Matematički fakultet Univerzitet u Beogradu

Master rad

Korisnički web servisi na platformi Android

Student: Jelena Nikolić, 1025/2011

Mentor: dr. Dušan D. Tošić

Beograd, 2015

Student:	Jelena Nikolić 1025/2011
Naslov	Korisnički web servisi na platformi Android
Mentor	dr. Dušan D. Tošić, redovni profesor Univerzitet u Beogradu – Matematički fakultet
Članovi komisije	dr. Nenad S. Mitić, vanredni profesor Univerzitet u Beogradu – Matematički fakultet
	dr. Filip Marić, docent Univerzitet u Beogradu – Matematički fakultet
Datum	Oktobar 2015.

Apstrakt

Kroz rad su predstavljeni istorija i razvoj web servisa. Navedene su osnovne karakteristike i vrste web servisa - svaka od njih je detaljno objašnjena. Akcenat je stavljen na REST-olike web servise obzirom na to da je ova vrsta web servisa trenutno najkorišćenija.

Većina web servisa za transport poruka koristi HTTP protokol (Hypertext Transfer Protocol). Najznačajniji primer REST-olikih web servisa je WWW (World Wide Web).

Web servisi su platformski i jezički transparentni, što znači da servis može biti napisan na jednom programskom jeziku, a da ga koristi klijent koji je napisan u nekom drugom programskom jeziku.

Kao primer upotrebe web servisa, u radu je predstavljen komunikacija web servisa napisanog na Java programskom jeziku, a klijent koji ga koristi je aplikacija za platformu Android. Aplikacija služi za praćenje rada autoservisa. Ime aplikacije, odnosno ime autoservisa kojem je aplikacija namenjena, jeste *Arizona*.

Abstract

Through this paper history and development of web services are presented. Characteristics and types of web services are listed, and each of them is explained in details. The accent is put on RESTful web services since this type of web services is the most used nowadays.

Almost all web services use the HTTP (Hypertext Transfer Protocol) as messaging transport protocol. The most important example of RESTful web services is WWW (World Wide Web).

Web services are platform and language transparent, which means that service can be implemented in one programming language, but it is used by the client written in other programming language.

As the example of web service usage, we presented the sample of web service written in Java programming language, but the client that consumes that service is the application written for Android platform. The application is used to track the work of the car service. The application name (in fact, the name of the car service where the application is implemented) is *Arizona*.

Sadržaj

UVOD

Većina aplikacija je razvijana da komunicira sa korisnicima; korisnik unosi ili zahteva podatke kroz korisnički interfejs, a aplikacija mu odgovara. Web servis radi manje-više isto, izuzev toga što web servis aplikacija komunicira samo od mašine do mašine ili od aplikacije do aplikacije. Često nema direktne komunikacije sa korisnikom. U suštini, web servis je kolekcija otvorenih protokola koji se koriste za razmenu podataka između aplikacija. Korišćenje otvorenih protokola omogućava web servisima da budu nezavisni od platforme. Dakle, softveri napisani u različitim programskim jezicima i koji rade na različitim platformama mogu da koriste web servise za razmenu podataka preko računarskih mreža kao što je Internet.

S druge strane, tehnologije mobilnih uređaja se neprestano razvijaju i usavršavaju, pa mobilni uređaji sve češće igraju ulogu prenosnih, džepnih, računara. Tome je dosta doprinelo korišćenje operativnih sistema koji su napredniji iz dana u dan. Operativni sistem Android je jedan od vodećih operativnih sistema na tržištu mobilnih uređaja. Za njegov razvoj i zastupljenost je najzaslužnija kompanija Google, u čijem je vlasništvu od 2005. godine.

U ovom radu će biti predstavljene vrste i karakteristike web servisa, kao i primer njihove primene na mobilnim uređajima koji koriste Android operativni sistem.

Web API-i korišćeni u ovom radu su razvijani u programskom jeziku Java, uz pomoć razvojnog alata Eclipse. Projekat se zasniva na „Spring Boot“ alatu, koji uključuje konfiguraciju i prateće biblioteke potrebne za razvoj REST web servisa u Spring okruženju (eng. Spring Framework). Za čuvanje podataka je korišćena MySQL relaciona baza podataka.

Android aplikacija koja koristi API-e je razvijana u okviru razvojnog alata Android Studio. Aplikacija treba da automehaničaru omogući laki pristup podacima o svojim mušterijama, odnosno autoservisima koji su rađeni u njegovoj radnji.

Motivacija za ovakvu aplikaciju je ta što je kancelarija sa računarom obično odvojena od same radionice, a mehaničaru može da bude veoma korisno da na licu mesta popuni podatke o novom servisu i vlasniku automobila. Interfejs aplikacije je jednostavan i lak za korišćenje, a svi podaci se čuvaju u bazi.

Istorija i razvoj web servisa

Prosta definicija glasi: “Web servis je aplikaciona komponenta dostupna preko otvorenih protokola”. (videti [14])

W3C (World Wide Web Consortium) je internacionalna organizacija koja ima za cilj da dizajnira i razvija Web standarde, uputstva i protokole koji podržavaju WWW (World Wide Web) kao sigurni alat za komunikaciju korišćen za pružanje servisa i kreiranje sadržaja. (videti [14])

W3C definiše web servis kao „softverski sistem dizajniran da podržava interoperabilnu interakciju između mašina preko interneta“.

Web servisi su evoluirali iz RPC (Remote Procedure Call) mehanizma u DCE (Distributed Computing Environment) - okruženje za softverski razvoj, u ranim 90-im godinama prošlog veka. DCE uključuje distribuirani fajl-sistem zasnovan na Kerberu. Iako je DCE nastao u UNIX sistemu, Microsoft je brzo napravio svoju implementaciju poznatu kao MSRPC, koja je služila kao infrastruktura za međuprocesnu komunikaciju na Windows-u.

Prva generacija okruženja (eng. framework) za distribuirane objektne sisteme, CORBA (Common Object Request Broker Architecture) i MSDCOM (Distributed COM), zasnovana je na DCE/RPC proceduralnom okruženju. Java RMI (Remote Method Invocation) takođe nastaje iz DCE/RPC, a pozivi metoda u JavaEE (Enterprise Edition), naročito u Session i EntityEJBs (Enterprise Java Bean), jesu Java RMI pozivi. Java EE i MS DotNet su druga generacija okruženja za distribuirane objektne sisteme.

Kasnih 1990-ih, Dave Winer iz Userland Software-a razvio je XML-RPC, inovativnu tehnologiju, toliko dobru da se označava kao rođenje web servisa. XML-RPC je veoma jednostavan RPC sistem sa podrškom za osnovne tipove podataka (u osnovi – C tipovi sa boolean i datetime tipovima) i sa nekoliko komandi. Dve ključne funkcionalnosti su korišćenje XML marshaling/unmarshaling-a, kako bi se postigla jezička neutralnost i oslanjanje na HTTP (a kasnije i na SMTP) za transport. Izraz „marshaling“ odnosi se na konverziju nekog objekta iz memorije (npr. objekat u Javi) u neki drugi format (npr. XML dokument); „unmarshaling“ je suprotan proces.

Dve ključne razlike između XML-RPC-a i DCE/RPC-a, kao i njegovih ogranaka su:

1) XML-RPC poruke (eng. payloads) predstavljaju tekst dok su poruke DCE/RPC-a binarne. Tekst je lako procesirati u alatima kao što su editori i parseri.
2) XML-RPC transport koristi HTTP. Da bi podržao XML-RPC, programski jezik zahteva jedino standardnu HTTP biblioteku zajedno sa bibliotekama za procesiranje XML-a.

XML-RPC podržava zahtev/odgovor (eng. request/response) obrazac. SOAP je direktno nastao od XML-RPC-a, ali je dosta teži za rad. (videti [1])

Web servisi pojednostavljuju stvari u distribuiranom programiranju. Procesiranje na klijentskoj, kao i na serverskoj strani, zahteva samo lokalno dostupne biblioteke. Zbog toga, kompleksnosti se mogu izolovati u krajnje tačke (eng. endpoints) – servis i klijentska aplikacija sa bibliotekama podrške – bez curenja razmenjenih poruka. Na kraju, web servisi su dostupni preko HTTP-a, javnog protokola koji je postao standard, sveprisutne infrastrukture. Servis i klijent ne moraju biti implementirani u istom jeziku, čak ni u istim stilovima jezika. Mogu biti implementirani u objektno-orijentisanim, proceduralnim, funkcionalnim i drugim stilovima. (videti [1])

Karakteristike web servisa

Objavljivanje web servisa se vrši preko web servera.

Tehnički: web servis je distribuirani softverski sistem čije se komponente mogu razvijati i izvršavati na fizički različitim uređajima. (videti [3])

Servis može imati dosta klijenata koji mu se obraćaju, a i sam može biti sačinjen od drugih servisa.

Jedan HTTP zahtev (eng. request) dolazi, po definiciji, od klijenta ka serveru, dok HTTP odgovor (eng. response) dolazi, opet po definiciji, od servera ka klijentu. Za web servise koji kao transport koriste HTTP, HTTP poruke su infrastruktura i one se mogu kombinovati u osnovne konverzacijske obrasce (eng. patterns) koji karakterišu web servise. Obrasci su:

1) Request/response – konverzacija počinje HTTP zahtevom i očekuje HTTP odgovor
2) Solicit/response – konverzacija počinje HTTP porukom sa servera i očekuje poruku od klijenta

Bogatiji konverzacijski obrasci se mogu dobiti kombinacijama ovih obrazaca. I oni sami su sačinjeni od primitivnih obrazaca:

1) One-way – poruka od klijenta ka serveru, bez odgovora
2) Notification – poruka od servera ka klijentu, bez klijentskog odgovora.

Web servisi nastoje da imaju jednostavnu strukturu. Prethodna četiri obrasca pokrivaju većinu modernih web servisa, a dominantan među njima je request/response obrazac.

Glavna karakteristika web servisa je jezička transparentnost – servis i klijenti ne moraju biti napisani u istom jeziku. Međutim, ako kllijenti nisu napisani u istom jeziku, mora postojati središnji sloj koji će rukovati razlikama u tipovima između servisnog i klijenstkog jezika. Taj sloj može biti XML (eXtensible Mark-up Language) ili JSON (JavaScript Object Notation). JSON je tekstualna reprezentacija nativnog JavaScript objekta. On se više koristi jer su web servis klijenti uglavnom JavaScript programi ugrađeni u HTML dokumente i tako se izvršavaju u pretraživaču

(eng. browser). Takođe, JSON je čitljiviji od XML-a jer ima znatno manje oznaka. Kod SOAP web servisa je dominantan format XML, iako podjednako dobro rade i sa JSON-om.

Web servisi imaju nekoliko odlika zbog kojih se razlikuju od drugih distribuiranih softverskih sistema:

- Otvorena infrastruktura (eng. Open infrastructure): Web servisi su razvijani pomoću standardizovanih (i nezavisnih od proizvođača) protokola i jezika kao što su HTTP, XML i JSON, a svi ovi su sveprisutni i lepo razumljivi. Web servisi se mogu osloniti na umrežavanje, formatiranje podataka, bezbednost i druge infrastrukture koje već postoje, što smanjuje troškove razvoja i povećava interoperabilnost među servisima.
- Platformska i jezička transparentost (eng. Platform and language transparency): Servisi i njihovi klijenti mogu da komuniciraju iako su napisani na različitim jezicima. Web servisi mogu biti objavljeni i konzumirani na različitim hardverskim platformama i različitim operativnim sistemima.
- Modularni dizajn (eng. Modular design): Web servisi su zamišljeni da budu modularni tako da novi servisi mogu da se nadograde na stare. Vodeći princip u dizajnu web servisa je: početi sa veoma jednostavnim servisnim operacijama, onda grupisati te operacije u servise, koji opet mogu biti organizovani da rade sa drugim servisima i tako u nedogled. (videti [1])

Mehanika razmena poruka je dokumentovana u WSD-u, koji je napisan na WSDL (Web Service Definition Language). WSD (Web Service Definition) definiše formate poruka, tipove podataka, protokole transporta i transportnu serijalizaciju formata koja bi trebalo da se koristi između klijenta i servera. Takokđe, WSD specificira jednu ili više internet lokacija preko kojih se server može kontaktirati, i može opisivati neke informacije o očekivanom obrascu razmene poruka. U osnovi, WSD predstavlja dogovor o vodećoj mehanici interakcije sa određenim servisom. (videti [4])

Semantika web servisa je očekivanje o ponašanju servisa, posebno o odgovoru na poruke koje su mu poslate. U praksi, ovo je „ugovor“ između klijenta i servera o svrsi i posledicama njihove interakcije. Ne mora biti napisana ili eksplicitno ugovorena. Može biti eksplicitna ili implicitna, usmena ili pismena, mašinski ili čovek-čitljiva, formalna (eng. legal agreement) ili neformalna.

Dok opis servisa predstavlja ugovor o mehanici interakcije sa određenim servisom, semantika predstavlja ugovor o značenju i svrsi te interakcije. Granica između ova dva ne mora da bude striktna. Kako se semantički bolji jezici sve više koriste za opis mehanike, više esencijalnih informacija može migrirati iz neformalne semantike u opis servisa. Kako se dešava migracija, više posla potrebnog za postizanje uspeha može biti automatizovano. (videti [4])

Pouzdanost web servisa (eng. web services reliability): sretanje sa greškama je neizbežno, naročito u kontekstu servisa globalne mreže koji pripadaju različitim ljudima. Kako greške ne

možemo eliminisati, cilj je da smanjimo njihovu učestalost u interakciji i, kada se dese, da obezbedimo što veću količinu informacija o neuspešnim, ali i uspešnim korišćenjima servisa.

Fokus na pouzdanosti se zapravo ne odnosi na probleme kao što su sintaksne greške, ili čak loše napisane aplikacije. Obiman je spisak uzroka koji mogu da dovedu do greške; neki od njih su prekid internet konekcije, gašenje servera usred transakcije ili pogrešan unos neke informacije u neki od opisnih dokumenata.

U kontekstu web servisa, probleme u pouzdanosti možemo podeliti na nekoliko nivoa:

- Pouzdana i predvidljiva isporuka infrastrukture servisa, kao što je transport poruka ili otkrivanje servisa;
- Pouzdane i predvidljive interakcije između servisa;
- Pouzdano i predvidljivo ponašanje individualnih klijenata i servis provajdera – servera.

Sama arhitektura web servisa nema podršku za pouzdani transport poruka ili za izveštavanje o neuspehu. Ipak, ima uputstva kako se ovo može postići.

Pouzdanost poruka se najčešće postiže preko AI (acknowledgment infrastructure), što je zapravo set pravila koja definišu kako učesnici – klijent i server – treba da komuniciraju međusobno, proveravajući format i validnost poruke. (videti [4])

Vrste web servisa

Postoje dva popularna tipa web servisa – SOAP i REST-oliki. SOAP je XML dijalekt sa gramatikom koja specificira strukturu koju dokument mora da ima da bi se računao kao SOAP. U tipičnom SOAP servisu, klijent šalje SOAP poruke servisu i servis mu takođe odgovara SOAP porukama. REST-olike servise je teško opisati u rečenici ili dve. Za sada, REST-oliki servis je onaj koji HTTP ne tretira samo kao protokol, već i kao set smernica za dizajniranje servisnih zahteva i servisnih odgovora. Kod REST-olikih servisa, HTTP se i sam može posmatrati kao API.

Istorijski gledano, REST-oliki pristup web servisima se može posmatrati kao reakcija na kompleksnost SOAP web servisa. (videti [1])

Arhitektura web servisa

Mogu se identifikovati tri opšte arhitekture web servisa: „RESTful resource-oriented“, RPC stil i hibridni REST-RPC.

REST-olike, „resource-oriented“ arhitekture

Tehnički, dosta arhitektura je REST-oliko. Međutim, kada se govori o REST-olikim web servisima, obično se misli na one koji liče na WWW. Ova vrsta arhitekture se obično naziva ROA (Resource Oriented Architecture).

Kod REST-olikih arhitektura, informacija o metodu dolazi kroz HTTP metod. Kod ROA ova informacija dolazi kroz URI.

Dobar primer ROA web servisa je Amazonov servis „Simple Storage Service“, popularni S3 (https://aws.amazon.com/s3).

RPC arhitekture

RPC web servis od klijenta prihvata „koverat“ pun dokumenata, a sličan vraća nazad. Informacije o metodu i obimu podataka su unutar samog koverta. HTTP je popularan format tog koverta, pošto svaki web servis vredan pomena mora da koristi HTTP na neki način. Drugi popularan format koverta je SOAP (prenos SOAP dokumenata preko HTTP-a stavlja SOAP koverat unutar HTTP koverta).

Svaki RPC web servis definiše svoj rečnik. I za računarske programe važi: svaki put kad pišemo program, mi definišemo funkcije sa različitim imenima. Suprotno ovome, REST-oliki web servisi dele standardni rečnik HTTP metoda.

Primeri RPC web servisa su:

- Svi servisi koji koriste XML-RPC protokol (koji je skoro pao u zaborav)
- Skoro svi SOAP servisi.

REST-RPC hibridna arhitektura

Ovo je arhitektura web servisa koji se uklapaju negde između REST-olikih i pravih RPC servisa. Ove servise često kreiraju programeri koji znaju dosta o „real-world“ web aplikacijama, ali ne tako mnogo o teoriji REST-a. Takvi su servisi koji koriste HTTP GET metod i kada žele da menjaju podatke, to jest, REST-oliki su kada zahtevaju neke podatke, a RPC kada te podatke menjaju. (videti [5])

Tehnologije

Nezavisno od arhitekture, tehnologije koje web servisi koriste su: HTTP, URI, XML-RPC, SOAP, WS-*, WSDL, WADL.

XML (eXtensible Mark-up Language) je jezik koji nam omogućava da identifikujemo i organizujemo informacije na ispravan i fleksibilan način. „Extensible“ (proširiv) je zato što nema fiksni format. To znači da se pomoću XML-a može definisati bilo koji element i može se nazvati kako god. Za razliku od XML-a, HTML je SML (Single Mark-up Language) i ima predefinisane elemente i atribute. (videti [15])

XML-RPC je format strukture podataka za reprezentaciju poziva funkcija i njihovih povratnih vrednosti. Kao što ime kazuje, ovaj format je posebno dizajniran da koristi RPC stil.

WS-* su standardi koji se koriste pri SOAP-u, analogni su HTTP zaglavljima.

WSDL se koristi da opiše SOAP web servise. Klijent može da učita WSDL fajl i da na osnovu njega tačno zna koje RPC metode može da pozove, koje argumente ti metodi očekuju i koje vrednosti vraćaju. Skoro svaki SOAP servis ima svoj WSDL fajl. One koji ga nemaju je jako teško koristiti. (videti [5]) Trenutna verzija je WSDL 2.0, gde je „Description“ zamenjen sa „Definition“, pa WSDL predstavlja Web Services Definition Language. (videti [8])

WADL (Web Application Description Language) je XML rečnik koji opisuje REST-olike web servise. Ima istu funkciju kao WSDL, s tim što je WADL REST-olikim web servisima skoro nepotreban (zbog njihovih jednostavnijih interfejsa).

Svaki URI predstavlja tačno jedan resurs. Da nije tako, ne bi bio Univerzalni Identifikator Resursa (eng. Universal Resource Identifier/ Uniform Resource Identifier). (videti [5])

HTTP protokol

Prva verzija HTTP-a, dokumentovana 1991. godine kao HTTP/0.9, bila je jednostavan „read-only“ protokol. Dozvoljavao je klijentima da pošalju, kao zahtev, znakovnu nisku (eng. string) koja se sastoji od reči GET koju prati razmak i adresa dokumenta (ono što danas zovemo URI), a server je mogao da odgovori vraćanjem tog dokumenta. HTTP/0.9 nije imao zaglavlja, a odgovor je podrazumevano bio u HTML formatu.

Do 1992. godine je dokumentovan „Basic HTTP“, koji je smatran potpunom specifikacijom. Ova verzija je imala nekoliko novih metoda kao što su HEAD, PUT, POST i DELETE. Drugi metodi, kao što su CHECKIN, CHECKOUT, LINK, UNLINK, SEARCH i drugi, ne postoje u današnjoj specifikaciji. Ovaj dokument je uključio i koncept status kodova, ranu formu „content-negotiation“-a i meta-informacije (ono što nam je danas poznato kao HTTP zaglavlja). Iako ovaj dokument iz 1992. nije

nikada zvanično odobren od strane IETF-a (Internet Engineering Task Force) i drugih, u narednih nekoliko godina klijenti i web serveri su uveliko počeli da koriste ove funkcionalnosti. Kao rezultat, 1996. je nastao HTTP 1.0, dokumentovan u RFC 1945. Nedugo nakon toga je kao RFC 2068 dokumentovan HTTP1.1. Ovaj dokument je ulepšao mnoge implementacione detalje HTTP-a, koje danas znamo kao takve. Još jedna dopuna dokumenta se desila 1999. godine i objavljena je kao RFC 2616. Ovo je verzija HTTP-a koja se i danas koristi.

Kroz celu svoju istoriju, HTTP je dizajniran da radi kao „client-initiated“ i bez pamćenja stanja (eng. stateless) protokol za transfer poruka između klijenta i servera kroz TCP/IP. (videti [13])

WWW (World Wide Web) je postao popularan jer ga obični ljudi mogu koristiti za veoma korisne stvari uz minimalnu obuku. Ali, pre svega, web je moćna platforma za distribuirano programiranje.

Web je zasnovan na tri tehnologije:

1) Konvencija imenovanja/kreiranja URL-a (eng. URL naming convention)
2) HTTP protokol
3) Format HTML dokumenata. (videti [7])

Web servis je vrsta aplikacije koja se isporučuje preko HTTP-a (HyperText Transport Protocol). HTTPS (HTTP Secure) dodaje na HTTP sigurnosni nivo, pa je i servis isporučen preko HTTPS-a takođe web servis. (videti [1])

Jedan web servis može da opslužuje dosta različitih URL-ova, a svaki URL omogućava pristup različitim podacima sa servera.

Svaki zahtev ima svoju HTTP sesiju. (videti [7])

HTTP je protokol aplikativnog sloja koji definiše operacije za transfer poruka između klijenata I servera. (videti [2])

HTTP standard (RFC 2616) definiše 8 metoda koje klijenti mogu da primene na resurse: GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS i TRACE.

„Hypermedia“ je generalni izraz za tehnike koje server koristi da bi objasnio klijentu šta sledeće može da uradi (npr. HTML linkovi i forme). (videti [7])

Svaki HTTP odgovor se može podeliti na 3 dela:

- Status kôd (eng. status code), poznat kao kôd odgovora; to je trocifreni broj koji nam govori kako je prošao zahtev. Status kôd je prva stvar koju klijent vidi i on postavlja principe obrade ostatka odgovora.
- Telo odgovora (eng. entity-body), ponekad se naziva samo telo; to je dokument pisan u nekom formatu za koji je očekivano da ga klijent razume.

- Zaglavlja odgovora (eng. response headers); to je set ključ-odgovor parova koji generalno opisuju HTTP odgovor. Zaglavlja se šalju između status koda i tela.Najvažnije HTTP zaglavlje je „Content-Type“, koje govori HTTP klijentu kako da razume telo odgovora. Kažemo da je vrednost ovog zaglavlja „media type“ tela. Najpoznatiji medija tipovi su text/html (za HTML format) i image/jpeg (za slike).

JSON je standard za reprezentaciju jednostavnih struktura podataka u običnom (eng. plain) tekstu.

Collection+JSON je standard za prikazivanje liste resursa koja može da se pretražuje. (videti [7])

Postoji, za sada, 41 HTTP status kôd. Neki od njih se uglavnom ne koriste, ali svi zajedno predstavljaju osnovni set semantike i definisani su u svim API standardima.

HTTP status kôd je trocifreni broj dodat HTTP odgovoru. Na najosnovnijem nivou, on javlja klijentu šta se desilo kada je server pokušao da obradi zahtev. HTTP specifikacija definiše pet familija status kodova na osnovu prve cifre:

- 1xx: Informativno (eng. Informational) – koriste se samo u pregovorima izmedju HTTP klijenta i servera
- 2xx: Uspešno (eng. Successful) – ako je server uspešno obradio zahtev
- 3xx: Redirekcija (eng. Redirection) – kada se zahtev koji je klijent poslao nije izvršio na serveru, ali ako klijent hoće da napravi blago različit zahtev, preko njega bi server trebalo da uradi šta klijent traži
- 4xx: Greška klijenta (eng. Client Error) – problem sa HTTP zahtevom; loše je formiran, loš URL ili bilo šta što server ne može da prihvati
- 5xx: Greška servera (eng. Server Error) – zahtev nije uspešno izvršen zbog greške na serverskoj strani. Ovde klijent ne može ništa da uradi, do da sačeka da se problem reši.

Najznačajniji status kodovi su 200 (OK), 301 (Moved Permanently), 400 (Bad Request), 500 (Internal Server Error). Odmah za njima su 404 (Not Found) i 409 (Conflict).

RFC 2616 definiše 47 HTTP zahtev/odgovor zaglavlja (eng. headers). Isto kao i kod status kodova, neki od njih su skoro beskorisni, ali i oni definišu osnovni set semantike od kog svaki API može da ima koristi.

Neka od najvažnijih zaglavlja su vezana za „Content Negotiation“ proces. Klijent može da koristi specijalna HTTP zaglavlja zahteva kako bi rekao serveru kakve reprezentacije želi. Ovaj proces se naziva „content negotiation“, a HTTP standard definiše 5 zaglavlja zahteva za njega. Ona se kolektivno zovu Accept-* zaglavlja. Najvažniji su Accept i Accept-Language.

Primer: `Accept: application/json`. Na ovaj način klijent saopštava serveru da razume samo JSON reprezentaciju.

Primer: `Accept-Language: en-us`. Na ovaj način klijent saopštava serveru na kom jeziku želi da budu reprezentacije.

Tekstualne reprezentacije, kao što su JSON i XML dokumenti, mogu da se kompresuju, kako bi se smanjila njihova originalna veličina. HTTP klijent može da traži od servera kompresovanu verziju, a onda da za svog korisnika to transparentno dekompresuje. Zaglavlje kojim klijent javlja serveru da želi kompresovanu verziju je `Accept-Encoding: gzip`. Pored gzip-a, IANA definiše još neke vrednosti, ali ovo je najbolji i najvažniji tip. Kada server šalje kompresovanu verziju, pored Content-Type zaglavlja koje bi i inače poslao, šalje `Content-Encoding: gzip`. Ova tehnika može znatno smanjiti protok, a time i ubrzati komunikaciju. (videti [7])

Bezbednost web servisa

Zbog obimnosti, razmatranje bezbednosti web servisa se obično deli na tri dela:

- Transportni nivo (eng. wire-level security): ovo je prvi nivo bezbednosti; sačinjen je od osnovnih protokola koji pokrivaju komunikaciju između web servisa i njihovih klijenata. Bezbednost na ovom nivou obezbeđuju tri servisa. Prvo, klijentu i servisu je na transportnom nivou potrebno uverenje da komuniciraju međusobno, a ne sa nekim „prevarantom“. Drugo, podaci koji se šalju s jedne strane na drugu, moraju biti dovoljno dobro enkriptovani tako da ih presretač ne može dekritpovati i tako pristupiti poverljivim informacijama. Treće, svakoj strani treba uverenje da je poruka koja je primljena ista kao poslata poruka.

- Autentifikacija korisnika i autorizacija (eng. user authentication and authirization): Web servisi obezbeđuju klijentima pristup određenim resursima. Ako je resurs zaključan (eng. secured), onda su klijentu potrebni odgovarajući akreditivi (eng. credentials) kako bi mogao da mu pristupi. Akreditivi su prezentovani i potvrđeni kroz proces koji obično ima dve faze. U prvoj fazi, klijent, odnosno korisnik, šalje informaciju kao što je korisničko ime zajedno sa akreditivom kao što je šifra. Ako akreditivi nisu prihvaćeni, pristup zahtevanim resursima je odbijen. Ova faza je poznata kao Korisnička autentifikacija. Druga, opciona, faza se sastoji od finih podešavanja i prava pristupa autentifikovanog korisnika. Ova faza je poznata kao autorizacija.

- Bezbednost web servisa (eng. Web Services security): bezbednost web servisa ili WSS je kolekcija protokola koji određuju koliko se različitih nivoa bezbednosti može primeniti u SOAP infrastrukturi pre nego kroz određeni transport, HTTPS, ili kroz određeni servisni kontejner, Tomcat. WSS bi trebalo da obezbedi jaku „end-to-end“ bezbednost bez obzira na transport i kontejner koji služi taj servis. (videti [1]) Izraz „end-to-end“ se koristi za kompletan transfer od klijenta do servisa ili obrnuto.

SOAP

SOAP web servisi su stariji, ali i dalje dosta korišćeni web servisi. SOAP predstavlja ime komunikacijskog protokola koji ovi servisi koriste – Simple Object Access Protocol. Ovaj protokol je odgovoran za dostavljanje poruka, obično preko HTTP protokola, koji se koristi zbog manje verovatnoće da će biti blokiran od strane „firewall“-a. (videti [9])

Počev od verzije 1.2, slova u akronimu nemaju neko specifično značenje. (videti [11])

Svaki SOAP web servis pruža javni interfejs koji je dostupan preko specificiranog URL-a. Ova ulazna tačka servisa se naziva „endpoint“. Kada korisnik želi da koristi usluge nekog SOAP web servisa, on mora da zna tačnu ulaznu tačku i definiciju interfejsa. Sve ovo je opisano u WSDL fajlu. (videti [9])

SOAP je glavna specifikacija web servisa. Sve što ona opisuje je jedan XML koverat i procesiranje modela za transfer poruka preko Interneta. Takođe, sadrži uputstva za SOAP programere o tome kako se transfer protokol može koristiti za transfer SOAP poruka.

Preko SOAP-a se ne mogu razrešiti problemi kao što su bezbednost ili transakcije i ne pokušava da nametne semantiku na aplikacionom nivou ili poručne obrasce na SOAP porukama. U tom smislu, SOAP je dosta jednostavniji od HTTP specifikacije. (videti [10])

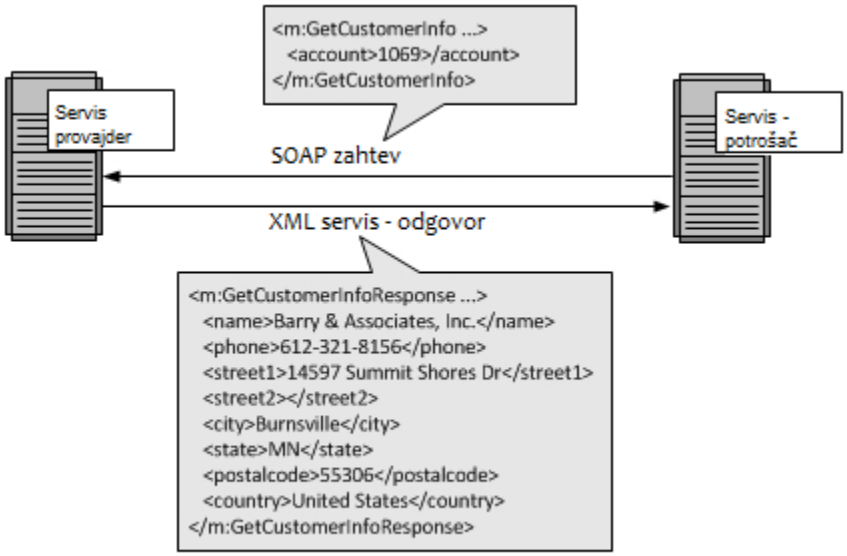
Tri glavne karakteristike SOAP-a su:

- Proširivost (eng. extensibility) – npr. bezbednost je jedna od ekstenzija – nije definisana u samoj specifikaciji
- Neutralnost (eng. neutrality) – SOAP može da radi kroz bilo koji transportni protokol, HTTP, SMTP, TCP, UDP ili JMS
- Nezavisnost (eng. independence) – SOAP dozvoljava bilo koji model programiranja. (videti [12])

SOAP koverat (eng. envelope) se sastoji od mesta (eng. placeholder) za zaglavlja koja sadrže metapodatke za postavljanje konteksta za procesiranje poruke, npr. bezbednosni kontekst (eng. security context), usmeravanje (eng. routing), i druge.

SOAP koverti se prenose kroz proizvoljne transportne protokole sa vezama definisanim kroz različite SOAP specifikacije za povezivanje. Od njih, jedino je široko prihvaćen SOAP preko HTTP-a.

SOAP poruke se mogu usmeravati kroz proizvoljan broj posrednika (na mreži ili unutar servisa), a svaki od njih procesira SOAP zaglavlja svake SOAP poruke koja prođe.



Slika 1:Korišćenje SOAP-a u web servisima

Dakle, SOAP samo definiše koverat i značenja transfera tog koverta preko mreže. Aplikacione semantike se održavaju unutar servisa, a determinisane su kroz poruku – zaglavlje i telo poruke.

SOAP je protokol za slanje poruka niskog nivoa koji ne nameće aplikacione semantike, već ostavlja servisima da definišu način interpretacije poruka koje primaju.

Kako se SOAP uglavnom koristi za transport HTTP koverata preko HTTP konekcije, javlja se dosta briga u web zajednici. Tvrdnje su da, pošto Web već omogućava proširive koverte, metapodatke, „entity body“ i podršku za posrednike, SOAP samo unosi preopširnost, kašnjenje i kompleksnost. Dalje, SOAP poruke se prenose preko HTTP POST, što znači da se gube postojeće pogodnosti infrastrukture Web-a, a posebno keširanje.

Koverat je fundamentalna struktura u oba, SOAP i HTTP - sveta i predstavlja prostor za podatke i metapodatke. Struktura SOAP koverta je slična strukturi HTTP koverta. (videti [10])

Na primer,

XML dokument:

```
<hello-world xmns="http://example.com">
```

postaje SOAP dokument na sledeći način:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soa/envelope/">
     <soap:Body>
          <hello-world xmns="http://example.com">
     </soap:Body>
</soap:Envelope>
```
(videti [5])

SOAP zaglavlja i telo su okruženi dodatnim SOAP tagovima `<soap:Envelope>` i `<soap:Body>`. Poruke mogu biti definisane preko WSDL specifikacije.

SOAP ima osnovni mehanizam za greške, nazvan SOAP fault, koji saopštava da li je greška nastala od strane korisnika, servera ili posrednika, kao i dodatne informacije o tome zašto se greška desila. Ali, nema standardizovane informacije o tome kako se oporaviti od neuspešne interakcije. Drugim rečima, ne postoji ekvivalent HTTP status kodovima ili link koji vodi do uputstva za interakciju.

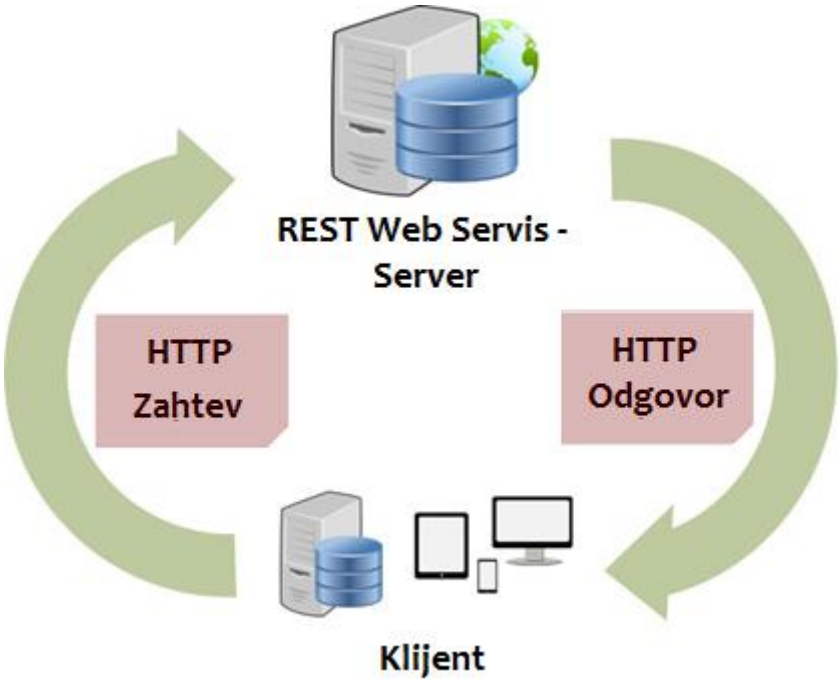
Primer SOAP fault-a:

```
<?xml version="1.0" ?>
<env:Envelope
xmlns:env="http://www.w3.org/2002/06/soap-envelope">
<env:Body>
<env:Fault>
<env:Code>
<env:Value>env:Receiver</env:Value>
</env:Code>
<env:Detail>
java.io.FileNotFoundException: db.txt
at java.io.FileInputStream.<init>(FileInputStream.java)
at java.io.FileInputStream.<init>(FileInputStream.java)
at Service.main(Service.java:15)
</env:Detail>
</env:Fault>
</env:Body>
</env:Envelope>
```

Ovo je sintaksno korektno, ali onaj ko dobije ovakvu poruku ne može s njom mnogo da uradi. Poruka ne opisuje problem u protokolu, već internu implementaciju. Ne postoji način da korisnik preduzme neku značajnu akciju nakon ovakve poruke, osim da je šalje dalje. (videti [10])

REST

Pojam web servisi je postao usko vezan za SOAP, tako da zajedno sa njim pada u zaborav. SOAP se još uvek koristi, ali samo u velikim kompanijama gde je teško promeniti tehnologiju. REST je doneo novo ime – API.



Slika 2: REST web servis

Jedan od aspekata REST-a, koji većina programera još uvek ne razume, jeste hipermedija (eng. hypermedia).

Multimedija predstavlja integraciju različitih medijskih elemenata koji su u osnovi samostalni sadržaji. To je informacija predstavljena ili memorisana u kombinaciji teksta, grafike, zvuka, animacije i videa objedinjenih pomoću računara. (videti [17])

Hipermedija je nadogradnja na multimediju, a označava multimedijske sadržaje koji su međunobno isprepleteni. Mreža se najčešće gradi poveznicama (eng. Hyperlink). Klasičan primer hipermedije je WWW. (videti [18])

Hipermedija u web API-ima je funkcionalnost koja čini web API sposobnim za izmene. (videti [7])

REST (Representational State Transfer) je stil softverske arhitekture za distribuirane hipermedijalne sisteme ili sisteme u kojima su tekst, grafika, audio i druge vrste medija

uskladištene kroz mrežu i međusobno povezane pomoću hiperlinkova. Najveći takav distribuirani sistem (koji radi sa web servisima) je WWW. (videti [1])

REST arhitekturalni stil je razvijan od strane W3C TAG (Technical Architecture Group), u paraleli sa HTTP 1.1, a koji je zasnovan na postojećem dizajnu HTTP 1.0. WWW (World Wide Web) je najveća implementacija ovog stila. (videti [6])

REST je široko prihvaćen kao jednostavna alternativa za SOAP i WSDL web servise.

REST sistemi obično, mada ne uvek, komuniciraju preko HTTP protokola, koristeći HTTP metode, a koriste ih web pretraživači za dobijanje web strana i slanje podataka do udaljenih servera. (videti [6])

U webu, HTTP je i transportni protokol i poručni sistem zato što su HTTP zahtevi i odgovori – poruke. Tela HTTP poruka (eng. payload) su obično MIME (Multipurpose Internet Mail Extension) tipa. HTTP odgovori sadrže i statusne kodove kojima informišu klijente da li je zahtev uspeo, i, ako nije, zašto. (videti [1])

URI (Uniform Resource Identifier) jeste niz karaktera koji se koristi da identifikuje resurse. Takva identifikacija omogućava interakciju sa reprezentacijama resursa preko mreže, uglavnom preko WWW-a, koristeći neki određeni protokol.

Minimalna informaciona jedinica u REST-u je resurs. Resursi su izvori podataka koji sadrže funkcionalnosti i aplikaciono stanje sistema. (videti [14])

Bilo šta može biti nazvano resursom: dokument, privremeni servis, kolekcija resursa, ne-virtuelni objekat (npr. osoba) i tako dalje. Resurs je konceptualno mapiranje na set entiteta, a ne entitet koji odgovara mapiranju u nekom određenom trenutku. (videti [16])

Resurs je izvor reprezentacije, a reprezentacija je samo neka informacija o trenutnom stanju resursa. Server može poslati podatke kao XML dokument, web stranu, JSON dokument… Ovo su različite reprezentacije istog resursa. Kao definiciju možemo da prihvatimo sledeće: Reprezentacija je svaka korisna informacija o stanju resursa. (videti [5])

O reprezentaciji razmišljamo kao o nečemu što server šalje klijentu. To je zato što, kada pretražujemo web, većina naših zahteva su GET zahtevi. Dakle, mi pitamo za reprezentaciju. Međutim, u PUT, POST ili PATCH zahtevu, klijent šalje reprezentaciju serveru. Tada je posao servera da promeni stanje resursa tako da se ono reflektuje na dolazeću reprezentaciju. (videti [7])

Karakteristike REST-a

REST ima četiri važne karakteristike. Dve najvažnije su adresiranje (eng. addressability) i nepostojanje stanja (eng. statelessness). Ostale su povezanost (eng. connectedness) i uniformni interfejs (eng. The Uniform Interface).

Adresiranje

Aplikacija je adresabilna ako otkriva interesantne aspekte svojih podataka kao resurse. Kako su resursi dostupni preko URI-a, adresabilna aplikacija obezbeđuje URI za svako parče informacije koje može da prikaže na razumljiv način.

Sa perspektive krajnjeg korisnika, adresiranje je najvažniji aspekt svakog web sajta ili aplikacije.

Moguće je ulančati URI-e (eng. to chain) i koristiti jedan URI kao ulazni parametar za drugi. Na primer, možemo koristiti neki eksterni web servis kako bismo validirali HTML, ili da bismo preveli tekst na drugi jezik. Ovi web servisi očekuju URI kao ulazni parametar. Da HTTP nije adresabilan, ne bismo imali načina da im kažemo koje resurse želimo da obradimo.

Adresiranje je jedna od najboljih stvari web aplikacija. Zahvaljujući njemu, klijenti mogu lako da koriste web sajtove.

Korišćenje ovog pravila donosi aplikaciji i njenim korisnicima mnoge pogodnosti REST-a.

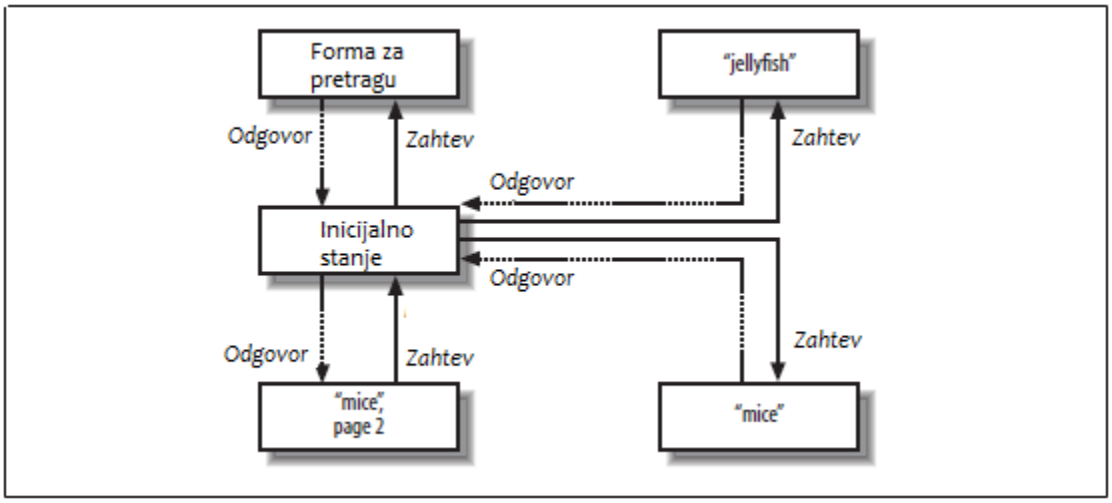
Nepostojanje stanja

Druga, od četiri najvažnije karakteristike ROA, je nepostojanje stanja. To znači da se svaki HTTP zahtev dešava u kompletnoj izolaciji. Kada klijent napravi HTTP zahtev, on uključuje sve informacije koje su potrebne serveru da obradi taj zahtev. Server se nikada ne oslanja na informacije iz prethodnog zahteva.

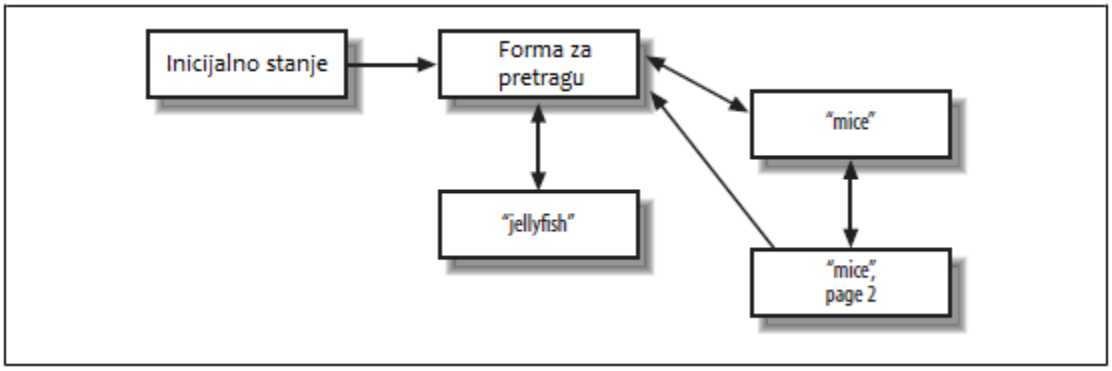
Posmatrajmo nepostojanje stanja u okviru adresiranja. Adresiranje kaže da svako interesantno parče informacije, koje server može da posluži, treba da bude prikazano kao resurs i da ima svoj URI. Nepostojanje stanja omogućava da su posebna stanja servera takođe resursi i da treba da imaju svoj URI. Klijent ne treba da utiče na server da pređe u određeno stanje kako bi ga učinio sposobnim da prihvati određeni zahtev.

Kao korisnici, često nailazimo na situacije kada „BACK“ dugme u našem pretraživaču ne radi korektno i ne možemo da idemo unazad i unapred kroz istoriju pretraživanja. Ponekad je to zato što smo odradili neku akciju koja ne može da se ponovo izvrši, kao što je postavljanje

komentara na blogu ili kupovina knjige, ali ovo se uglavnom dešava kada taj web sajt narušava principe nepostojanja stanja. Takvi sajtovi očekuju da klijent šalje zahteve u tačno određenom redosledu: A, B, pa zatim C. Većina desktop aplikacija radi ovako.



Slika 3: "stateless" pretraga



Slika 4: "stateful" pretraga

Kod aplikacija, koje poštuju principe nepostojanja stanja, svaki zahtev je totalno nezavisan od drugih i klijent ih može izvršavati nebrojeno puta i u bilo kom redosledu.

Na primer, FTP (File Transfer Protocol) ima stanja. Možemo se ulogovati na FTP server, zatim pomoću „cd“ komande doći do određenog direktorijuma i onda pomoću druge komande pročitati fajl odatle. Ako želimo da koristimo još neki fajl iz tog direktorijuma, ne moramo ponovo izvršavati „cd“ komandu kako bismo došli do njega.

Stanje bi učinilo HTTP zahtev jednostavnijim, ali bi onda sam HTTP protokol bio mnogo kompleksniji. FTP klijent je mnogo komplikovaniji od HTTP klijenta, upravo zbog toga što stanje sesije mora ostati sinhronizovano između klijenta i servera. Ovo je kompleksan zadatak čak i za mreže kojima se može verovati, a Internet to svakako nije.

Kad se eliminiše stanje iz protokola, eliminiše se i dosta uslova za neuspeh. Server ne mora da brine o klijentovom vremenu trajanja, odnosno o njegovoj sesiji, zato što nijedna interakcija između klijenta i servera ne traje duže od jednog zahteva. Klijent nikada neće završiti sa nekim zahtevom u „pogrešnom“ direktorijumu zbog toga što server pamti neko njegovo prethodno stanje.

Nepostojanje stanja donosi i nove funkcionalnosti. Ovakve aplikacije je lakše distribuirati kroz „load-balanced“ servere. Ovi serveri se nalaze na više različitih fizičkih ili virtualnih mašina, a “load-balancer” onda preusmerava saobraćaj aplikacije ka nekom od tih servera, u zavisnosti od opterećenja samih servera, tako da se kompjuterski resursi koriste optimalno. Kako ne postoje dva zahteva koji zavise jedan od drugog, njih mogu obrađivati dva različita servera koja međusobno nikad ne koordiniraju. Skaliranje (eng. scaling up) je lako, kao i dodavanje novog servera u „load balancer“. Aplikacije bez stanja je takođe lako iskeširati: parče softvera može da odluči da li da kešira rezultat nekog HTTP zahteva prihvatajući samo taj jedan zahtev.

Da bi servis postao adresabilan, mora se nešto uraditi – prebaciti podatke iz aplikacije u set resursa. HTTP je u osnovi „stateless“ protokol (protokol bez stanja), tako da kad pišemo web servise, po pravilu, dobijamo nepostojanje stanja. Ali, da bi se to promenilo, mora se nešto uraditi, a najčešći način je implementiranje naše verzije HTTP sesije.

Povezanost

Ponekad su reprezentacije nešto više od serijalizovanih struktura podataka. Kod većine REST-olikih servisa, reprezentacije su hipermedijske: dokumenti koji ne sadrže samo podatke, već i linkove ka drguim resursima.

Server šalje klijentu uputstva o tome koja su stanja blizu trenutnog. Primeri su „next“/“previous“ linkovi, zatim „page 1, 2, 3, …“ prilikom Google pretrage, i drugi.

Ova karakteristika se zove povezanost. Web servis je povezan sa drugim resursima do kojih klijent može da dođe prateći samo linkove.

Uniformni interfejs

Četiri najvažnija HTTP metoda su GET, POST, PUT i DELETE. Malo manje važni metodi su PATCH, HEAD i OPTIONS.

- GET služi za dobijanje reprezentacije nekog resursa
- POST služi za kreiranje novog resursa za postojeći URI

- PUT služi za kreiranje novog resursa za novi URI ili modifikovanje postojećeg resursa za postojeći URI
- DELETE služi za brisanje postojećeg resursa
- HEAD služi za dobijanje metapodataka nekog resursa (ekvivalentno zaglavlju koje se dobija kroz GET)
- OPTIONS služi za proveru koje HTTP metode određeni resurs podržava
- PATCH služi za parcijalno modifikovanje postojećeg resursa
- TRACE se koristi za debagovanje proxy-a

Ranije je postojao još jedan metod – CONNECT, koji se koristio da vrati neki drugi protokol kroz neki HTTP proxy.

Dve važne karakteristike metoda HTTP protokola jesu bezbednost (eng. safety) i idempotentnost (eng. idempotence).

Bezbednost znači da koliko god puta klijent pošalje isti zahtev, neće promeniti stanje servera ili resursa. GET i HEAD su bezbedne operacije. Dakle, isto je da li je klijent poslao deset ili nijedan GET zahtev.

Idempotentnost: Neka operacija nad resursom je idempotentna ako je slanje jednog zahteva isto kao i slanje serije identičnih zahteva. Drugi i ostali zahtevi ostavljaju stanje resursa potpuno isto kao što je bilo posle prvog zahteva. Idempotentni su GET, HEAD, PUT i DELETE.

Bezbednost i idempotentnost dozvoljavaju klijentu da šalje pouzdane HTTP zahteve preko nepouzdane mreže.

POST nije ni bezbedna ni idempotentna operacija. Slanje dva identična POST zahteva ka istom resursu (URI-u) bi verovatno rezultiralo sa dva nova resursa istog tipa sa istim informacijama.

Dakle, ROA je zasnovana na četiri koncepta:

- Resursi
- Njihova imena (URI-i)
- Njihove reprezentacije
- Linkovi među njima

i četiri karakteristike:

- Adresiranje
- Nepostojanje stanja
- Povezanost
- Uniformni interfejs. (videti [5])

Autentifikacija:

Većina API-ja, da bi vratila odgovor, zahteva autentifikaciju.

Postoje dva koraka do autentifikacije. Prvi korak se dešava samo jednom (eng. one-time step) i tada korisnik postavlja svoje akreditive sa servis provajderom. To obično znači da čovek preko web pretraživača kreira nalog na API serveru. Drugi korak je automatsko prezentovanje korisničkih akreditiva sa svakim zahtevom upućenim nekom API-u. Zašto svaki put? Zbog karakteristike REST-olikih servisa koja se zove nepostojanje stanja (eng. statelessness), što dozvoljava serveru da potpuno zaboravi na klijenta između dva zahteva. U implementaciji REST-olikih servera nema sesija.

Tri najpopularnije tehnike autentikacije su „Basic Auth“, „OAuth 1.0“ i „OAuth 2.0“.

Basic Auth

Ova tehnika autentifikacije je opisana u RFC 2617. To je jednostavna korisničko ime/ lozinka (eng. username/password) šema. Korisnik API-ja šalje korisničko ime i lozinku, koje je verovatno dobio registrovanjem na neki sajt povezan sa tim API-jem. Basic Auth je jednostavna tehnika autentifikacije, ali ima 2 velika problema. Prvi problem je što ova tehnika nije bezbedna, to znači da svako ko „špijunira“ nečiju internet konekciju može da ukrade te akreditive. Ovaj problem nestaje ako API koristi HTTPS umesto HTTP-a. Drugi problem nije tako bitan za WWW, ali je veoma ozbiljan u svetu API-ja: ljudi koji koriste API ne mogu verovati svojim klijentima. Na primer, ako se koristi neki popularan API kao što je Twitter API. Recimo da je neko toliko aktivan da ima 10 različitih klijenata za ovaj jedan API – nekoliko telefona, desktop računar, laptop računar i slično, a pri tom je drugim web sajtovima dao pravo da koriste ovaj API u njegovo ime. Šta ako neki klijent počne da postavlja „spam“ na ovaj nalog? Korisnik mora da promeni lozinku da „zao“ klijent više ne bi imao validne akreditive. Ali, svih deset klijenata ima istu lozinku. Devet klijenata je od poverenja, ali promena lozinke blokira svih 10. Korisnik prolazi kroz 9 klijenata i postavlja novu lozinku. Ali, šta ako jedan od tih devet postane zao? Korisnik opet mora da prolazi kroz isto. Ovde ne bi bilo problema da je svaki klijent dobio različiti set akreditiva. Ovo se rešava upotrebom OAuth-a.

OAuth

U OAuth-u, korisnik svakom klijentu dodeljuje individualni set akreditiva. Ako odluči da ukloni nekog klijenta, jednostavno opozove njegove akreditive, a ostalih 9 klijenata će ostati netaknuti.

Postoje dve verzije OAuth-a. OAuth 1.0 (RFC 5849) radi dobro za integraciju korisničkih web servisa sa određenim API-em. Ne radi kada je u pitanju integracija API-ja sa desktop, mobilnim ili „in-browser“ aplikacijama. OAuth 2.0 (RFC 6749) ima podršku za ove scenarije.

Android i web servisi

Operativni system Android je trenutno najrasprostranjeniji operativni sistem za mobilne uređaje. Zasnovan je na Linux jezgru i prilagođen tako da se može koristiti na većini mobilnih uređaja, uključujući mobilne telefone, tablet računare, prenosive računare, netbuk računare, smartbuk računare, čitače elektronskih uređaja, pa čak i ručne satove i druge uređaje.

Android je razvijan kao projekat otvorenog koda (eng. Open Source Project), što znači da je njegov izvorni kôd dostupan svima. Kao takav, projekat je okupio veliki broj programera i IT (Information Technology) stručnjaka koji svakodnevno rade na njegovom razvoju.

U novembru 2007. godine osnovano je udruženje OHA (Open Handset Alliance), koje je sada zaduženo za pružanje podrške i dalji razvoj sistema. (videti [19])

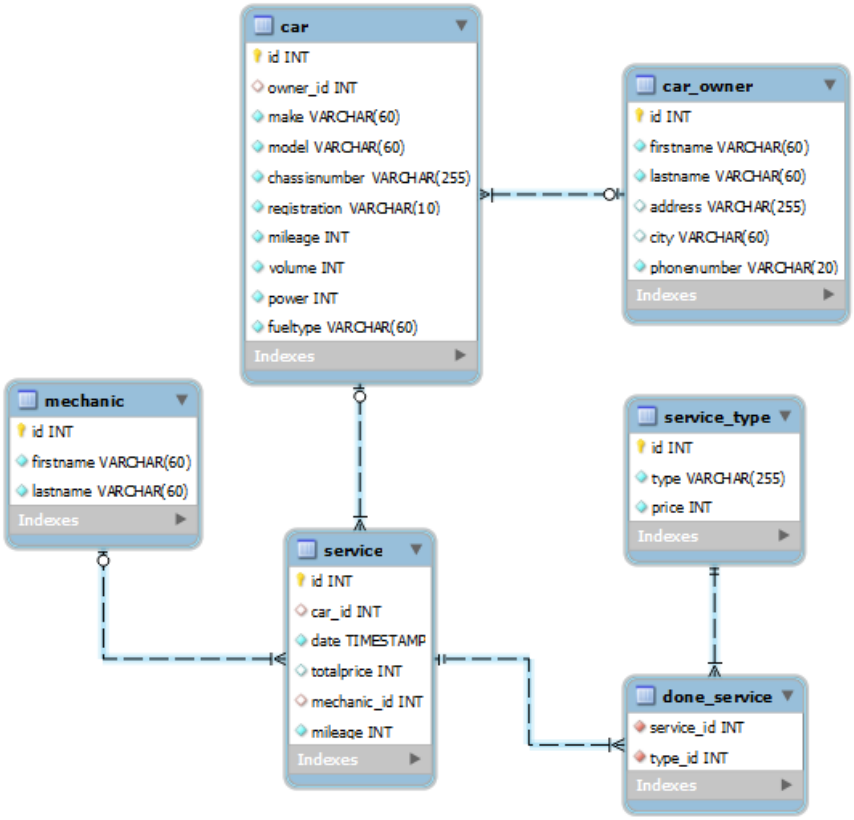
OHA ističe sledeće karakteristike Android operativnog sistema:

- Otvorenost – programer ima potpunu slobodu u razvoju novih i menjanju postojećih aplikacija, a proizvođač ima pravo na slobodno korišćenje i prilagođavanje platforme bez plaćanja autorskih prava
- Sve aplikcije su ravnopravne i svim aplikacijama je omogućen ravnopravni pristup resursima mobilnog uređaja
- Automatsko upravljanje životnim ciklusom aplikacije – postoji nadzor pokretanja i izvršavanja aplikacija na sistemskom nivou, a sve u cilju optimizovanog korišćenja memorije i snage uređaja
- Uklanjanje granica “klasičnih” aplikacija – mogućnost razvoja aplikacija zasnovanih na međusobnoj kolaboraciji tehnologija
- Brz i jednostavan razvoj aplikacija – zahvaljujući bogatoj bazi korisnih programskih biblioteka i alata za izradu aplikacija
- Visokokvalitetni grafički prikaz i zvuk
- Kompatibilnost sa većinom sadašnjeg i budućeg hardvera (videti [20])

Serverska strana – REST-oliki web servis

Na programskom jeziku Java, razvijeni su REST API-i koji korisniku omogućavaju da rukuje podacima sačuvanim u bazi podataka.

Na sledećoj slici je prikazan model baze podataka:

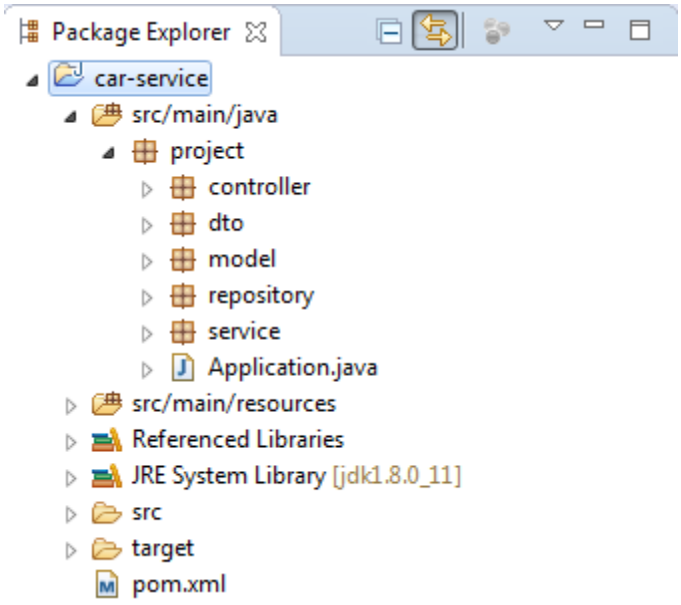


Slika 5:Model baze

U tabeli *mechanic* se čuvaju podaci o zaposlenima. Tabela *car* sadrži podatke o automobilima kojima je rađen servis, dok tabela *car_owner* čuva podatke o njihovim vlasnicima. U tabeli *service* se nalaze osnovni podaci o samom servisu, kao što su: datum, ukupna cena, mehaničar koji je obavljao servis, dok se u tabeli *done_service* nalaze stavke usluga koje su obuhvaćene jednim servisom (*service_type* tabela je zadužena za čuvanje mogućih stavki).

Dakle, resursi u ovom web servisu su car, carOwner, service, serviceType i mechanic. Za rad sa reprezentacijama ovih resursa kreirani su URI-I poput *<domain.com>/cars*.

Na sledećoj slici je prikazana struktura projekta:



Slika 6:Struktura Web API projekta

Projekat ima nekoliko slojeva:

- *Repository* – sloj zadužen za komunikaciju sa bazom podataka
- *Service* – sloj zadužen za biznis logiku
- *Controller* – tanak sloj zadužen za dostupnost API-a spoljnom svetu, odnosno klijentima.

U paketu *model* nalaze se domenski objekti koji predstavljaju tabele u bazi, a u paketu *dto* su takozvani "data transfer objects", odnosno reprezentacije resursa. Reprezentacija resursa ne mora nužno da sadrži sva polja koja ima sam domenski objekat. U nastavku sledi po jedan primer domenskog objekta i reprezentacije.

Domenski objekat:

```
@Entity
@Table(name = "service")
public class Service {
        @Id
        @GeneratedValue
        private Long id;

        @Column(nullable = false)
        private Date date;

        @Column(nullable = false, name = "totalprice")
        private Integer totalPrice;

        @ManyToOne
        @JoinColumn(name = "car_id")
        private Car car;

        @ManyToMany(fetch = FetchType.LAZY)
        @JoinTable(name = "done_service", schema = "masterproject",
                        joinColumns = {@JoinColumn(name = "service_id", referencedColumnName = "id")},
inverseJoinColumns = {@JoinColumn(
                                        name = "type_id", referencedColumnName = "id")})
        private List<ServiceType> types = new ArrayList<ServiceType>();
```

```
	@ManyToOne
	@JoinColumn(name = "mechanic_id")
	private Mechanic mechanic;

	@Column(nullable = false)
	private Long mileage;

	public Service() {
	}

	public Service(ServiceDto dto) {
		this.date = dto.getDate();
		this.mileage = dto.getMileage();
	}

	public Service(Date date) {
		this.date = date;
	}

	// getters and setters...

	public ServiceDto transferToDto() {
		ServiceDto dto = new ServiceDto();
		dto.setId(getId());
		dto.setDate(getDate());
		dto.setTotalPrice(getTotalPrice());
		dto.setMileage(getMileage());
		dto.setCarId(getCar().getId());
		dto.setMechanicId(getMechanic().getId());
		return dto;
	}
```

Reprezentacija:

```
public class ServiceDto {
	private Long id;
	private Date date;
	private Integer totalPrice;
	private Long mileage;

	private Long carId;
	private Long mechanicId;

	public ServiceDto() {
	}

	public ServiceDto(Date date) {
		this.date = date;
	}

	// getters and setters

}
```

Svi API-i vraćaju odgovor u JSON formatu. Na primer, ako se pošalje GET zahtev ka *<domain.com>/services/1*, odgovor će biti nalik ovom:

{
"id": 1,
"date": 1440695294000,
"totalPrice": 4000,

"mileage": 110000,
"carId": 2,
"mechanicId": 1
}

Obezbeđeni su svi API-i potrebni za funkcionisanje klijentske aplikacije, a biće navođeni kroz opis same Android aplikacije.

Klijentska strana – Android aplikacija za praćenje rada auto servisa

U narednom poglavlju je opisana aplikacija za praćenje rada autoservisa *Arizona*.

Izrada aplikacije se može podeliti na dva dela – funkcionalnost aplikacije i korisnički interfejs aplikacije.

Što se funkcionalnosti tiče, korisniku treba da bude omogućeno da registruje novog klijenta (u ovom slučaju automobil) i upiše važne podatke o njemu, pregled podataka o klijentu i pregled prethodnih, odnosno upis novog autoservisa koji će se obaviti nad tim klijentom. Svaki automobil je jednoznačno određen svojim brojem šasije, a od ostalih podataka se unose podaci o vlasniku, registracioni broj i podaci neophodni za izvršavanje servisa – vrsta goriva, snaga motora, zapremina… Pre bilo kakvog korišćenja, automehaničar mora da unese svoj identifikacioni broj. Zbog bezbednosnih razloga nije dozvoljeno registrovanje novih automehaničara kroz aplikaciju.

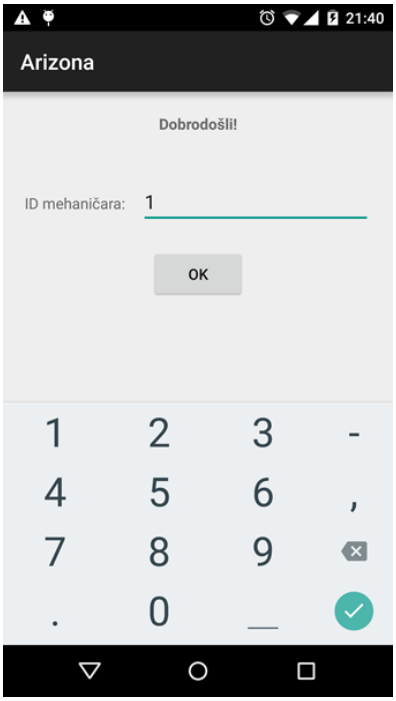
Korisnički interfejs treba da bude što jednostavniji za korišćenje. Kao i kod drugih Android aplikacija, i ovde smo ograničeni veličinom i karakteristikama ekrana uređaja. Slučajevi upotrebe će biti opisani u okviru pregleda aplikacije koji sledi.

Za rad aplikacije je neophodna konekcija sa internetom odnosno serverom na kom radi Web API aplikacija.

Opis aplikacije

1. Pokretanje aplikacije.

   Korisnik – automehaničar mora da unese svoj identifikacioni broj (u nastavku ID), kako bi mogao dalje da koristi aplikaciju.



Slika 7: Aplikacija - ekran 1

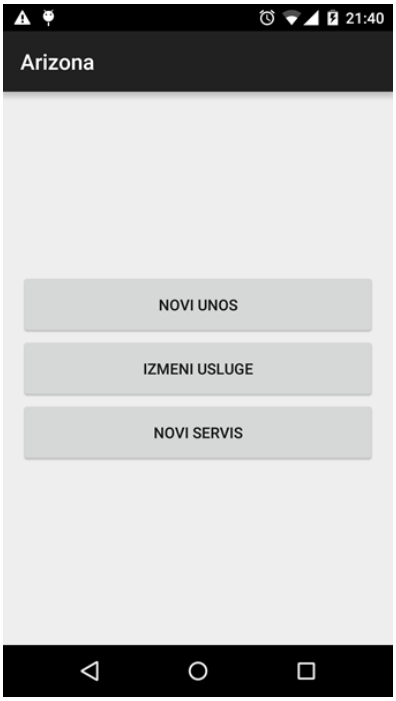
Klikom na dugme „OK“, aplikacija upućuje serveru GET zahtev:

<domain.com>/mechanics/{id}, gde je {id} zamenjeno unešenim ID-em korisnika.

Odgovor koji aplikacija dobija od servera sadrži status kôd ***200 (OK)*** i telo odgovora koje je u JSON formatu, nalik ovom:

```
{
  "id": 1,
  "firstName": "Jelena",
  "lastName": "Nikolic"
}
```

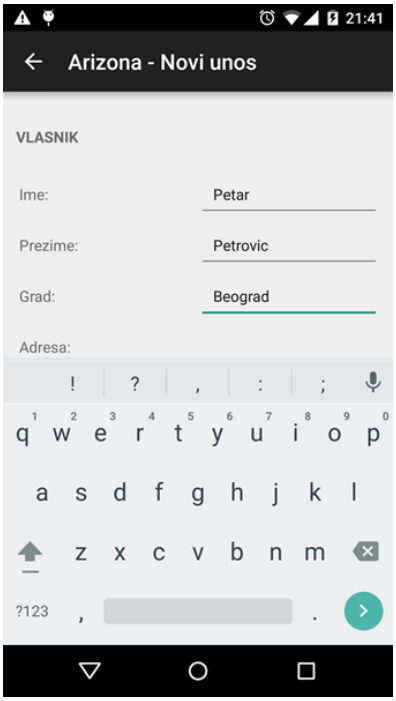
2. Nakon odgovora servera da je korisnik validan (registrovan u bazi podataka), korisnik može izabrati akciju koju želi da izvrši. On može odabrati da unese podatke o novom klijentu – automobilu, da promeni spisak usluga koje se mogu obaviti u njegovom autoservisu ili da unese podatke o novom servisiranju automobila koji je već u bazi.



Slika 8: Aplikacija - ekran 2

3. Korisnik je izabrao neku od ponuđenih akcija.
   3.1. Novi unos

   Korisnik unosi podatke o automobilu i vlasniku automobila na osnovu saobraćajne dozvole i odgovora koje dobija od vlasnika automobila. Nakon što je popunio formu, korisnik može izabrati da sačuva podatke ili da otkaže unos.



Slika 9: Aplikacija - ekran 3

3.1.1. Sačuvaj:

Podaci se šalju serveru, unos se snima u bazu i nakon toga prelazi na korak 4.

Server dobija POST zahtev na ***<domain.com>/cars***, sa telom poruke nalik ovom:

```
{
"owner" : {
   "firstName": "Jelena",
   "lastName": "Nikolic",
   "address": "Bulevar Mihaila Pupina 17",
   "city": "Beograd",
   "phoneNumber": "0649383243"
},
"car" : {
   "make": "Renault",
   "model": "Clio",
   "chassisNumber": "neka_tamo_sasija",
   "registration": "BG 821-CH",
   "mileage":107125,
   "volume": 1645,
   "power": 50,
   "fuelType": "diesel"
}
```

}

Odgovor servera je u obliku status kôda ***201 (Created)*** i tela poruke koji sadrži ID dodatog automobila.

3.1.2. Otkaži:

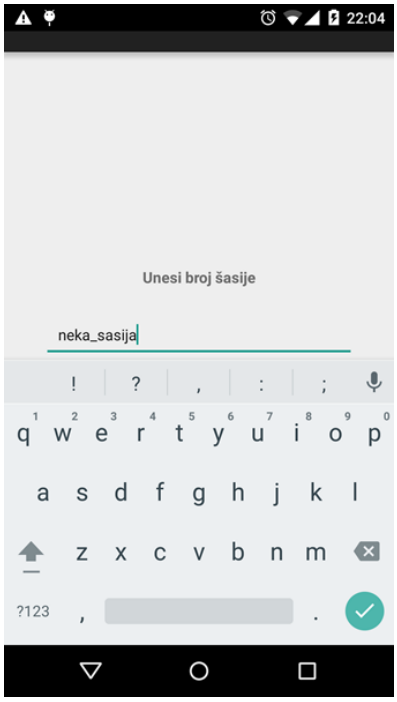
Unos se poništava, a aplikacija se vraća na korak 2.

3.2. Izmeni usluge

Otvara se ekran sa izlistanim postojećim uslugama i navedenim cenama za svaku od njih – korak 5.

3.3. Novi servis

Otvara se ekran sa poljem za unos broja šasije, nakon čega se prelazi na korak 4.

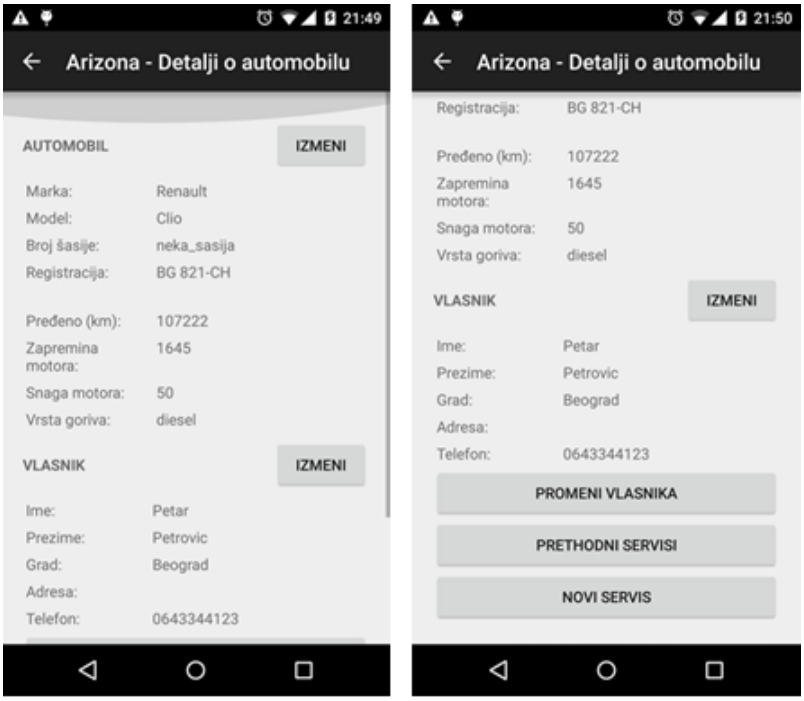


Slika 10: Aplikacija - ekran 4

Zahtev je GET ***<domain.com>/cars/{chassisNumber}***, gde je {chassisNumber} uneti broj šasije, a kao odgovor se dobijaju identifikacioni brojevi automobila i vlasnika automobila, koji se zatim čuvaju u memoriji same aplikacije.

4. Otvara se ekran koji prikazuje unete podatke o automobilu i vlasniku.

Ovde se mogu promeniti podaci o automobilu (registracija ili kilometraža) ili podaci o vlasniku. Takođe, ako se promenio vlasnik automobila, ovde je moguće uneti podatke o novom vlasniku. Druge opcije su pregled prethodnih servisiranja i unos novog servisa.



Slika 11: Aplikacija - ekran 5

Podaci prikazani na ovom ekranu su dobijeni kao odgovori na 2 odvojena GET zahteva.

Prvi je GET ***<domain.com>/cars/{chassisNumber}***, pri čemu je odgovor servera nalik:

```
{
  "id": 1,
  "make": "Renault",
  "model": "Clio",
  "chassisNumber": "neka_tamo_sasija",
  "registration": "BG 821-CH",
  "mileage": 107125,
  "volume": 1645,
  "power": 1151,
  "fuelType": "diesel",
  "ownerId": 1
},
```

a drugi GET ***<domain.com>/ carOwners/{id}***, gde je odgovor servera:

```
{
```

"id": 1,
"firstName": "Jelena",
"lastName": "Nikolic",
"address": "Bulevar Mihaila Pupina 17",
"city": "Beograd",
"phoneNumber": "0649383243"
}.

Status kôd je u oba slučaja ***200 (OK)***.

4.1. Izmeni (podatke o automobilu)

Nakon odabira ove opcije, na ekranu se pojavljuju samo podaci o automobilu, gde je moguće promeniti samo kilometražu i registracioni broj.

Slika 12:Aplikacija - ekran 6

4.1.1. Sačuvaj:

Podaci se šalju serveru, unos se snima u bazu i nakon toga se aplikacija vraća na korak 4 i prikazuje izmenjene podatke.

Server dobija PUT zahtev na ***<domain.com>/cars/{id}***, sa telom poruke nalik ovom:

{
"registration": "BG 821-CH",
"mileage":117125
}

Odgovor servera na PUT zahtev je status kôd **204 (No Content)**.

4.1.2. Otkaži:

Unos se poništava, a aplikacija se vraća na korak 4.

4.2. Izmeni (podatke o vlasniku)

Nakon odabira ove opcije, na ekranu se pojavljuju samo podaci o vlasniku, koje je moguće izmeniti.

Slika 13:Aplikacija - ekran 7

4.2.1. Sačuvaj:

Podaci se šalju serveru, unos se snima u bazu i nakon toga se aplikacija vraća na korak 4 i prikazuje izmenjene podatke.

Server se kontaktira zahtevom PUT ***<domain.com>/ carOwners/{id}*** sa odgovarajućim telom poruke, a njegov odgovor je isti kao u prethodnom slučaju, **204 (No Content)**.

4.2.2. Otkaži:

Unos se poništava, a aplikacija se vraća na korak 4.

4.3. Promeni vlasnika

Otvara se forma za popunjavanje podataka o vlasniku automobila (pogledati ekran iz koraka 4.2).

4.3.1. Sačuvaj:

Podaci se šalju serveru, unos se snima u bazu i nakon toga se aplikacija vraća na korak 4 i prikazuje izmenjene podatke.

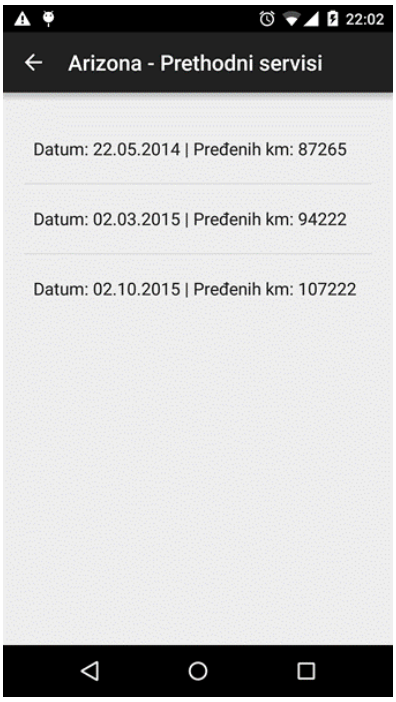
Zahtev serveru je POST ***<domain.com>/ carOwners***, sa telom poruke koji odgovara reprezentaciji vlasnika automobila.
Odgovor servera je u obliku status kôda ***201 (Created)*** i tela poruke koji sadrži ID dodatog automobila.

4.3.2. Otkaži:

Unos se poništava, a aplikacija se vraća na korak 4.

4.4. Prethodni servisi

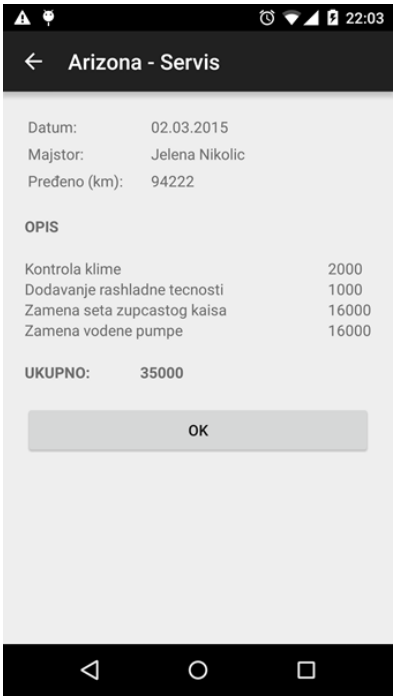
Nakon odabira ove opcije dobijamo izlistane datume svih prethodnih servisiranja obavljenih u ovom autoservisu i za automobil odabran u koraku 3.



Slika 14:Aplikacija - ekran 8

Podaci prikazani na ovom ekranu su dobijeni od servera kao odgovor na zahtev GET ***<domain.com>/ cars/{id}/services***, gde je {id} zapravo ID automobila čije prethodne servise želimo da izlistamo.

Klikom na neki od datuma dobijamo sve informacije o uslugama obavljenim u okviru naznačenog servisiranja.



Slika 15:Aplikacija - ekran 9

Sve ove informacije dobijamo pomoću tri odvojena zahteva.

Prvi je GET ***<domain.com>/services/{serviceId}***, gde je odgovor servera u obliku:

{
"id": 1,
"date": 1440695294000,
"totalPrice": 4000,
"mileage": 110000,
"carId": 2,
"mechanicId": 1
}.

Drugi zahtev je upućen zbog informacija o mehaničaru koji je bio zadužen za taj servis, a to je GET ***<domain.com>/mechanics/{id}***, a treći zahtev nam je potreban da bismo dobili spisak stvari obavljenih u okviru servisa. To je GET ***<domain.com>/services/{serviceId}/types***. Ono što server šalje kao odgovor na poslednji zahtev je u obliku:

```
[
 {
  "id": 2,
  "type": "Zamena filtera ulja",
  "price": 2000
 },
 {
  "id": 4,
  "type": "Zamena filtera goriva",
  "price": 2000
 },
 {
  "id": 5,
  "type": "Kontrola klime",
  "price": 2000
 }
].
```


Klikom na dugme „Nazad“, aplikacija se vraća na korak 4.4.

4.5. Novi servis

Na ovom ekranu se mogu izabrati usluge koje treba obaviti u okviru servisiranja automobila.

Slika 16:Aplikacija - ekran 10

Ovde je bilo potrebno serveru uputiti GET ***<domain.com>/serviceTypes*** zahtev, kako bismo dobili kompletnu listu mogućih usluga prilikom servisiranja.

4.5.1. Sačuvaj:

Podaci se šalju serveru, unos se snima u bazu i nakon toga se aplikacija vraća na korak 4.4 i prikazuje podatke koji su upravo uneti.

Zahtev serveru je POST ***<domain.com>/services*** sa telom poruke:

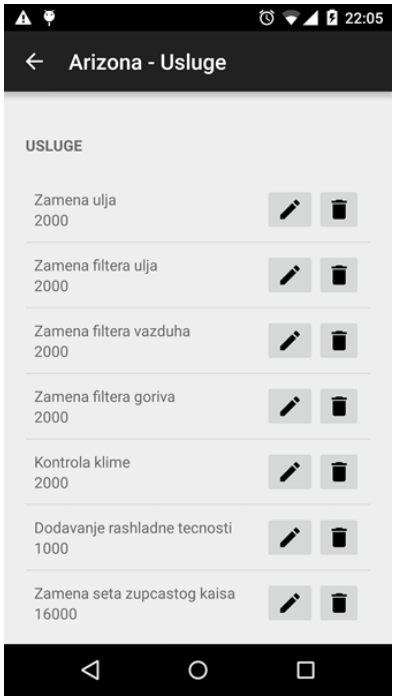
```
{
   "service": {
      "mileage": "110000",
      "carId": 2,
      "mechanicId": 1
   },
   "typeIds": [
      4,5,7
   ]
}.
```

Odgovor servera je u obliku status koda ***201 (Created)*** i tela poruke koji sadrži ID dodatog servisa.

4.5.2. Otkaži:

Unos se poništava, a aplikacija se vraća na korak 4.

5. Odabir opcije „Izmeni usluge“ iz koraka 3 otvara ekran sa izlistanim svim uslugama koje ovaj autoservis pruža svojim klijentima.



Slika 17:Aplikacija - ekran 11

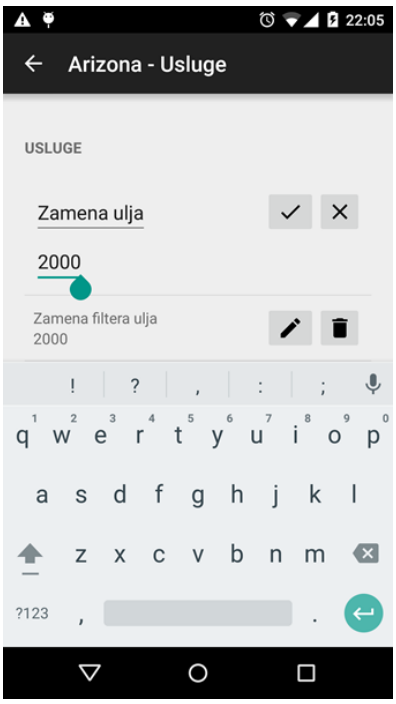
Za prikaz podataka na ovom ekranu se koristi već pomenuti zahtev GET ***<domain.com>/serviceTypes***, a odgovor servera je status kôd ***200 (OK)*** uz telo poruke :

```
[
 {
  "id": 1,
  "type": "Zamena ulja",
  "price": 2000
 },
 {
  "id": 2,
  "type": "Zamena filtera ulja",
  "price": 2000
 },
 {
  "id": 3,
  "type": "Zamena filtera vazduha",
  "price": 2000
 },
 {
  "id": 4,
  "type": "Zamena filtera goriva",
  "price": 2000
```

```
},
{
  "id": 5,
  "type": "Kontrola klime",
  "price": 2000
},
{
  "id": 6,
  "type": "Dodavanje rashladne tecnosti",
  "price": 1000
},
{
  "id": 7,
  "type": "Zamena seta zupcastog kaisa",
  "price": 16000
},
{
  "id": 8,
  "type": "Zamena vodene pumpe",
  "price": 16000
},
{
  "id": 9,
  "type": "Balansiranje guma",
  "price": 2000
},
{
  "id": 10,
  "type": "Zamena gume",
  "price": 1000
}
].
```


Naziv, odnosno cena, svake usluge se mogu izmeniti. Svaka od usluga se može obrisati ukoliko autoservis prestaje sa radom te vrste. Realan primer je da je u okviru autoservisa postojala i vulkanizerska radnja, ali je vlasnik servisa odlučio da zatvori taj sektor. Usluge poput zamene guma više neće postojati, pa ih je potrebno obrisati sa spiska.

Za brisanje se serveru šalje zahtev DELETE ***<domain.com>/serviceTypes/{id}***, gde se kao {id} prosleđuje ID stavke koju korisnik želi da obriše.



Slika 18:Aplikacija - ekran 12

Za izmenu stavke server dobija zahtev PUT ***<domain.com>/serviceTypes/{id}*** sa telom poruke koje odgovara reprezentaciji jednog tipa servisa. U oba slučaja (DELETE i PUT) server kao odgovor šalje status kôd **204 (No Content)**.

Svi status kodovi korišćeni u odgovorima se nalaze u HTTP specifikaciji. Za PUT, POST i DELETE zahteve najbolja je praksa koristiti gore navedene kodove, 201 i 204, ali nije zabranjeno koristiti i druge, poput status koda 200.

U slučaju GET zahteva za nepostojećom reprezentacijom (na primer, mehaničar pokušava da pokrene aplikaciju pomoću ID-a koji ne postoji u bazi podataka), server će odgovoriti HTTP status kodom **404 (Not Found)**. Ostali nevalidni tipovi zahteva završiće se odgovorom **400 (Bad Request)**.

ZAKLJUČAK

U prethodnim poglavljima smo se upoznali sa istorijom i razvojem web servisa. Zaključujemo da su web servisi programske komponente koje omogućavaju pravljenje skalabilnih, slabo povezanih i platformski nezavisnih aplikacija. Opisano je koje vrste web servisa postoje, koje su više, a koje manje korišćene.

Kao primer, naveden je web servis koji je implementiran na Java programskom jeziku, a kojeg poziva Android aplikacija. Prikazan je primer aplikacije i navedeni alati koji su potrebni da bi se naše ideje sprovele u delo.

Dokumentacija o Android operativnom sistemu je dobro organizovana i vrlo su jasno objašnjene smernice za razvoj jedne aplikacije. Zahvaljujući ovome, relativno brzo je razvijena aplikacija *Arizona*. Poznavanje osnovnih karakteristika web servisa i dobra dokumentovanost Androida omogućavaju programeru da prihvatljive aplikacije razvije za kratko vreme.

Iako *Arizona* predstavlja samo primer korišćenja REST API-a, može se proširiti i unaprediti tako da postane korisna u stvarnom svetu, obzirom na to da bi mogla nekom uštedeti vreme i olakšati mu administraciju, i naravno, na činjenicu da je Android postao neprikosnoven na tržištu mobilnih uređaja.

Glavno unapređenje bi se moglo odnositi na dodavanje svojevrsnog čitača/skenera saobraćajne dozvole. Nakon skeniranja dozvole, aplikacija bi sama mogla da popuni podatke o automobilu i eventualno o vlasniku automobila, što bi dodatno ubrzalo proces i uštedelo vreme i automehaničaru i osobi koja je dovezla auto na servisiranje. Takođe bi se smanjile eventualne greške u kucanju.

Literatura

[1] Java Web Services: Up and Running, Second Edition, Martin Kalin, 2013

[2] RESTful Web Services Cookbook, First Edition, Subbu Allamaraju, 2010

[3] URL: https://en.wikipedia.org/?title=web_service

[4] URL: www.w3.org/TR/ws-arch

[5] RESTful Web Services, First Edition, Leonard Richardson and Sam Ruby, 2007

[6] https://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services

[7] RESTful Web APIs, First Edition, Leonard Richardson and Mike Amundsen with a Foreword by Sam Ruby, 2013

[8] https://en.wikipedia.org/wiki/Web_Services_Description_Language

[9] Mobile system for management of EEG/ERP experiments, Master thesis, Petr Miko, 2013

[10] REST in Practice, First Edition, Jim Webber, Savas Parastatidis, and Ian Robinson, 2010

[11] URL: www.service-architecture.com/articles/web-services/soap.html

[12] URL: https://en.wikipedia.org/wiki/SOAP

[13] Building Hypermedia APIs with HTML5 and Node, First Edition, Mike Amundsen, 2012

[14] Design and development of a REST-based Web service platform for applications integration, Master thesis, Luis Oliva Felipe, 2010

[15] URL: www.webreference.com/authoring/web_service/index.html

[16] Architectural Styles and The Design of Network-based Software Architectures, PhD Dissertation, R. T. Fielding, 2000

[17] http://www.pmf.ni.ac.rs/pmf/predmeti/4604/doc/08-Multimedija.pdf

[18] https://hr.wikipedia.org/wiki/Hipermedija

[19] https://en.wikipedia.org/wiki/Open_Handset_Alliance

[20] http://www.openhandsetalliance.com