UNIVERZITET METROPOLITAN BEOGRAD

IT255 - VEB SISTEMI 1

Web arhitekture

Lekcija 02

PRIRUČNIK ZA STUDENTE

IT255 - VEB SISTEMI 1

Lekcija 02

WEB ARHITEKTURE

UVOD

Nastavak sticanja znanja neophodnog za razvoj veb sistema

Lekcija nastavlja nastavak sticanja znanja neophodnog za razvoj veb sistema sa akcentom na sledećim temama:

- *klijent - server* arhitektura (dvoslojna i troslojna);
- softverski okvir (*framework*);
- *SOA* - servisno orijentisana arhitektura;
- *brokeri* servisa.

Savladavanjem ove lekcije student će da razume benefite dizajniranja softverske veb arhitekture, kao i razlike između veb arhitektura pout *SOA*, *MVC* i sličnih.

Poglavlje 1

Dvoslojna Klijent-Server (K-S) arhitektura

NAJČEŠĆE KORIŠĆENI ARHITEKTONSKI STILOVI

Mnogi sistemi imaju sličnu strukturu. Na primer, distribuirani sistemi obično imaju klijent-server strukturu u kojoj klijent upućuje upite, a server procesira te upite i odgovara na njih.

Na početku razvoja sistema potrebno je izabrati arhitekturu koja će biti primenjena. Mnogi sistemi imaju sličnu strukturu. Na primer, distribuirani sistemi obično imaju klijent-server strukturu u kojoj klijent upućuje upite, a server procesira te upite i odgovara na njih. Dosadašnja iskustva po pitanju arhitekture sistema dovela su do pojave različitih stilova u projektovanju. Stil podrazumeva strukturalnu organizaciju softverskog sistema. On uključuje izbor komponenata (podsistema), kao i definisanje njihovih uloga.

Na primer, u klijent-server arhitekturi razlikuju se dva podsistema: klijent (kojih može biti više) i server (koji je jedinstven). Uloga klijenta može biti prikazivanje korisničkog interfejsa korisniku, a uloga servera procesiranje brojnih zahteva i zaštita podataka važnih za korisnika.

Stil, takođe, odražava i veze između podsistema, tj. način njihovog povezivanja, uz eventualna ograničenja. Na primer, veza između klijenta i servera se ostvaruje tako što klijent postavlja pitanja, a server odgovara na njih. Nekoliko najčešće korišćenih stilova, koje smo već pomenuli ranije, su:

- cevi i filtri (pipe-filter)
- slojevita (layers) arhitektura
- klijent-server (client-server) arhitektura
- ravnopravan (peer-to-peer) pristup
- arhitektura zasnovana na događajima (event-bus)
- objektno-orijentisani (object-oriented) pristup

Različite osobe uključene u razvoj sistema vide ga u različitim etapama životnog ciklusa projekta na različit način. Arhitektura sistema je najvažnija tvorevina koja se može upotrebiti da bi se kontrolisao postepeni razvoj sistema tokom njegovog životnog ciklusa. Arhitektura sistema je skup važnih odluka o:

- organizaciji softverskog sistema;

• izboru strukturnih elemenata od kojih je sistem sastavljan i njihovim interfejsima;

• ponašanju elemenata kako je to specifikovano u društvu saradnika koje sačinjavaju ti elementi;

• sastavljanju tih elemenata kako strukturnih tako i onih za opis ponašanja i formiranju podsistema;

KLIJENT-SERVER ARHITEKTURA

Arhitektura koja se zasniva na ravnopravnom pristupu može se shvatiti kao simetrična klijent-server arhitektura.

• stilu koji se usvaja u organizaciji (statičkim i dinamičkim elementima i njihovim interfejsima, njihovim društvima saradnika i odnosima između njih).

Softverska arhitektura se ne odnosi samo na strukturu i ponašanje već i na funkcionalnost, performanse, dogradljivost, ponovno korišćenje, razumljivost, ekonomska i tehnička ograničenja, određene kompromise i estetiku.

Klijent-server arhitektura

U klijent-server arhitekturi, serverska komponenta pruža usluge većem broju klijentskih komponenata. Klijentske komponente zahtevaju usluge od servera. Serveri su stalno aktivni i osluškuju da li ima zahteva od klijenata. Zahtevi se šalju komunikacionim kanalima koji zavise od mašina na kojima se server i klijent nalaze. Opšti izgled ove arhitekture prikazanje na slici 1.

Tipični primeri klijent-server arhitekture su aplikacije sa udaljenim pristupom bazama podataka (kada klijentska aplikacija zahteva uslugu od servera baze podataka), udaljeni fajl sistemi (klijentska aplikacija pristupa datotekama na serveru i u lokalu transparentno), ili web aplikacije (čitači zahtevaju podatke od web servera). Prispeli zahtevi se obično opslužuju u odvojenim nitima na serveru.

U komunikaciji često ima „praznog hoda" koji nastaje kako zbog saobraćaja na mreži, tako i zbog neophodnog transformisanja zahteva i rezultata u formate koji su često različiti na serverskoj i klijentskog strani. U distribuiranim sistemima sa više servera koji obavljaju istu funkciju, mora se obezbediti transparentnost, što znači da klijenti ne treba da razlikuju servere. Na primer, ako se u Google pretraživaču zahteva neki podatak unošenjem URL adrese, klijent ne treba da zna na kojoj tačno mašini je taj podatak lociran, koja je platforma primenjena, ili koja je putanja korišćena.

Ravnopravni pristup

Arhitektura koja se zasniva na ravnopravnom pristupu može se shvatiti kao simetrična klijent-server arhitektura. U ovoj arhitekturi, ista komponenta može da radi i kao klijent (zahteva usluge od drugih komponenata) i kao server (pruža usluge drugim komponentama). Takođe, komponenta može i da zadrži samo jednu funkcionalnost, bilo klijentsku, ili serversku. Osim toga, komponenta može dinamički da menja svoju ulogu između navedene tri.

Prednost ove arhitekture je u tome što komponente mogu da koriste ne samo svoje, već i kapacitete kompletne mreže u kojoj se nalaze.

KLIJENT-SERVER ARHITEKTURA - NASTAVAK

Za jednostavne statičke Web aplikacije dovoljni su klijent-server, dok se za dinamičke i poslovne aplikacije mora posvetiti posebna pažnja o izboru odgovarajuće softverske arhitekture.

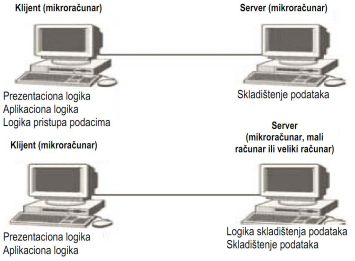
Nakon pažljive analize korisničkih zahteva, neophodno je doneti odluku o arhitekturi sistema. Odluka mora biti doneta na osnovu trenutnih potreba projekta i budućeg razvoja. Klijent-server je arhitektura gde su korisnik (klijent) i server odvojeni ili neravnopravni. Najočitiji je primer pregledanja Internet stranica. Korisnikov računar i Internet pregledač su klijent – oni zahtevaju, dok su računar i baza podataka koji čine web stranicu server – on opslužuje. Klijent je obično aktivan korisnik, koji šalje zahteve i čeka dok se isti ne ispune, dok je server pasivan, čeka na zahteve te ih ispunjava i šalje korisniku. Serveri su obično veoma jake mašine sa dobrim konfiguracijama i karakteristikama zbog toga što istovremeno moraju preraditi mnogo zahteva koji rastu iz dana u dan. Obično servere pogone i posebni operativni sistemi za razliku od običnih – klijent operativnih sistema, serverski operativni sistemi su u više segmenata bolji i sadrže naprednije opcije.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Klijen-server arhitektura jee razvijena kao: a) višenamenska, b) modularna infrastruktura, c) zasnovana na slanju i primanju poruka sa ciljem: unapređenja upotrebljivosti, fleksibilnosti, interoperabilnosti i skalabilnosti. Klijent je jednokorisnički računar. Sadrži interfejs, a omogućava obradu i skladištenje podataka. Poseduje mogućnost povezivanja na servere (prema potrebi i na druge klijente).

Server je višekorisnički računar. Omogućava rad sa deljenom bazom podataka, obradu podataka i servisiranje interfejsa. Poseduje mogućnost povezivanja sa klijentima i drugim serverima. Korisnicima izgleda kao da jedan računar obavlja celi posao.

Prednosti dvoslojne arhitekture klijent-server (slika 1) su u izolaciji promena pojedinom sloju, kvalitetnijoj (lakšoj) obradi i središnjem upravljanju integritetom podataka na serveru.

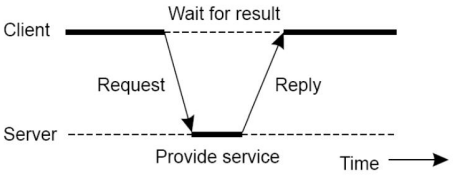


Slika 1.1 Dvoslojna arhitektura klijent-server

„MRŠAVI“ I „DEBELI“ KLIJENT

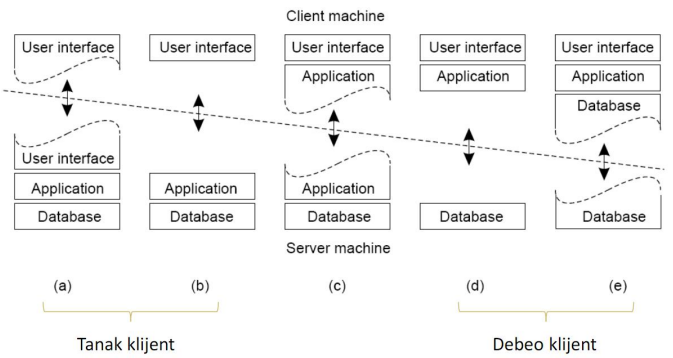
Tanki (mršavi) klijent je onaj klijent kod koga se logika podataka nalazi na serveru, a Debeli klijent je onaj klijent kod koga je integrisana logika podataka.

Debeli klijent je onaj klijent kod koga je integrisana logika podataka. Nema obrade podataka na serveru ili je obrada minimalna. Poseduje minimalnu ili nikakvu elastičnost na promenu poslovne politike. Prednosti debelog klijenta su: brzi početni razvoj aplikacije, veća samostalnost klijenta i rasterećenje glavnog računara (servera). Može imati lokalnu bazu podataka. Kao debeli klijenti mogu se koristiti jeftini računari sa snažnim procesorima. Nedostatak je da je poslovna logika integrisana na klijentu. Promena poslovne logike znači instalisanje nove verzije aplikacije na svim klijentima. Velika je mogućnost rada sa zastarelim podacima. Ako sa vremenom aplikacija postane spora (zbog količine podataka), treba promeniti sve klijente. Razvoj velike aplikacije sa vremenom postaje vrlo kompleksan (sav programski kod je na klijentu). Klasična šema funkcionisanja klijent-server arhitekture može se videti na Slici 2. Pritom, klijent šalje upit ka serveru, a server šalje odgovor klijentu, što iziskuje određeno vreme čekanja.



Slika 1.2 Upit-odgovor ponašanje u K-S arhitekturi

Tanki (mršavi) klijent je onaj klijent kod koga se logika podataka nalazi na serveru. Osnovna namena tankog klijenta je prikaz podataka. Većinom se koriste u poslovnim sistemima. Prednosti tankog klijenta su: promena poslovne logike ne znači obavezno i promenu u klijentskom delu aplikacije, promena poslovne logike može se obaviti centralizovano, računari ne moraju imati veliku procesorsku snagu, ukoliko sa vremenom obrada postane spora (zbog količine podataka) može se jednostavno povećati snaga središnjeg računara. Kao tanki klijent može se koristiti npr. web pretraživač (dobro definisano i svima dostupno). Nedostaci su: veliko opterećenje glavnog računara, a to znači skupi glavni računar. Grafički prikaz tankog i debelog klijenta može se videti na slici 3.



Slika 1.3 Grafički prikaz debelog i tankog klijenta

Poglavlje 2

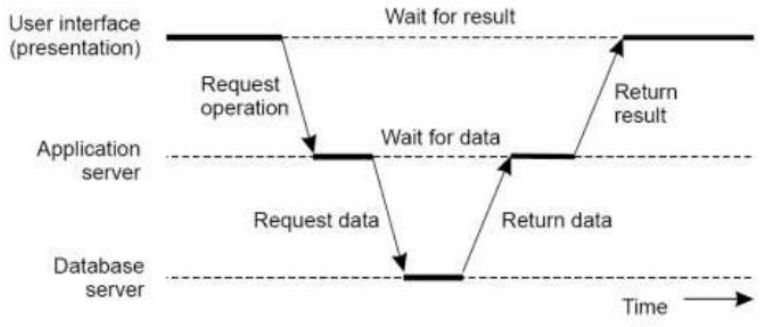
Troslojna i višeslojna arhitektura klijent-server

TROSLOJNA KLIJENT-SERVER ARHITEKTURA

Prednosti troslojne arhitekture su: bolja raspodela opterećenja, veća skalabilnost, odnosno mogućnost ekspanzije.

Kod troslojne arhitekture klijent-server (slika 1) distribucija baza podataka i poslovne logike je izvršena na zasebne servere, čime je dobijena arhitektura: server aplikacija + server baza podataka + klijent. Namena pojedinog sloja je sledeća:

- Server aplikacija obavlja upravljanje transakcijama "preuzetih" sa servera podataka. Deo ili čitava poslovna logika je "preuzeta" sa klijenta;
- Server baza podataka vrši upravljanje podacima;
- Klijent sadrži korisnički interfejs. Takođe sadrži deo poslovne logike, i to onaj deo koji se ne menja, ili logiku ličnog karaktera.



Slika 2.1 Komunikacija između slojeva kod višeslojne K-S

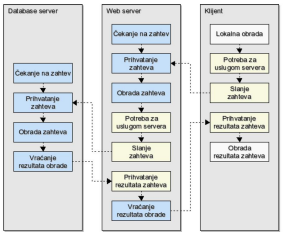
Prednosti troslojne arhitekture su: bolja raspodela opterećenja, veća skalabilnost, odnosno mogućnost ekspanzije (npr. povećanja broja korisnika, bez preopterećenja ili potrebe za promenom procedura). Nedostaci su: složeni (komplikovani) dizajn i razvoj, problem raspodele podataka, procesa, interfejsa, kao i veće opterećenje mreže.

Višeslojna arhitektura se koristi za razvoj složenih aplikacija. Programski kod se može podeliti u više nivoa, npr.:

- Kod na prezentacionom sloju - formi (GUI - Graphic User Interface);
- Kod u sloju poslovne logike (BLL - Business Logic Layer);

- Ko;d u sloju pristupa podacima (DAL - Data Access Layer);
- Kod na bazi podataka (stored procedure).

Detaljnija komunikacija između slojeva je prikazana na slici 2.



Slika 2.2 Detaljnija komunikacija između slojeva gde je Web server aplikacioni

BITNIJI KRITERIJUMI ZA ODABIR PRAVE KLIJENT-SERVER ARHITEKTURE

Često se vrši podela u tri nivoa, i to: klijent - GUI (u web aplikaciji to je web pretraživač), server aplikacija odnosno BLL (npr. web servis) i baza podataka (npr. SQL Server).

Kriterijumi za odabir prave klijent-server arhitekture mogu biti za:

Dvoslojna K-S arhitektura sa **tankim** klijentima

Aplikacije:

- Nasleđeni aplikativni sistemi gde je nepraktično i neisplativo odvojiti aplikativne obrade i upravljanje podacima.
- Računarsko-zahtevne aplikacije sa vrlo malo ili bez obrade podataka.
- Podacima bogate aplikacije (pretraživanje i upiti) sa veoma malo ili bez aplikativne obrade.

Dvoslojna K-S arhitektura sa **debelim** klijentima

Aplikacije:

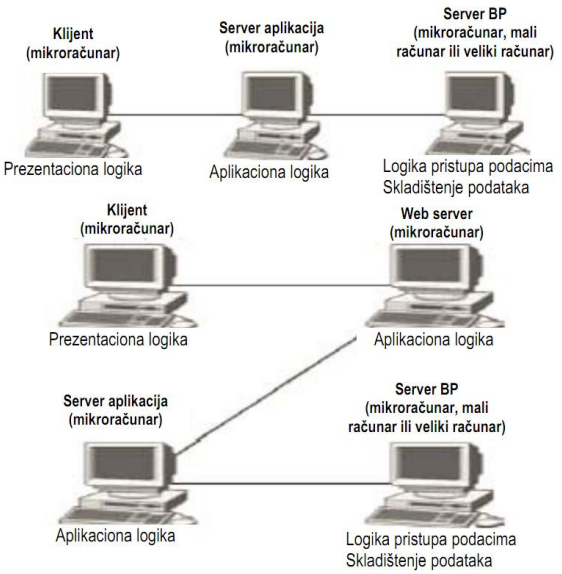
- Aplikacije gde se aplikativna obrada izvodi na klijentu sa COTS (Commercial Off-The Shelf Software) programskom podrškom.
- Aplikacije koje zahtevaju računarsko zahtevne obrade podataka (npr. vizualizacija podataka – interaktivno ili u izveštajima).
- Aplikacije sa relativno čvrstom krajnje - korisničkom funkcionalnošću, korišćene u sredini gde je dobro uspostavljeno upravljanje sistemom.

Troslojna ili višeslojna K-S arhitektura (Slika 3)

Aplikacije:

- *Aplikacije velikog opsega sa stotinama ili hiljadama klijenata*
- *Sistemi u kojima su i podaci i aplikacije promenljivi.*

- *Aplikacije u kojima se integrišu podaci iz višestrukih izvora*



Slika 2.3 Troslojna arhitektura klijent-server

Poglavlje 3

Softverski okvir - framework

SOFTVERSKI FRAMEWORK - OSNOVE

Softverski frejmvork-ovi nude rešenja za mnoge od problema sa kojima se svakodnevno susreću programeri koji se bave razvojem web softverskih sistema.

Obzirom da se ova lekcija bavi analizom razvoja posebne vrste softvera, tzv. Softverskih framework-a, u ovom odeljku detaljnije je objašnjeno šta taj pojam podrazumeva. Uopšteno posmatrano, softverski framework sadrži skup softverskih komponenti koje se mogu koristiti iz nekog spoljnog programa. Ono što je pri tom značajno istaći, jeste da framework ne obezbeđuje samo neki skup funkcionalnosti koje se koriste iz spoljnog programa (kao softverska biblioteka), već dizajn frameworka određuje arhitekturu programa koji ga koristi, odnosno koji je na njemu zasnovan.

Osnovna svrha softverskih framework-a je da ubrzaju i olakšaju razvoj određene vrste softvera, tako što obezbeđuju standardni osnovni dizajn i konkretna rešenja, za probleme iz određene oblasti (npr. sistema agenata), ili za neke specifične probleme u vezi sa razvojem određene vrste aplikacija (npr. Web aplikacija).

Frameworkobezbeđuje osnovni dizajn tako što sadrži skup povezanih klasa, koje zajedno u celini čine odgovarajuće rešenje za oblast ili problem od interesa za koje je framework namenjen. Pri tom, glavna karakteristika framework-a je da sadrži skup apstrakcija (interfejsa, apstraktnih i osnovnih klasa) i definiše tačke proširenja (extension points) koje se koriste za kreiranje proširenja i prilagođavanje framework-a potrebama konkretne primene odnosno aplikacije.

To praktično znači da framework određuje osnovni dizajn sistema, pri čemu se specifični elementi mogu redefinisati ili proširiti od strane korisničkog programa.

Framework obezbeđuje osnovne klase i interfejse koje korisnički softver nasleđuje, i implementira odgovarajuće komponente koje se uklapaju u postojeći dizajn frameworka. Uopšteno, može se reći da framework sadrži apstrakcije za višekratnu upotrebu (reusable abstractions), sa dobro definisanim programskim interfejson (API).

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

GLAVNE KARAKTERISTIKE FRAMEWORK-A

Korišćenje okvira (framework) predstavlja rešenje problema koji se javljaju tokom razvoja web softverskih sistema su čuvanje, izmena i prikazivanje velike količine podataka.

Većina softverskih okvira ima sledeće karakteristike:

1. Inverzija kontrole toka izvršavanja - IoC (inversion of control). Kod biblioteka i korisničkih aplikacija kontrolu toka izvršavanja definiše pozivaoc funkcije (ili korisnička aplikacija), dok kod framework-a osnovni tok izvršavanja definiše framework. Neki autori smatraju [Riehle, 2000] de je ovo ključna karakteristika frameworka po kojoj se razlikuju od softverskih biblioteka. Kod softverskih biblioteka, korisnički kôd poziva biblioteku, dok kod framework-a, framework poziva korisnički kod

2. Definisano standardno ponašanje (default behaviour) – framework treba da definiše standardno/podrazumevano (default) ponašanje u situaciji kada ga korisnik nije definisao. Ovo standardno ponašanje treba da bude nešto korisno što ima smisla u kontekstu određene primene/problema, a ne samo niz praznih operacija.

3. Proširivost (extensibility) - framework se može proširiti od strane korisnika redefinisanjem postojećih funkcionalnosti (primenom tehnike nasleđivanja i preklapanja/override metoda) ili dodavanjem specijalizovanih implementacija određenih komponenti frameworka (npr. interfejsa, apstraktnih klasa).

4. Nepromenljivi delovi framework-a: osnovna struktura i funkcionalnosti framework-a su nepromenljivi. Korisnici mogu da proširuju framework, ali ne mogu da menjaju njegov kod i osnovnu logiku.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ARHITEKTURA FRAMWORK-A

Softverski framework sastoj se od nepromenljivih (frozen spots) i promenljivih delova (hot spots, extension points).

Nepromenljivi delovi frejmvork-a definišu celokupnu arhitekturu sistema, odnosno osnovne komponente sistema i veze između njih. Promenljivi delovi (ili tačke proširenja) framework-a podrazumevaju mesta na kojima programeri koji koriste framework, mogu da dodaju sopstveni programski kod koji obezbeđuje funkcionalnosti potrebne za aplikaciju koja se razvija. Na taj način framework definiše mesta u arhitekturi na kojima je moguće izvršiti prilagođavanje odnosno proširenje. Ova mesta se formalno kreiraju eksplicitnim definisanjem potrebnog interfejsa komponente, ili kreiranjem klase koja faktički definiše interfejs, a čije se metode mogu redefinisati u izvedenim klasama.

Kada se konkretan softverski sistem razvija pomoću framework-a, programeri koriste tačke proširenja u skladu sa zahtevima sistema koji razvijaju. Osnovnu arhitekturu aplikacije i tok izvršavanja definiše framework, i framework poziva/koristi komponente koje programeri kreiraju obično nasleđivanjem apstraktnih klasa i interfejsa iz framework-a.

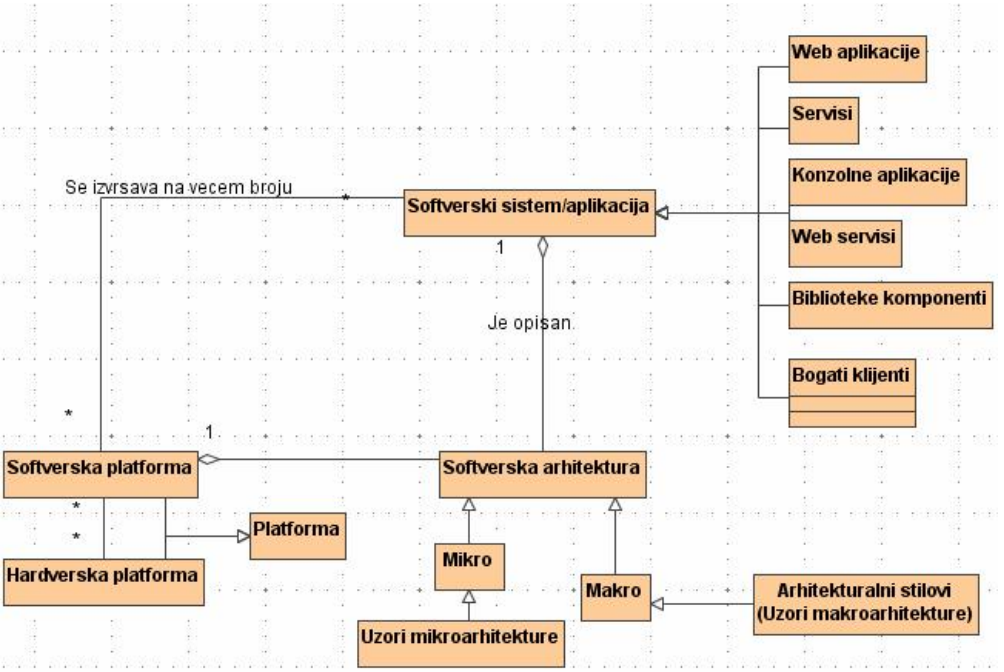
Softverski veb sistemi najčešće koriste tronivojsku softversku arhitekturu. Sastoji se iz tri nivoa i to: nivoa korisničkog interfejsa, nivoa aplikacione logike i nivoa podataka.

Na prezentacionom nivou najčešće se koristi Angular Okvir, na nivou aplikacione logike Spring Okvir, a na nivou za pristup podacima koristi se *Hibernate*. ***Angular*** okvir se koristi za izradu prezentacionog sloja zbog svoje jednostavnosti prilikom izrade korisničkog interfejsa i zato što se pomoću ovog okvira jasno odvaja nivo korisničkog interfejsa od nivoa aplikacione logike. Na nivou aplikacione logike koristi se Spring okvir pomoću koga jednostavno manipulišemo domenskim Java klasama. Hibernate okvir je izabran kao jedan od najpopularnijih okvira za pristup podacima. Hibernate okvir sadrži klase za upravljanje skladištenjem i prikazivanjem podataka kojima se veoma jednostavno manipuliše.

SOFTVERSKE ARHITEKTURE, SISTEMA, APLIKACIJA I PLATFORMI

Osnovnu arhitekturu aplikacije i tok izvršavanja definiše framework, i framework poziva/koristi komponente koje programeri kreiraju obično nasleđivanjem apstraktnih klasa i interfejsa.

Softverska arhitektura predstavlja opis arhitekturalno značajnih elemenata softverskog sistema, ali jednako koliko predstavlja opis, ona predstavlja i konkretne elemente. Preciznije rečeno, čak i kada softverska arhitektura nekog sistema nije formalno opisana, odnosno kada ne postoji njena formalna dokumentacija, realizovan softverski sistem ipak ima arhitekturu, jer ona predstavlja stub i osnovu postojanja svakog softverskog sistema kao što se sa na sledećoj slici vidi.



Slika 3.1 Odnos Softverske arhitekture, sistema, aplikacija i platformi

Poglavlje 4

SOA - Servisno Orijentisana Arhitektura

ISTORIJAT SOA (SERVICE ORIENTED ARCHITECTURE)

Moglo bi se reći da je SOA nastala evolutivnim procesom iz XML i Web Servis tehnologije, koje su svojom pojavom i filozofijom otvorile mnoge mogućnosti.

SOA (Service Oriented Architecture) je novi pristup pri projektovanju informacionih sistema od starih i već postojećih resursa. Sam cilj SOA-e je da odvoji poslovne procese od IT-a i da IS učini fleksibilnim i dovoljnim da ispuni sve zahteve modernih tokova poslovanja. Kao šro ćemo videti u ovoj lekciji, SOA nije potpuno nova ideja, ona predstavlja početak realizacije jedne ideje koja ima za cilj da spoji ceo IT sektor u jednu celinu i potčini ga interesima poslovanja.

Kao relativno mlada, u praktičnoj primeni, ova filozofija zahteva izvesnu edukaciju i upoznavanje sa terminologijom, tehnologijom, metodama i pristupima izrade ovakvog sistema.

Da bi bilo šta rekli o SOA istorijatu moramo se prvo osvrnuti na XML i Web Servise.

Moglo bi se reći da je SOA nastala evolutivnim procesom iz ove dve tehnologije, koje su svojom pojavom i filozofijom otvorile mnoge mogućnosti. XML je razvijen kao odgovor na eBiznis pokret, koji je nastao kasnih devedesetih.

Korišćenjem XML-a omogućeno je informacijama, koje se šalju internet protokolima, dati smisao i kontekst. XML nije samo poslužio za standardizaciju podataka već je kao jezik iskorišćen za razvoj dodadnih

standarda: XSD (XML Schema DefinitionLanguage) i XSLT (XSL Transformation Language). Pored XML jezika ova dva standarda su danas ključni delovi XML tehnologije.

Ova arhitektura predstavljanja podataka je bazni sloj SOA tehnologije. U okviru nje, XML određuje format i strukturu poruke koja se kreće između servisa. XSD šeme čuvaju integritet i validnost podataka poruke i XSLT se, kroz šematsko mapiranje koristi da bi se omogućila komunikacija između različitih formata podataka. Dave Winer, Don Box, Bob Atkinson i Mohsen Al-Ghosein su uz pomoć Microsoft-a 1998. godine dizajnirali SOAP(*Simple* Object Acces Protocol). W3C je 2000. godine usvojio SOAP standard. Prevashodno je zamišljen da ujedini RPC (Remote Procedure Call) komunikaciju. Ideja je bila da se podaci, koji razmenjuju komponente serijalizuju pomoću XMLa, transportuju i onda deserijalizuju u prvobitni format. Uskoro su korporacije i prodavci softvera uvideli ogroman potencijal za unapređenje eBiznis

tehnologije kreiranjem okvira za komunikaciju koji nema vlasništvo. Ovo je dovelo do ideje da se stvori distribuirana tehnologija bazirana na internet-u, koja ne koristi samo postojeće internet protokole već pruža standardizovani okvir komunikacije da bi se premostile razlike u komunikacije u okviru i među organizacijama.

DEFINISANJE SOA-E

SOA je arhitektura koja je tu da pospeši poslovanje razlaganjem poslovnih procesa na više servisa koji međusobno sarađuju da bi se izvršila neka aktivnost.

Ovaj koncept je nazvan WEB Servisi.Najbitniji deo WEB Servisa je javni interfejs koji on pruža, to je ono što pruža servisu identitet i omogućava njegovo pozivanje. Baš zato je jedna od prvih inicijativa, za podršku tehnologije WEB Servisa, bila WSDL (*Web Service Definition Language*). W3C je 2001. godine primila specifikaciju za WSDL i nastavila je da razvija ovaj standard. WSDL datoteka je centralni deo informacije koja opisuje veb servis.

Posledjni u nizu standarda, veb servisa prve generacije, jeUDDI (Universal Description Discovery and Integration) specifikacija. Razvijena je od strane UDDI.org, kasnije predata OASIS-u (*Organization* for the Advancement of Structured Information Standards), koji je nastavio da je razvija u saradnji sa UDDI.org. Ova specifikacija omogućava kreiranje standardizovanih registara za opis servisa, u okviru i van organizacije. ovo pruža mogućnost registracije veb servisa na centralizovanoj lokaciji gde ih mogu otkriti korisnici.

Uskoro su organizacije počele da shvataju da veb servisi, pored prilagođavanja postojećih distribuiranih aplikacija, mogu postati osnova za potpuno novu platformu, koja bi omogućila koncept deljenja poslovanja na seriju autonomnih servisa.

Tako je nastao koncept koji danas zovemo SOA.

Autori knjige, Service Oriented Architecture for Dummies,definišu SOA kao "softversku arhitekturu za pravljenje aplikacija koje implementiraju poslovne procese ili servise koristeći skup labavo povezanih crnih kutija kao komponente usklađene da pruže zadovoljavajući nivo usluge" (Hurwitz i saradnici, 2007).

Čak nam i WIKIPEDIA nudi svoju definiciju pojma SOA. "Možemo definisati SOA kao grupu servisa koji međusobno komuniciraju. Proces komunikacije može biti jednostavno razmenjivanje podataka ili koordinisanje dva ili više servisa u izvrašavanju neke aktivnosti. Međusobna komunikacija ukazuje na to da su nam neophodna sredstva za međusobno povezivanje dva ili više servisa".

Sve ove definicije su ispravne i manje više razumljive, sve ukazuju na donekle slične stvari. SOA je arhitektura koja je tu da pospeši poslovanje razlaganjem poslovnih procesa na više servisa koji međusobno sarađuju da bi se izvršila neka aktivnost. Međutim da bi se SOA razumela u potpunosti neophodno je bolje upoznavanje sa njenim komponentama, čemu služe, kako funkcionišu i na koji način su povezane.

OSNOVNE KOMPONENTE SOA

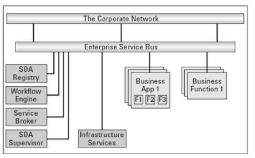
Osnovne komponente SOA arhitekture su:ESB (Enterprise Service Bus), SOA Registar, Workflow Engine, Servis Broker i SOA Supervizor.

Osnovne komponente SOA arhitekture su:

- *ESB (Enterprise Service Bus)*
- *SOA Registar*
- *Workflow Engine*
- *Servis Broker*
- *SOA Supervizor*

Svaka od ovih komponenti (Slika 1) ima određenu, nezavisnu od drugih komponenti, ulogu u sistemu kao i u saradnji sa ostalim delovima. ESB se stara o porukama koje se prenose između komponenti SOA sistema. SOA registar sadrži informacije o lokacijama SOA komponenti.

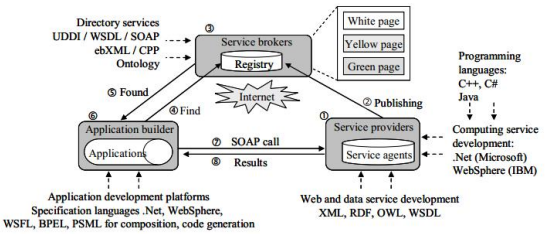
Workflow engine je tehnoloigja koja povezuje ljude sa ljudima, ljude sa procesima i procese sa procesima, dok servis broker povezuje servise sa servisima što na kraju omogućava rad poslovnih procesa. Uloga SOA supervizora je da se postara da ceo sistem funkcioniše skladno i predvidivo.



Slika 4.1 Komponente SOA arhitekture

U tradicionalnom razvoju softvera programer uzima zahteve i pretvara ih u specifikacije. Postoji nekoliko pristupa za prevod zahteva u operativni sistem uključujući model vodopada, inkrementi razvoj, objektno orijentisano računarstvo**,** računarstvo zasnovano na komponentama. Svaki proces ima svoje procese i tehnike.

Servisno orijentisano računarstvo je nova paradigma koja je evoluirala iz objektno orijentisanog računarstva i računarstva zasnovanog na komponentama deljenjem programera u tri grupe: oni koji prave aplikacija, brokeri servisa i programeri servisa (Slika 2). Odgovornost programera servisa je da razvije softverske servise koji su labavo povezani. Broker servisa objavljuje dostupne servise. Oni koji prave aplikacije nalaze dostupne servise pomoću brokera servisa i koriste ih za razvoj novih aplikacija. Razvoj aplikacija se vrši, pre, pomoću otkrivanja i kompozicijie nego uobicajenim dizajnom i kodiranjem.



Slika 4.2 Tipična web servis arhitektura

ESB (ENTERPRISE SERVICE BUS)

ESB predstavlja centar za komunikaciju servisa u okviru SOA sistema. Dizajniran je da funkcioniše kao posrednik između SOA komponenti, infrastrukturnih servisa i poslovnih procesa.

Pod SOR paradigmom, pojedini servisi su razvijeni nezavisno sa standardnim interfejsima. Oni su dostavljeni brokerima servisa. Programeri servisa traže, nalaze, povezuju, testiraju, potvrđuju i izvršavaju servise u svojim aplikacijama dinamički u toku rada. Takva servisno orijentisana arhitektura pruza programerima aplikacija maksimalnu fleksibilnost u biranju najboljeg brokera servisa i najbolje servise.

Provajderi veb servisa su razvili softevrske komponente da bi pruzili različite usluge korišćenjem programskih jezika kao sto su C++, C# i JAVA. Trenutni brokeri servisa koriste UDDI i ebXML protokole koji pružaju skup standardnih interfejsa za registraciju i objavljivanje web servisa. Za UDDI, informacije potrebne za registrovanje servisa obuhvataju: informacije bele strane: ime provajdera servisa, identifikaciju npr. DUNS broj; informacije zute strane: vrstu industrije, tip proizvoda i usluga, geografsku lokaciju; informacije zelene strane: tehnički detalj kako drugi web servisi mogu pristupiti servisima kao sto je API.

Jednom kada broker servisa pronađe servis u svom registru, on vraća podatke o servisu onima koji prave aplikaciju. Oni koji prave aplikacije koriste dostupne servise da bi sastavili traženu aplikaciju. Ovo je viši nivo programiranja korišćenjem modula servisa za izgradnju većih aplikacija.

Na ovaj način oni koji prave aplikacija ne moraju da znaju niži nivo programiranja. Broker servisa može testirati servis pre nego što ga registruje. Servis koji nisu uspeli se ne registruju.

ESB (Enterprise Service Bus)

ESB predstavlja centar za komunikaciju servisa u okviru SOA sistema. Dizajniran je da funkcioniše kao posrednik između SOA komponenti, infrastrukturnih servisa i poslovnih procesa. Povezuje se sa različitim tipovima srednjeg sloja (middleware) sistema, skladištima definicijama metapodataka, registrima i interfejsima aplikacija. Bitno je shvatiti da ESB nije hardverska komponenta već skup softverskih komponenata.

Postoje različite vrste ESB-a i razlikuju se po svojim mogućnostima. Svaki ESB ima apstraktni sloj koji je odgovoran za upravljanje porukama i na taj način omogućava spajanje i komunikaciju softverskih komponenti. Neki od njih mogu da funkcionišu sa raznim vrstama poruka od e-mail do SOAP-a, neki čak implementiraju i enkripciju. ESB može funkcionisati i kao servis broker, može posedovati i svoj registar pa čak može i preuzeti neke funkcije SOA supervizora. Ovo je dobar pokazatelj da se ESB razvijao nezavisno od SOA-e. Tako da je moguće imati ESB bez implementacije SOA sistema i obrnuto, naravno ovo je moguće samo u manjim SOA sistemima.

ESB SERVISI

ESB izvršava infrastrukturne poslove koji bi inače morali da budu implementirani u samu aplikaciju.

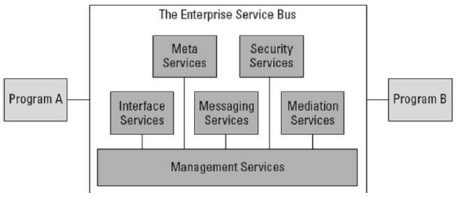
Servisi ESB-a

ESB izvršava infrastrukturne poslove koji bi inače morali da budu implementirani u samu aplikaciju. Servisi koje nudi ESB su (Slika 3):

- ***Messaging service*** **(Servis poruka)** – podržava širok spektar poruka, pruža inteligentno rutiranje na osnovu konteksta i garantuje isporuku. Takođe može kombinovati i deliti poruke.

- ***Managment service*** **(Upravljački servis)** – prati performanse i reaguje u slučaju velike *latencije,* implementira prioritete i globalna pravila aplikacijama i svim komponentama koje povezuje

- ***Interface service*** **(Interfejs servis)** – potvrđuje validnost poruka upoređivanjem sa XSD-om (XML Schema Definition) koja se nalazi u ESB registru. Podržava standarde veb servisa i pružaju adaptere za interfejse koji nemaju podršku za veb servise.



Slika 4.3 Povezvanje dva programa pomoću ESB-a

- ***Mediation service*** **(Posrednički servis)** – menja format poruke radi usklađivanja sa aplikacijom.

- ***Metadata service*** **(Servis meta podataka)** - povezan sa posredničkim servisom, takođe menja formate poruka koristeći se definicijama metapodataka koje se nalaze u registru.

- ***Security service*** **(Sigurnosni servis)** – vrši enkripciju podataka i uključuje standardizovani sigurnosni metod autorizacije, autentifikacije i pregleda svih aktivnosti ESB-a.

Servis poruka

Postoje šest osnovnih tipova poruka koje ESB koristi, oni se mogu kombinovati i mogu se primenjivati razna pravila da bi se dobili kompleksniji prenosi. Osnovni tipovi poruka su:

- *Point-to-point* (poruke od tačke-do-tačke) – Najjednostavniji vid poruka. Šalje je jedna osoba drugoj i nije neophodan odgovor.

- *Point-to-point request/response* (poruke od tačke-do-tačke sa zahtevom za odgovor) – Jedna osoba šalje poruku drugoj sa očekivanim odgovorom. ESB je svestan da je odgovor očekivan i poruke se šalju dok se komunikaicja ne završi, ovo predstavlja vrstu transakcije.

UPRAVLJAČKI SERVIS

Sve komponente ESB-a treba da funkcionišu zajedno da bi izvršavali svoje funkcije i neophodan je skup servisa koji kontroliše rad u okviru ESB-a.

- *Broadcast* (difuzni prenos poruka) – Jedna osoba šalje poruku ka više ljudi, slično kao prvi primer samo što poruka ide na više lokacija.

- *Broadcast request/response* (difuzni prenos poruka sa očekivanim odgovorom) – Jedna osoba šalje poruku ka više ljudi sa očekivanimodgovorom, transakcija se ne završava dok svaki primalac ne odgovori.

- *Publish subscribe* (objavljivanje poruka na oglasnoj tabli) – Poruka se postavlja na „oglasnu tablu“ i vide je samo „pretplatnici“.

- *Store and forward* (skladištenje i prosleđivanje poruka) – ESB skladišti poruke ako korisnik nije trenutno u mogućnosti da ih primi i kasnije ih prosleđuje korisniku.

Upravljački servis

Sve komponente ESB-a treba da funkcionišu zajedno da bi izvršavali svoje funkcije i neophodan je skup servisa koji kontroliše rad u okviru ESB-a. U „saobraćaju“ podataka uvek može doći do izvesnih grešaka, neki servis dolazi u koliziju sa drugim, transakcija prestaje bez ikakvog razloga ili nagli pad performansi. U ovakvim situacijama servisi moraju da se isprave i ponovo počnu sa radom.

Upravljački servis pre svega mora upravljati sobom, ovim pruža visoko pouzdani servis za sve poruke sa kojima radi. ESB je sposoban da balansira svoj rad upravljajući svim raspoloživim resursima i ima mogućnost da se oporavi od greške.

Interfejs servis

Pored upravljanja sobom i porukama glavni zadatak ESB-a je da omogući komunikaciju dva ili više programa. Programi međusobno pričaju prosleđivanjem informacija i komandi jedni drugima. Ova komunikaicja nije jednostavna i zahteva izvesna pravila komunikacije, pre pojave XML-a korišćene su druge metode. ESB povezuje Web servise koristeći SOAP i WSDL ali može podržati i druge načine povezivanja, CORBA (*Common Object Request Broker Architecture*) – standard koji je povezan sa objektno-orjentisanim programiranjem, JMS (*Java Messaging Service*) – standard povezan sa Java programerskim okruženjem.

Broj različitih interfejsa koje ESB pruža je veoma bitan zato što to daje mogućnost povezivanja većeg broja programa bez dodatnog kodiranja. Interfejs servisi su skup adaptera koji omogućavaju povezivanje programa na ESB. Bitno je razlikovati SOA interfejs servise koji su takođe adapteri ali daleko kompleksniji od ESB adaptera.

SIGURNOSNI SERVIS

U okviru sigurnosnog servisa ESB-a ne postoje sami sigurnosni mehanizmi nego ovaj servis pruža okvir za softver koji je posebno namenjen da se bavi pitanjem sigurnosti.

Posrednički servis

Računarski sistemi se sastoje od računarskog hardvera, operativnog softvera i lokalnog softvera. Računarska okruženja se razlikuju i povezati ih međusobno nije jednostavno. ESB može predstavljati posrednika između ovako različitih okruženja kao što su IBM mejnfrejm i Windows server. Ova dva okruženja se potpuno razlikuju ali ESB treba da ih poveže tako da program pod jednim okruženjem može da komunicira sa programom iz drugog. Ovo se izvršava tako što ESB prevodi iz jednog protokola u drugi. Takođe je neophodno posredovati u samoj sadržini poruke. Ako npr. Jedan program definiše datum u formatu dd/mm/yy i poveže se sa programom koji definiše datum u formatu mm/dd/yy neophodno je promeniti format datuma i to izvršava posrednički servis.

Servis meta podataka

Meta podaci predstavljaju informaciju o strukturi i značenju podataka. Ako npr. imamo podatke o mušterijama takođe je neophodno da znamo kako je definisana mušterija u sistemu.

Da bi programi komunicirali međusobno neophodno je da imaju istu definiciju podataka, sve se još više komplikuje kada je neohodno da naš softver komunicira sa softverom druge firme

koja je saradnik. Jedini način za prevazilaženje ovog problema je da čuvamo podatke o definiicjama svih podataka za svaki program sa kojim naši programi komuniciraju, upravo ovo izvršava servis meta podataka. ESB može imati svoj registar svih programa koje povezuje, može da koristi eksterni SOA registar ili može kombinovati ova dva pristupa.

Sigurnosni servis

Sigurnost je jako bitna stavka u bilo kom računarskom sistemu tako da se i ESB bavi pitanjem sigurnosti. Sigurnost nije tako bitna tema kada povezujemo programe u okviru jedne organizacije ali čim se naši programi povežu sa programima drugih organizacija ovo postaje bitan prioritet.

U okviru sigurnosnog servisa ESB-a ne postoje sami sigurnosni mehanizmi nego ovaj servis pruža okvir za softver koji je posebno namenjen da se bavi pitanjem sigurnosti. Bitne stavke koje se tiču sigurnosti u okviru ESB-a su:

- **Autentifikacija –** provera autentičnosti korisnika ili poruke

- **Autorizacija –** provera prava pristupa korisnika

- **Privatnost –** zaštita informacija od neautorizovanih korisnika

- **Integritet –** provera autentičnosti podataka.

KARAKTERISTIKE ESB-A

Federacija je skup ESB sistema koji pružaju uslugu na nivou celog poslovanja. Svaki ESB poseduje svoja pravila i polise ali ona potpadaju pod viši skup SOA pravila.

Iako sigurnosne servise posmatramo kao zasebne servise oni su samo specijalizovani posrednički servisi.

Karakteristike ESB-a

Kada kompanija implmentira ESB obično počinje sa jednim ESB-om i on je sasvim dovoljan da zadovolji trenutne potrebe kompanije. Kako raste SOA sistem tako je neophodno uvesti nove ESB sisteme i povezati ih federaciju. Federacija je skup ESB sistema koji pružaju uslugu na nivou celog poslovanja. Svaki ESB poseduje svoja pravila i polise ali ona potpadaju pod viši skup SOA pravila.

U suštini ESB pruža mogućnost povezivanja softverskih komponenti i servisa u fleksibilnu vezu (*loose coupling*) . Pod ovim se podrazumeva da ESB pruža sloj koji omogućava komponentama da pozivaju i koirste servise drugih komponenti na jednostavan način. ESB pruža mogućnost korišćenja već postojećih i starih aplikacija njihovim povezivanjem. Fleksibilna veza omogućava povezivanje neograničenog broja korisničkih interfejsa, koji mogu pozivati aplikacije i servise koji se nalaze na različitim paltformama, u zajedničku mrežu.

SOA Registar

SOA Registar je centralana veza koja omogućava otkrivanje poslovnih servisa i pruža opise svih servisa koji imaju referencu u tom registru. Svaki servis koji je zabeležen u registru je morao da prođe kroz verifikaciju od strane poslovnog i IT menadžmenta. U registru je takođe zabeležena istorija svakog servisa, ko je kreator servisa, ko ima mogućnost promene servisa, kako se koristi i ko ima pravo pristupa.

SOA Registar nije pasivni direktorijum već se menja u realnom vremenu(real-time registry), menja se u skladu sa promenama poslovnih pravila. Poseduje tri ključne funkcije:

- *Objavljivanje i pružanje mogućnosti otkrivanja servisa*
- *Sakupljanje i upravljanje meta podacima poslovnih servisa*
- ***Upravljanje korišćenjem poslovnih servisa***

Objavljivanje

SOA Registar pruža mogućnost pregleda raspoloživih servisa od strane korisnika. Servisi mogu biti od koristi i osobama koje rade u drugim organiazacijama kao što su mušterije ili poslovni partneri.

KOMPONENTE SOA REGISTRA

Svaka promena se mora kontrolisati da ne bi došlo do grešaka koje se mogu odraziti na funkcionalnost celog sistema.

Pored smeštanja definicija i opisa servisa, registar služi i za objavljivanje (*publish*) poslovnih servisa, na taj način im mogu pristupiti svi zainteresovani za usluge koje pruža neki servis.

Upravljanje meta-podacima

SOA Registar upravlja meta-podacima koji se odnose na neki poslovni servis. Metapodaci predstavljaju opise poslovnih servisa, poslovnih pravila vezanih za servis i pravila za upravljanje serivisima. Meta-podaci uključuju tehničke opise kako se jedan servis povezuje sa drugim.

Upravljanje korišćenjem

Neophodno je obezbediti da servisi koji su objavljeni budu u skladu sa poslovnim pravilima organizacije. Svaki servis koji je objavljen mora proći kroz sva pravila registra ili razvojnih delova SOA-e.

Komponente SOA Registra

Sve što je smešteno u registru je u formi meta-podataka i tiče se pravila upravljanjaservisima u okviru SOA okruženja. SOA Registar se sastoji od:

- **Komponente za opisivanje interfejsa Web servisa –** Koristi se UDDI (*Universal Description, Discovery, and Integration*)

standard koji podržavaju svi SOA Registri. Kreiran je pomoću XML tehnologije

- **Komponente za opisivanje ostalih tipova iterfejsa –** Pored interfejsa kreiranih XML-om moguće je koristiti već postojeće interfejse. Dovoljno je imati podatke o pravilima korišćenja postojećih interfejsa.

- **Definicije poslovnih procesa –** Za izvršenje nekog poslovnog procesa neophodno je povezati više komponenti. Da bi se one povezale na pravi način i da bi se izvršio poslovni proces neophodno je da imamo mapu i potpunu definiciju poslovnih procesa.

- **Pravila poslovnih procesa –** Svaki proces mora se izvršavati u okviru određenih pravila. Kako ne bi morali da za svaki proces definišemo posebno pravila ona su centralitovana i smeštena u SOA Registar. Ovo takođe olakšava pronalaženje pravila koja se tiču određenog poslovnog procesa.

- **Opisa nivoa performansi i dostupnosti servisa –** Skup pravila koja opisuju nivo preformansi i dostupnosti koje računarska mreža mora obezbediti za rad određenog servisa.

- **Pravila upravljanja –** Sadržaj registra mora se povinovati određenim pravilima upravljanja.

Poglavlje 5

Servis Broker

SERVIS BROKER JE KOMPONENTA

Servis broker je komponenta koja omogućava funkcionalnost veza između svih ostalih komponenti.

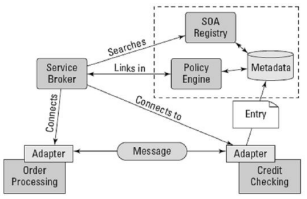
Servis Broker

Servis broker je komponenta koja omogućava funkcionalnost veza između svih ostalih komponenti. Spaja više komponenti u niz koje čine poslovni proces. Koristi informacije o komponentama koje nalazi u SOA registru i spaja ih pripremajući ih za rad w*orkflow engine-a*.

Servis broker započinje sve procese i kada završi sa spajanjem komponenti u poslovnom procesu prelazi na sledeći zadatak spajanja.

Servis broker je neophodna komponenta zato što je SOA sistem sačinjen od slabo povezanih (*loosely coupled*) komponenti koje se moraju povezati u smislenu celinu da bi se izvršio neki poslovni proces. On orkestrira rad komponenti praveći vezu između njih pomoću informacija iz SOA registra.

Na slici 1 je ilustrativno prikazano kako broker funkcioniše u saradnji sa registrom.



Slika 5.1 Rad servis brokera u saradnji sa SOA registrom

Slika prikazuje povezivanje servisa porudžbine sa servisom provere kreditne sposobnosti. Servis porudžbine, preko brokera koji koristi informacije iz registra o servisu provere kreditne sposobnosti, vrši zahtev za usluge servisa provere kreditne sposobnosti.

Servis broker koristi informacije iz registra kako da pronađe ova dva servisa ali i po kojim pravilima rade i kako da objedini dva skupa pravila u jednu funkcionalnu celinu. Na slici vidimo *policy engine* koji implementira pravila rada i koji je zapravo deo registra. Informacija o servisu provere kreditne sposobnosti je objavljena u registru, što se takođe vidi na slici.

SOA Supervizor

Sistem slabo povezanih komponenti (*loose coupling*) ne može biti toliko efikasan i pružiti nivo performansi kao sistem čvrsto povezanih komponenti (*tight coupling*). Zato je uloga SOA supervizora toliko bitna, on omogućava prihvatljiv nivo usluga i performansi servisa.

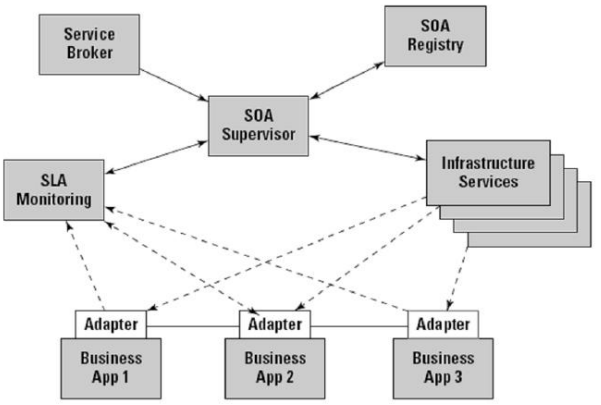
SOA SUPERVIZOR

SOA tehnologija je još uvek mlada, većina firmi još uvek je ne implementira u potpunosti, tako da rad SOA supervizora nije zahtevan toliko koliko će biti u budućnosti.

Takođe je bitno napomenuti da SOA supervizor mora biti aktivan dok god je neki servis aktivan i vrši se neki poslovni proces.

SOA tehnologija je još uvek mlada, većina firmi još uvek je ne implementira u potpunosti, tako da rad SOA supervizora nije zahtevan toliko koliko će biti u budućnosti. Kakokompanija sve više implementira SOA sistem tako će se povećavati i zahtevi koje SOA supervizor mora da ispuni da bi se održao prihvatljiv nivo performansi.

Slika 2 prikazuje rad superizora u saradnji sa ostalim komponentama, ovakav sistem se ne može još uvek naći u praktičnoj primeni ali je to ono čemu se teži.



Slika 5.2 Rad SOA supervizora u SOA okruženju

Ceo proces započinje servis broker slanjem poruke SOA supervizoru da je spojio određene servise i tako omogućio početak nekog poslovnog procesa. SOA supervizor se konsultuje sa registrom oko detalja vezanih za poslovni proces da bi mogao da postavi softver za nadgledanje svih neophodnih komponenti. Ovo nadgledanje izvršava SLA (*Service Level Agreement*) monitoring komponenta koja preko lokalnih adaptera aplikacija dobija izveštaje o nivou performansi. Izveštaji se prosleđuju SOA supervizoru koji ih prikazuje na konzoli da bi se mogli pratiti u realnom vremenu.

Kada se pojavi problem SOAsupervizor obaveštava infrastrukturne servise, koji bi trebali da reše problem. Ako ovi servisi nisu umogućnosti da reše problem obaveštava se osoblje o problemu koji se pojavio.

Stvari u realnosti izgledaju dosta drugačije nego što su prikazane na dijagramu. Većina kompanija nije za sada u mogućnosti da obezbedi rad SOA supervizora kako je prikazano

na slici. Ovo ne znači da one nemaju funkcionalan SOA sistem, samo da im za sada to nije neophodno zato što nisu implementirali SOA-u u potpunosti.

BEZBEDNOST SOA SISTEMA

Bitna stavka svakog informacionog sistema je bezbednost, pogotovo kada se radi o otvorenom informacionom sistemu koji ima veliki broj korisnika i koji pruža veliki broj usluga.

Ali vremenom će se SOA sistem širiti i da bi se postigla ovakva funkcionalnost SOA supervizora pre svega je potrebno obezbediti:

- **Dobro definisane nivoe prihvatljivih performansi – SLA(*Service Level Agreement*)**

- **Potpuno funkcionalan infrastrukturni softver**

- **Mape poslovnih procesa**

- **Popis hrdverske opreme i softvera koji se koristi**

Bezbednost SOA sistema

Bitna stavka svakog informacionog sistema je bezbednost, pogotovo kada se radi o otvorenom informacionom sistemu koji ima veliki broj korisnika i koji pruža veliki broj usluga.

Ovo pravilo takođe važi i za SOA sisteme. Nijedna tehnologija ne nudi savršeno rešenje i da bi se postigao zadovoljavajući nivo bezbednosti, SOA sitema, neophodno je razumeti osnovne principe sigurnosti informacija i razumeti dizajn i arhitekturu SOA sistema. Ključno za dobar sistem bezbednosti je da se sigurnosna strategija i plan razvoja usvoje u ranim fazama implementacije SOA sistema. Dobri sigurnosni sistemi omogućavaju poslovnim aplikacijama da ispune sva očekivanja

organizacije uz implementaciju osnovnih bezbednosnih ciljeva, kao što su:

- Autentifikacija

- Autorizacija

- Federativni identitet

- Privatnost

- Integritet - Nemogućnost poricanja(non-repudiation) – Garancija da pošiljalac poruke ne može moreći da je poslao poruku i da primalac nemože poreći da je primio poruku - Preglednost

- Dostupnost

Poglavlje 6

Vežba 2

UVOD U VEŽBU - WEB SERVISI U APLIKACIJAMA

Cilj ove sekcije je da upozna studenta šta će raditi u ovoj vežbi

Šta ćemo naučiti?

Na ovom času ćemo naučiti kako se koriste web servisi u web aplikacijama. Koji su to tipovi servisa koji postoje. Studenti će proći kroz par primera korišćenja web servisa i provežbaće iste.

Zašto ovo učimo?

Servisno orijentisana arhitektura (SOA) je jedna od najzastupljenih arhitektura danas. Back end sistema često prestavljaju REST servisi koji koriste XML ili JSON kao tip podataka dok se za front end web aplikacija koriste front end okviri kao što su:

- Angular
- Ember JS
- React

Danas se SOA arhitektura koristi zbog lake centralizacije serverske strane aplikacija pa klijentska strana mogu biti mobilna aplikacija, aplikacija za sat, tv, aplikacija računar ili web aplikacija.

PRIMERI GET WEB SERVISA

Cilj ove sekcije je da da primer studentu šta je to GET servis i kako radi

GET web servisi su web servisi koji se pozivaju sa HTTP Get metodom. Pošto koristimo HTTP Get metodu moguće je da iz samog web čitača vidimo izlaz to je odgovor od samog web servisa. Jedan primer ovakvog servisa je:

http://www.thomas-bayer.com/sqlrest/

Kada otvorite ovaj web servis preko web čitača pokrenuće te HTTP GET metodu nad ovim linkom. Kao dogovor dobićete XML. U ovom slučaju dobijate sve postojeće resurse Thomas-bayer web servisa tj listu servisa koje možemo pozvati.

```
<resource xmlns:xlink="http://www.w3.org/1999/xlink">
<CUSTOMERList xlink:href="http://www.thomas-bayer.com/sqlrest/
CUSTOMER/">CUSTOMER</CUSTOMERList>
<INVOICEList xlink:href="http://www.thomas-bayer.com/sqlrest/
INVOICE/">INVOICE</INVOICEList>
<ITEMList xlink:href="http://www.thomas-bayer.com/sqlrest/ITEM/">ITEM</ITEMList>
<PRODUCTList xlink:href="http://www.thomas-bayer.com/sqlrest/
```

```
PRODUCT/">PRODUCT</PRODUCTList>
</resource>
```

Veb servisi su napravljeni tako da imaju raspored sličan hijerarhiji fajlova. Postoji osnovni folder u kome postoje drugi folderi koji predstavljaju različite vrste fajlove, svaki od fajlova predstavlja jedan entitet kome se može pristupiti. Svaki fajl predstavlja jednu krajnju tačku (end-point). Krajnja tačka predstavlja adresu koja vraća neki odgovor.
Iz prethodnog primera moguće je zaključiti da postoje 4 entiteta a to su:
CUSTOMER, INVOICE, ITEM, PRODUCT
Adresa svakog od ovih je osnovna adresa/naziv entieta. Primer:
http://www.thomas-bayer.com/sqlrest/CUSTOMER/
http://www.thomas-bayer.com/sqlrest/INVOICE/
http://www.thomas-bayer.com/sqlrest/ITEM/
http://www.thomas-bayer.com/sqlrest/PRODUCT/
Svaki od ovih krajnjih tačaka sa GET zahtevom vratiće nam listu entiteta tipa koji smo zahtevali pa će adresa http://www.thomas-bayer.com/sqlrest/CUSTOMER/ vratiti sledeću listu:

```
<CUSTOMERList xmlns:xlink="http://www.w3.org/1999/xlink">
<CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/0/">0</CUSTOMER>
<CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/1/">1</CUSTOMER>
<CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/2/">2</CUSTOMER>
<CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/4/">4</CUSTOMER>
</CUSTOMERList>
```

Kada bi hteli da pristupimo informacijama o nekom od kupaca preuzeli bi link sa atributa xlink:href u Customer tagu u XML-u. Npr: http://www.thomas-bayer.com/sqlrest/CUSTOMER/3/. Odgovor na ovaj link biće upravo informacije o ovom kupcu:

```
<CUSTOMER xmlns:xlink="http://www.w3.org/1999/xlink">
<ID>3</ID>
<FIRSTNAME>Michael</FIRSTNAME>
<LASTNAME>Clancy</LASTNAME>
<STREET>542 Upland Pl.</STREET>
<CITY>San Francisco</CITY>
</CUSTOMER>
```

PARSIRANJE GET SERVISA KROZ PROGRAMSKI JEZIK JAVA

Cilje ove sekcije je da pokaže studentu kako se poziva GET servis kroz Javu

Kako bi iz Jave lako pristupali i upravljali XML servisima možemo korsititi Jsoup biblioteku. Ova biblioteka omogućava lako upravljanje sa XML i HTML fajlovima. Jsoup možete preuzeti sa sledeće strane:
http://jsoup.org/download
Kako bi dodali biblioteku Jsoup u NetBeans razvojno okruženje potrebno je da napravite u Java projektu folder libs prebacite jar fajl u taj folder a potom dodate biblioteku sledećim koracima:

1. U projektu sa leve strane pronađite Libraries folder
2. Kliknite desni klik na ovaj folder i idite na opciju Add Jar
3. Selektujte Jsoup JAR fajl I kliknite open

Primer izvlačenje podataka sa web servisa iz prethodnog primera u Javi:

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        new Main();
    }

    public Main() {
        try {
            //Konektujemo se na servis i preuzimamo ga kao dokument
            Document doc = Jsoup.connect("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/3/").get();
            Elements elements = doc.select("customer"); //Selektujemo tag customer
unutar XML-a
            for (Element element : elements) // Za svakog kupca i kupcima
ispisujemo njegovo ime, prezime, ulicu i grad
            {
                String ime = element.select("firstname").get(0).text();
                String prezime = element.select("lastname").get(0).text();
                String adresa = element.select("street").get(0).text();
                String grad = element.select("city").get(0).text();
                System.out.println(ime);
                System.out.println(prezime);
                System.out.println(adresa);
                System.out.println(grad);
            }
        } catch (IOException ex) {
        }
    }
}
```

Zadatak 1
Koristeći Jsoup preuzeti informacije o svim postojećim kupcima sa liste kupaca na sajtu:
http://www.thomas-bayer.com/sqlrest/CUSTOMER/

JSON GET servis
Pored XML restful servisa postoje i JSON web servisi. JSON je Java Serialized Object Notation ili notacija seriazacije java objekata u tekstualnom obliku. Kao i XML Json se lako može parsirati ili pretvoriti u Java objekat ili objekat nekog drugog objketno-orijetisanog programskog jezika. Primer JSON servisa je:
https://www.w3schools.com/js/tryit.asp?filename=tryjson_parse

Ovaj servis će na GET zahtev vratiti JSON sa podacima o tutorijalima na sajtu W3C.

```
[
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
},
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "jQuery Tutorial",
"url": "http://www.w3schools.com/jquery/default.asp"
},
{
"display": "JSON Tutorial",
"url": "http://www.w3schools.com/json/default.asp"
},
{
"display": "AJAX Tutorial",
"url": "http://www.w3schools.com/ajax/default.asp"
},
{
"display": "SQL Tutorial",
"url": "http://www.w3schools.com/sql/default.asp"
},
{
"display": "PHP Tutorial",
"url": "http://www.w3schools.com/php/default.asp"
},
{
"display": "XML Tutorial",
"url": "http://www.w3schools.com/xml/default.asp"
}
]
```

PARSIRANJE GET SERVISA KROZ PROGRAMSKI JEZIK JAVA U FORMATU JSON

Cilje ove sekcije je da pokaže studentu kako se poziva GET servis kroz Javu i parsira JSON odgovor

Ovaj JSON se može parsirati na različite načine. Jedan je da za ovaj JSON napravite Java klase a potom da ove podatke parsirate u tu klasu. JSON lako možete pretvoriti u java klase koristeći sledeći alat:
https://codebeautify.org/json-to-java-converter
Klase dobijene od ovog JSON-a su:

```
public class RootObject
{
  private String display;
  public String getDisplay() { return this.display; }
  public void setDisplay(String display) { this.display = display; }
  private String url;
  public String getUrl() { return this.url; }
  public void setUrl(String url) { this.url = url; }
}
```

Kada imamo klasu koju možemo koristiti za parsiranje možemo pristupiti parsiranju iz Jave. Kako bi lakše parsirali JSON iz jave potrebna nam je GSON bibloteka koju možete preuzeti sa sajta:
https://mvnrepository.com/artifact/com.google.code.gson/gson
JAR fajl dodati isto kao što ste dodali i Jsoup.
Primer parsiranja u Javi:

```
public class Proba {

    public static void main(String[] args) {
        new Proba();
    }
    public Proba() {
        try {
            URL url = new URL("http://www.w3schools.com/json/myTutorials.txt");
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "application/json");
            if (conn.getResponseCode() != 200) {
                throw new RuntimeException("Pokusaj je uspeo : HTTP error  : "
                        + conn.getResponseCode());
            }
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    (conn.getInputStream())));
            String json = "";
            String output;
            while ((output = br.readLine()) != null) {
                json += output;
            }
            conn.disconnect();
            Gson gson = new Gson();
            ArrayList<OneUrl> lista = gson.fromJson(json, new
TypeToken<ArrayList<OneUrl>>() {
            }.getType());
            for (OneUrl urlone : lista) {
                System.out.println("ime je " + urlone.getDisplay());
                System.out.println("url je " + urlone.getUrl());
            }
        } catch (MalformedURLException e) {
        } catch (IOException e) {
        }
```

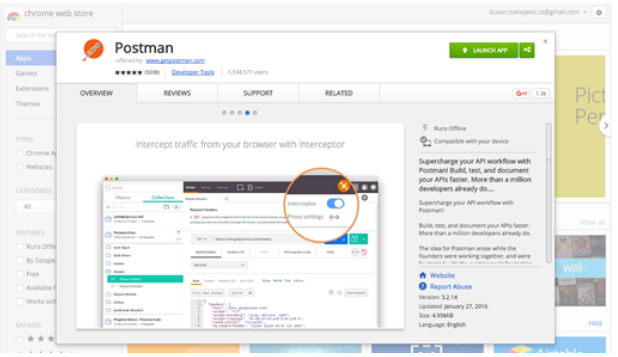
```
    }
}
```

Zadatak 2
Koristeći GSON preuzeti informacije o trenutnom vremenu i datuma sa sajta:
http://date.jsontest.com/

JSON POSTMAN GET POST PUT DELETE

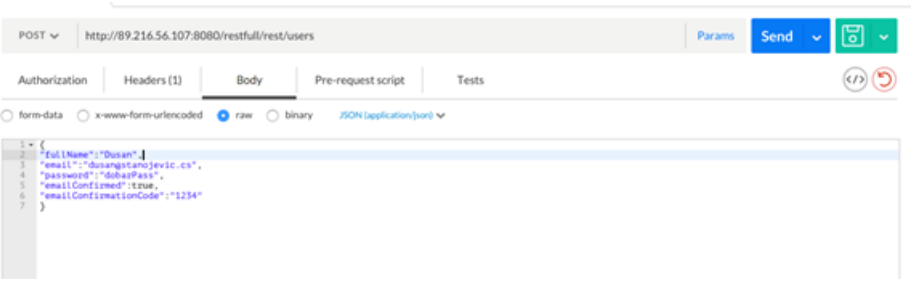
Kroz ovu sekciju student će naučiti da poziva HTTP POST kroz POSTMAN

Za naredni primer koristićemo Postman da bismo demonstrirali pozive svih http metoda sa restful json servisima.
Postman je ekstenzija za Google Chrome koju možete skinuti sa lokacije:
https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=sr



Slika 6.1 Preuzimanje POSTMan-a

Nas endpoint koji pozivamo za rad sa korisnicima je na lokaciji /users na rest servisu dostupnom na http://89.216.56.107:8080/restfull/rest. Kako bi dodali korisnika potrebno je da prosledimo informacije o korisniku na adresu. Za dodavanje na sistem uvek se koristi POST zahtev,



Slika 6.2 Primer post servisa

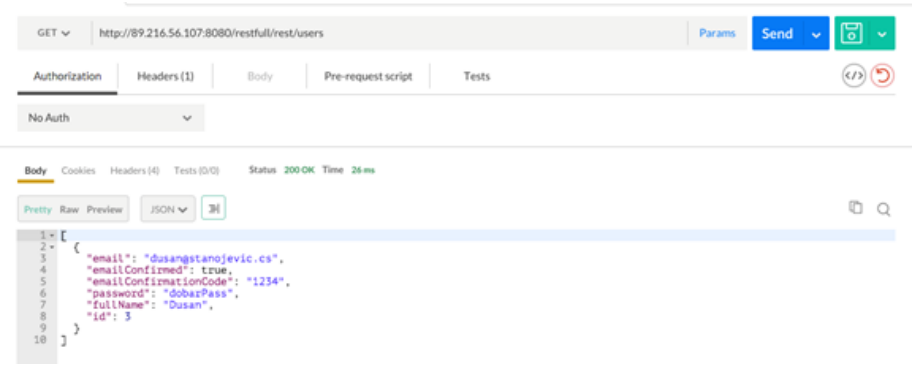
Na slici 2 vidimo da je u POSTMan aplikaciji obeležen POST kao tip zahteva, da je upisana adresa na koju se zahtev šalje i da je kao tip podataka koji se šalju stavljen raw i application/json. Svaki korisnik je opisan sa više atributa, ekvavilent ovog JSON-a kao java klasa bio bi:

```
public class User {
    private String fullName;
    private String email;
    private String password;
    private boolean emailConfirmed;
    private String emailConfirmationCode;
}
```

JSON POSTMAN GET

Kroz ovu sekciju student će naučiti da poziva HTTP GET kroz POSTMAN

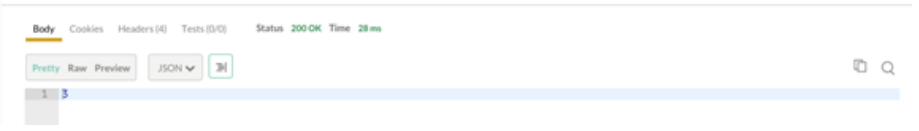
Pozivom istog endpoint-a GET zahtevom nam vraća sve korisnike dosada upisane u bazu:



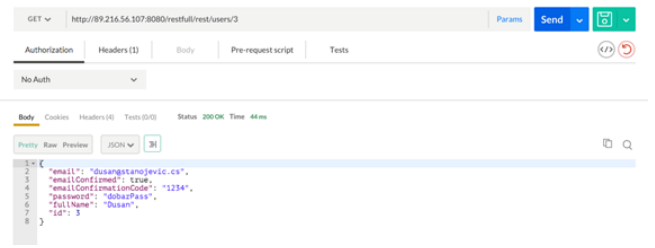
Slika 6.3 Svi korisnici sistema

Sa ove slike možete videti da se radi o nizu korisnika, pošto smo dodali samo jednog korisnika imamo samo jedan element u ovom nizu.

Ukoliko prosledimo broj id-a sačuvanog korisnika kao kontekst parametar /users endpoint-u dobićemo tog konkretnog korisnika. Ukoliko korisnik tog id-a ne postoji dobićemo grešku.



Slika 6.4 Greška



Slika 6.5 Informacije o korisniku

Zadatak 1
Skinite i instalirajte Postman kao google chrome ekstenziju.
Zadatak 2
Istestirajte 3 osnovne operacije sa korisnicima koje su prethodno **prikazane.**
Zadatak 3
Ukoliko je Java klasa koja predstavlja jedno pitanje:

```
public class Question {
    private long userId;
    private String title;
    private String text;
    private String correctAnswer;
}
```

I ukoliko je end point za servise vezane za pitanje
http://89.216.56.107:8080/restfull/rest/questions
Korišćenjem Postman ekstenzije za Google Chrome prikažite čitanje svih pitanja sa servisa, dodavanja pitanja u bazu predstavljanu ovim servisom i prikazivanje određenog pitanja po njegovom id-u.

JAVA PRIMERI POST I GET

Kroz ovu sekciju student će videti prethodne POSTMAN primere implementirane kroz Javu

Iz prethodnih primera možete videti zvanje web servisa putem POSTman aplikacije ovo takođe može biti urađeno i iz jave.

Primer 1: Kreiranje korisnika putem Jave i Gson biblioteke

Klasa **User**:

```
public class User {
    private String fullName;
    private String email;
    private String password;
    private boolean emailConfirmed;
    private String emailConfirmationCode;

    public String getFullName() {
        return fullName;
    }
    public void setFullName(String fullName) {
        this.fullName = fullName;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
```

```
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public boolean isEmailConfirmed() {
        return emailConfirmed;
    }
    public void setEmailConfirmed(boolean emailConfirmed) {
        this.emailConfirmed = emailConfirmed;
    }
    public String getEmailConfirmationCode() {
        return emailConfirmationCode;
    }
    public void setEmailConfirmationCode(String emailConfirmationCode) {
        this.emailConfirmationCode = emailConfirmationCode;
    }
}
```

Ova klasa mapira JSON iz primera koji je dat za POST preko POSTman aplikacije. Koristeći ovu klasu možemo pozvati ovaj servis putem Jave:

```
User u = new User();
u.setEmail("D@d.com");
u.setEmailConfirmationCode("123");
u.setFullName("Dusan");
u.setPassword("1234");
try {
    URL url = new URL("http://89.216.56.107:8080/restfull/rest/users");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Accept", "application/json");
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Greška : HTTP error: "
+conn.getResponseCode());
    }
    PrintWriter pw = new PrintWriter(conn.getOutputStream());
    pw.print(new Gson().toJson(u));
    pw.close();
    pw.flush();

    BufferedReader br = new BufferedReader(new InputStreamReader(
    (conn.getInputStream())));
    String output;
    while ((output = br.readLine()) != null) {
        System.out.println(output);
    }
    conn.disconnect();
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
```

```
    e.printStackTrace();
}
```

Primer 2: Get svih korisnika preko web servisa i Jave

```
try {
    URL url = new URL("http://89.216.56.107:8080/restfull/rest/users");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.setRequestProperty("Accept", "application/json");
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : "
                + conn.getResponseCode());
    }

    BufferedReader br = new BufferedReader(new InputStreamReader(
    (conn.getInputStream())));
    String output;
    while ((output = br.readLine()) != null) {
        System.out.println(output);
    }
    conn.disconnect();
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Primer 3: Get specifičnog korisnika preko identifikatora

```
public static void get(String id) {
        try {
            URL url = new URL("http://89.216.56.107:8080/restfull/rest/
users/"+id);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "application/json");
            if (conn.getResponseCode() != 200) {
                throw new RuntimeException("Failed : HTTP error code :
"
                        + conn.getResponseCode());
            }

            BufferedReader br = new BufferedReader(new InputStreamReader(
            (conn.getInputStream())));
            String output;
            System.out.println("Output from Server .... \n");

            while ((output = br.readLine()) != null) {
                System.out.println(output);
            }
            conn.disconnect();
```

```
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

POSTMAN - VIDEO TUTORIAL

Video tutorial za POSTman

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER ZA SAMOSTALAN RAD - INDIVIDUALNA VEŽBA

Pokušajte uraditi primere upotrebe:GET web servisa, JSON PostMan GET POST PUT DELETE i dr.

Odaberite neku od veb aplikacija i:

1. Uradite primere GET web servisa

2. Uradite primere JSON PostMan GET POST PUT DELETE

3. Kreiranje korisnika putem Jave i Gson biblioteke.

Poglavlje 7

Domaći zadatak 2

DOMAĆI ZADATAK

Cilj domaćeg zadatka je da student provežba naučeno na vežbama

Uraditi domaći zadatak po sledećim zahtevima:

- Za domaći zadatak broj 3 potrebno je da student pronađe otvoreni web servis
- Prikazati rad *POST* i *GET* http metoda kroz Javu i Postman.

Domaći zadatak smestiti u zip datoteku sa imenom *IT255-DZXX-ime-prezime-brojindeksa.zip* i postaviti na *GitHub* kao nov *commit*.

Obaveštenje o urađenom domaćem zadatku poslati na mejl predmetnom asistentu.

Zaključak

VEB ARHITEKTURE

Savladavanjem ove lekcije student razume benefite dizajniranja softverske veb arhitekture, kao i da razume razlike između veb arhitektura pout SOA, MVC i sličnih.

Lekcija je nastavila vođenje studenata ka sticanju znanja neophodnog za razvoj veb sistema i pri tome je akcenat stavila na sledeće teme:

- klijent - server arhitektura (dvoslojna i troslojna);
- softverski okvir;
- SOA - servisno orijentisana arhitektura;
- brokeri servisa.

Savladavanjem ove lekcije student sada razumeju benefite dizajniranja softverske veb arhitekture, kao i jasne razlike između veb arhitektura pout SOA, MVC i sličnih.

LITERATURA ZA LEKCIJU 02

U izradi ove lekcije korišćena je navedena literatura.

Obavezna literatura:

1. Patterns of Enterprise Application Architecture, Martin Fowler, Addison-Wesley, 2005.
2. Web Engineering: The Discipline of Systematic Development of Web Applications, Edited by Gerti Kappel, Birgit Proll, Siegfried Reich, Werner Retschitzegger, ISBN: 3-89864-234-8, Translation copyright 2006 by John Wiley & Sons Ltd.

Dopunska literatura:

1. Service-oriened Architecture, Concept, Technology, and Design, autora T. Erl u izdanju Pretince Hall, 2005, ISBN 0-13-185858-0
2. Ian Gorton, Essential Software Architecture, Second Edition, Springer-Verlag Berlin Heidelberg 2006, 2011.

Veb lokacije:

1. https://www.w3schools.com/angular/
2. https://www.packtpub.com/