenta čita niz podataka tijekom njihovog unosa i stvara niz podataka na njihovim izlazima obično u isto vrijeme kada primjenjuje transformaciju na ulazni tok podataka te ih procesira postupno na način da izlazni tok započinje prije nego što se ulazni tok u potpunosti obradi. Ograničenje je stavljeno na to da filter mora biti u potpunosti neovisan o drugim filterima. Stil jedinstveni cijev i filter dodaje ograničenje na filtere koji moraju imati isto sučelje. Ograničenje sučelja dopušta neovisno razvijenim filterima da budu posloženi po volji kako bi formirali novu aplikaciju.

3.2 Replikacijski stil (Replycation Styles)

Sistem baziran na stilu replikacijski repozitorij poboljšava pristupnost podacima i skalabilnost usluga na način da postoji više od jednog procesa koji pružaju iste usluge. Glavna prednost je poboljšanje korisnički percipirane performanse. U stilu priručne memorije može se naći više replikacijskih repozitorija i replikacija rezultata individualnih zahtjeva tako da se mogu koristiti u kasnijim zahtjevima. Ova vrsta replikacije najčešće se pronalazi u slučajevima gdje set potencijalnih podataka nadmašuje kapacitet klijenta (poput www-a) ili gdje je kompletan pristup repozitoriju nepotreban.

3.3 Hijerarhijski stil (Hierarchical Styles)

Stil klijent - poslužitelj najčešći je stil za mrežno temeljene aplikacije. Komponenta poslužitelj nudi niz usluga te čeka zahtjev vezan uz te usluge. Komponenta klijent koja želi da se usluga obavi šalje zahtjev poslužitelju preko konektora. Poslužitelj ili obavi ili odbije zahtjev te šalje odgovor klijentu. Odvajanje problema sastavni je dio ograničenja na stilu klijent - poslužitelj. Dobro odvajanje funkcionalnosti trebalo bi pojednostavniti komponentu poslužitelja kako bi se popravila skalabilnost. Slojeviti sistem i slojeviti klijent poslužitelj stilovi su gdje je slojeviti sistem organiziran hijerarhijski tako da svaki sloj pruža usluge sloju iznad njega i koristi usluge sloja ispod njega. Slojeviti sistem smanjuje vezanje preko više slojeva skrivanjem unutarnjeg sloja od svega osim susjednog vanjskog sloja. Slojeviti klijent – poslužitelj stil dodaje posrednike stilu klijent - poslužitelj. Stil udaljena sesija je varijanta stila klijent poslužitelj koja pokušava minimalizirati kompleksnost ili maksimalizirati višekratno korištenje komponenta klijenta umjesto komponenta poslužitelja. Stil se obično

[6] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 41-59.

koristi kada se želi pristupiti udaljenoj usluzi koristeći generički klijent ili prijeko sučelja koje oponaša generičkog klijenta.

3.4 Stil pokretni kod (Mobile Code Styles)

Stil pokretni kod koristi mobilnost kako bi dinamički promijenio udaljenost između izvora podataka ili destinacije rezultata. Predstavljanjem koncepta lokacije postaje moguće da se modelira cijena interakcije između komponenta na nivou dizajna. U stilu kod na zahtjev komponenta klijenta ima pristup setu resursa ali ne zna kako ih obraditi. Ona šalje zahtjev udaljenom poslužitelju za kod koji će joj omogućiti obradu, te ih obrađuje nakon primitka koda.

3.5 Stil ravnopravan ravnopravnom (Peer-to-Peer Styles)

Stil integracije temeljene na događaju također poznat i kao stil sistem događaja smanjuje vezanje između komponenta tako što uklanja potrebu za identifikacijom na sučelju konektora. Umjesto da pozove druge komponente direktno komponenta može najaviti jedan ili više događaja. Druge komponente mogu registrirati interes za taj tip događaja te kada je najavljen sistem pozove sve registrirane komponente. Stil C2 je usmjeren na podržavanje ponovnog korištenja i fleksibilno oblikovanje komponenta sistema provođenjem neovisnosti podloge. To radi kombinacijom stila integracija temeljena na događaju sa slojevitim klijent – poslužitelj stilom. Asinkrone poruke koje idu prema dolje i asinkrone poruke koje idu prema gore su glavna sredstva komunikacije između komponenta. To se provodi laganim vezanjem ovisnosti na višim slojevima i bez vezanja na nižim slojevima što poboljšava kontrolu nad sistemom bez gubljenja glavnih prednosti integracije temeljene na događaju.

4. Proces nastajanja REST-a

Poznajući zahtjeve www arhitekture, te probleme koji mogu nastati u dizajnu i evaluaciji predloženih poboljšanja moguće je korištenjem poznatih stilova arhitekture te uvidom u njihove mogućnosti napraviti hipotetsku metodu za razvoj i evaluaciju arhitektonskog stila koji bi vodio dizajn poboljšanja za modernu arhitekturu World Wide Weba. Za stvaranje novog stila arhitekture potrebno je definirati željena svojstva i značajke, te kombinirati ranije stilove i ostale potrebne elemente koji bi pripomogli u stvaranju novog stila.

Hipermedija je odabrana kao korisničko sučelje zbog svoje jednostavnosti i općenitosti; isto sučelje može biti korišteno neovisno o izvoru informacija, fleksibilnost hipermedijskih veza dopušta kompleksnim vezama unutar sučelja da vode čitatelje kroz aplikaciju. Za autore primarni zahtjev bio je da djelomična dostupnost cjelokupnog sistema ne sprječava stvaranje sadržaja. Hipertekst jezik za stvaranje sadržaja trebao bi biti jednostavan, te bi se trebao moći stvoriti uz pomoć postojećih alata za obradu. Jednostavnost je također bila jedna od ciljeva za stvaranje aplikacija. [7]Hipermedija je definirana prisutnošću kontrole aplikacijskih informacija ugrađenom unutar, ili kao sloj iznad prezentacije informacija. Korisničke aplikacije unutar distribuiranog hipermedija traže prijenos velike količine podataka iz mjesta gdje su spremljeni, do mjesta gdje će biti korišteni. Web je zbog toga morao biti dizajniran za prijenos velikih količina informacija.

Kako je stil imenovani set ograničenja na elementima arhitekture koji indiciraju set željenih svojstva bilo je potrebno identificirati ograničenja smještena unutar rane web arhitekture. Sljedeći korak bio je identificirati svojstva koja su željena na internet skali distribuiranog hipermedijskog sistema, odabrati dodatne arhitektonske stilove koji induciraju ta svojstva, te ih kombinirati sa ranijim web ograničenjima. Koristeći novi stil arhitekture kao vodič moguće je usporediti predložena proširenja i modifikacije web arhitekture sa ograničenjima unutar stila. Sukobi pokazuju da će prijedlozi prkositi nekima od svojstva dizajna weba. Te sukobe bilo je potrebno ukloniti ili otkriti prije implementacije. Poznajući to bilo je moguće postaviti tri hipoteze za razvijanje novog stila: [8]

1. Logička podloga dizajna iza www arhitekture može biti opisana kao stil arhitekture koji se sastoji od seta ograničenja primijenjenih na elementima unutar web arhitekture.
2. Ograničenja se mogu dodati www arhitektonskom stilu kako bi proizvela novi hibridni stil koji bolje prikazuje željena svojstva moderne web arhitekture
3. Prijedlog za modificiranje web arhitekture može se usporediti sa novijim www stilom arhitekture te analizirati za sukobe prije implementacije.

Poznajući tražena i željena svojstva web arhitekture te njihovim kombiniranjem sa postojećim stilovima koji ih podržavaju došlo je do nastanka novog hibridnog stila koji predstavlja model kako bi moderan web trebao funkcionirati.

[7] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 67-68.
[8] Ibid, str. 73-74.

5. Roy Thomas Fielding i predstavljanje REST arhitekture

Roy Thomas Fielding rođen je 1965. Slavan je američki računalni znanstvenik. Doktorirao je 2000 godine na Sveučilištu u Kaliforniji (University of California, Irvin). Poznat je kao jedan od glavnih autora HTTP specifikacija, ko-osnivač projekta Server Apache HTTP, te akademik i stručnjak u području arhitekture računalnih mreža. Radi kao glavni znanstvenik u tvrtci Adobe Systems (San Jose, California). Godine 1999 imenovan je kao jedan od top 100 inovatora u dobi od 35 godina. Fielding je REST arhitekturu predstavio u svojoj doktorskoj disertaciji Stilovi arhitekture i dizajn mrežno baziranih softvera arhitekture (Arhitectural styles and the Design of Network-based Software). U toj disertaciji REST je predstavio kao ključni pristup arhitekture World Wide Webu. Danas je REST postao temeljan pristup razvoju web servisa te alternativa drugim distribuiranim računalnim specifikacijama poput SOAP-a. Svoju disertaciju je opisao kao tekst koji je fokusiran na logiku koja stoji iza dizajna moderne web arhitekture, te kako se ona razlikuje od ostalih stilova arhitekture.

5.1 Fieldingov opis REST-a: [9]

Stil arhitekture za distribuirane hipermedijske sustave koji opisuje
načela softverskog inženjerstva kojima se vodi REST i interakcijska ograničenja odabrana da podržavaju ta načela, istodobno ih razlikujući od ograničenja drugih stilova arhitekture.

REST je hibridan stil nastao iz nekoliko mrežnih stilova arhitekture te je kombiniran sa dodatnim ograničenjima koja definiraju jednoličan konektor sučelja.

Stil arhitekture REST bio je zamišljen kao model koji opisuje kako bi web aplikacije trebale funkcionirati.

5.2 Fieldingov retrospektivan pogled na nastanak REST-a:[10]

„Tijekom procesa standardizacije HTTP-a pozvan sam da obranim izbor dizajna weba. To je bilo vrlo teško za učiniti unutar procesa koji prihvaća prijedloge od bilo koga, na temu koja je ubrzano postajala središte cijele industrije. Imao sam komentare od više od 500 izumitelja, od kojih su mnogi bili cijenjeni inženjeri sa nekoliko desetljeća iskustva a morao sam objašnjavati sve, od osnovnih apstraktnih pojmova web interakcije do najmanjih detalja HTTP sintakse. Taj je proces sveo moj model na samu jezgru, temeljan skup načela, svojstva i ograničenja koja se danas nazivaju REST.“

[9] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 76.

[10] https://en.wikipedia.org/wiki/Representational_state_transfer (8. 9. 2016)

6. REST ograničenja

Dizajn arhitekture weba može se opisati stilom arhitekture koji se sastoji od niza ograničenja primijenjenih na elemente unutar arhitekture. Ispitivanjem utjecaja svakog ograničenja njihovim dodavanjem na razvijajući stil možemo identificirati svojstva izazvana tim ograničenjima. Ta ograničenja se tada mogu iskoristiti za formiranje novog stila arhitekture koji bi bolje odražavao poželjna svojstva moderne web arhitekture. Svojstva arhitekture REST-a nastala su primjenom specifičnih ograničenja interakcije na komponente, konektore i podatkovne elemente.

6.1 Službena REST ograničenja[11]

Klijent – Poslužitelj (Client – Server)

Prvo ograničenje dodano je arhitektonskom stilu klijent – poslužitelj. Ono razdvaja klijenta od poslužitelja na način da svatko ima svoje zadatke. Klijent se ne bavi pohranom podataka već to radi poslužitelj čime se poboljšava prijenos podataka klijenta. Poslužitelji se ne brinu o korisničkom sučelju ili korisničkom stanju što ih čini jednostavnijima i skalabilnijima. Poslužitelj i klijent mogu se mijenjati i razvijati neovisno jedan o drugome tako dugo dok se sučelje između njih ne mijenja. Ono što je na webu najznačajnije jest da odvajanje klijenta i poslužitelja omogućava komponentama da se razvijaju samostalno, čime se podržava zahtjev internet skale za višestrukom organizacijskom domenom.

Neovisnost između zahtjeva (Stateless)

To ograničenje dodaje se interakciji između klijenta i poslužitelja. Komunikacija se mora odvijati na način da je svaki novi zahtjev neovisan o onom prijašnjem što čini zahtjev i odgovor neovisnim parom. Svaki poslani zahtjev od klijenta mora sadržavati sve potrebne informacije kako bi ga se razumjelo jer se ne može iskoristiti bilo kakav prošli zahtjev ili kontekst za bolje razumijevanje. Ta ograničenja potiču svojstva vidljivosti, pouzdanosti i skalabilnosti. Vidljivost je poboljšana time da se ne mora tražiti ni jedan prijašnji zahtjev kako bi se shvatio sadašnji. Pouzdanost je poboljšana zato što je omogućeno lakše oporavljanje od djelomičnog neuspjeha. Skalabilnost je poboljšana time što se ne mora pohranjivati stanje između zahtjeva što omogućava poslužitelju da brže oslobodi resurse i time poboljšava implementaciju, zato jer poslužitelj ne mora upravljati korištenim resursima tijekom novog zahtjeva. Nedostatak ovog ograničenja je u tome što može smanjiti performanse mreže ponavljanjem podataka u različitim zahtjevima ili odgovorima, što je potrebno pošto se kontekst ne može iščitati iz prošlog razgovora.

Priručna memorija (Cache)

Ograničenje stavljeno na priručnu memoriju zahtjeva da se podaci unutar odgovora na zahtjev implicitno ili eksplicitno označavaju kao podaci koji se spremaju u priručnu memoriju ili kao oni koji se ne spremaju. Podaci koji se ne spremaju su podaci čija se vrijednost mijenja te ih

[11] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 78-85.

stoga nema svrhe koristiti. Ako se podatak označi kao onaj koji se sprema onda ga klijent može ponovno koristiti za kasniji odgovarajući zahtjev. Prednost dodavanje ograničenja priručne memorije je u tome da poboljšava učinkovitost i skalabilnost na način da djelomično ili potpuno uklanja neke interakcije što smanjuje prosječno vrijeme odaziva niza interakcija.

Jedinstveno sučelje (Uniform interface)

Jedinstveno sučelje je od temeljne važnosti za dizajn bilo kojeg REST servisa te ono razdvaja njegov stil arhitekture od ostalih mrežnih stilova. Ono pojednostavnjuje i odvaja arhitekturu što omogućava da se svaki dio razvija samostalno. Također poboljšava vidljivost interakcija. Problem je u tome što jedinstveno sučelje smanjuje učinkovitost pošto se podaci prenose u standardiziranom obliku, a ne onom specifičnom za potrebe aplikacije. Pošto REST sučelje nije optimalno za druge oblike interakcije arhitekture kako bi se dobilo jedinstveno sučelje potrebno je više ograničenja arhitekture kojima bi se vodile komponente. Definirana su četiri ograničenja sučelja;

Identifikacija resursa – Pojedini resursi identificirani su u zahtjevima korištenjem URI-a u na primjer web baziranim REST sustavima. Resursi su konceptualno odvojeni od reprezentacija koje su vraćene klijentima. Na primjer, poslužitelj može slati podatke iz svoje baze kao HTML, XML ili JSON.

Manipulacija resursima kroz manipulaciju – Kada klijent drži reprezentaciju resursa uključujući metapodatke u prilogu onda ima dovoljno informacija da izmjeni ili izbriše resurse.

Samo opisne poruke – Svaka poruka sadrži dovoljno informacija koje opisuju kako je obraditi.

Hipermedija kao motor aplikacije – Klijenti obavljaju stanje prijelaza samo kroz akcije koje su dinamički identificirane unutar hipermedije od strane poslužitelja. Osim jednostavnih fiksnih ulaznih točaka aplikacije klijent ne pretpostavlja da je bilo koja određena radnja dostupna za bilo koji resurs osim onih opisanih u prikazu koji je prethodno primljen od poslužitelja. Ne postoji univerzalno prihvaćeni format za predstavljanje veze između dva resursa. RFC 5988 i JSON Hipermedija API Jezik su dva popularna oblika za određivanje REST hipermedijske veze

Slojeviti sustav (Layered system)

Slojeviti stil sustava omogućava arhitekturi da bude sastavljena od hijerarhijskih slojeva tako što ograničava ponašanje komponente na način da svaka komponenta ne može vidjeti dalje od trenutnog sloja sa kojim je u interakciji. Ograničavanjem poznavanja sustava na jedan sloj povezuje se ukupna složenost sustava i promovira se neovisnost podloge. Primarni nedostatak slojevitih sustava je što povećavaju vrijeme obrade podataka što smanjuje performansu.

Kod na zahtjev (Code on demand)

REST omogućava proširenje klijentove funkcionalnosti preuzimanjem i izvršavanjem koda u obliku skripte ili appleta. (Java appleti, JavaScript). Tako se smanjuje broj značajki koje se moraju provesti. Dopuštanjem da se značajke preuzmu nakon implementacije dopušta proširenje sustava ali i smanjuje vidljivost, no to je samo dodatno ograničenje unutar REST-a. Opcionalno ograničenje omogućava da se dizajnira arhitektura koja podržava željeno ponašanje u općem slučaju ali s razumijevanjem da se ono može onemogućiti u nekim kontekstima.

Slika 2: Kod na zahtjev

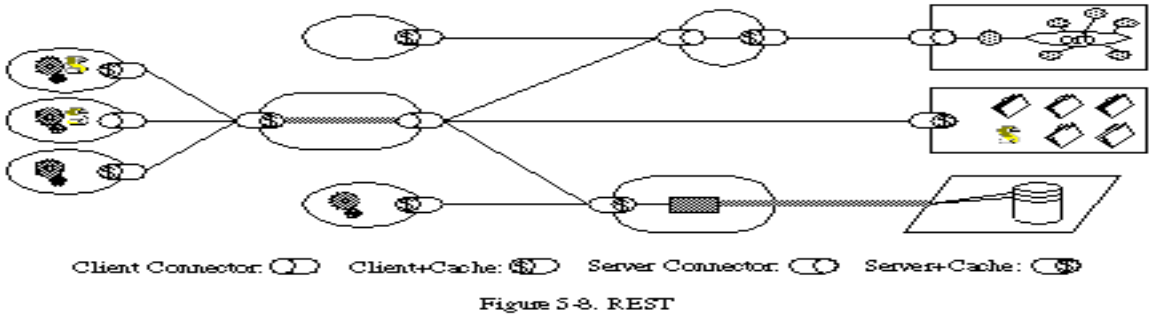


Figure 5-8. REST

Preuzeto iz [2]

7. REST elementi arhitekture

Elementi arhitekture su jedinstveni detalji i dijelovi komponenata koji zajedno formuliraju stil arhitekture.

REST stil je apstrakcija elemenata arhitekture unutar distribuiranog hipermedijskog sustava. REST ignorira detalje implementacije komponenata i sintakse protokola kako bi se fokusirao na uloge komponenata, ograničenja u njihovoj interakciji s drugim komponentama i interpretaciju značajnih elemenata podataka. REST obuhvaća temeljna ograničenja na komponentama, konektorima i podacima koji definiraju temelj web arhitekture, a time i suštinu svog ponašanja kao aplikacije temeljene na mreži.

REST razlikuje tri klase elemenata arhitekture a to su: elementi podataka, konektori i komponente.[12]

7.1 Elementi podataka

Za razliku od distribuiranog objekta gdje su svi podaci enkapsulirani i skriveni od komponenata za obradu, priroda i stanje arhitekture podatkovnih elemenata je ključni aspekt REST-a. Razlog za taj dizajn pronalazi se u prirodi distribuiranog hipermedija. Odabirom veze informacije trebaju biti poslane iz lokacije gdje su spremljene na mjesta gdje će biti korištene. Arhitekt distribuirane hipermedije ima tri opcije; 1) Preurediti podatke na mjestu gdje su spremljeni te poslati preuređeni format primatelju. 2) Zahvatiti podatke sa programom za preuređivanje te oboje poslati primatelju. 3) Poslati nepreuređene podatke primatelju zajedno sa metapodacima koji opisuju tip podataka kako bi primatelj mogao sam izabrati program za preuređivanje. Svaka opcija ima prednosti i nedostatke. REST pruža hibrid svih triju opcija tako što se fokusira na zajedničko razumijevanje tipa podataka uz pomoć metapodataka, istovremeno ograničavajući ono što je vidljivo standardiziranom sučelju. REST komponente komuniciraju tako što prenose reprezentaciju resursa u formatu koji odgovara razvijajućoj skupini standardiziranih tipova podataka, odabranog dinamički i temeljenog na mogućnostima ili željama primatelja te prirode resursa. Dali je prikaz u istom formatu kao i original ili je izveden iz originala ostaje skriveno iza sučelja. Na taj način REST pruža odvajanje briga stilu klijent – poslužitelj bez problema sa skalabilnošću poslužitelja, dopušta skrivanje informacija kroz generičko sučelje kako bi onemogućio enkapsulaciju usluga, te pruža raznoliki set funkcionalnosti kroz programe koje je moguće skinuti.

Tablica 1 : REST elementi podataka

Elementi podataka	Primjeri
Resursi (Resource)	konceptualna meta hipertekst reference
Identifikator resursa (Resource identifier)	URI (URL ili URN)
Metapodaci resursa (Resource metadata)	veza izvora, alternative,
Reprezentacije (Representation)	HTML, XML, JSON, PPG
Metapodaci reprezentacije (Representation	vrsta medija, posljednji put modificirano

[12] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 86-97.

metadata)	
Kontrolni podaci (Control data)	ako-modificirano-od, kontrola priručne memorije

Preuzeto iz [1]

7.2 Resursi i identifikatori resursa

Ključna apstrakcija informacija u REST-u je resurs. Bilo koja informacija koja se može imenovati može biti i resurs; dokument, slika, privremene usluge, kolekcije drugih resursa, ne virtualni objekti... Svaki koncept koji može biti meta autorove hipertekst reference mora se uklapati u definiciju resursa. Resurs je konceptualno mapiranje seta entiteta, a ne entitet koji odgovara na mapiranje u bilo kojem vremenskom trenutku. Preciznije, resurs R je vremenski različita funkcija članstva koja za vrijeme (t) mapira set entiteta ili vrijednosti koje su ekvivalentne. Vrijednosti u setu mogu biti reprezentacija resursa i/ili identifikatori resursa. REST koristi identifikator resursa kako bi identificirao određeni resurs, uključen u interakciju između komponenata. REST konektori pružaju generičko sučelje za pristup i manipulaciju vrijednosnog seta resursa neovisno o tome kako je funkcija članstva definirana, i vrsti softvera koji rukovodi zahtjev. REST se obično oslanja na to da autor odabere identifikator resursa koji se najbolje uklapa u prirodu koncepta za identifikaciju.

7.3 Reprezentacije

REST komponente obavljaju akcije na resursu korištenjem reprezentacije kako bi uhvatile trenutačno ili buduće stanje resursa, te prenosile tu reprezentaciju između komponenata. Reprezentacija je sekvenca bitova zajedno sa reprezentacijskim metapodacima koji opisuju te bitove. Druga imena za reprezentaciju su dokumenti, instance, varijance... Reprezentacija se sastoji od podataka, metapodataka koji opisuju te podatke a ponekad i od metapodataka koji opisuju metapodatke (obično kako bi se provjerio integritet poruke). Metapodaci su u obliku para ime-vrijednost gdje ime odgovara standardu koji definira vrijednosnu strukturu i semantiku.

Kontrolni podaci definiraju svrhu poruke između komponenata kao što su tražene akcije ili značenje poruke. Također su korišteni za definiranje parametara zahtjeva te kako bi nadvladali zadano ponašanje elemenata. Na primjer, ponašanje priručne memorije može biti modificirano uz pomoć kontrolnih podataka uključenih u poruku odgovora ili zahtjeva.

7.4 Konektori

REST koristi nekoliko tipova konektora kako bi enkapsulirao aktivnosti obuhvaćanja resursa te prijenos reprezentacije resursa. Konektori predstavljaju apstraktno sučelje za komunikaciju komponenata povećavajući jednostavnost pružanjem jasnog odvajanja problema te skrivanjem temeljne implementacije resursa i mehanizma komunikacije. Sve REST interakcije su međusobno neovisne, što znači da svaki zahtjev mora sadržavati sve informacije potrebne kako bi konektor razumio zahtjev bez korištenja prošlih zahtjeva. To ograničenje potiče četiri funkcije;

1) Uklanja potrebu za tim da konektori zadržavaju aplikacijsko stanje između zahtjeva čime se smanjuje potrošna resursa te povećava skalabilnost. 2) Omogućava da se interakcije

procesiraju paralelno bez zahtjeva da mehanizam za procesiranje razumije semantiku interakcije. 3) Dopušta posredniku da pregledava i dopušta zahtjev u izolaciji, što može biti korisno i potrebno kada su usluge dinamički preuređene. 4) Tjera sve informacije koje bi mogle biti faktor u ponovnoj upotrebi spremljenog zahtjeva da budu prisutne u svakom zahtjevu.

Tablica 2 : REST konektori

Konektor klijent (Client connector)	Inicira komunikaciju i izvršava zahtjeve
Konektor poslužitelj (Server connector)	Čeka vezu i odgovara na zahtjeve
Konektor priručne memorije (Cache connector)	Pridružuje se klijentu ili poslužitelju, sprema odgovore u zahtjeve za kasnije korištenje
Rješavač (Resolver)	Prevodi identifikatore resursa u mrežnu adresu za stvarnu komunikaciju
Tunel (Tunnel)	Prenosi komunikaciju preko granica komunikacije

Preuzeto iz [2]

Primarni tipovi konektora su klijent i poslužitelj. Temeljna razlika između njih je ta što klijent pokreće komunikaciju slanjem zahtjeva dok poslužitelj odgovara na zahtjev kako bi omogućio pristup svojim uslugama. Komponenta može sadržavati konektore i od klijenta i od poslužitelja.

Treći tip konektora, konektor priručne memorije lociran je na sučelju klijenta ili poslužitelja kako bi u priručnu memoriju spremio odgovore koji se mogu koristiti za daljnje interakcije. Priručna memorija može biti korištena od strane klijenta kako bi izbjegao ponavljanje mrežne komunikacije ili od strane poslužitelja kako bi izbjegao proces generiranja odgovora. Oba slučaja služe kako bi se smanjilo vrijeme čekanja. Priručna memorija je obično implementiran a unutar mjesta za adresu konektora koji je koristi.

Rješavač prevodi djelomične ili cijele identifikatore resursa u informacije mrežne adrese koje su potrebne kako bi se uspostavila inter-komponentna veza. Na primjer, većina URI-a uključuje DNS (Domaine Name System) ime hosta kao mehanizam za identifikaciju autoriteta za imenovanje resursa. Kako bi inicirao zahtjev web pretraživač će odvojiti ime hosta od URI-a i iskoristiti DNS rješavač kako bi dobio adresu internet protokola tog autoriteta.

Tip konektora zvan tunel (Tunell) prenosi komunikaciju preko granica povezivanja poput vatrozida ili niže razine mrežnog pristupa. Razlog zašto je uključen u REST a ne odbačen kao dio mrežne infrastrukture je taj što se neke REST komponente mogu dinamički prebaciti iz aktivnog ponašanja komponente na ono od tunela. Temeljan primjer za to je HTTP posrednik koji se prebacuje u tunel kao odgovor na zahtjev CONNECT metoda, što omogućava klijentu direktnu komunikaciju sa udaljenim poslužiteljem koristeći drugačiji protokol, poput TLS koji ne dozvoljava posrednike. Tunel nestaje kada obje strane okončaju komunikaciju.

7.5 REST komponente

Tabica 3 : REST komponente

Korisnički agent (User agent)	Koristi konektor klijenta kako bi izvršio

	zahtjeve te je posljednji primatelj odgovora
Izvorni poslužitelj (Origin server)	Koristi konektor poslužitelj za procesiranje zahtjeva i pružanje odgovora.
Posrednik (Proxy)	Posrednik odabran od klijenta za poboljšanje performanse, prevođenje podataka...
Poveznik (Gateway)	Posrednik nametnut od mreže ili izvornog poslužitelja sa sličnim funkcijama kao i posrednik.

Preuzeto iz [3]

Korisnički agent koristi klijent konektor kako bi inicirao zahtjev te postao konačan primatelj odgovora. Najčešći primjer jest web pretraživač koji pruža pristup informacijskim uslugama i čini da pružanje usluga bude u skladu s aplikacijskim potrebama.

Izvorni poslužitelj koristi konektor poslužitelj za upravljanje prostorom imena za zatraženi resurs. On je konačan izvor za reprezentaciju svojih resursa i mora biti krajnji primatelj bilo kojeg zahtjeva koji namjerava modificirati vrijednosti svojih resursa. Svaki izvorni poslužitelj pruža generičko sučelje svojim uslugama kao hijerarhiju resursa. Detalji implementacije resursa su skriveni iza sučelja.

Komponente posrednika ponašaju se i kao klijent i kao poslužitelj kako bi prenijeli, sa mogućim prijevodom, zahtjeve i odgovore. Komponenta posrednik je posrednik od klijenta kako bi pružio enkapsulaciju sučelja drugih usluga, prevođenja podataka, zaštitu sigurnosti...

Komponenta poveznik je posrednik nametnut od mreže ili izvornog poslužitelja kako bi pružio enkapsulaciju sučelja drugih usluga, za prevođenje podataka, povećanje performanse ili uvođenje sigurnosti. Razlika između posrednika i poveznika je ta što klijent određuje kada će koristiti posrednika.

8. REST pogled arhitekture

Uz pomoć REST elemenata arhitekture može se koristiti pogled arhitekture za opisivanje kako elementi funkcioniraju zajedno u formiranju arhitekture. Postoje tri tipa pogleda; proces, konektor, i podaci za ilustraciju načela dizajna REST-a.[13]

8.1 Pogled na proces

Pogled na proces arhitekture posebno je efektivan za izazivanje interakcije odnosa među komponentama, otkrivajući tok podataka dok se kreću kroz sistem. Nažalost, interakcija pravih sistema obično uključuje poveći broj komponenata što rezultira pogledom koji je zaklonjen zbog previše detalja.

Odvajanje problema u REST-u u stilu klijent-poslužitelj pojednostavnjuje implementaciju komponenta, smanjuje složenost semantike konektora, poboljšava efektivnost podešavanja performanse, te povećava skalabilnost komponenata poslužitelja.

Ograničenja slojevitog sustava dopuštaju posrednicima poput proxya, gatewaya i vatrozida da budu predstavljeni nekoliko puta tijekom komunikacije bez mijenjanja sučelja između komponenata što im dopušta da pomažu u prevođenju komunikacije ili poboljšavanju performanse. REST omogućava procesiranje posrednika na način da ograničuje poruke na to da budu samo opisne što znači da je interakcija između zahtjeva neovisna, a standardne metode i vrste medija su korišteni kako bi naznačili semantiku i razmijenili informacije.

Kako su komponente povezane dinamički njihove funkcije za određene akcije aplikacije imaju karakteristike slične stilu „cijev i filter“. Generičko sučelje konektora dopušta komponentama da budu smještene na tok temeljen na svojstvima svakog zahtjeva i odgovora. Kako su u REST-u sve interakcije međusobno neovisne to uklanja potrebu za informiranosti o cjelokupnom razmještaju komponenta, i dopušta komponentama da se ponašaju kao posrednici ili destinacije, što se utvrđuje dinamički, ovisno o meti određenog zahtjeva.

8.2 Pogled konektora

Pogled konektora arhitekture usredotočuje se na mehanizme komunikacije između komponenti. REST temeljenoj arhitekturi posebno su zanimljiva ograničenja koja definiraju generičko sučelje resursa. Konektori klijenta proučavaju identifikatore resursa kako bi izabrali prikladan komunikacijski mehanizam za svaki zahtjev. Na primjer, klijent bi mogao biti konfiguriran da se poveže na specifičnu komponentu posrednika, moguće onu koja se ponaša kao anotacijski filter kada identifikator pokazuje da je resurs lokalan. Isto tako klijent može biti konfiguriran da odbije zahtjev za neki podskup identifikatora. REST ne ograničava komunikaciju na određeni protokol ali ograničava sučelje između komponenata a time i opseg interakcije i pretpostavke implementacije koje bi inače bile između komponenata. Na primjer, temeljni prijenosni protokol weba je HTTP, ali arhitektura također uključuje nesmetan pristup

[13] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 97-103.

resursima koji postoje na prijašnjim mrežnim poslužiteljima uključujući FTP, Gopher i WAIS. Interakcija sa tim uslugama je ograničena na semantike REST konektora.

8.3 Pogled podataka

Pogled podataka arhitekture otkriva stanje aplikacije dok informacije teku kroz komponente. Kako je REST posebno usmjeren na distribuirane informacijske sustave on gleda aplikaciju kao koherentnu strukturu informacija i alternative kontrole kroz koje korisnik može izvršiti željeni zadatak.

REST usmjerava dva kontrolna stanja u reprezentacije primljene kao odgovor na interakcije. Cilj je poboljšanje skalabilnosti poslužitelja eliminirajući bilo kakvu potrebu za tim da poslužitelj prati stanje klijenta van trenutnog zahtjeva. Aplikacijsko stanje je definirano svojim iščekivanim zahtjevima, razmještajem povezanih komponenata, aktivnim zahtjevima na komponentama, tokom podataka reprezentacije kao odgovor na zahtjeve i obrade tih zahtjeva kako dolaze korisničkom agentu.

Korisnički doživljaj aplikacije ovisi o vremenu čekanja između ustaljenog stanja; periodu između selekcije veze na web stranici i trenutka kada su dobivene korisne informacije. Kako REST temeljene arhitekture komuniciraju prvenstveno kroz prijenos reprezentacije resursa, vrijeme čekanja može ovisiti i o dizajnu komunikacijskih protokola i o dizajnu formata za reprezentaciju podataka. Sposobnost postupnog povećavanja prikazivanja podataka kako su primljeni utvrđeno je dizajnom vrste medija i dostupnosti formata podataka unutar svake reprezentacije. Sljedeće kontrolno stanje aplikacije nalazi se u reprezentaciji prvog zatraženog resursa tako da je dobivanje prve reprezentacije prioritet. REST interakcija je tako popravljena uz pomoć protokola koji „ prvo odgovaraju a zatim razmišljaju“. Drugim riječima, protokol koji zahtjeva više interakcija po radnji korisnika kako bi radio stvari poput pregovarao o značajkama sposobnosti prije slanja odgovora, bit će puno sporiji od protokola koji šalje prvi optimalan odgovor a zatim pruža listu alternativa za klijenta, u slučaju da prvi odgovor nije zadovoljavajući.

Slika 3 : Pogled na proces

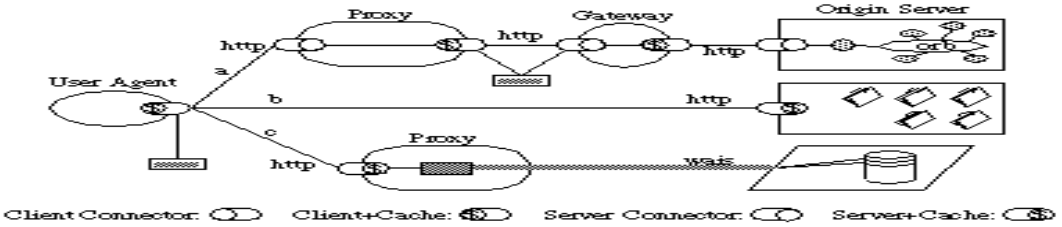


Figure 5-10. Process View of a REST-based Architecture
A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

Preuzeto iz [3]

9. Svojstva REST arhitekture

Performansa – Interakcije komponente mogu biti dominantan faktor u korisničkom doživljaju izvedbe i mrežnoj učinkovitosti. Način interakcije komponenta utječe na performansu.

Skalabilnost – Služi za potporu velikoj količini komponenata i interakciji između komponenata.

Jednostavnost – Između interakcije sučelja

Modifikacija – Modifikacija komponenta kako bi se zadovoljile potrebe za promjenom čak i u vrijeme rada programa.

Vidljivost – Prozirnost komunikacije između komponenata od strane predstavnika.

Prenosivost – Prenošenje komponenata pomicanjem programskog koda sa podacima.

Pouzdanost – Otpornost na neuspjeh na sistemskoj razini prilikom nazočnosti kvarova unutar komponenata, konektora i podataka. [14]

[14] https://en.wikipedia.org/wiki/Representational_state_transfer (8. 9. 2016)

10. API – Aplikacijsko programsko sučelje

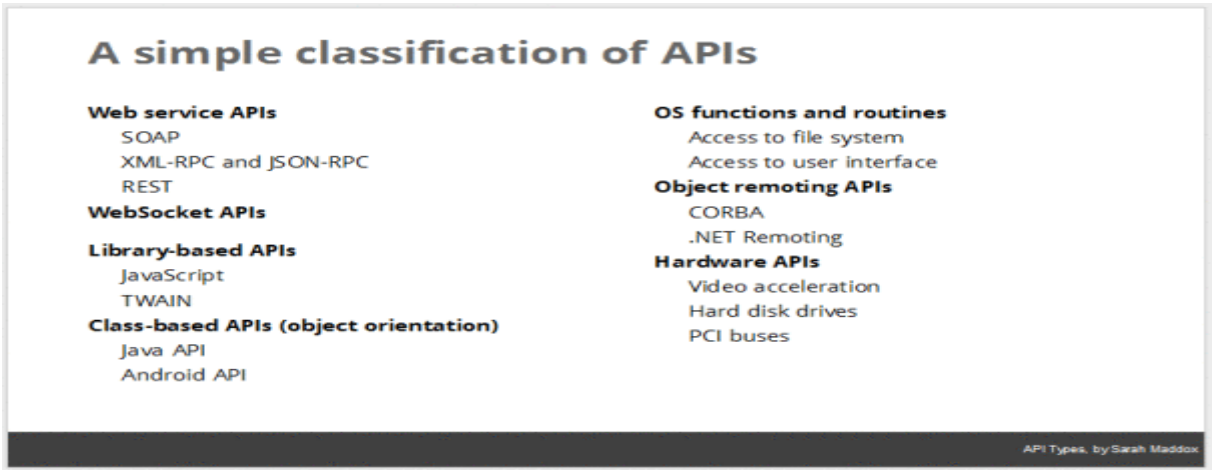
API je skup određenih pravila i specifikacija koje programeri slijede tako da se mogu služiti uslugama ili resursima operacijskog sustava ili nekog drugog složenog programa kao standardne biblioteke rutina, struktura podataka, objekata i protokola.

U računalnom programiranju API je skup rutinskih definicija, protokola i alata za izgradnju softvera i aplikacija.

API se može specificirati na različite načine, primjerice kao međunarodni standard, kao dokumentacija proizvođača softvera, kao biblioteka programskog jezika.

Dobro API olakšava izradu programa tako što pruža potrebne „građevinske blokove“ koje sastavljaju programeri. API može biti za web bazirane sustave, operacijske sustave, sustave baza podataka, hardvere ili softverske biblioteke. Specifikacije API dolaze u mnogo oblika a najčešće sadrže specifikacije za rutine, strukture podataka, klase, objekte, varijable… Različiti oblici API-a su; POSIX, JavaAPI, Microsoft Windows API… API je obično povezano sa bibliotekom softvera. Ono opisuje i propisuje pravila očekivanog ponašanja no biblioteka je ta koja ta pravila implementira.[15] Jedno API može imati više implementacija u obliku različitih biblioteka koje dijele isto programsko sučelje. U većini proceduralnih jezika specifikacije API su funkcije ili rutine koje obavljaju specifične zadatke ili im je dozvoljena interakcija sa specifičnim komponentama softvera. Te specifikacije dolaze u obliku papirnate knjige ili u elektroničkom formatu (eBooks). U objektno orijentiranom jeziku, u svojem najjednostavnijem obliku API opisuje kako rade objekti. Obično se izražava kao skup klasa povezanih sa listom metoda klasa.

Slika 4 : Klasifikacija API



Preuzeto iz [4]

[15] https://en.wikipedia.org/wiki/Application_programming_interface (28. 9. 2016)

11. Web usluge

Web usluga je usluga koju jedan elektronički uređaj nudi drugome tijekom komunikacije putem World Wide Weba.

Softverski sistem dizajniran da podrži interoperabilne stroj na stroj interakcije preko mreže.

Pojam web usluge opisuje standardizirani način integracije web baziranih aplikacija koristeći XML, SOAP, WDSL i UDDI otvorene standarde preko glavnog podatkovnog smjera internetskog protokola (Internet backbone). XML se koristi za označavanje podataka, SOAP za prijenos podataka, WDSL za opis dostupnih usluga, a UDDI nudi listu dostupnih usluga. U praksi web usluge obično pružaju objektno orijentirano web temeljeno sučelje poslužitelju baze podataka koje se može koristiti od strane drugog poslužitelja ili mobilne aplikacije koje pružaju korisničko sučelje krajnjem korisniku.

Mogu se razlikovati dva glavna razreda web usluga: [16]

RESTful web usluge čija je primarna svrha manipulacija predstavljenih web resursa koristeći jedinstveni skup međusobno neovisnih operacija.

Proizvoljne web usluge u kojima web usluga može izložiti proizvoljan niz operacija.

Web API predstavlja novi dio web usluga fokusiran na jednostavniju, REST baziranu komunikaciju. REST APIs ne trebaju XML bazirane protokole web usluga (SOAP , WDSL) kako bi podržale njihova sučelja.

[16] https://en.wikipedia.org/wiki/Web_service (28. 8. 2016)

12. RESTful web usluge

REST se primarno koristi za razvoj web usluga koje su lake za održavanje i skalabilne. Usluga bazirana na REST arhitekturi naziva se RESTful uslugom. REST je neovisan o bilo kojem protokolu ali skoro svaka REST usluga koristi HTTP kao temeljni protokol**.**

12.1 RESTful dizajn

Svaki sustav koristi resurse. Ti resursi mogu biti slike, video, web stranice, poslovne informacije itd. Svrha usluge jest da pruži „prozor“ ili „okvir“ svojim klijentima, tako da oni mogu pristupiti tim resursima. Arhitekti usluga žele da te usluge mogu lako implementirati, da budu održive, skalabilne, nadogradive.. To je ono što bi trebala pružiti svaka usluga koja je RESTful.

12.2 Svojstva RESTful usluga: [17]

Reprezentacija

Fokus REST usluga su resursi i kako pružiti pristup tim resursima. Jedan resurs se može sastojati od više drugih resursa. Prilikom dizajniranja sistema prvo je potrebno identificirati resurse i utvrditi kako se oni međusobno povezuju. Nakon identifikacije resursa potrebno je naći način za reprezentaciju ili prikaz tih resursa. Dozvoljeno je korištenje bilo kojeg formata za reprezentaciju jer REST na njih ne stavlja ograničenja. Ovisno o zahtjevu može se koristi XML ili JSON.

Poruke

Klijent i poslužitelj razgovaraju putem poruka. Klijent pošalje zahtjev a poslužitelj na njega odgovara. Osim podataka te poruke sadrže i metapodatke. Kako bi se dizajnirala RESTful usluga potrebno je poznavati HTTP 1.1 format za zahtjeve i odgovore.

URI

REST zahtjeva da svaki resurs ima barem jedan URI (niz znakova koji se koriste za identifikaciju resursa). URI služi za identifikaciju resursa ili skupa resursa.

Jedinstveno sučelje

RESTful sistemi bi trebali imati jedinstveno sučelje. HTTP 1.1 pruža skup metoda nazvanih riječima. Najpoznatije riječi su GET, PUT, POST, DELETE, OPTIONS, HEAD… REST preporuča jedinstveno sučelje dok ga HTTP pruža.

Neovisnost između zahtjeva

U REST stilu interakcija je neovisna što znaći da se ne podržava pohrana podataka za nijednog klijenta. Zahtjev ne može biti ovisan o prošlom zahtjevu te usluga tretira svaki novi zahtjev kao samostalan.

[17] http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069 (23,9,2014)

Veza između resursa

Reprezentacija resursa može sadržavati veze do drugih resursa kao što i web stranica pruža vezu do druge stranice. Ovdje se te veze nalaze u HTTP dokumentima.

Privremeno skladištenje sadržaja

To je koncept memoriranja generiranih rezultata i korištenja tih rezultata umjesto da se oni iznova generiraju. To se može napraviti na klijentu, poslužitelju ili bilo kojoj drugoj komponenti između njih kao što je na primjer posrednik poslužitelj. Privremeno skladištenje sadržaja je odličan način za poboljšavanje korisničkog doživljaja ali ako se ne koristi primjereno može doći do toga da klijent koristi zastarjele podatke. Ono se može kontrolirati korištenjem sljedećih HTTP zaglavlja; Date, Last modified, Cache-control, Age, Expires.

13. HTTP

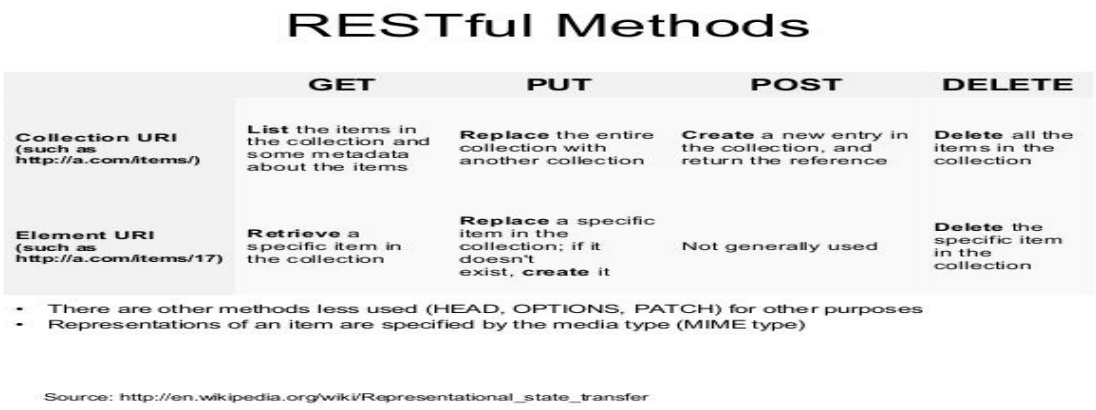
HTTP Hyper Text Transfer Protocol je aplikacijski protokol za distribuirane, suradničke, hipermedijske informacijske sustave. HTTP je temelj komunikacije podataka za www. Hipertekst (Hypertext) je strukturirani tekst koji koristi logičke veze između čvorova koji sadrže tekst. HTTP je protokol za zamjenu ili prijenos hiperteksta.

HTTP funkcionira kao protokol zahtjev – odgovor u računalnom modelu klijent – poslužitelj. Kao primjer, web preglednik može biti klijent a aplikacija koja smješta web stranicu može biti poslužitelj. Klijent podnosi HTTP zahtjev u obliku poruke poslužitelju. Poslužitelj koji osigurava resurse kao što su HTTP datoteke i drugi sadržaji ili obavlja različite funkcije u ime klijenta, vraća poruku odgovora klijentu. Taj odgovor sadrži informacije o zahtjevu, a može sadržati i informacije o zahtjevu u tijelu poruke.

HTTP definira metode koje ukazuju na željene akcije koje bi se trebale izvršiti nad identificiranim resursom. HTTP 1.0 definira metode GET, POST i HEAD. HTTP 1.1 dodaje pet novih metoda a to su OPTIONS, PUT, DELETE, TRACE I CONNECT. Najčešće korištene HTTP metode su POST, GET, PUT, PATCH, DELETE.[18]

REST definira način na koji bi se trebao koristiti HTTP. On HTTP metode koristi kao „gradbene blokove“. Većina, ako ne i sve RESTful aplikacije koriste HTTP. REST koristi pristup sličan CRUD (Create, Read, Update, Delete) pristupu kod SQL jezika te za svaku operaciju koristi metode iz HTTP protokola. GET se koristi za dohvat, DELETE za brisanje, a za stvaranje i ažuriranje koriste se POST i PUT. [19]

Slika 5 : HTTP metode za RESTful aplikacije



Preuzeto iz [5]

GET – Metoda koja zahtjeva prikaz određenog resursa. Zahtjevi koji koriste GET trebali bi samo dohvatiti podatke, bez bilo kakvih drugih učinaka.

[18] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol (6. 9. 2016)
[19] http://www.restapitutorial.com/lessons/httpmethods.html (14. 2. 201)

PUT – Metoda traži da se entitet u prilogu skladišti samo u pruženom URI-u. Ako se URI odnosi na već postojeći resurs onda je on modificiran, a ako se ne odnosi na postojeći resurs onda poslužitelj može kreirati novi resurs s tim URI.

POST – Metoda zahtjeva da poslužitelj prihvati entitet zatvoren u zahtjevu kao novi podređeni entitet web resursa identificiran od strane URI. Podaci koji su POSTed mogu biti poruke za oglasnu ploču, označavanje postojećih resursa, interesne grupe…

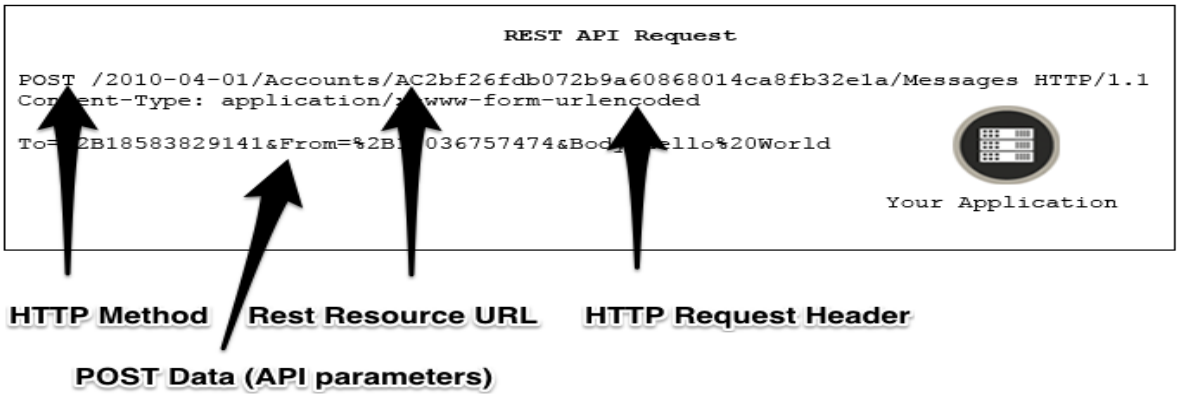
DELETE – Metoda briše specificirani resurs.

Tablica 4 : HTTP metode za CRUD

Create	HTTP Put
Read	HTTP Get
Update	HTTP Post
Delete	HTTP Delete

Preuzeto iz [4]

Slika 6 : REST zahtjev preko HTTP metode



Preuzeto iz [6]

14. REST primijenjen na HTTP

Hyper Text Transfer Protocol (HTTP) ima posebnu ulogu u web arhitekturi kao primarni protokol na razini aplikacija za komunikaciju između web komponenta i kao jedini protokol dizajniran za prenošenje reprezentacije resursa. Za razliku od URI bio je potreban veliki broj promjena kako bi HTTP podržavao modernu arhitekturu. [20]

REST je korišten kako bi identificirao probleme unutar postojećih HTTP implementacija. On specificira interoperabilni podskup tog protokola kao HTTP 1.0, analizira predložena proširenja i nadogradnje za HTTP 1.1 i osigurava motivirajuće argumente za uvođenje HTTP 1.1. Ključni problemi koji su identificirani uz pomoć REST-a uključuju planiranje uvođenja nove verzije protokola, odvajanje poruka analiziranjem iz HTTP semantike i temeljnog transportnog sloja, razlikovanje logičkih i nelogičkih odgovora, kontrolu skladištenja sadržaja u priručnu memoriju i razne aspekte protokola koji nisu samo opisni. REST je također korišten za modeliranje performanse web aplikacija baziranih na HTTP-u i predviđanje utjecaja tih nadogradnja. Uz to REST je korišten kako bi ograničio opseg standardiziranih HTTP proširenja na ona koja se uklapaju u model arhitekture kako se ne bi dopustilo da aplikacije koje zloupotrebljavaju HTTP imaju jednak utjecaj na standard.

Jedan od glavnih ciljeva REST-a je podržati postupno i fragmentirano uvođenje promjena unutar već uvedene arhitekture. HTTP je modificiran kako bi podržao te ciljeve. HTTP1.1 jest poboljšani protokol protokola HTTP1.0 i uveden je kako bi podržao REST arhitekturu i RESTful APIs. Temeljne razlike između dva protokola nalaze se u područjima proširenja, prenošenja poruka, očuvanja internet adrese, notifikacija o pogreški, pregovora o sadržaju, sigurnosti, integriteta, autentičnosti itd.

14.1 Nepodudaranja između REST-a i HTTP-a [21]

Postoji nekoliko nepodudarnosti između REST-a i HTTP-a, neka zbog nadograđivanja od treće strane koja su uvedena van standardnog procesa a druga zbog nužnosti da HTTP ostane kompatibilan sa komponentama HTTP 1.0.

Razlikovanje ne vjerodostojnih odgovora (Differentiating Non-authoratative responses)

Jedna slabost koja još uvijek postoji unutar HTTP-a je ta da ne postoji konzistentan mehanizam za razlikovanje vjerodostojnih odgovora koji su generirani od strane izvornog poslužitelja kao odgovor na trenutan zahtjev, i ne vjerodostojnih odgovora koji su dobiveni od posrednika ili priručne memorije bez pristupa izvornom poslužitelju. Razlika može biti važna za aplikacije koje traže vjerodostojne odgovore vezane uz kritične sigurnosne informacije i kada klijent nije siguran dali je do pogreške došlo zbog poslužitelja ili podrijetla.

[20] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 116.
[21] Ibid, str. 129-134.

Kolačići (Cookies)

Primjer gdje je neprimjereno proširenje napravljeno protokolu kako bi podržalo svojstva koja su kontradiktorna željenim svojstvima generičkog sučelja jest predstavljanje i uvođenje u cijeli web informacije u obliku HTTP kolačića. Interakcija kolačića ne odgovara REST modelu pohranjivanja podataka aplikacije što često dovodi do zabune kod preglednika. Kolačići se također ne slažu sa REST-om zato što dopuštaju da se podaci prenose bez dovoljne identifikacije njihove semantike, što dovodi do zabrinutosti za sigurnost i privatnost, jer oni omogućuju praćenje korisnika.

Obavezna proširenja (Mandatory expansion)

Imena zaglavnih polja HTTP-a mogu se proširivati po volji ali samo kada informacije koje sadržavaju nisu potrebne za točno razumijevanje poruke. Obavezna proširenja zaglavnih polja traže veću reviziju protokola ili veću promjenu metoda semantike. To je aspekt moderne web arhitekture koji se još ne podudara sa ograničenjima na samo opisnim porukama REST stila arhitekture. Razlog tome je prvenstveno taj da bi cijena obaveznog proširenja unutar postojeće HTTP sintakse nadmašila druge beneficije.

Miješanje metapodataka (Mixing metadata)

HTTP je dizajniran da proširi generički priključak sučelja preko cijele mrežne veze. Kao takav namjena mu je da odgovara karakteristikama sučelja uključujući opisivanje parametara kao kontrolnih podataka, metapodataka i reprezentacija. Dva najznačajnija ograničenja protokola HTTP 1. x su ta da ne mogu sintaktički razlikovati reprezentaciju metapodataka i kontrolu informacija u porukama, te ne dopušta da metapodaci budu efektivno posloženi za provjeravanje integriteta poruke. REST je ta ograničenja u protokolu identificirao rano u standardizacijskom procesu očekujući da će dovesti do problema u uvođenju novih značajka poput konstantne veze i sažete provjere autentičnosti.

MIME sintaksa (Mime sintax)

HTTP je naslijedio svoju sintaksu poruka od MIME-a kako bi zadržao povezanost s drugim internetskim protokolima te ponovno iskoristio mnogo standardiziranih polja za opisivanje medijskih tipova u porukama. Nažalost, MIME i HTTP imaju vrlo drugačije ciljeve a sintaksa je dizajnirana za ciljeve MIME-a. U MIME-u korisnik šalje nepoznatom primatelju sa kojim nije u interakciji mnoštvo informacija koje se tretiraju kao cjelina. MIME pretpostavlja da će korisnik slati sve informacije u jednoj poruci pošto je slanje nekoliko poruka preko interneta manje efektivno. MIME sintaksa je konstruirana kako bi „pakirala“ poruke. U HTTP-u „pakiranje“ različitih poruka nema smisla jer je efikasnije napraviti nove zahtjeve za one dokumente koji još nisu pohranjeni.

15. REST primijenjen na URI

URI (Uniform Resource Identifiers) su istovremeno najjednostavniji ali i najbitniji elementi web arhitekture. URI su poznati pod mnogo imena; www adrese, Universal Document Identifiers, te kao kombinacija Uniform Resource Locators (URL) i Uniform Resource Names (URN). Sintaksa URI-a ostala je relativno nepromijenjena još od 1992 godine no to nije bio slučaj sa semantikom i opsegom resursa. REST je korišten kako bi definirao izraz resurs za URI standard, kao i cjelokupnu semantiku generičkog sučelja za manipulaciju resursa pomoću njihovih reprezentacija.[22]

15.1 Redefiniranje resursa

Rana web arhitektura definirala je URI kao identifikator dokumenata. Autori su definirali identifikatore sukladno lokaciji dokumenata na mreži. Tada su se web protokoli mogli koristiti kako bi povratili dokument. Ta definicija pokazala se nezadovoljavajućom iz mnogo razloga. Definicija resursa u REST-u temelji se na jednostavnoj premisi. Identifikatori bi se trebali mijenjati što rjeđe. Kako web koristi ugrađene identifikatore autori trebaju identifikatore koji se poklapaju sa traženim semantikama reference hipermedija, što dopušta referenci da ostane statična iako se rezultat pristupanja referenci može mijenjati sa vremenom. REST je to postigao definirajući resurs da bude semantika onoga što autor namjerava identificirati, umjesto vrijednost koja odgovara semantici u vrijeme kada je referenca kreirana.[23]

Definiranje resursa na način da REST identificira koncept umjesto dokument nameće mnoga pitanja. Kako će korisnik pristupiti, manipulirati ili prenijeti koncept na način da dobiju nešto korisno kada je hipertekst veza odabrana. REST ogovara na to tako što definira stvari koje su manipulirane da budu reprezentacije identificiranih resursa, a ne sam resurs. Izvorni poslužitelj održava mapiranje iz identifikatora resursa do seta reprezentacija koje odgovaraju svakom resursu. Resurs je manipuliran prijenosom reprezentacije kroz generičko sučelje definirano identifikatorom resursa.

REST definicija resursa proizlazi iz središnjih zahtjeva weba; samostalno autorstvo međusobno povezanih hipertekstova preko više povjerljivih domena. Prisiljavajući definicije sučelja da se poklapaju sa zahtjevima sučelja čini protokol nejasnim, ali to je samo zato što je manipulirano samo sučelje a ne implementacija. Protokoli su specifični u namjeri akcije aplikacije ali mehanizam iza sučelja mora odlučiti kako će ta namjera utjecati na temeljnu implementaciju mapiranja resursa u reprezentacije. Skrivanje informacija je ključan princip softverskog inženjeringa koje motivira jedinstveno sučelje REST-a. Kako je klijent ograničen na manipulaciju reprezentacije umjesto da ima direktan pristup implementaciji resursa, implementacija može biti sagrađena u bilo kojem obliku po želji autoriteta imenovanja bez utjecaja na klijenta koji mogu koristiti svoje reprezentacije.

[22] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 110.
[23] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 111.

15.2 REST nepodudarnosti s URI

Komponente korištene web arhitekture ne podržavaju svako ograničenje koje postoji u dizajnu arhitekture. REST je korišten kao sredstvo za definiranje poboljšanja arhitekture i kako bi identificirao nepodudaranja arhitekture. Iako se dizajn URI-a slaže sa REST pojmom arhitekture, identifikatora sama sintaksa nije dovoljna da natjera autoritete imenovanja da definiraju svoj URI prema modelu resursa.[24]

Jedan oblik zlostavljanja jest uključivanje informacija koje identificiraju trenutačnog korisnika unutar svih URI-a referiranih od strane reprezentacije odgovora hipermedija. Ugrađeni korisnički identifikatori mogu se koristiti za održavanje stanja sesije na poslužitelju, praćenje korisničkog ponašanja, evidentiranje njihovih akcija. No kršenjem REST ograničenja sistemi također uzrokuju da zajednički spremljeni podaci postanu neučinkoviti, smanjuju skalabilnost poslužitelja, te rezultiraju nepoželjnim efektima kada korisnik dijeli te reference sa drugima.

Drugi konflikt sa REST sučeljem resursa događa se kada softver pokuša tretirati web kao distribuirani sistem datoteka. Kako sistem datoteka otkriva implementaciju informacija postoje alati koji prikazuju te iste informacije preko više stranica kao sredstvo za balansiranje opterećenja i raspodjelu sadržaja bliže korisniku. No, to mogu učiniti samo zato jer datoteke imaju imenovane sekvence bajtova koji se lako mogu duplicirati. Sukladno tome, pokušaj prikazivanja sadržaja web poslužitelja kao dokumenta ne može uspjeti jer se sučelje resursa ne poklapa uvijek sa semantikom sistema datoteka, i zato što su podaci i metapodaci uključeni unutar i važni semantikama reprezentacija. Sadržaj web poslužitelja može se zamijeniti na udaljenim stranicama ali samo ako je dupliciran cijeli mehanizam i konfiguracija poslužitelja, ili ako se selektivno dupliciraju samo oni resursi koji imaju statične reprezentacije.

[24] Roy Thomas Fielding PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine 2000, str. 115.

16. Kršenje REST ograničenja

Od nastanka REST-a sve više web usluga i aplikacija nazivaju se RESTful iako to vrlo često nisu. Kršenje bilo kojeg od REST ograničenja onemogućuje usluzi ili APIs da budu RESTful. Najčešća zabuna dolazi iz razloga što REST stil arhitekture još uvijek nije u potpunosti shvaćen, te se kao RESTful API smatra svako HTTP bazirano sučelje. Drugi razlog zašto većina APIs nisu RESTful je taj što je hipertekst ograničenje, što znači da se mehanizmom aplikacijskog stanja upravlja sa hipertekstom. Svako ograničenje specificirano u REST-u sadrži svoja pravila kojih se treba pridržavati ako se želi stvoriti RESTful API.

16.1 REST API specifikacije [25]

REST APIs ne bi trebala biti ovisna ni o kojem komunikacijskom protokolu, iako bi njihovo uspješno mapiranje na određeni protokol moglo ovisiti o dostupnosti metapodataka, izboru metoda… Općenito, svaki element protokola koji koristi URI kao identifikator mora dopustiti bilo kojoj URI shemi da se koristi u svrhu te identifikacije.

REST API ne bi trebalo sadržavati bilo kakve promjene komunikacijskog protokola osim popunjavanja ili popravljanja detalja nedovoljne specifikacije bitova standardnih protokola poput PATCH metode HTTP-a.

REST API bi trebalo utrošiti gotovo sve svoje deskriptivne trudove na definiranje tipa medija korištenog za reprezentaciju resursa ili na definiranje produženja imena veza. Vrijeme utrošeno na opisivanje koje metode koristiti na kojem URI-u trebalo bi biti u potpunosti definirano unutar opsega prava obrade vrste medija.

REST API ne smije definirati popravljena imena resursa ili hijerarhije. Poslužitelji moraju imati slobodu kontroliranja njihovog prostora za ime. Umjesto toga treba se dopustiti poslužitelju da daje instrukcije klijentu o tome kako konstruirati prikladan URI, kako je to i napravljeno u HTML formama i URI predlošcima, definiranjem tih instrukcija unutar vrste medija i veze linkova.

REST API ne bi nikada smio imati tip resursa koji je značajan za klijenta. Autori specifikacija mogu koristiti tipove resursa za opisivanje implementacije poslužitelja iza sučelja ali ti tipovi moraju biti nevidljivi i neznačajni za klijenta. Jedini tipovi važni za klijenta su trenutna medijska vrsta reprezentacije i standardizirana imena veza.

U REST API bi se trebalo ulaziti bez prethodnog znanja van inicijalnog URI-a i seta standardiziranih tipova medija koji su prikladni namijenjenoj publici. Od te točke nadalje svo aplikacijsko stanje prijelaza mora biti vođeno izborom klijenta između mogućnosti koje pruža poslužitelj a koje su prisutne u primljenim reprezentacijama, ili implicirane korisničkom manipulacijom tih implementacija. Prijelaz može biti utvrđen klijentovim poznavanjem vrste medija i mehanizma komunikacije resursa, od čega oboje može biti poboljšano tijekom rada aplikacijskog programskog sučelja.

[25] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (20. 10. 2008)

17. Zaključak

Još od 1994 godine REST stil arhitekture korišten je kako bi vodio dizajn i razvoj arhitekture za moderan web. Prva verzija REST-a razvijena je prvenstveno kao sredstvo komunikacije web koncepta. Nakon toga REST se neprestano razvijao i primjenjivao u mnogobrojnim revizijama i proširenjima standarda web protokola. Namjena REST-a nije ta da obuhvati sve moguće načine korištenja standarda web protokola, već je važno da REST obuhvaća sve aspekte distribuiranog hipermedijskog sustava koji se smatraju najbitnijim za ponašanje i performansu zahtjeva web-a na način da će optimizacija ponašanja unutar modela rezultirati optimalnim ponašanjem unutar korištene web arhitekture.

REST je postao poznat širokoj javnosti te učestalo primjenjivan nakon što ga je Roy Fielding predstavio u svojoj doktorskoj disertaciji. Unutar nje REST je opisan kao hibrid prošlih stilova arhitekture koji sadrži ograničenja kojih bi se trebalo pridržavati. Prvo ograničenje naziva se jedinstveno sučelje koje definira sučelje između klijenta i poslužitelja. Ono pojednostavnjuje i razvija arhitekturu te omogućava klijentu i poslužitelju da se razvijaju neovisno. Četiri vodeća principa jedinstvenog sučelja su; manipulacija resursa kroz reprezentaciju, temeljenost na resursima, samo opisne poruke i hipermedija kao mehanizam aplikacijskog stanja. Sljedeće ograničenje je neovisnost između zahtjeva. To ograničenje nam govori da poslužitelj ne bi smio pratiti stanje aplikacije što znači da bi klijent u svojoj poruci trebao poslati sve potrebne informacije za izvedbu zato jer poslužitelj ne može razumjeti poruku uz pomoć konteksta prošle poruke. U ograničenju priručne memorije podaci unutar odgovora na zahtjev označuju se kao podaci koji se spremaju u priručnu memoriju ili kao podaci koji se ne spremaju. Ako je odgovor spremljen u priručnu memoriju onda ga klijent može koristiti za daljnje ekvivalentne zahtjeve. Ograničenje slojeviti sustav postoji zato jer postoji mnogo slojeva između klijenta i poslužitelja. Oni se nazivaju posrednicima te se mogu dodati putu zahtjev – odgovor bez da izazovu promjenu sučelja između komponenata. Zadnje ograničenje jest kod na zahtjev. Ono je opcionalno te omogućava klijentu da skine i izvrši kod od poslužitelja. Sve aplikacije koje su nastale pridržavajući se tih ograničenja nazivaju se RESTful a takve aplikacije trebale bi biti jednostavne i skalabilne.

Iako se danas mnoge aplikacije nazivaju RESTful one to zapravo i nisu zato jer krše neka od REST-ovih ograničenja Razlog tome je što je REST kao stil arhitekture unatoč svojoj primijenjenosti još uvijek vrlo neshvaćen te se RESTful aplikacijom nazivaju sve one koje imaju HTTP bazirano sučelje.

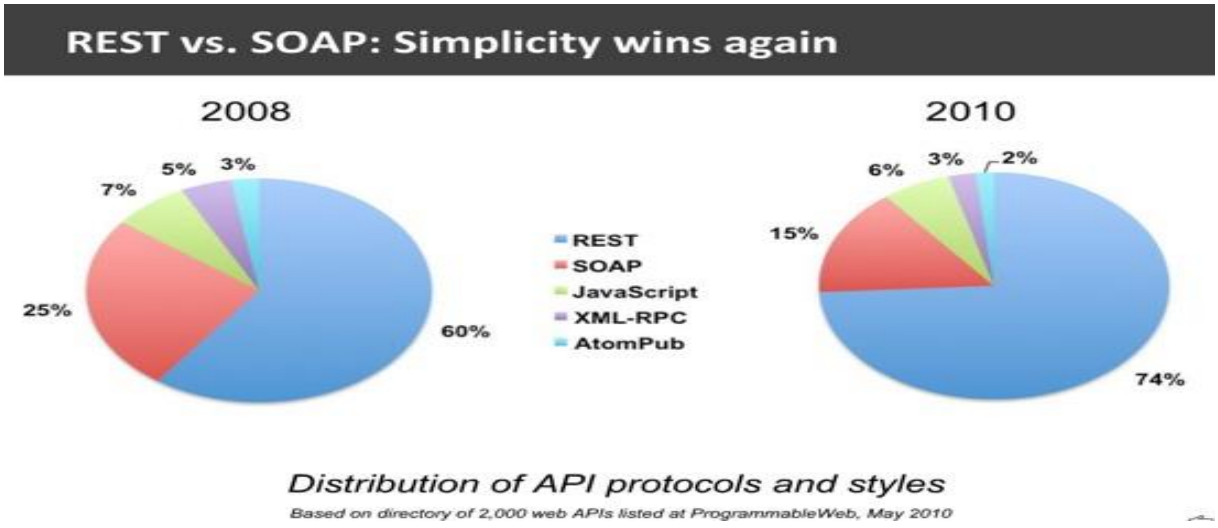
Unatoč tome što REST predstavlja suvremeni način dizajna web aplikacije te je time zamijenio stare protokole poput SOAP-a i PRC-a on još uvijek nije usavršen i primjeren za sve. SOAP je komunikacijski protokol neovisan o platformi, baziran na XML-u te se koristi za razmjenu informacija između aplikacija preko HTTP-a. REST se smatra mnogo jednostavnijim iz nekoliko razloga. On koristi HTTP standard. Dopušta korištenje različitih formata podataka dok SOAP koristi samo XML. Najčešće korišten format kod REST-a je JSON koji se smatra boljim formatom za podatke te ih brže obrađuje. REST ima bolju performansu i skalabilnost te za razliku od SOAP-a dijelovi poruke REST-a mogu biti

spremljeni u privremenu memoriju a SOAP-ovi ne. Unatoč tome SOAP se još uvijek smatra sigurnijim te se bira ovisno o potrebama određene aplikacije. REST nema direktnu podršku za generiranje klijenta iz generiranih metapodataka poslužitelja dok SOAP to podržava uz pomoć WDSL-a (Web Service Description Language).

REST je često korišten za mobilne aplikacije, socijalne mreže, alate za web aplikacijske hibride… Poznate RESTful aplikacije su Facebook, Twitter, Myspace, Photobucket iako se tvrdi da i one krše REST ograničenja.

REST će se i dalje nastavljati razvijati i poboljšavati. Prvi korak k tome je rad na nepodudarnostima između REST-a, HTTP-a i URI-a, te poboljšanje i eliminacija svih REST-ovih slabosti. REST je jedan od najpopularnijih stilova arhitekture te će biti korišten i dalje u sve boljim i naprednijim verzijama kako sve više poduzeća traži način za pružanje otvorenih i definiranih sučelja za aplikacije i infrastrukturne usluge.

Slika 7 : Primijenjenost REST stila arhitekture



Preuzeto iz [7]

Literatura

1. Roy Thoma Fielding (PhD Dissertation) –Architectural Styles and the Design of Network-based Software Architectures.

2.Wikipedia (https://en.wikipedia.org/wiki/Representational_state_transfer)- Representational state transfer (8,9,2016)

3. Wikipedia (https://hr.wikipedia.org/wiki/World_Wide_Web) – World Wide Web (10,9,2016)

3. Wikipedia (https://en.wikipedia.org/wiki/Application_programming_interface) – Application Programming Interface (28,9,2016)

4. Wikipedia (https://en.wikipedia.org/wiki/Web_service) – Web service (28,8,2016)

5. RESTful Web services: A tutorial Dr Dobbs (http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069) (23,9,2014)

6. Wikipedia (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) – HTTP (6,9,2016)

7. Roy Thomas Fielding (http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven) – REST APIa must be hypertext-driven (20,10,2008)

8. Using HTTP Methods for RESTful Services (http://www.restapitutorial.com/lessons/httpmethods.html) (14,2,201)

Prilog

Slike

1. Slika 1 - Definicija stila arhitekture
https://www.google.hr/search?q=fielding+architectural+style+definition&client=firefox-b&biw=797&bih=368&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjT36Xd6_rOAhWiE5oKHcssB3wQ_AUIBigB#imgrc=945Otp0xE082RM%3A

2. Slika 2 - Kod na zahtjev
Roy Thomas Fielding PhD dissertation „Architectural Styles and the Design of Network-based Software Architectures“

3. Slika 3 - Pogled na proces
Roy Thomas Fielding PhD dissertation „Architectural Styles and the Design of Network-based Software Architectures

4. Slika 4 - Klasifikacija API
https://www.google.hr/search?q=rest+connectors&client=firefox-b&biw=797&bih=368&source=lnms&tbm=isch&sa=X&sqi=2&ved=0ahUKEwiVypnX9vrOAhXiJ5oKHW5HCHwQ_AUIBigB#tbm=isch&q=a+simple+classification+of+api&imgrc=MX-Ptl6Yov9DrM%3A

5.Slika 5 - HTTP metode za RESTful
aplikacijehttps://www.google.hr/search?q=rest+connectors&client=firefox-b&biw=797&bih=368&source=lnms&tbm=isch&sa=X&sqi=2&ved=0ahUKEwiVypnX9vrOAhXiJ5oKHW5HCHwQ_AUIBigB#tbm=isch&q=restful+api+http+methods+collection+uri+such+as&imgrc=tmOpYn6JZ_4HdM%3A

6. Slika 6 - Rest zahtjev preko HTTP
metodehttps://www.google.hr/search?q=rest+request+response&client=firefox-b&biw=797&bih=368&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjW_dWe8_rOAhWLCCwKHV2jC9MQ_AUIBigB#imgrc=icqeRDSqNZgXvM%3A

7. Slika 7 - Primijenjenost REST stila
arhitekturehttps://www.google.hr/search?q=rest+vs+soap+simplicity&client=firefox-b&biw=797&bih=368&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjape609PrOAhVhSZoKHTTwDXoQ_AUIBigB#imgrc=Pchw_N3faYz4ZM%3A

Tablice

1.Tablica 1 – REST elementi podataka
Originalna izrada

2. Tablica 2 - REST komponente
Originalna izrada

3. Tablica 3 - REST konektori
Originalna izrada

4.Tablica 4 - HTTP metode za CRUD
Originalna izrada