XML

1. Šta je markup?

Markup je oznaka koja opisuje deo sadržaja. On daje upustvo kako neki sadržaj obraditi ili prikazati. Obično se ne prikazuje eksiplictino krajnjem čitaocu - krajnji čitaoc ga jedino uočava kroz izgled pojedinih grafičkih elemenata, ukoliko je makrup korišten za definisanje prezentacije dokumenta.

2. Ko može definisati markup? Prednosti otvorenih standarda?

Markup-e mogu definisati firme (Microsoft-ov doc, Adobe-ov pdf) ili tela za standardizaciju (otvoreni standardi kao XML i HTML). Rad sa više markup-a istovremeno je zahtevan. Velike su šanse da dođe do problema u konverziji. Zahteva ručne korekcije i tehnički je komplikovan. To je prednost otvorenih standarda za markup. Oni bi trebalo da pojednostave razmenu podatka između različitih sistema, različitog softvera, različitih učesnika u proizvodnji sadržaja.

3. Šta je nejasan markup? Osnovni razlozi za pojavu nejasnog markup-a?

Nejasan markup je onaj markup koji je nedorečen ili višeznačan. Na primer, oslanja se na vizuelni izgled, nedostaju mu tagovi, nema dobro definisanu strukturu. Osnovni razlozi za pojavu nejasnih markup-a su fokus na prikaz dokumenta i ograničenost na sopstveni, zatvoreni sistem. Otvoreni standardi zasnivaju se na konceptima markiranja strukture i markiranja značenja.

4. Istorija standarda za strukturni markup?

- GML (Generic Markup Language) - razvijen u okviru IBM, namenjen za definisanje različitih tipova dokumenata. Predstavljao je dokument kao hijerarhiju elemenata.
- SGML (Standard Generalized Markup Language) - nastao kao otvoreni standard baziran na GML. Fleksibilan i imao široko područje primene.
- HTML (HyperText Markup Language) - razvijen u CERN-u kao jednostavan, fiksan skup tagova. Kasnije definisan kao SGML gramatika.
- XML (Extensible Markup Language) - Definisan kao podskup SGML koji je znatno jednostavnije implementirati.

5. Šta je dokument? Navesti tri dimenzije dokumenta?

Dokument je osnovni nosilac informacija ili osnovna jedinica razmene informacija. Osnovne dimenzije dokumenta su sadržaj, struktura i prezentacija.

6. Šta je XML? Koji su osnovni ciljevi XML-a?

XML je jezik za definisanje markup jezika, tj. metajezik – ne nudi tagove unapred, već ih korisnik sam definiše. XML podrazumeva familiju standarda (XSL – vizuelizacija, XPath – struktura, XLink – povezivanje dokumenata, XQuery - pretraživanje dokumenata po sadržaju i strukturi). Osnovni ciljevi XML-a:

- da odgovara upotrebi na Internetu
- mogućnost upotrebe od različitih aplikacija
- jednosmerna kompatibilnost sa SGML
- da se programi za obradu XML-a lako pišu
- jednoznačnost prilikom obrade XML dokumenta
- čitki i rezonski jasni dokumenti

- dizajn XML-a se vrši brzo
- dizajn XML-a je formalan i konzistentan
- kreiranje XML dokumenta je lako
- konciznost je od minimalne važnosti

7. **Šta je elementa, šta tag, a šta atribut?**

Element je čvor u hijerahijskoj strukturi dokumenta. Tag je tekstualna oznaka za početak i kraj elementa. Sadržaj elementa nalazi se između otvarajućeg i zatvarajućeg taga. Sadržaj elementa može biti tekst, drugi elementi, mešavina teksta i elemenata ili ništa. Element može da ima atribute. Atrubuti imaju svoj naziv i sadržaj, koji je isključivo tekstualni.

8. **Navesti pravila dobro formiranog XML dokumenta?**

- Ima tačno jedan korenski element
- Elementi se mogu ugnježdavati, ali ne preklapati
- Vrednosti atributa moraju biti unutar navodnika
- Element ne može imati dva atributa sa istim nazivom
- Komentar i procesne instrukcije se ne smeju nalaziti unutar taga

9. **Kada koristiti atribute, a kada elemente?**

Nema univerzalnog odgovora. Podaci bi trebalo da budu smešteni u elementima, a metapodaci u atributima. Elementi se mogu lakše prilagoditi izmenama u budućnosti i mogu se struktuirati.

10. **Pravila za imenovanje XML atributa i elemenata?**

Imena su casesensitive. Mogu se koristiti slova, brojevi i znakovi (_, -, :, .). Ime može početi slovom ili donjom crtom. Ime ne može početi rezervisanom reči xml.

11. **Osim atributa i elemenata koje delove može imati XML dokument?**

- Komentar – navodi se između <!-- i -->. Predstavlja komentar autora XML dokumenta. Ignorišu se kao sadržaj, ne prikazuju se i ne obrađuju se. Mogu se nalaziti bilo gde van taga.
- Procesne instrukcije – navode se između <? i ?>. Predstavljaju instrukciju softveru. Nisu namenjene čoveku, niti su deo sadržaja. Mogu se nalaziti bilo gde izvan taga. Primer je XML deklaracija na početku dokumenta.
- Entiteti
- CDATA sekcije - navode se između <![CDATA[i]]>. Sadrži tekst koji se interpretira direktno, bez izmene entiteta.

Document Type Definition (DTD)

12. **Šta je DTD i koju ulogu ima?**

Za uspešnu razmenu podataka potrebno je da svi učesnici koriste isti format. DTD je deo osnovnog XML standarda i to je dokument sa opisom formata klase/familije/tipa XML dokumenta. Sadrži pravila koji elementi i entiteti se mogu pojaviti na kom mestu u dokumentu i šta je sadržaj elemenata i atributa.

13. **Vrste deklaracija u DTD?**

- Deklaracija elementa – oblik <!ELEMENT naziv (specifikacija sadržaja)>. Naziv mora da odgovara pravilima. Sadržaj predstavlja kombinaciju sekvenci, izbora i sufiksa. Sufiksi (+, ? i *) su oznake za broj pojava nekog podelementa, i ukoliko se izostave podrazumeva se tačno jedan. Oznaka za izbor je | i izbor može da obuhvati dva ili više podelementa. Mogući tipovi sadržaja su drugi elementi, PCDATA, ANY i EMPTY.
- Deklaracija atributa – oblik <!ATTLIST imeElementa imeAtributa tipAtributa default>. Naziv atributa mora da odgovara pravilima za formiranje imena. Tip se bira iz konačnog skupa (CDATA, NMTOKEN, NMTOKENS, nabrajanje mogućih vrednosti, ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION). Obaveznost se bira iz konačnog skupa (IMPLIED, REQUIRED, FIXED, default). Jedna deklaracija može da sadrži više atributa istog elementa.
- Deklaracija entiteta – oblik <!ENTITY naziv "sadržaj">. Mogu se koristiti predefinisani, i definisati novi. Sadržaj može da sadrži druge entitete (rekurzija i cirkularno refrenciranje nisu dozvoljeni). Prilikom procesiranja xml dokumenta u skladu sa datim DTD, svuda gde se javlja naziv entiteta, on će biti zamenjen sa sadržajem entiteta.
- Deklaracija notacije – oblik <!NOTATION naziv PUBLIC "pubid">. Notacije predstavljaju pomoćne podatke za XML aplikaciju prilikom rukovanja sa neparsiranim entitetima.

14. **Nedostaci DTD?**

- Neuobičajna, ne-XML sintaksa
- Slaba podrška za prostore imena
- Nema tipizacije podataka (nema datumskih i numeričkih polja)
- Ograničena proširivost
- Ograničene mogućnost za opisivanje strukture podataka (broj podelemenata se ne može definisati bez nametanja redosleda, ako se koristi mešani sadržaj ne može se nametnuti broj i redosled pojavljivanja).

XML Namespaces

15. **Razlozi za uvođenje namespaces?**

Svako može da definiše sopstvenu XML gramatiku. Ukoliko želimo da koristimo različite gramatike može doći do poklapanja imena. Poklopljena imena se formalno ne razlikuju, aplikacije nemaju način da ih razlikuju i pretraga može biti dvosmislena.

16. **Šta je namespace? Koji odnos imaju dokument i namespace?**

Skup elemenata koji imaju isti prefiks zove se prostor imena (namespace). Prostor imena je skup elementa. Jedan dokument može sadržati više prostora imena. Elementi iz jednog prostora imena mogu se koristiti u više dokumenata.

17. **Kako se identifikuju prostori imena?**

Prostori imena identifikuju se nazivom. Naziv je niz znakova u URI (Uniform Resource Identifier) formatu. URI može biti formiran kao URL (protokol: putanja do resura) i kao URN (urn:nid:nss). U praksi se za identifikaciju prostora imena najčešće koriste URL. Razlozi:

- lakše je napraviti jedinstveno ime
- izbegavaju se slučajne kolizije
- DTD koji definiše elemente iz datog prostora imena se može zaista i postaviti na lokaciju na koju pokazuje URL - na taj način definicije elemenata postaju javne i odmah dostupnim.

18. **Šta je kvalifikovano ime? Zašto je uvedeno?**

Puno ime elementa iz nekog namespace bi zahtevalo navođenje identifikatora prostora imena (najčešće URL) i naziva elementa. Takva imena ne bi bila validna po XML (nedozvoljeni znakovi poput .) i nezgrapna za pisanje. Kvalifikovano ime (QName) se sastoji iz dva dela - lokalno ime i namespace. U XML dokumentu se umesto namespace koristi lokalno ime.

XML Schema

19. **Šta je šema? Poređenje DTD-a i šeme?**

Šema je dokument koji opisuje strukturu drugih elemenata. Šema prevazlizazi nedostatke DTD-a (videti pitanje 14). Prednosti DTD-a:

- Šeme nemaju funkcionalnost rada se entitetima
- DTD može direktno da ugradi u dokument koji opisuje, šema mora biti odvojen dokument
- Softver i dalje potpuno podržava rad sa DTD.

20. **Šta je XML Schema?**

Postoji više standarda za definisanja šema za XML dokumente. XML Schema je standard propisan od strane W3C XML Schema Group. On je najraširenij, najmoćniji i najkomplikovaniji standard za šeme.

21. **Koje elementa ima šema?**

- xsd:element - deklariše element i dodeljuje mu tip
- xsd:attribute - deklariše atribut i definiše mu tip
- xsd:simpleType - definiše novi prosti tip
- xsd:complexType - definiše novi složeni tip

22. **Koja dva pristupa dodele tipa elementu postoje? Koji je kada bolje koristiti?**

Postoje anonimni i imenovani tipovi. Anonimni su definisani u okviru elementa. Imenovi su definisani izvan elementa i dodeljen im je naziv. Ako očekujemo da če se isti tip pojaviti na više mesta bolje je koristiti imenovani. Ako želimo da definicija tipa ne bude dostupna korisniku šeme bolje je koristiti anonimni.

23. **Kako se kreiraju novi prosti tipovi?**

Dekalracija tipa oslanja se na neki od postojećih tipova. U okviru XML Schema standardna unapred su definisani logički, tekstualni, numerički, URI, vremenski i XML tipovi. Mehanizmi za kreiranje novih tipova su:

- restrikcija - navođenje ograničena na vrednost novog tipa (primeri: definisanje minimalne i maksimalne dužine, enumeracije, regularni izrazi itd)
- lista – definisanje konačne liste mogućih vrednosti (primeri: navodi se tip, a vrednost elementa će biti vrednosti tog tipa razdvojene razmakom)
- unija – kombinovanje vrednosti više različitih osnovnih tipova.

24. **Kako se kreiraju novi složeni tipovi?**

Deklaracija složenog tipa koristi jedan od modela: sekvenca, izbor, neuređeni skup podelemenata. Deklaracija složenog tipa može da sadrži i deklaracije atributa.

25. **Kako se deklarišu atributi?**

Prilikom deklarisanja atributa navode se naziv, tip, forma (qualified i unqualified), use (optional, required, prohibited), podrazumevana vrednost atributa, fiksna vrednost atributa.

XML Parseri

26. **Šta je XML parsiranje? Šta je XML parser?**

Podrazumeva prolazak kroz XML dokument sa ciljem:

- pristupa određenim ili svim podacima u njemu
- modifikacije podataka

XML parser je program koji putem odgovarajućih funkcija omogućama laku manipulaciju sadržajima XML dokumenta.

27. **Navesti vrste XML parsera?**

- SAX Parser - obrađuje dokument na osnovu događaja. Za određeni tip događaja piše se odgovarajući handler. Ne učitava odjednom ceo dokument u memoriju (ne pravi njegovu memorijsku prestavu).
- DOM Parser – obrada XML dokumenta se obavlja tako što se ceo dokument učita u memoriju – formira se struktura hijerarhijskog stabla. Sva manipulacija se zatim obavlja nado ovom memorijskom manipulacijom dokumenta.
- StAX Parser – Koristi slične principe kao SAX, bolje je optimizovan.
- XPath Parser – Odradu dokumenta vrši na osnovu odgovarajućih izraza.

28. **Osnovne karakteristike SAX Parsera? Prednosti i mane?**

SAX (Simple Api for XML) Parser je stream orjentisan interfejs za obradu XML-a. Čitanje dokumenta se vrši od root elementa. Događaj se generiše za svaki token koji parser prepozna u dobro formiranom dokumentu. Parser obaveštava aplikaciju o tipu tokena koji je pronađen. Da bi obrada bila smislena potrebno je da aplikacija registruje event handlere. Mogući su event handleri za sadržaj dokumenta (startDocument, endDocument, startElement, characters, processingInstruction), ErrorHandleri (obrada događaja vezanih za greške tokom parsiranja), DTDHandleri (obrada događaja vezanih za parsiranje DTD-a), EntityResolver (pribavljanje eksternih entiteta). Kada se određeni token detektuje, poziva se odgovarajući event handler.

Pogodan za upotrebu kada:

- Obradu je moguće vršiti linearno
- Dokument nije duboko ugnježden
- Procesira se veliki dokument
- Problem koji se rešava zahteva korištenje samo dela dokumenta
- Podaci su dostupni čim ih parser prepozna, što je idealno za obradu podataka koji se “streamuju”.

Nedostaci:

- Nije moguć direktan pristup elementu
- Ukoliko je potrebno imati naknadni pristup podacima koje je parser već prošao, potrebno je pisati sopstveni kod koji bi takve podatke negde privremeno sačuvao.

28. **Osnovne karakteristike DOM Parsera?**

DOM (Document Object Model). Definiše interfejse koji aplikacijama omogućavaju pristup i manipulaciju sadržaja XML dokumenta. Parseri koji implementiraju DOM implementiraju interfejse Node, Element, Attr, Text, Document, itd... Po završenom parsiranju aplikaciji je na raspolaganju objektna reprezentacija sadržaja dokumenta, formirana kao hijerarhijsko stablo. DOM obezbeđuje veliki broj funkcija za pristup i manipulaciju nad DOM strukturom i sadržajem.

Osnovna prednost: DOM je standarni interfejs za manipulaciju strukturom XML dokumenta. Kod napisan da koristi jedan DOM parser, trebalo bi da radi i sa drugim DOM parserima.

Glavni nedostatak: Nepogodan je za jako velike dokumente, jer formiranje in-memory strukture može biti prezahtevno. Nepogodan je kod serverskih aplikacija zbog puno istovremeno parsiranih dokumenata.

Pogodan za upotrebu kada:

- Za obradu je neophodno sagledati i dobro poznavati strukturu celog dokumenta
- Neophodno je reorganizovati dokument (premeštati, sortirati, dodavati elemente...)
- Velika je verovatnoća da će nam određene informacije iz dokumenta biti potrebne više puta tokom obrade.

29. **Osnovne karakteristike StAX parsera? Koja dva načina parsiranja omogućava?**

StAX parser je streaming parser, koji implementiri pull koncept (program traži podatke od parsera kada su mu potrebni). Može da se koristi i za čitanje i za pisanje XML dokumenta. Parsriranje se može vršiti pomoću:

- Cursors Api koristi kursor koji se kreće od početka do kraja XML dokumenta i pokazuje na elemente XML informacionog skupa.
- Iterator Api predstavlja XML dokumenta kao niz diskretnih događaja, koji se mogu obraditi.

30. **Osnovne karakteristike JAXB parsera?**

JAXB (Java Architecture for XML Binding) je frejmvork za generisanje Java klasa na osnovnu DTD ili XML Schema šema (i obrnuto) i za transformaciju XML dokumenta u graf Java objekata (i obrnutno). Može se posmatrati kao jedan od načina (de)serijalizacije Java objekata. Koristi se za parsiranje transakcionih dokumenata koji se lako mapiraju na Java objekte. Nije pogodan za parsiranje narativnih dokumenata.

XPath

31. **Šta je XPath? Koje elemente sadrži XPath?**

XPath je jezik za označavanje delova XML dokumenta. Zasnovan je konceptu navigacije kroz stablo dokumenta. W3C standard je. Elementi:

- Standardne funckije (operacije sa stringovima, operacije sa numeričkim vrednostima, itd...)
- XPath izraz – namenjen za označavanje čvora ili skupa čvorova u dokumentu

- XPath čvor - sa stanovišta XPath XML dokument je stablo čvorova (element, atribut, tekst, namespace, ...).
- XPath putanja – sastoji se iz više koraka razdvojenih znakom /. Svaki korak izračunava se u odnosu na čvorove u tekućem skupu čvorova.

32. **Od čega se sastoji korak XPath putanje?**

Korak XPath putanje ima oblik osa::test[predikat].

- Osa – definiše skup čvorova u odnosu na tekući čvor (ancestor, child, attribute, following, self, parent, itd...). Ako se ne navede podrazumeva se child. Postoje verzije za skraćeno pisanje i za ostale (npr. tačka (.) za self, @ za attribute)
- Predikat - piše se unutar [] i služi za pronalaženje čvorova koji zadovoljavaju dati uslov.

XPointer

33. **Šta je XPointer? Čime proširuje XPath?**

XPointer je jezik za označavanje delova XML dokumenta, koji proširuje XPath. Proširuje XPath:

- pronalaženjem podataka poređenjem teksta
- može da se doda na kraj URI-ja
- označava ne samo cele čvorove u dokumentu, već i delove čvorova.

34. **Koji je opšti oblik XPointer izraza? Šta sve može biti lokacija?**

Opšti oblik je xpointer(lokacija). Lokacija može biti:

- XPath čvor
- Tačka - definiše se pomoću čvora koji je sadrži i indeksa i. Za struktuirani sadržaj tačka je mesto odmah nakon i-tog podelementa. Za tekstualni sadržaj tačka je mesto odmah nakon i-tog karaktera.
- Opseg – interval između dve tačke. Definiše se funkcijama range-to(), range(), range-inside() i string-range().

XLink

35. **Šta je XLink? Objasniti razliku između HTML linka i XLink linka?**

XLink je jezik za definisanje linkova između (delova) XML dokumenata. XLink je opštiji. HTML link povezuje dva resursa jednom usmerenom granom od polazišta ka odredištu. XLink link povezuje dva ili više resursa i ima jedan ili više lukova.

36. **Kako se definiše XLink? Objasniti uloge atributa?**

Ne postoji posebni element za definisanje linka, svaki element u dokumentu može da posluži toj svrsi. Za definisanje se koriste globalni atributi definisani od strane XLink-a (type, href, role, title, itd...).

- xlink:role – opisuje ulogu elementa u linku. Od njega zavisi koji drugi atributi će biti mogući. Može da ima jednu od 6 predefinisanih vrednosti: simple, extended, locator, arc, resource, title. Namenjen je za mašinsku obradu.
- xlink:href - sadrži adresu resursa koji učestvuje u linku. Adresa je u URI obliku.
- xlink:title – predstavlja tekstualni opis uloge, prilagođen čoveku.

- xlink:actuate – navodi kada će resurs biti dobavljen. Moguće vrednosti onLoad, onRequest, undefined i sopstevene vrednosti.
- xlink:show – opisuje kako će se resurs prikazati kada se učita. Moguće vrednosti new, replace, embed, undefiend, sopstvene vrednosti.
- xlink:from i xlink:to - definišu luk.

37. **Vrste linkova u XLink?**

Vrste linkova su:

- simple – povezuje dva resursa jednosmernim lukom
- extended - opšti oblik linka.

XSL Transformations

28. **Šta je XSLT?**

XSLT je jezik za opis transformacija XML dokumenta u druge XML dokumenta. Predstavlja W3C standard. Transformacije se opsije XSLT dokumentom, a izvodi je XSLT procesor.

29. **Šta je šablon u XSLT?**

U okviru XSLT fajla nalaze se definicije šablona. Šablon opisuje transformaciju dela polaznog dokumenta. Za identigfikaciju dela polaznog dokumenta na koji se šablon odnosi koristi se XPath izraz.

30. **Navesti elemente XSLT?**

- <xsl:stylesheet> - korenski element svakog XSLT dokumenta. Atribut version govori koja verzija XSLT-a se koristi.
- <xsl:template> - definiše šablon
- <xsl:value-of> - sadržaj datog elementa dodaje u rezultujući dokument
- <xsl:for-each> - izdavaja element iz skupa čvorova
- <xsl:sort> - sortira sadržaj tekućeg konteksta prema datom kriterijumu
- <xsl:if> - prikazuje sadržaj samo ako je uslov ispunjen
- <xsl:choose> - izvršava više uzastopnih testova
- <xsl:apply-templates> - pokreće ponovnu primenu šablona u tekućem kontekstu
- <xsl:call-template>-poziva imenovani šablon na datom mestu.
- parametri
- promenljive

XQuery

31. **Šta je XQuery? Objasniti pojmove sekvence, singletona i item-a?**

XQuery je standardni upitni jezik za XML (i W3C srandard).

- Sekvenca – svaka XQuery vrednost je sekvenca sa 0 ili više elemenata. Sekvenca ne može da sadrži druge sekvence.
- Singleton – svaki element sekvence je Singleton. Svaki Singleton ima svoj tip izveden iz item().

- item() - slično kao Object klasa u Javi, ali apstraktno (ne može se instancirati). Postoje dve vrste item()-a: XML čvorovi - node() i atomičke vrednosti – anyAtomicType.

32. **XQuery izraz?**

Svaki XQuery izraz može imati statički tip (compile-time) i dinamički tip (run-time). Dinamički tip predstavlja tip dobijenog rezultata, tj. vrednost rezultata je instanca tog tipa.

33. **Šta je XQuery prolog?**

XQuery prolog je neobavezni deo Xquery izraza. Definiše kontekst za upit (compile-time) - namespaces, korisničke funkcije, importovani šema tipovi, importovani modulu, promenljive.

34. **FLWOR izrazi?**

FLWOR je centarlni izraz u XQuery. Obuhvata for (uteracija kroz sekvencu), let(dodeljuje promenljivoj vrednost izraza), where (filtrira), order by (sortira), return (konstruiše rezultat). Dozvoljeno je da se for i let pojavljuju u bilo kom redosledu i broju, sve dok postoji barem jedna let ili for klauzula. Namena mu je:

- definisanje promenljivih
- iteracija kroz sekvencu
- filtriranje rezultata
- sortiranje sekvenci
- spajanje različitih izvora podataka.

Skraćena verzija FLOWR izraza su some i every. Vraćaju logičku vrednosti – some (postoji), every (za svaki).

Service-Oriented Architecture (SOA)

1. **Šta je servis, a štao SOA?**

Servise predstavlja funkcionalnu celinu – posao koji davalac usluge obavlja sa ciljem da postigne neki željeni rezultat za korisnika usluge. Servisno orjentisane arhiterktura (SOA) predstavlja sistem koji se sastoji od modularnih softverskih komponenti, sa jasno definisanim i standardizovanim načinom pristupa i interfejsima koje su nezavisne od specifičine platforme i tehnologije implementracije. U najkraćem obliku, SOA predstavlja kolekciju stanardizovanih servisa dostpunih preko mreže, koji međusobno komuniciraju u kontekstu izvršenja poslovnog procesa.

2. Razlike OO i SOA? Razlike između Web Service i SOA?

OO sugeriše da se uvežu podaci i operacije nad njima. Na primer, kod OO bi se uz svaki CD/DVD dobijale funkcionalnosti za reprodukciju - CD/DVD player. SOA nalaže da postoji CD/DVD Player koji poštuje standard i može da obavlja reprodukciju bilo kog CD/DVD-a.

Web servisi predstavljaju kolekciju tehnologija (XML, SOAP, WSDL, REST) koji omogućavaju da se povežu programska rešenja u specifičan sistem razmene poruka i omogućavaju integraciju aplikacija. Web servisi su implementacija SOA principa, dok je SOA tip arhitekture sistema. SOA je na višem nivou apstrakcije – ne predstavlja bilo koji specifičan skup tehnologija.

3. **Prednosti SOA?**

- Interoperabilnost – rezultujuće sisteme karakteriše slaba povezanost i modularnost komponenti koji komuniciraju preko standardizovanih komunikacionih metoda i kanala. Povezivanje aplikacije je pojednostavljeno i ne zahteva prepravljanje velikih delova aplikacije, kada se mala izmena unese u sistem.
- Sistemi su više agilni i prilagodljivi izmenama poslovnih procesa – rekonfigurisanje slabo povezanih komponenti (servisa) je jednostavno, relativno brzo i jeftino. Time SOA omogućava da podrška brže reaguje na izmenjene procese, nove zahteve ili nove procese.

4. **Procesi na visokom nivou (top-level)?**

- Proces isporuke servisne implementacije – tradicionalni razvoj, programiranje, automatizacija web servisa uz pomoć alata.
- Osiguravanje isporuka servisa (usluge) - životni ciklus servisa i ponovna iskoristivost artifakata.
- Proces korištenja servisa (usluge) - u osnovu vođen poslovnim procesom.

5. **Koraci SOA procesa?**

- SOA Planning (planiranje) – na osnovu poslovnog procesa definišu se i kreiraju servisi (usluge).
- SOA Enablement (osposobljavanje za korištenje SOA servisa) - na bazi prihvaćenih standarda formira se registrar svih servisa. Kreira se infrastrktura za podršku, npr. za upravljanje web servisima, upravljanje entitetima, razmenu poruka.
- SOA Publishing (publikovanje) - objavljuju se definicije procedura ili politika. Politike sadrže ko sme da objavi novi servis u registru, koje procedure objavljivanja novih verzija se moraju poštovati, kako će različite politike dizajna, primene standarda i sigurnosti biti sprovedene unutar SOA rešenja.
- SOA Discovery (pronalaženje servisa) - koristi se UDDI.
- SOA Management (upravljanje servisima) - obuhvata upravljanje međusobnim vezama poslovnih servisa, upravljanje verzijama i zavisnostima komponenti, sprovođenje rekonfiguracije različitih aspekata aplikacije (lokacija, transport, bezbednost, ...)
- SOA Analysis (analiza korištenja servisa) -
    - obuhvata nadgledanje i prikupljanje povratnih informacija (radi optimizacije rešenja),
    - nadgledanje statusa servisa, korisnika, načina korištenja i prikupljanje metrike koja omogućava zaključivanje relativne uspešnosti poslovnih servisa,
    - obezbeđivanje analize uticaja i zavisnosti individualnih servisa, grupisanje servisa na bazi zajedničkih osobina
    - širok spektar izveštaja - poslovnih, IT, SOA, o ponovnoj iskoristivosti, narušavanjima politike koriptenja, ...

6. Navesti slojeve SOA aplikacije. Navesti osnovne komponente SOA aplikacije?

Višeslojne aplikacije obično sadrže prezentacioni sloj, sloj poslovne logike i sloj za skladištenje podataka. Dva ključna sloja u SOA arhitekturi su servisni sloj i sloj poslovnih procesa. Osnovne komponente su:

- Service providers (ponuđač servisa) - komponenta ili skup komponenti koji izvršava određene poslovne funkcije u stateless režimu, prihvatajući predefinisane ulaze i poizvodeći odgovarajuće izlazne parametre.
- Service consumers (korisnik servisa) - skup komponenti koje koriste jedan ili više servisa koji nude ponuđači servisa.

- Servise repository (katalog servisa) - sadrži opise servisa. Ponuđači svoje usluge prijavljuju katalog, a korisnici pretražuju katalog kako bi pronašli odgovarajuću uslugu.

7. Šta podrazumeva dinamičko pronalaženje servisa? Prednosti?

Ponuđači svoje usluge prijavljuju katalog, a korisnici pretražuju katalog kako bi pronašli odgovarajuću uslugu. Katalozi organizuju servise po određenim pravilima kategorizacije i nude pretraživače korisnku.

Prednosti:

- Skalabilnost
- Potpuno razdvajanje korisnika i ponuđača servisa
- Brzi updejt servisa
- Korisnik ima na raspolaganju servis za pregled servisa
- Korisnik može i u vremenu izvršavanja da izbaere ponuđača čiji servis im u tom trenutku najbolje odgovara.

8. Izazovi i rešenja za SOA?

- Identifikacija servisa - šta je servis, koja je poslovna funkcionalnost koji bi on obezbeđivao, koja je optimalna granularnost
- Lokacija servisa – gde servis treba da se nalazi u poslovnom sistemu
- Definicija domena servisa – kako se servisi grupišu u logične domene
- Pakovanje servisa – kako postojeće servise reinženjeringom zapakovati u ponovo iskoristive servise
- Orkestracija servisa – kako je moguće orkestracijom postojećih servisa kreirati nove
- Rutiranje servisa – kako zahteve klijenata rutirati do ogovarajućeg servisa
- Upravljanje servisom – kako će poslovni sistem upravljati administracijom i održavanjem servisa
- Usvajanje standarda za razmenu poruka u okviru servisa – kako će poslovni sistem konzistentno usvojiti određeni standard razmene poruka

9. Koji problem rešava distribuirana dinamička rekonfiguracija? Koje zahteve treba da ispuni?

Trenutno rešenja često nisu u stanju da zadvolje zahteve izdržljivosti. Ključna osobina izdržljivosti je da je sistem sposoban da se dinamički rekonfiguriše u slučaju otkaza ili preopterećenja. Često ne uspeva automatsko pronalaženje i povezivanje na alternativni servis. Rešenje je distribuirana dinamička rekonfiguracija.

Zahtevi koje treba da ispuni:

- Efikasnost – rekonfiguracioni algoritam treba da unosi minimalan poremećaj u sistem
- Konzistentnost – akacija rekonfiguracije treba da obezbedi konzistentnost svih komponenti nakon rekonfiguracije
- Ispravnost – uslovi dati za rekonfikuraciju moraju biti ispunjeni u svakom momentu
- U realnom vremenu – rekonfiguracija se mora sprovesti u realnom vremenu tokom rada sistema
- Upravljan odgovarajućom politikom – sistemom upravlja poslovna politika, verovatno i politika upravljanja distribuiranim sistemom

- Distribuirano i paralelno izvršavanje - rekonfiguracija se sprovodi pomoću distribuiranih agenata i akcije rekonfiguracije se sprovode istovremeno
- Servisa orjentacija – server koji upravlja dinamičkom rekonfiguracijom ili agent su i sami servisi u SOA arhitekturi, i sami se mogu rekonfigurisati u slučaju potrebe
- Dinamička rekonfiguracije organizovana hijerarhijski - ugrađuje se u svaki sloj sistema kako bi se izbegla jedinstvena tačka otkaza.

10. Koje funkcije obezbeđuje DRS u okviru SOA? Kako se on implementira?

DRS kao servis obezbeđuje funkcije:

- Dinamički pregled servisa, objavljivanje servisa, povezivanje na servisive, profiliranje servisa
- Registraciju i odjavu servisa
- Servise za verifikaciju tokom izvršavanja, koji uključuju i proveru ograničena (bezbednosnih ograničenja, provere interoperabilnosti, nadgledanje performansi).

DRS se implementira kao kritični servis i strategije rekonfiguracije i algoritmi se mogu adaptirati tokom izvrešenja. Ovo omogućava da se može tokom rada adaptirati ponašanje SOA sistema, kao i samog DRS.

11. **Šta su ontologije? Za šta nam treba ontologija za SOA?**

Ontologije služe da opišu značenje (semantiku) podataka – metapodatke. Opisivaje semantike podataka vrši se kako bi se obezbedio jedinstven način razumevanja podataka od strane različitih učesnika, sa ciljem razumevanja. U okviru SOA mogu se koristiti da:

- Opišu veze i zavisnosti između servisa i njihove interakcije
- Prilikom dinamičkog utvrđivanja odgovarajućeg servisa
- Da bi se unapredila ponovna iskoristivost servisa.

12. Problemi sa testiranjem SOA?

Male promene mogu proizvesti značajan uticaj na aplikacije. Brzina promena uslovljava da tradicionalne arhitekture za testiranje, ne mogu dovoljno brzo da obave svoj posao.

Indirektno testiranje je pristup da se problem pokušava otkriti simuliranjem rada aplikacije ponuđača i klijenta. Problemi:

- Primena moguća samo u kasnim fazama razvoja – potrebno je da servis postoji, a tada su izmene skupe
- Podaci nisu dovoljni da se pronađe gde nastaje greška - u aplikaciji ponduđača servisa, načinu transporta i kodiranja podataka, sadržaju poruka, klijentaskoj aplikaciji
- SOA ne pretpostavlja klijenta servisa
- SOA zahteva sprovođenje pojedinačnih, a zatim i integracionih testova
- Testiranje integracije podrazumeva da postoje test interfejsi na svakom od slojeva, da bi se testiranje moglo sprovesti od početka do kraja
- Za testiranje razmene XML poruka moraju postojati interfejsi za kreiranje, slanje, primanje i verifikaciju poruka.

Web Services

13. **Na čemu su zasnovani Web Services? Koje probleme rešavaju?**

Namena Web Services je da omoguće računarima (softverskim komponenta) da direktno komuniciraju putem interneta. Zasnovani su na prethodnim tehnologijama iste namene – RPC i ORPC (DCOM, CORBA, JAVA RMI). Rešavaju probleme:

- Interoperabilnosti – raniji sistemi su imali probleme sa interoperabilnošću jer je svaki proizvođač implementirao sopsteve formate za razmenu poruka između distribuiranih objekata.
- Firewall – sistemi su koristi nestandardne portove. Web Services koriste HTTP kao transportni protokol, a većima firewall-a propšuta konekcije na port 80 (HTTP).
- Kompleksnost – web servisi su developer friendly. Prethodne tehnologije zahtevale su više učenja i navikavanja. Za različite projekte bilo je potrebe učiti dodatne jezike i tehnologije.

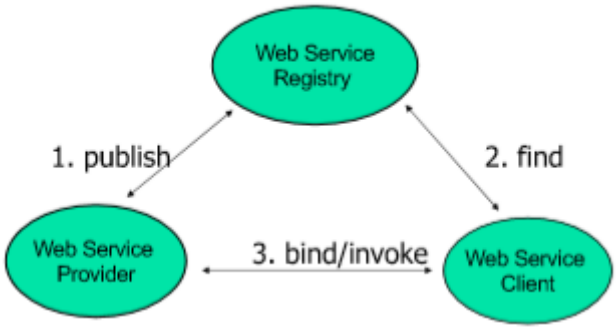
14. **Šta su Web Servisi? Koje komponente čine web servise?**

Web servisi su softverske komponente opisane pomoću WSDL, a kojima je moguće pristupiti preko standardnih mrežnih protokola (SOAP over HTTP). Komponente:

- XML – koristi se kao jedinstven način reprezentacije i razmene podataka
- SOAP (Simple Object Access Protokol) - standardni način kompunikacije. Obezbeđuje pravila za kodiranje zahteva i njegovih argumenata – format pakovanja poruke.
- UDDI (Universal Description, Discovery and Integration specification) - mehanizam za registraciju i lociranje servisa u katalog
- WSDL (Web Services Description Language) - strandardni metajezik za opis ponuđenih servisa.

15. **Objasniti model rada Web servisa? Na čemu se zasniva komunikacija kod Web servisa?**

Model koristi publish, find i bind paradigmu. Ponuđač servisa se registruje kod kataloga (publish). Klijent koristi katalog da nađe odgovarajući servis (find), klijent se zatim povezuje na servis (bind).



Web servisi su dokument centrični. Komunikacija kod Web servisa obavlja se slanje dokumenata na server i primanjem dokumenta kao odgovora od servera.

16. **Prednosti Web servisa? Nedostaci?**

Prednosti:

- Obezbeđuju interoperabilnost između heterogenih sistema, koji se izvršavaju na heterogenim platformama.
- Koriste otvorene standarde i protokole. Kada god je moguće protokoli i formati su tekst bazirani.
- Zbog upotrebe HTTP-a, Web servisi mogu raditi kroz većinu uobičajnih firewall-a, a da pri tome nije potrebno posebno nameštanje.

Nedostaci:

- Koverzija podataka u/iz XML – nije pogodna za sisteme gde su performanse komunikacije od posebnog značaja
- Nepotrebno komplikovanje implementacije sistema – kod sistema koji su homogeni i u celosti implementirani na jednoj ravojnoj platformi.

17. **Objasniti princip rada Web servisa? Koji su potencijalni problemi?**

Klijent prvo pronalazi servis. Tipično se potom izvršava povezivanje na servis (binding) uspostavljanjem TCP veze ka adresi pronađenoj u opisu servisa. Klijent potom kreira SOAP zahtev (marshalling) - popunjava informacije o tome koji servis je potrebno izvršiti, kao i argumente. Zapakovan zahtev se šalje serveru. SOAP ruteri usmeravaju zahtev na odgovarajući server, ako ih ima više. Server raspakuje zahtev (unmarshaling). Rezultat se šalje nazad.

Potencijalni problemi:

- Marshalling – podaci koji se razmenjuju moraju biti predstavljeni u platformski nezavisnom formatu (endian-ness, preciznost, floating point, pokazivači)
- Pronalaženje pravog servisa – mogu se koristiti ili URL ili URN. Uobičajni način je da se koristi posrednik – servis za pronalaženjer servisa (katalog).

18. **Šta je katalog? Zašto je otkrivanje servisa problematično?**

Katalog je baza koja izlistava spisak servera koji nude servisa. Opisan najčešće upotrebom UDDI jezika (XML notacija). UDDI se koristi da opiše informacije koje predstavljaju jedan zapis u katalogu. WSDL dokumentuje interfejse i tipove podataka koje servis koristi.

Problemi:

- klijent ima svoje viđenje stavri
- service ima svoje viđenje stvari
- DNS ima svoje
- Internet ima svoje
- Web servis ne opisuje stardandizovani način kako obaviti traženje odgovarajućeg servisa, nego pretpostavlja da je to moguće
- Network address translation – interne adrese servera na mreži često nisu dostupne spolja
- Firewalls

SOAP

19. **Objasniti razliku između RPC i document style komunikacije? Prednosti?**

RPC (Remote Procedure Calls) podržavaju sintakstu i semantiku pozivanja funkcija i metoda. Kod slanja poruka (document-style) komunikacija je zasnovana na slanju strukturanih podataka. RPC je jednostavan za učenje, efikasan za kodiranje i tipičan za sinhronu komunikaciju. Document-style bolje razdavaja delove sistema, obezbeđuje veću nezavisnost sistema i tipičan je za asinhronu komunikaciju.

20. **Razlike između API-level specifikacije i wire-level specifikacije? Prednosti i nedostaci API-level specifikacija za pristup wire-level protokolima?**

API-level specifikacije predstavlja definiciju klasa, interfejsa, metoda, itd. koje su načinjene za date potrebe (JDBC, JNDI, JDO itd). Omogućavaju pojednostavljenu zamenu programskig modula koji su

apstrahovani datim API-jem (npr. JDBC omogućava zamenu baze podataka, bez značajnih promena programa). Konkretna implementacija API-level specifikacije predstavlja programsku biblioteku koja podržava API.

Wire-level specifikacije predstavljaju definicije formata poruka koje se razmenjuju između učesnika u komunikaciji i postupka razmene poruka (HTTP, SMTP itd). Omogućavaju komunikaciju heterogenih delova sistema. Konkretna implementacija podrazumeva biblioteku koja implementira dati protokol.

Prednosti API-level specifikacija za pristup wire-level protokolima: najbolje rešenje u smislu prenosivosti i interoperabilnosti.

Nedostaci API-level specifikacija za pristup wire-level protokolima: komplikovano za upotrebu sa stanovišta programera (više isprepletanih standarda).

21. **Šta je SOAP? Odnos SOAP i HTTP?**

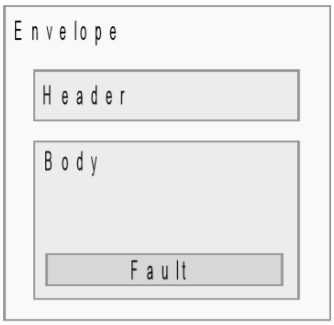
SOAP (Simple Object Access Protocol) je protokol za komunikaciju sa Web servisima. Definiše format poruka koji razmenjuju učesnici. Oslanja se na neki transportni mehanizam za prenos SOAP poruka.

HTTP može biti transportni mehanizam za prenos SOAP poruka - "HTTP binding". Za slanje poruka tada se koristi POST komanda.

22. **Kakvu strukturu ima SOAP poruka?**

SOAP poruka je XML dokument:

- Envelope – korenski dokument. Obuhvata celu SOAP poruku. Atribut encodingStyle definiše namespace sa tipovima podataka koji se koriste u dokumentu - definiše način serijalizacije podataka iz aplikacije u XML.
- Header – opcioni element u okviru Envelope. Sadrži podatke koji opisiju kontekst u kome se šalje poruka ili upustva za posrednike u komunikaciji. Ovi podaci ne predstavljaju samu poruku, već utiču na način obrade poruke. Na primer: podaci za autentifikaciju, podaci za praćenje sesije, podaci za upravljanje transakcijama. Svaki SOAP procesor može da dodaje elemente u Header na putu od pošiljaoca do konačnog primaoca. Svaki procesor je dužan da ukloni elemente namenjene njemu (označeni sa @actor atributom). Elementi u Headeru se dodatno mogu označiti sa @mustUnderstand – da li poruka može biti obrađena ako SOAP procesor ne razume i ne zna da obradi element Header-a.
- Body – obavezni element u okviru Envelope. Sadrži konkretan SOAP zahtev ili odgovor.
- Fault – opcioni podelement elementa Body. Sadrži podatke o nastalim greškama, namenjenim klijentu. Ima četiri podelementa faultcode (indikacija greške namenjena mašinskoj obradi), faultstring (tekstualni opis greške namenjen čoveku), faultactor (indikacija čvora u komunikaciji koji je uzork greške), detail (opisuje greške koje su posledica neispravnog sadržaja u Body).



23. **Dve mogućnosti za SOAP reprezentaciju podataka?**

- SOAP encoding – mapiranje podataka iz Jave/C++/... u skladu sa SOAP specifikacijom. Podaci u okviru Body nastaju serijalizacijom podataka iz aplikacije u XML format u skladu sa nekom šemom. SOAP ne defeniše podrazumevanu šemu. Dozvoljena je upotreba više šema u jednoj SOAP poruci. Izbor aktivne šeme se vrši izborem atributa encodingStyle. Svi podaci u telu poruke su predstavljeni kao sadržaj elemenata.
- XML Schema – podaci koji se prenose definisani su XML šemom.

24. **Tipovi podataka u SOAP-u?**

- Jednostavi (simple) - atomičke vrednosti, bez atributa i podelemenata
- Agregirani (compound) - sadrži više jednostavnih podataka organizovanih u neku strukturu
- Predefinisani tipovi
- Novi tipovi definisani XML Schema jezikom
- Reference – prazni su i imaju atribut href čiji sadržaj je id nekog drugog elementa, element na koji ukazuje referenca mora imati atribut id tipa ID, korisni za serijalizaciju grafova
- null vrednosti.

25. **SOAP u Web servisima RPC tipa i SOAP u Web servisima document-style tipa?**

U Web servisima RPC tipa SOAP zahtev predstavlja poziv metode, a SOAP odgovor rezultat metode. Za poziv je potrebno identifikator objekta kome je poziv upućen, naziv metode, parametri metode, podaci za SOAP zaglavlje. Identifikacija objekat kome je poziv upućen (naziv objekta) ugrađuje se POST komandu HTTP zahteva. Naziv metode određen je podelemetom Body elementa – servis prepoznaje dati element i poziva odgovarajuću metodu. Parametri su dobijeni serijalizacijom podataka iz aplikacije i smešteni u SOAP Body.

Kod Document-style Web servisa ne postoji koncept metode i parametara. Servisu se šalje XML dokument. Metoda servisa koja će obraditi dokument se ipak mora navesti – prvi Body podelement navodi nju, dok njegov namespace navodi servis. Komunikacija nije obavezno po modelu zahtev/odgovor. Moguće verzije su:

- One Way – klijent šalje zahtev
- Request-response – klijent šalje zahtev, servis odgovor
- Solicit response – servis šalje request, klijent response
- Notification – servis šalje request

WSDL

26. **Šta je WSDL?**

WSDL (Web Service Description Language) je XML gramatika za opisivanje Web servisa kao skupa krajnjih pristupnih tačaka koji mogu da razmenjuju poruke na RPC ili document-style način. WSDL opisuje šta servisa radi, kako pozvati njegove operacije i gde ga pronaći.

27. **Koka/jaje problem u WSDL?**

Na osnovu WSDL specifikacije servisa moguće je generisati delimičnu implementaciju servisa i klijentske klase za pristup servisu. Na osnovu interfejsa u implementaciji servisa moguće je generisati WSDL koji opisuje dati servis. *(Na projektu smo koristili generisanje klijentskih klasa za pristup na agentu i generisanje WSDL-a na osnovu interfejsa na backu).*

Koka/jaje problem je da li:

- prvo pisati WSDL, pa generisati servis – bolje kada komuniciraju delovi sistema zasnovani na različitim tehnologijama
- prvo pisati servis, a WSDL generisati – bolje kada komuniciraju servisi u istoj implementacinoj tehnologiji.

28. **Struktura WSDL dokumenta?**

- wsdl:definitions – korenski element WSDL dokumenta. Često se koristi za globalne namespace deklaracije.
- wsdl:import – dodaje sadržaj datog namespace-a u tekući WSDL. Predstavlja način za modularizaciju WSDL dokumenta. Dodatni namespace definisan je u datom fajlu. Tipično služi za dodavanje XML Schema definicija tipova.
- wsdl:types – kontenjer za definicije tipova podataka koji se dalje koriste u dokumentu. Kao jezik se koristi W3C XML Schema.
- wsdl:message – za definisanje poruka koje se razmenjuju u komunikaciji sa Web servisom. Poruka se sastoji iz delova, svaki deo pripada nekom tipu podataka.
- wsdl:portType - definiše skup operacija koje se mogu izvršiti nad Web servisom. Operacija može imati ulaznu i izlaznu poruku.
- wsdl:binding - definiše konkretan portokol i format podataka za portType. Mogu se koristiti standardni protokoli i definisati novi. Po jedan element u binding navodi se za svaki element u portType.
- wsdl:service – koristi se kada je potrebno definisati konkretan endpoint za Web servis. *(U okviru njega je bedni Location)*

UDDI

29. **Šta je UDDI? Šta omogućava?**

UDDI (Universal Description, Discovery and Integration) je standardno okruženje za publikovanje i otkrivanje informacija o Web servisima. Predstavlja standard za objavljivanje podataka o sopstvenoj poslovnoj organizaciji zajedno sa podacima o pristupu softverskim servisima koje organizacija nudi. Omogućava formiranje registra na globalnom nivou.

30. **Tipovi informacija u UDDI registru? Tipovi korisnika UDDI servisa?**

Tipovi informacija:

- White pages – osnovne informacije za kontakt, poreski identifikacioni broj (PIB) i slično. Ovi podaci omogućavaju pronalaženje Web servisa na osnovu podataka o konkretnom poslovnom partneru
- Yellow pages – informacije koje opisuju usluge Web servisa pomoću svrstavanja u razne kategorije. Ovi podaci omogućavaju pronalaženje Web servisa po vrsti proizvoda/usluga koje organizacija nudi.
- Green pages - tehničke informacije koje opisuju funkcije Web servisa, uključujući podatke o lokaciji servisa. Ovi podaci omogućavaju pristup konkretnim Web servisima.

Tipovi korisnika:

- Poslovni službenici - koriste UDDI registar slično kao pretraživač. Rezultat pretrage je skup podataka o organizaciji. Pošto nisu svi podaci u registru user friendly, pristup se najčešće vrši putem portala.
- Softverski inženjeri - koriste UDDI registar da pronađu podatke o servisima ili da registruju sopstveni servis.

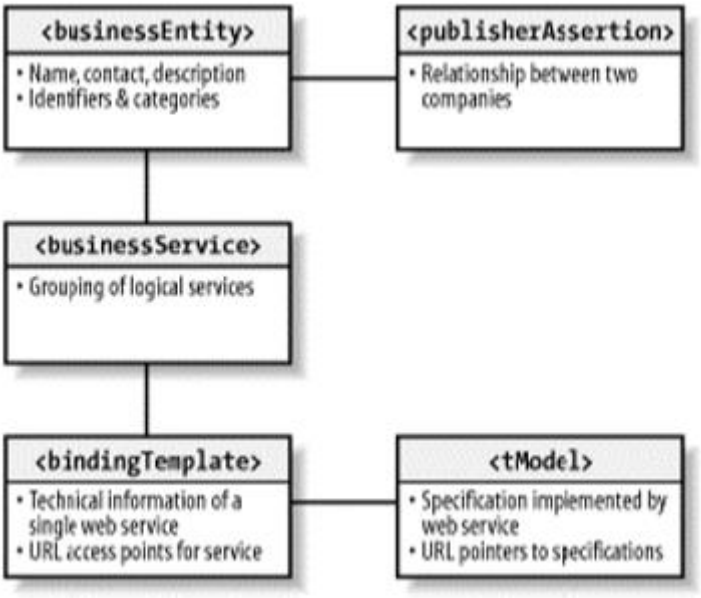
31. **UDDI arhitektura?**

UDDI je jedan jedinstveni servis na svetskom nivou. Sastoji se iz više čvorova. Svaka organizacija može da napravi sopstveni UDDI čvor i uključi ga u UBR (UDDI Business Registry).

Lokalne baze podataka u čvorovima se sinhronizuju replikacijom. Dodavanje podataka u registar se odvija na jednom čvoru. Taj čvor postaje vlasnik i kasnije izmene su moguće samo preko tog čvora. Pristup bilo kom čvoru registra omogućava pristup svim podacima registra.

32. **UDDI model podataka?**

- businessEntity – opisuje organizaciju
- businessService – opisuje servise koje nudi organizacija
- bindingTemplate – opisuje tehničke informacije potrebne za pristup servisu
- tModel – opisuje tehnički model nekog koncepta, npr. tip Web servisa ili protokol koji se koristi
- publisherAssertation – opisuje vezu jednog businessEntity sa drugim
- subscription – opisuje zahtev za praćenjem promena u okviru nekog entiteta



33. **Šifrarnici nad UDDI modelom podataka?**

Nad skupom podataka UDDI registra moguće je definisati više taksonomskih klasifikacija. Ovo daje mogućnost za šifriranje delatnost, proizvoda, geografskog položaja. Postoje kontrolisani i nekontrolisani šifrarnici. Kod kontrolisanih unete vrednosti moraju pripada nekom konačnom skupu, kod nekontrolisnih se ne proveravaju.

RESTful Web servisi

34. **Šta je REST? Koja ograničenja specificira? Koje su prednosti upotrebe REST-a? Šta su sve resursi u REST-u?**

REST (Representational State Transfer) je stil softverske arhitekture namenjen distribuiranim hipermedijalnim sistemima (poput World Wide Web). Specificira sledeća ograničenja:

- Način identifikacije resursa
- Uniformisani interfejs – GET, PUT, DELETE, POST, (HEAD, OPTIONS)
- Samoopisive poruke
- Stanje aplikacije upravljanje hipermedijom
- Stateless interakcija.

Poštovanje datig ograničenja pojačava pozitive osobine, kao što su performanse, skalabilnost, izmenjivost.

U okviru REST-a kao resursi se podrezumevaju podaci i funkcionalnosti. Rest je klijent/server arhitektura i dizajnirana je da koristi stateless komunikacioni protokol. Klijent i server razmenjuju reprezentacije resursa koristeći standardizovani protokol i interfejs.

35. **Navesti korake prilikom dizajna REST Web servisa?**

- Identifikacija resursa koji treba da budu vidljivi kao servis
- Modelovanje relacija između resursa kao hiperlinkova, koji se mogu slediti da bi se dobilo više detalja

- Definisanje URI-ja za adresiranje resursa
- Definisanje smisla GET, POST, PUT i DELETE zahteva za svaki od resursa, kao i da li su svi dozvoljeni
- Dizajn i dokumentovanje reprezentacije resursa
- Implementacija i postavljanje na Web servis
- Testiranje

36. **Šta je URI?**

URI je Internet Standard za imenovanje i identifikaciju resusa. Osobine koje treba da poštuje "lep" URI:

- Upotreba segmenata, a ne key=value parova
- Upotreba imenica, a ne glagola
- Kraći URI su boji
- Ako je moguće koristiti pozicionu šemu prosleđivanja parametara, a ne query string
- Ponekad se koriste URI postfixi da se specificira tip sadržaja
- Ne menjajte URI-je za resurse. Kada je neophodno koristi se redirekcija.

37. **URI šabloni?**

URI Templates (šabloni) specificiraju kako konstruisati i parsirati parametrizovane URI-je. Na serverskoj strani se obično koriste za pravila za rutiranje, a na klijentskoj da bi se konstruisao URI do resursa, na osnovu lokalnih parametara.

38. **Šta je Content Negotiation? Prednosti?**

Različiti klijenti mogu da traže da šalju i primaju stanje objekta u različitim formatima. Servis bi trebalo da može da podrži različite korisnike, bez potrebe da pravi zaseban interfejs za svakoga od njih. Content Negotiation podrazumeva da se specifični formati i sadržaj reprezentacije podataka koji će biti prihvaćeni i vraćani od strane servisa mogu dogovarati u trenutku izvršavanja servisa, kao deo njegovog poziva. Dogovaranje se zasnima na tipovima sadržaja - media types. Dogovoranje je podržano u samom HTTP zahtevu, i ne zahteva slanje dodatnih zahteva.

Prednosti ovog pristupa: obezbeđuje slabu povezanost, povećana interoperabilnost, povećava agilnost za adaptiranje.

39. **Namena metoda?**

CRUD	REST	
CREATE	POST	Krerira (pod)resurs
READ	GET	Preuzima trenutno stanje resursa
UPDATE	PUT	Ažurira stanje resursa na zadatom URI-ju
DELETE	DELETE	Uklanja resurs. Nakon toga URI više nije validan.

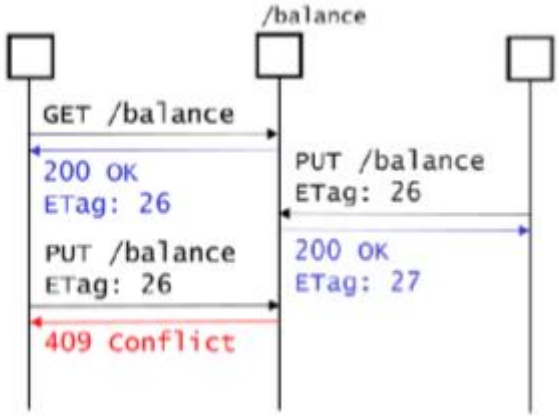
40. **Vrste zahteva nad resusima?**

- Idempotentni - mogu se ponavljati vise puta, a da ne izazivaju negativne bočne efekte na serveru. Ukoliko server ne funkcioniše kako treba (čitaj baca hysterix time out), mogu se ponavljati dok server ne proradi. Idempotentni su GET, PUT i DELETE. Sigurni su oni idempotentni zahtevi koji ne menjaju stanje samog resursa na serveru – GET.
- Nesigurni – modifikuju stanje servera i ne mogu se ponavljati bez da izazovi neželjene efekte na serveru. Za ovakve zahteve neophodne su dodatne akcije u posebnim situacijama. Takav je POST.

41. **Problemi konkurentnosti kod REST-a?**

Problem konkurentnosti se odnosi na situaciju kada jedan korisnik pokušava da čita, a drugi da menja ili briše resurs, ili više korisnika da menja resurs istovremeno. Moguća rešenja su:

- Pesimističko zaključavanje resursa
- Upotreba statusa 409 – prenosi korisniku informaciju da bi izvršavanje zahteva dovelo do nekonzistentnog stanja.



42. **Blokirajući ili neblokirajući zahtevi?**

HTTP je u suštini sinhron, ali ne mora biti blokirajući. Za zahteve za koje znamo da mogu dugo da traju može se vratiti kod 202 sa informacijama gde se može naknadno preuzeti rezultat i koliko često korisnik da pokušava čitanje rezultata.

43. **Richardsov model zrelosti REST servisa?**

- Level 0: HTTP se koristi kao RPC protokol, koristi se za udaljeni pristup, ali bez korištenja mehanizama weba. Obično se sve operacije obavljaju preko jedne adrese.
- Level 1: Identifikacija resursa - uvođenje više URI-ja za različite resurse.
- Level 2: Uniformno korištenje HTTP metoda. Za svaku metodu se očekuje određeno ponašanje. HTTP statusni kodovi se ispravno koriste.
- Level 3: Hypermedia Controls. Na ovom nivou i same osobine servisa i mogućnosti koje nudi klijenu mogu da se odrede pristupom servisu.

Mikroservisi

44. **Šta su mikroservisi? Osnovne karakteristike?**

Još jedan stil arhitekture softverskih sistema, u kome se velike složene aplikacije komponuju skladanje pojedinačnih servisa. Mikroservisi su još jedan pristup implementaciji SOA. Karakteristike:

- Mogu se razvijati u programskom jeziku koji je najpogodniji, nezavisno od ostalih
- Komunikacija između mikroservisa se obavlja programskim interfejsima API-jima koji su nezavisni od programskog jezika
- Mikroservisi imaju potpuno ograničen konkest – ne moraju biti svesni nikakvih implementacionih detalja i arhitekture drugih mikroservisnih modula.

45. **Nedostaci monolitnih aplikacija? Prednosti mikroservisa u odnosu na monolitne aplikacije?**

Nedostaci monolitnih aplikacija:

- Dodavanjem funkcionalnosti raste broj modula i složenost srednjeg sloja
- Skaliranje se obavlja pokretanjem više instanci celokupne aplikacije, iako nam nisu sve funkcije jednako potrebne.

Prednosti mikroservisa:

- Mikroservisi se koncentrišu na jednu poslovnu funkcionalnost i samim tim nema velikog broja modula i velike složenosti
- Komponente se mogu nezavisno razvijati i testirati – u različitim programskim jezicima i okruženjima
- Komponente se mogu lako menjati – svaka se može pojedinačno u celosti zameniti ili ažurirati
- Moguće je različito skaliranje različitih komponenti.

46. **Kada koristiti mikroservise, a kada monolitnu aplikaciju?**

Monolitnu kada:

- Za početne faze razvoja

Mikroservisnu kada:

- Nam treba parcijalna implementacija
- Nam treba dostupnost čak i kada jedan mikroservis otkaže
- Nam treba da razvijamo na različitim platformama
- Želimo da forsiramo modularnost.

47. **Ponovna iskoristivost i granularnost mikroservisa?**

- Ponovna iskoristivnost - poželjna je, ali nije obavezna i nije jedini razlog uvođenja mikroservisa
- Granularnost - određuje se na osnovu poslovnih procesa. Prevelika granularnost može da dovede do latentnosti servisa – ukoliko je servis previše usitnjen i zahteva previše poziva ka drugim mikroservisima može se osetiti problem usporenja apilakacije.

48. **Pretpostavke za uspešnu mikroservisnu arhitekturu?**

- 1 komponenta = 1 servis
- Pametni endpointi i i glupi komunikacioni kanali
- Decentralizovano upravljanje i decentralizovano upravljanje podacima – Odluke o načinu implementacije se donose decentralizovano. Servisi podatke razmenjuju isključivo putem javno dostupnih API-ja, nema oslanjanja na deljenje baze podataka. Ovo omogućava svakom servisu da upravlja na način najpogodniji sa stanovišta tog mikroservisa.
- Automatizacija infrastrukture – Za razvoj mikroservisne arhitekture verovatno je potrebno da se servisi razvijaju i puštaju u rad nezavisno. Deployment novih i ažuriranih mikroservisa treba

da bude brz. Radi upotrebe skalabilnosti, potrebno je da se serverski kapaciteti dobijaju brzo. Potreban je dobar monitoring, kako bi se mogla testirati međusobna komunikacija servisa. Ključna stvar za uspeh je automatizacija svega navedenog.

- Dizajniranje sistema tako da bude otporan na greške - Koncept mikroservisa može da zakomplikuje strukturu, jer stvara mnogo delića koji mogu da otkažu. Ključna stvar je dizajnirati svaki servis pretpostavljajući da u nekom momentu sve ono od čega taj servis zavisi može da nestane i bude nedostupno. Servis tada treba da otkaže gracefully.
- Održavanje konzistencije – Jedno od bitnih pravila mikroservisa je ne koristiti deljenu bazu podataka. Neke podatke ipak verovatno koristi više mikroservisa. Ažuriranja se šalju preko AJAX poziva – nema garancije da će svi modulu obaviti ažuriranje u istom trenutku. Garantuje se konzistentnost u nekom trenutku (eventual consistency).

49. **Šta je DevOps?**

Kovanica nastala iz “software DEVelopment” i “information technology OPerationS”. Termin koji se koristi za praksu u razvoju softvera i koji naglašava kolaboraciju i neophodnost konstante komunikacije između programera (developers) i IT profesionalaca koji nadziru operativnu upotrebu softvera. Ova saradnja je usmerena na automatizaciju procesa isporuke softvera i izmena infrastukture.

50. **Šta obično podrazumeva DevOps?**

- Prakse agilnog razvoja softvera
- Kontinuiranu integraciju
- Automatizaciju isporuke softvera
- Funkcionalno testiranje modula
- Testiranje integracije sistema
- Nadzor sistema i infrastrukture.

51. **Značaj DevOps za mikroservise?**

- Lakša realizacija isporuke i monitoringa mikroservisa
- Brži odgovori na nove zahteve
- Brži odgovori na probleme u produkciji
- Ako nije usvojen DevOps svaki tim mora da razvija svoje principe, što dovodi do nekonzistentnosti po pitanju kvaliteta, sigrurno i dostupnosti različitih mikroservisa
- Obezbeđuje se kontinualna isporuka

52. **Šta je uobičajni zadatak DevOps tima?**

- Obezbeđuju alate, praćenje zavisnosti, upravljanje, upravljanje vidljivošću servisa
- Obezbeđuju bolji uvid u rad mikroservisnih timova i samim tim bolju povratnu informaciju klijentima
- Obezbedđuju neophodne informacije o dostupnosti mikroservisa, kada se oni isporučuju i koji su korišteni od strane drugih servisa i klijenata

53. **Koja tri principa DevOps koristi da bi obezbedio kontinualnu isporuku?**

- Automatizacija – automatizacija tetsiranja, automatizacija obezbeđivanja infrastukture, automatizacija konfigurisanja razvojinih platformi, automatizacija isporuke, automatizracija izmena u bazi i drugih konfiguracionih parametara aplikacija

- Standardizacija – upotreba standardnih platformi, pouzdani i ponovoljivi procesi, visok stepen pouzdanosti u momentu kada se dostigne vreme isporuke
- Česte isporuke koda - omogućavaju da aplikacija ostane relevantna i dobro usaglašena sa potrebama klijenata.

54. **Kontinunualno poslovanje?**

Continuous business planning (continuous steering) omogućava kontinualno planiranje, merenje i uticaj poslovne strtegije i povratnih informacija od klijenta na razvojni ciklus aplikacije. Na ovaj način razvoj se koncentriše na aktivnosti koje donose najveći pozitivan efekat u datom momentu:

- Najvredniji i najriskantiniji zahtevi se "napadaju" prvi
- Predviđa se i kvantifikuje rezultat i resursi
- Pomaže da se dobro shvati šta klijent stvarno želi
- Omogućava agilno upravljanje razvojem.

55. **Kontinualna integracija i kolaborativni razvoj?**

Kontinualna integracija - opšteprihvaćena praksa pri kojoj se kod koji ceo tim razvija integriše u redovnim intervalima, kako bi se potvrdilo da sve zajedno dobro radi. Integracija se proverava automatizovanim build-om koji izvršava i regresionim testovima kako bi se eventualne greške brzo uočile. Pokazalo se da ovakav pristup značajno smanjuje probleme pri integrisanju komponenti prilikom realeasa.

Kolaborativni razvoj - omogućava saradnju između poslovodstva, razvoja, i QA kako bi se obezbedila konstantna isporuka novih funkcionalnih celina. Ovo podrazumeva i podršku za programiranje na različitim jezicima, razvoj na različitim platformama, razmenu ideja, kreiranje korisničkih priča i kompletno upravljanje razvojnim ciklusom.

56. **Kontinualno testiranje?**

Smanjuje troškove testiranja i istovremeno pomaže razvojnim timovima da izbalansiraju uložene napore, kvalitet rezultata i brzinu razvoja. "Uska grla" za testiranje se izbegavaju tako što se kreiraju virtuelizovani neophodni servisi, a kreiraju se i virtuelizovana test okruženja koja se mogu lako deployovati,deliti i ažurirati kako se celokupni sistem menja.

Prednosti kontinualnog testiranja:

- Izbegavanje neočekivanih poremećaja poslovanja
- Pouzdanije aplikacije
- Proaktivno rešavanje problema
- Efikasnije i brže rešavanje problema
- Manje ponovljenog posla
- Efikasniji i zdraviji timovi (valjda od manje stresa, pa zdraviji)
- Manji troškovi.

57. **Kontinualna isporuka?**

Predstavlja praksu koja omogućava da se kod može brzo i sigurno primeniti na produkciono okruženje, tako što se svaka promena isporučuje na okruženje koje je skoro identično produkcionom i rigoroznim automatizovanim testovima proverava se da aplikacije i servisi funkcionišu kao što je i očekivano. Nakon uspešnog testa na ovakvom okruženju, date izmene se na produkciju mogu prebaciti "na klik".

Prednosti ovog pristupa:

- Koriste se usvojene i standardne prakse za deployment
- Svaki deployment pokreće i sesiju standardnih tetsova
- Svaka izmena se automatski isporučuje na staging okruženje
- Isporuka koda na produkciju je moguća na zahtev.

58. **Kontinualni razvoj?**

Omogućava nadzor koji pomaže ravojnim timovima i testerima da shvate performanse i dotupnost aplikacija, čak i pre nego se isporuče u produkciju. Rana dostupnost ovih povratnih informacija je bitna za smanjenje troškova ispravljanja grešaka i nakandnih neophodnih izmena, i pomaže da se projekti uspešno privode kraju (kao npr da lepo vidimo kada se desi hysterix time out).

Prednosti kontinualnog nadzora:

- Ovo je vodeća praksa za kvalitetno upravljanje infrastrukturom, platformama, aplikacijama, kako bi se obezbedilo da funkcionišu optimalno
- Moguće je postaviti pragove koji definišu dozvoljene minimume i maksimume određenih veličina
- Svaki detektovani incident pokreće alarm ili pokušaj oporavka sistema
- Logovi se koriste za analizu funkcionisanja sistema
- Razvojni timovi mogu takođe pratiti ponašanje aplikacija i analizirati realno ponašanje aplikacije.

59. **Kontinualno prikupljanje povratnih informacija i optimizacija?**

Efikasan DevOps mora omogućiti brzo prenošenje povratnih informacija od klijenta. Na ovaj način razvojni tim dobija uvid u ponašanje korisnika kao i slabe tačke aplikacije koje korisniku predstalvjaju problem za korišćenje.

Prednosti:

- Mmogućava klijentu da stekne iskustvo korektnog prijavljivanja problema što pomaže optimizaciji
- Daje određeni nivo kontrole putem povratnih informacija svim zainteresovanim
- Omogućava identifikovanje važnih prijava - od ideja kako bi nešto moglo da radi do informacija o relanom korišćenju aplikacije
- Omogućava brz odgovor na zahteve
- Omogućava da održite kopetitivne prednosti aplikacije.

Cloud Computing

1. Šta je Cloud Computing? Osnovne osobine?

Cloud Computing je model isporuke IT servisa koji omogućava jednostavan, “na zahtev” pristup konfigurabulnim računarskim resursima. Obezbeđuje visok nivou apstrakcije procesnih resursa, kao i resursa za skladištenje podataka. Zasniva se na konceptu vizuelizacije.

Osobine:

- Samouslužni servis “na zahtev” - korisnik sam može odlučiti koje računarske resurse želi da pribavi. Postupak je automatizovan i nije potrebna aktivnost administratora sistema da se novi resurs alocira i dodeli korisniku.
- Heterogeni pristup – resursima se pristupa preko mreže (uključujući i Internet). Za pristup se koriste standardizovani mehanizmi, koji obezbeđuju pristup iz različitih vrsta klijenata.
- Objedinjavanje resursa (Resource Pooling) - resursi koji se nude se objedinjavaju u veće bazene resursa, koji mogu da opsluže više korisnika (multi-tenancy)
- On-demand - različiti fizički i/ili virtuelni resursi se dinamički dodeljuju i oslobađaju prilagođavajući se trenutnim potrebama korisnika.
- Upotreba servisa se kontroliše i meri – sistemi koriste alate za merenje performansi. Merenje i kontrolisanje služe za optimizaciju i naplaćivanje. Merenje i kontrola obezbeđuju predvidivo ponašanje računarske platforme, usmereno na to da u svakom momentu zadovolji potrebe korisnika.

2. **Tipični modeli Cloud sistema?**

- IaaS (Infrastructure as a Service) - obezbeđuju krajnjem korisniku infrastrukturne komponente (procesne module, kapacitete za smeštaj podataka, mrežne komponente i druge osnovne infrastrukturne komponente). Korisnik ove resurse koristi da na njima pokrene softver po svom nahođenju. Korisnik nema kontrolu nad hardverskim resursima, ali ima nad OS, alokacijom memorije, instaliranim aplikacijama, konfiguracijom mrežnih uređaja.
- PaaS (Platform as a Service) - korisniku se obezbeđuje mogućnost da na infrastrukturi koja je kreirana po korisnikovom zahtevu razvija, instalira i koristi aplikacije koje je kreirao koristeći alate i razvojna okruženja koje mu je Cloud provider obezbedio. Korisnik ne upravlja i ne kontroliše infrastrukturu. Korisnik upravlja svojim aplikacijama i konfiguracijom tih aplikacija. (Cloud za komentare je ovo)
- SaaS (Software as a Service) - korisniku se obezbeđuje mogućnost da koristi aplikacije koje je ponuđač usluga razvio, na Cloud infrastrukturi. Aplikacijama na Cloudu je moguće pristupiti preko različitih klijentskih aplikacija. Korisnik nema pristup, niti kontrolu nad infrastrukturom komponenata, serverima, OS, skladištenjem podataka. Jedino čemu ima pristup je podešavanje aplikacije koju koristi (Firebase gde držimo komentare je ovo, Google maps isto).

3. **Deployment modeli Cloud computing-a?**

- Private cloud – Cloud infrastruktura se koristi isključivo za potrebe jedne organizacije. Može biti u samoj organizaciji, ili u nekom data centru ponuđača usluge, ali je u celosti posvećena jednom korisniku.
- Community cloud – Cloud infrastruktura se koristi od strane više ogranizacija i podržava rad u određenim interesnim grupama sa zajedničkim ciljevima. Upravljanje može da vrši neka od članica ili treća strana.
- Public cloud – Cloud infrastruktura se obezbeđuje kao javni servis koji mogu da koriste svi, a upravljanje i kontrolu radi ponuđač usluge.
- Hibridni cloud – Cloud infrastruktura predstavlja kombinaciju neke od svih prethodnih modela.

4. **Pogodnosti Cloud Computinga?**

- Ne mora svaki pojedinačni korisnik da kupuje high-end opremu – Oprema u data centrima Cloud providera jeste high-end, ali je koristi veliki broj korisnika. Cena opreme koja se kupuje na veliko je povoljnija od one kada pokušate istu opremu da kupite pojedinačno.

- Ne moraju se kupovati resursi unapred – Planiranje resursa ne mora biti za dug period unapred. Smanjenje početnih troškova. Plaćanja se onoliko koliko se troši.
- Data centri obezbeđuju sigurno smeštanje podataka – podaci se po pravilu repliciraju (Firestore sa našeg projekta isto pravi repliku)
- Skraćuje se vreme da se obezbede resursi i pokrene aplikacija – nove resurse je moguće alocirati u nekoliko minuta.
- Pošto je bazen resursa veliki možete da zakupite na zahtev kakav god resurs želite i otkažate ga kada želite.
- Skaliranje je jednostavno – resursi se mogu dinamički alocirati kako bi odgovarali tekućim potrebama.

5. **Koje su tehnologije omogućile Cloud Computing?**

- SOA, Web servisi, RESTful servisi
- Širokopojasne mreže i Internet arhitekture – Svi Cloud sistemi isporučuju se isključivo putem mreže (javne ili privatne). Zbog potencijalo velikog broja korisnika i velike količine podataka, kao i potrebe da celokupna infrastruktura bude dostupna preko mreže, potrebna je velika propusnost mreže. Svi proizvođači Cloud usluga imaju sopstvene veze između data centara. Širokopojasne mreže i Internet zasnovani su na dva koncepta: paketna razmena podataka (podaci se dele na pakete ograničene veličine) i rutirane interkonekcije (ruter usmerava pakete tako da ne moraju svi paketi putovati istom rutom).
- Tehnologije virtuelizacije - Omogućava da se jedan fizički resurs namapira (najčešće na više) virtuelnih resursa. Moguće je virtuelizovati servere, kapacitete za čuvanje memorije, mrežne uređaje. Dve opcije prilikom virtuelizacije su da hipervizor bude deo host OS-a (kao kod VirtualBox) ili da hipervizor bude na hardware-u.
- Data-centri – Veliki data centri na jednom mestu smeštaju veliki broj servera i prateće opreme. Rad data centra zasniva se na virtuelizaciji, modularizaciji i standardizaciji, automatizaciji, upravljanju i održavanju sa udaljene lokacije.
- Web tehnologije – Web tehnologije se koriste i kao medij putem koga se obavlja isporuka servisa, i kao interfejs za upravljanje servisima. Intenzivno se koriste URL (za pristup resursima), HTTP (za razmenu podatak) i markup jezici. Aplikacije koje upravljaju Cloud servisima su Web aplikacije. Većina SaaS ponuda su Web aplikacije.
- Multitenancy – Omogućava da više korisnika istovremeno pristupe resursima. Aplikacije koje koriste ovakav pristup moraju da se pobrinu da postoji dobra izolacija. Korisnici mogu da pristupe samo svojim resursima. Ni pod kojim uslovima ne mogu videti ili menjati podatke drugih korisnika.