1. **Šta je softverska komponenta i koje su njene karakteristike?**

● Nezavisna jedinica softvera koja obuhvata neku zaokruženu funkcionalnost i može da su uklapa sa drugim komponentama.

● Karakteristike:

○ Reusability - mogucnost da se koristi u različitim situacijama i različitim aplikacijama

○ Zamenljivost - komponenta može da se zameni drugom komponentom

○ nezavisne od konteksta - komponente se dizajniraju da mogu da rade u različitim okruženjima i kontekstima

○ Prosirive - komponenta može da se proširi dodatnim funkcionalnostima

○ skrivena implementacija - spoljašnjem svetu je vidljiv interfejs i opis korišdenja komponente, a detalji implementacije i stanja unutrašnjih promenljivih su skrivena

○ nezavisne - dobro konstrisana komponenta ne zavisi mnogo od drugih komponenti

1. **Šta je modularnost? Razlika izmedju logicke I fizicke modularnosti na primeru iz Jave.**

● Modularnost je osobina softverskog sistema da se sastoji od skupa logicki nezavisnih delova - modula, koji imaju jasnu logicku granicu (programski kod moze da pripada moduli, ili da bude izvan njega). U svakom modulu postoji deo koji je vidljiv spoljašnjem svetu: public API, preko kog se modul ukljucuje u softverski sistem.

-Logicka modularnost: odnosni se na vidljivost softverskog koda. Fizicka modularnost se odnosi na nacin na koji je kod organizovan I spakovan za dalju distribuciju. Fizicki modul ne mora istovremeno biti I logicki modul. Dok vise logickih modula moze biti spakovano u jedan fizicki model.

* + Java klasa je primer logicke modularnosti - ogranicava vidljivost koda.
  + Paket u javi je primer logicke modularnosti – odnosi se na vidljivost softverskog koda.

1. **Pojmovi kohezija modula i coupling** 
   * + Kohezija - stepen povezanosti elemenata unutar modula. Treba teziti visokoj koheziji, modulima koji imaju jednu ulogu.
     + Coupling - stepen medjusobne povezanosti izmedju modula, treba teziti manjoj povezanosti izmedju modula i praviti sto nezavisnije module.
2. **Šta su build alati i koje operacije izvršavaju?**

● Alati za kreiranje izvršne verzije programa na osnovu izvornog koda. Izvrsna verzija moze biti desktop app, web app i softverske komponenta. 2 osnovne komponente: build skripta I executable. Operacije:

○ kompajliranje izvornog koda

○ kopiranje resursa

○ kompajliranje i izvrsavanje testova

○ pakovanje projekta

○ deploy

○ brisanje nepotrebnih resursa

○ upravljanje povezanim softverskim komponentama (dependencies)

1. **Šta su dependencies i koji su neki od najčešcih problema u upravljanju dependency-jima?** 
   * Softverske komponente koje koristimo za razvoj našeg softvera. Jedan od osnovnih zadataka build toolsa je kako da uravljaju dipendensijima.
   * ● Problemi - dependency hell, konfilikt verzija, tranzitivnost dependency-ja, shadowing
2. **Objasniti osnovne elemente ant skripta: target, task i property, navesti jedan primer za svaki od ovih koncepata.** 
   * Target - jedan element target može da zavisi od nekog drugog target elementa. zavisnost između targeta definiše redosled izvršavanja. sastoji se od pojedničanih taskova:

○ <target name=”init”>

○ <mkdir dir=”$(build)”/>

○ </target>

* + Task - Parce koda koji se izvrsava. Postoje ugradjeni taskovi (javac,delete,jar,mkdir) I taskovi koje sami pisemo.

○ <name attribute1="value1" attribute2="value2" ... /> <mkdir dir=”$(build)”/>

* + Property - promenljive koje se koriste u build fajlu u paru name/value. Mogu se navesti i eksterno

○ ${name}

1. **Objasniti osnovne pojmove maven-a: pom, plugin, goal**

● POM - XML reprezentacija Maven projekta (pom.xml). Svaki Maven projekat mora sadrzati ovaj fajl. U njemu su specificirani svi artifakti vezani za projekat(organizacija programskog koda, dipendensi za projekat, URL adrese, licence, konfiguracioni fajlovi…)

* + Plugin – Program koji moze da se pokrene sa mavenom. Svaki plugin podržava i neku dodatnu konfiguraciju. Svaki zadatak se izvršava preko maven plugin-a. 2 vrste:

○ build plugins - služe za pakovanje i kreiranje izvršne verzije projekta. konfigurišu se u elementu <build> u pom.xml (clean, compile, install, deploy, resources, jar, rar, war)

○ reporting plugins - generisanje izveštaja, dokumentacije. konfigurišu se u elementu <reporting> u pom.xml (Javadoc,changelog,project-info-reports)

* + Goal - predstavlja specifičan zadatak plugina koji je deo procesa pakovanja aplikacije. Svaki plugin ima svoje goolove. Goal je komanda plugina. Kada pokrecemo maven, u stvari pokrecemo odredjeni goal. Prilikom pokretanja goal-a maven proverava pom fajl i traži da li postoje dodatne konfiguracije za plugin kome pripada goal, i ako postoje njih koristi u izvršavanju goal-a

1. **Kako maven upravlja dependencijima? Šta je dependency scope?**

● Maven koristi lokalni repozitorijum biblioteka. Automatsko skidanje navedenih projekata i biblioteka u lokalni repozitorijum (ukoliko vec nisu tamo). Povlače se i dodatne biblioteke potrebne za rad biblioteka povezanih sa našim projektom (tranzitivne biblioteke)

* + Dependency scope je u kom delu "života" softvera de se koristiti biblioteka (compile – u toku kompajliranja, provided, runtime – u toku izvrsavanja, test – u testnoj fazi). Svaki opseg razlicito tretira dipendensije.

1. **Šta je optional depenedency i dependency exclusion kod maven-a?** 
   * Optional dependency

○ Projakt A -> Projekat B (projekat B je označen sa optional)

○ Projekat X -> Projekat A

■ projekat B nece biti uključen na classpath projekta X

○ štedi memoriju i sprečava neke probleme tranzitivnih depenedency-ja

* + Dependency exclusion

○ Koristimo ako želimo da isključimo neki od tranzitivnih dependency-ja

○ koristi se kada proizvođač dependency-ja nije postavio parametar optional

1. **Objasniti pojmove nasleđivanja i agregacije maven projekata.** 
   * Nasledjivanje - pogodno za razvoj modularnih aplikacija .Super POM je jedan primer nasleđivanja projekata i POM-ova (Super POM nasledjuju svi POM fajlovi koje pravimo. U super pom fajlu se navodi defaultna konfiguracija). Postoji mogucnost da I sami definisemo nasledjivanja izmedju pomova I projekata. Elementi POM-a koji se nasleđuju su (dependencies, developeri, lista pluginova i njihova konfiguracija, resources)
   * Agregacija

○ Koncept slican nasledjivanju

○ umesto specifikacije parent POM-a u modulima u parent POM-u se navode moduli

○ parent projekata sada "zna" koji su mu moduli i kada se pozove maven komanda nad parent projektom, ona će se izvršiti i nad modulima

1. **Objasniti osnovne pojmove gradle-a: plugin i task** 
   * Plugin je ekstenzija Gradle koji daje određenu vrstu templejta projekta i predefinisan skup zadataka koji su deo build procesa. Postoji veliki broj ugradjenih pluginova (java, android). Pisu se na pocetku skripte.
   * Task - jedan zadatak koji se izvršava u procesu build-a (npr. Kompajliranje izvornog koda). Pise se u grooviju. Taskovi mogu da zavisi jedan od drugog, I na taj nacin definisati raspored izvrsavanja ( ant target je ekvivalent gradle tasku).
2. **Šta je konfiguracija dependency-ja kod Gradle-a i navesti jedan primer korišdenja?** 
   * Svi dependency-ji u gradle build fajlu grupišu se u konfiguracije. Konfiguracije imaju ime, i još neke osobine, i mogu da nasleđuju jedna drugu.
   * configurations { compile { description = 'compile classpath'

transitive = true }}

1. **Šta u Gradle može biti dependency i navesti primere za dve različite vrste dependencyja.** 
   * komponenta iz repozitorijuma
   * drugi projekat u istom build procesu ● fajl

Primeri:

○ Drugi projekat: dependencies { compile project(‘:shared’) }

○ File: dependencies { runtime files(‘libs/a.jar’, ‘libs/b.jar’) }

1. **Šta su moduli u Python-u i kako se u definiše vidljivost na nivou modula?** 
   * Modul je py fajl koji sadrži Python definicije i naredbe. Veci programi mogu da se razdvoje na manje module (moduli mogu da se importuju u druge module).
   * Ne postoji klasican privatni indicator vidljivosti. Ako ima pocinje sa donjom crtom, ne preporucuje se da se koristi izvan modula (ne pripada javnom API-ju modula)

○ ne moze se importovati kad kazes `from file import \*`

○ ali ce se importovati ako kazes from file import \_var

1. **Koji alat se koristi za instalaciju Python komponenti i koji su koraci u instalaciji?** 
   * Pip. Instalira se zajedno sa pyhonom. Korisne komande: $pip install, uninstall, list, show, freeze, search
   * Koraci:

○ Identifikacija elemenata za instalaciju i obrada argumenata

○ Razresenje dependencya i odredjivanje sta ce se instalirati

○ Preuzimanje paketa

○ Instalacija paketa

1. **Čemu služi setup.py fajl I requirements.txt?**

**Setup.py** - fajl za specifikaciju Python komponente (specifikacija rejuzabilnit komponenti (biblioteka)). Sadrži metapodatke o komponenti neophodne za njenu distribuciju (ime, verzija, dipendensi). Apstrakna specifikacija dipendensija (nepotpuna). Pip koristi ovu specifikaciju za instalaciju dipendensija.

**Requirements**.txt – lista pip komandi za instalaciju aplikacija koje se izvrsavaju. Konkretnija specifikacija dipendensija.

1. **Šta je CommonJS i kako funkcioniše?** 
   * CommonJS - projekat čiji je cilj razvijanje alata i tehnika za korišdenje JavaScript-a izvan brauzera
   * CommonJS modul- rejuzabilno parce JavaScript koda koje eksportuje određeni objekat koji drugi moduli ili programi mogu da importuju preko komande require

Primer Node.js : var http = require(“http”)

1. **Šta je npm, od čega se sastoji i kako se koristi?** ● Npm je package manager za JavaScript ● sastoji se od:

○ klijenta za komandanu liniju

○ npm repozitorijuma besplatnih i pladenih paketa - npm registry

○ npm veb stranica - za korisničko pretraživanje registra

-Koristi se tako sto komandom ‘npm install’ instalira paket, skida se i smesta u poseban folder koji npm kreira. U programu se taj paket koristi preko ‘require’.

1. Inverzija kontrole – realizacija, prednosti, koriscenja

-IoC je princip u razvoju softvera gde je kontrola nad objektima ili delom programa prebacuje na neko odgovarajuce softversko okruzenje (framework ili container). Koristi se kod OOP-a. Umesto da nas program kontrolise tok izvrsavanja I poziva elemente, framework kontrolise tok programa I pozivva nas kod.

-Realizacija: kljucni concept jeste apstrakcija. Ideja: dodajemo ponasanje sistemu tako sto nasledimo apstrakte klase ili interfejse koji su deo frejmvorka. To nije jedini nacin. Pisannje klase mora da postuje odredjene konvencije kao Java bean. Realizuje se preko: Service locator pattern, Factory pattern, Depdendency injection, Strategy design pattern.

-Prednosti: odvajanje izvrsavanje zadatka od njegove implementacije; jednostavnija promena implementacije; bolja modularnost programa; jednostavnije testiranje delova programa; slabija povezanost.

1. DEPENDENCI INJECTION

Pojam depencendija izmedju klasa. Jedna objekat moze da ima druge objekte kao dipendensije u okviru istog projekta.

-Ideja: IOC se postoze tako sto se objektu dodeljuje njegov dipendensi (drugi objekat koji koristi). Umesto da objekat sam pravi isntacu drugog objekta koji mu je potreban, taj objekat mu se dodeljuje ‘spolja’.

-Prednosti: objekat klase ima klasu kao dipendensi, ali ne I konkretnu instance klase. Objekat joj se dodeljuje u runtime. Nedostatak je to sto promene u implementaciji su vidljive u runtime-u. Problemi u dipendensiju objekta se moze otkriti u runtime-u (NoClassDefinition error).

1. 3 VRSTE REALIZACIJE DI

* Constructor injection – objekat se prosledjuje u konstruktoru
* Setter injection – objekat se prosledjuje u set metodi
* Interface injection – klijent implementira interface koji ima set metodu koja kao argument prima servis objekta.

1. Distribuirana softverska arhitektura I njene alternative

-Softverska arhitektura zasnovana na servisima. Njeni primeri: servisno-orijentisana arhitektura I mirkoservisna arhitektura. Mogu se posmatrati I kao distribuirane arhitekture: udaljeni pristup komponentama koje obezbedjuju servis koriscenjem nekog protokola za udaljeni pristup: REST, SOAP, RMI.

-Njene alternative: monolitna arhitektura, slojevita softverska arhitektura (dvoslojna, troslojna).

-Prednosti: mogucnost prosirenja, odvajanje modula I slaba povezanost

1. **Objasniti osnovne koncepte servisno-orijentisane softverske arhitekture** 
   * dizajn softvera kod koga više softverskih komponenti zajedno odrađuje neku funkcionalnost. svaka komponenta odrađuje određeni zadatak – servis.
   * komunikacija između komponenti odvija se preko mreže korišdenjem nekog komunikacionog protokola
   * komponenta je realizovana kao "crna kutija" i jedino što korisnik servisa vidi je interfejs
   * Prednosti: (reusability, jednostavnije odrzavanje ili zamena komponente)
2. **Objasniti osnovne koncepte SOAP tehnologije: SOAP, WSDL i UDDI.** 
   * SOAP – specifikacija koja definise rutinu I format za prenost XML podataka izmedju aplikacija na mrezi. SOAP je nezavisan od OS-a I programskog jezika. SOAP poruka je XML document.
   * WSDL – tehnologija koja se koristi sa opis veb stranica (opis sadrzi sve sto je neophodno za koriscenje veb servica). Omogudava proizvođačima veb servisa da opišu tipove XML dokumenata i SOAP poruka koji se moraju koristiti da bi se pozvao njihov veb servis.

* UDDI – tehnologija koja omogucava otkrivanje veb servisa, UDDI definise skup operacija koje se mogu pozivati putem SOAP poruka (operacije za pretrazivanje I objavljivanje veb servica). UDDI je jedan od nacina za kreiranje servis registra.

1. **Šta je REST? Objasniti povezanost REST-a sa pojmovima resurs, HTTP, URI.**

* softverska arhitektura zasnovana na HTTP protokolu koja propisuje određena ograničenja i pravila
* Resurs - fokus REST-a je na reprezentaciji i pristupu resursima . može se kombinovati i više formata. Slican konceptu objekata u OOP. Reprezentuje se preko XML ili JSON
* HTTP - slanje poruka preko HTTP protokola. klijent šalje HTTP zahtev serveru, server vraca HTTP odgovor klijentu
* URI - URL je jedna vrsta URI. defniše način pristupa resursu. Pomocu URL-a se identifikuje lokacija resursa.

1. **Šta su mikroservisi i koje su prednosti mikroservisne softverske arhitekture?** 
   * softverska aplikacija se sastoji od manjih nezavisnih celina (softverskih komponenti, servisa), koji se nezavisno distribuiraju, svaka komponenta odrađuje određeni proces i komunicira sa drugim komponentama preko dobro definisanih mehanizama i na taj način se relizuje poslovna funkcija softvera ● Prednosti:
     + sloboda u izboru tehnologija

○ lakša promena tehnologija

○ brže iteracije (kreiranje nove verzije)

○ bolji kvalitet koda i čitljivost

○ jednostavnije organizovanje posla u timove i nezavisnost timova

1. **Koji su kriterijumi za izbor tehnologije za integraciju mikroservisa i koje su opcije integracije mikroservisa?** 
   * Kriterijumi

-tehnologija koja omoguditi da promene u servisu ne utiču mnogo na korisnike servisa

- izbegavati tehnike koje namedu korišdenje određene tehnologije za implementaciju servisa (API servisa treba da bude nezavisnastan od tehnologije - technology agnostic)

-servisi treba da budu jednostavni za korišdenje i klijenti treba da budu slobodni u izboru tehnologije

- omoguditi skrivanje detalja implementacije

* + Opcije

-pristup istoj bazi podataka

-komunikacija između servisa (sinhroni i asinhroni)

1. **Objasniti dva stila komunikacije između servisa: sinhroni i asinhroni.** 
   * sinhrona komunikacija - kada se pošalje poruka, čeka se na odgovor, pre nego što se nastavi sa radom. Na ovaj nacin znamo da li znamo da li je operacija uspesno obavljena.
   * asinhrona komunikacija - poruka se šalje i ne čeka se odgovor. Korisna je za operacije koje se dugo izvrsavaju.
   * Dva stila komunikacije:

–zahtev/odgovor: klijent salje zahtev I ceka odgovor. Obicno se koristi kod sinhrone kom.

–Zasnovan na dogadjaju: umesto da se trazi da se nesto uradi, klijent obavestava da se desio dogadjaj I ocekije da druge komponente znaju kako da reaguju-asinhrona kom.

1. Implementacija asinhrone komunikacija izmedju miktroservisa I

- jedan na jedan: jedan servis objavljuje poruku na red cekanja, drugi je procita

- publist/subscribe : servis koji objavljuje objavljuje poruku na topic, a vise servisa moze da dobije poruku.

-Implementacija: najcesce resenje je koriscenje message brokera. SIstemi za implementaciju brokera: Amazon SQS, Apache ActiveMQ, RabbitMQ, Apache Kafka

1. Dizajn sabloni za otpornost mikroservisa

Sabloni za otpornost resavaju problem odrzavanja servisa koji poziva drugi servis koji nije dostupan ili ima lose performance. Potreba za ovim sablonima je nastala zbog primrose mikroservisne arhitekture koja zahteva ucestalu komunikaciju izmedju servisa.

-Curcuit breakers: ukoliko poziv traje predugo vraca gresku. Ako ima veliki broj problematicnih poziva prelazi u stanje open. Kada je u stanju open, svaki put kada treba da pozove drugi servis, on vraca gresku umesto da ga pozove. Postoji I stanje half-open kada pokusava da pozove udaljeni resurs.

-Fallback: ako udaljeni servis otkaze, ne baca greske vec pokusava da nadje alternative (dohvatavanje podataka sa drugog servisa, ili stavlja poziv na cekanje)

-Bulkheads: svaki poziv na udaljeni servis se izvrsava u odredjenom thread poolu. Ako postoji problem sa pozivom, zatrpace se jedan thread pool, dok ce se ostali pozivi izvravati u svojim thread poolovima.

1. **Šta je API Gateway i koje su njegove prednosti?** 
   * omogudava da se aplikacija implementirana preko više nezavisnih servisa koristi kao jedna Celina. Klijent salje zahtev na jedno mest, a ApiGateway ih prosledjuje odgovarajucem servisu. ● Prednosti:
     1. klijent ne mora da ima informaciju o lokaciji svakog mikroservisa

○ ako se mikroservis promeni ne moraju se menjati svi klijenti

○ sigurnosni mehanizmi se implementiraju na jednom mestu i ne moraju biti implementirani u svakom mikroservisu

○ mikroservisi se fokusiraju samo na poslovnu logiku

**33.Objasniti dva načina registracije servisa u registru servisa?**

* + Samoregistracija - komponenta sama registruje servis. Instanca servisa je odgovorna za komunikacija sa servis registrom.Periodcno salje registru request kao dokaz da je active
  + registracija od strane trede komponente - posebna komponenta registruje servise. Ona vrsi I unregister, ukoliko je servis nedostupan.

1. **Objasniti dva načina za implementaciju pronalaženja servisa u registru servisa?** 
   * pronalaženje servisa na klijentskoj strani
     1. klijent komunicira sa registrom servisa

○ potrebna dodatna implementacija na klijentu

○ klijent je I servis koji poziva drugi servis

* + pronalaženje servisa na serverskoj strani
    1. load balancer: rasporedjuje pozive kako bi se efikasnije obradili. Moze da bude deo API Gateway-a

1. **Objasniti različite načine komunikacije korisničkog interfejsa sa mikroservisima.** 
   * Prva opcija: korisnički interfejs komunicira direktno sa servisima
     1. razmena XML ili JSON poruka preko HTTP protokola. Servisi ne mogu jednostavo da konfigurišu različite odgovore za različite vrste klijenata. Za neke klijente ovakav način komunikacije može biti opteredenje (mobilne)
   * Druga opcija: servis obezbeđuju gotove delove korisničkog interfejsa
     1. korisnički interfejs se sklapa od gotovih formi. Potrebna je implementacija koja sklapa ovakve UI komponente. Dobra raspodela posla po timovima. Neujednačenost delova korisničkog interefejsa
2. **Mikroservisi i baze podataka - problemi, izazovi i mogucnosti.** 
   * Problem - SQL Join

-u decentralizovanim softverskim sistemima treba izbegavati JOIN upite.

Rešenje sa više REST poziva često nije prihvatljivo jer zahteva veliki broj poruka između klijenta i servera pre nego što se korisniku prikažu podaci

- U mikroservisnoj arhitekturi SQL Join nije moguc. Jednoj tabeli pristupa samo jedan mikroservis I ni jedan mikroservis nema pristup svim tabelama. Moguce rešenje: kreiranje novog mikroservisa za timeline resurs (servis koji pristupa drugim servisima, prikuplja potrebne podatke I vraca klijentu). Prednost: protok info izmedju mikroservisa je brzi.

* + Problem - konzistentnost podataka
    1. u monolitnim aplikacijama: DBMS upravlja transakcijama i na taj način je obezbeđena konzistentnost

○ u distribuiranim aplikacijama ne postoji mehanizam koji ovo reguliše

○ decentralizovano čuvanje podataka zahteva dodatno upravljanje ovakvim situacijamam (kada se stanje izmedju 2 requesta promeni)

1. **Objasniti neke od osnovnih problema u dekompoziciji baze podataka prilikom prelaska na mikroservisnu softversku arhitekturu.** 
   * veze preko stranog ključa - primer: id knjige u tabeli za primercima (hocemo da napravimo poseban servis koji upravlja knjigama i poseban koji upravlja primercima)
   * ista tabela se koristi u više komponenti - primer tabela sa primercima se koristi kod unosa inventarnih podataka i kod izdavanja članovima
2. **Šta je distribuirana transakcija?** 
   * više povezanih transakcija kojima upravlja menadžer transakcija. Vodi se racuna da podaci ostanu u konzistentnom stanju, samo sto obde imamo vise procesa koji komuniciraju preko mreze. Najcesci algoritam: komit u dve faze.
3. **Izveštavanje i mikroservisi - problemi i neka rešenja.** 
   * povlačenje potrebnih podataka kroz API servisa
     1. za izveštaje koji zahtevaju veliku količinu podataka neizvodljivo

○ alati za kreiranje izveštaja zahtevaju podatke u određenom obliku

○ API vrlo često nije dizajniran za izveštavanje

* + servis šalje podatke u komponentu za izveštavanje
    1. utiče negativno na slabu povezanost

○ u slučaju izveštaja se toleriše

* + slanje podataka za izveštaje – event-based (kada se napravi, azurira, ili izbrise neki entitet, salje se poruka registrovanim servisima koji na to reaguju. Medu njima se moze naci I servis za izvestavanje).

1. **Objasniti šta je u OSGi-ju komponenta/modul?** 
   * U OSGI-u jedan modul predstavlja jedan bundle. Bundle je osnovni concept OSGI-a. To je fizicka jedinica modularnosti u obliku JAR fajla koji sadrzi class fajlove, resurse i metapodatke. Ne predstavlja celu app, vec modul koji sa drugim modulma cini aplikaciju.
2. **Kako OSGi upravlja dependency-jima?** 
   * Export-Package I Import-Package su osnova za OSGI dependency model. Bundle dependency resolution jeste process upravljanja dipendensijima (sofisticiran algoritam). Upravljanje dipendensijima je process uparivanja import paketa sa export paketima druge komponente na konzistentan nacin, tako da svaka komponenta ima pristup samo jednoj verziji druge komponente. Nakon procesa uparivanja, svaka komponenta se povezuje sa drugim komponentama od kojih zavisi (import paketi se poklapaju sa export paketima) i pravi se graf povezanih komponenti. Sve povezane komponente moraju biti instalirane u okruzenje, kako bi komponenta mogla da se pokrene.
3. **Objasniti životni ciklus OSGi komponente i klasu Activator.** 
   * Svaka komponenta ima zivotni ciklus koji se tipicno sastoji od: instalacije, pokretanja i stopiranja programa. Da bi se OSGI komponenta pokrenula, mora se prvo instalirati u OSGI okruzenje. OSVI okruzenje ima podrsku za upravljanje komplentim zivotnim ciklusom komponente. Komponenta se ne moze pokrenuti dok nisu instalirane sve njene povezane komponente (import paketi).
   * Bundle activator se specificira kao metapodatak u manifest fajlu i odredjuje sta se desava pri pokretanju i stopiranju komponente (bundle-a). Komponenta ne mora imati activator.
4. **Nacin funkcionisanja OSGI servisa?** 
   * OSGI framework obezbedjuje centralni registar servisa. To je mesto gde proizvodjaci objavljuju servise, a korisnici pretrazuju servise. Operacije nad registrom servisa: find, publish, bind.
   * Publish: potrebni podaci: ime interfejsa, klasa koja implementira interfejs, recnik metapodataka(opciono)
   * Find: postoji nekoliko nacina za pronalaznje: samo na osnovu imena I na osnovu imena I dodatnog upita nad metapodacima
   * Bind: koriscenje servisa – kada se servis pronadje iz registra, preuzima se njegova implementacija. Kada zavrsimo sa upotrebom registra, potrebno je pzovati ungetService
5. **Objasniti šta se definiše softverskom licencom.** 
   * pravni instrument koji regulise upotrebu i distribuciju softvera. Postoje 2 kategorije:

* Licence koje garantuju specificna prava korisnika softvera I odnose se na softver koji je u necijem vlasnistvu – komercijalni softver
* Licence slobodnog softvera I licenca otvorenog koda

Razlikuju se u nacinu dobijanja prava za izmenu I modifikaciju softvera. Sadrza prava I obaveze korisnika I izdavaoca licence.

1. **Objasniti način dodele verzija komponenti prema semantic versioning sistemu.** 
   * verzija se sastoji od tri broja: major, minor, patch

-Major - kada u APIju vrsimo promene tako da on nije kompatibilan sa prethodnim API-jem

- Minor - dodaju se funkcionalnosti ali je verzija kompatibilno sa prethodnim API-jem

- Patch – uglavnom se odnose na popravke bagova

- Pored ova tri broja se mogu dodati I neki izrazi koji oznacavaju faze u razvoju (snapshot(maven), alpha, beta). Softver koji koristi semant versioning mora deklarisati javni API u kodu ili u obliku dokumentacije. API mora biti razumljiv I precizan. Kada se objavi softver sa odredjenom verzijom, nikakve promene na toj verziji nisu moguce. Svaka promena se objavljuje pod novom verzijom.. Verzija koja pocinje sa 0 je izuzetak, stalne promene na kodu i API-ju su dozvoljene jer se odnose na inicijalni razvoj. Kada se promeni major, minor pocinje od 0, kada se promeni minor, patch pocinje od 0.

===================================================================

1. **Da li se mogu kombinovano koristiti Ant i Gradle u istom projektu?** 
   * Da
2. **Koji od navedenih build alata nema skript fajl u xml-u?** 
   * a) Apache Maven o b) Gradle o c) Apache Ant
3. **U kom jeziku se piše skript fajl u Gradle-u?** 
   * Groovy
4. **Koji build alat nema podršku za upravljenje dependency-jima?** o Ant
5. **Da li se ant-om može pakovati veb aplikacija?** o Moze
6. **Kako se definiše redosled izvršavanja target-a u ant skripti?** o Zavisnost između targeta(ima vrednost koja odredjuje sl. parametar) definiše redosled izvršavanja
7. **Šta radi slededi deo ant skript-a?**

**<path id="build.classpath">**

**<fileset dir="${libs}">**

**<include name="\*\*.\*.jar"/>**

**</fileset>**

**</path>** o Dodaje sve jar fajlove iz libs foldera.

1. **Šta je Apache Ivy?** o Alat za upravljanje povezanim softverskim komponentama U Ivy-ju je dodato upravljanje dependency-jima
2. **šta od navedenog je validna vrednost elementa packaging u maven-u?** a) plugin
3. pom
4. jar

**10. Šta od navedenog je obavezni element u pom.xml fajlu?** a) build

1. packaging
2. version
3. groupId
4. **Šta je Super POM?** o To je poseban POM fajl u kojem se nalazi default konfiguracija koja ce se koristiti ako u našem POM-u nije specificirano drugačije.
5. **Šta radi maven goal install?** o Instalira kreirane artefakte I smesta ih u lokalni repozitorijum
6. **Čemu služi element resources u pom.xml?** o Specificira koji resursi se koriste u procesu bildovanja aplikacije
7. **Objasniti razliku između elementa dependencies i dependencyManagement?**

Svi elementi u dependencies iz parent projekta se nasleđuju u child projektu.

Elementi iz dependencyManagement-a u parent projektu se nasleđuju samo ako su navedeni u POM-u child projekta .

1. **Dato je stablo dependencija maven projekata. Koja verzija biblioteke D ce se koristiti u projektu A na osnovu pravila dependency mediation?**

**A -> B -> C -> D 2.0**

**A -> E -> D 1.0**

o  U maven-u gledamo plice, bice 1.0. U gradle-u gledamo vecu.

1. **Kako se definiše redosled izvršavanja Gradle taskova?** 
   * Taskovi zavise jedan od drugog.
2. **Šta radi slededa Gradle naredba? configurations.runtime.each { File file -> println file.name }** o Ispisuje imena svih fajlova u runtime-u.
3. **Objasniti šta znači slededa Gradle specifikacija dependency-ja:**

**dependency{**

**compile project(':projekat')**

**}**

* + Dodali smo projekat sa imenom "projekat" kao compile dependency.

1. **Objasniti šta znači slededa Gradle specifikacija dependency-ja:**

**dependencies {**

**runtime files('libs/a.jar', 'libs/b.jar')**

**}**

* + Dodaje libs/a.jar', 'libs/b.jar kao runtime dependency.

1. **Šta je executable jar?** 
   * Jar fajl koji sadrzi putanju ka main klasi i moze da se pokrene.
2. **Koja je razlika između OSGi i Import-Package i java import-a?** o Import u Javi predstavlja importovanje paketa i tiče se upravljanja prostorom imena. OSGi Import-Package daje okruženju informaciju šta je komponenti potrebno od eksternog koda.
3. **Šta znače slededi fragment u OSGi manifest fajlu?**

**Export-Package: org.foo.shape,org.foo.other**

o Eksportuje pakete org.foo.shape,org.foo.other iz komponente. Omogucava da druge komponente importuju org.foo.shape,org.foo kao dependency.

1. **Šta znače slededi fragment u OSGi manifest fajlu?**

**Import-Package: org.mojaorg.paket; version="[1.3.0,2.0.0)"** o Importuje paket sa nekom od verzija u intervalu [1.3.0,2.0.0).

1. **Kako se pokrece OSGi komponenta?** 
   * Sve povezane komponente moraju biti "instalirane" o Komanda list lista sve komponente i njigove id-jeve o Komanda: `start id` pokrece datu komponentu
2. **Kako se registruje OSGi servis?** o Preko centralnog registar servisa
   * Metodom registerService iz BoundleContext-a. Prosledjuje se Interfejs i drugi parametri za koje ce se vracati konkretna implementacija pozivom metode getService

**28. Kako se postavljaju metapodaci za OSGi servis?**

U manifest fajlu. Podela na 3 skupa:

* Informacija za coveka
* Informacija o identifikaciji modula
* Informacija o import/export-u

1. **Kojom tehnologijom se predstavlja interfejs servisa u SOAP tehnologiji?** –XML, SOAP, UDDI, WSDL
2. **Šta čini dobar servis (komponentu)?** o loose coupling - promene u jednoj komponenti ne zahtevaju promene na drugim komponentama
   * high cohesion - jedna komponenta je zadužena za jednu stvar
3. **Koji su problemi u situaciji kada više mikroservisa koristi istu bazu podataka?** o sve komponente su vezane za detalje implementacije baze podataka, nametnuto je korišcenje određene tehnologije, mešanje logike u različitim komponentama
4. **Da li event-based komunikacija između mikroservisa spada u sinhronu ili asinhronu Komunikaciju?** o Asinhronu
5. **Koja HTTP metoda se u REST-u koristi za kreiranje resursa?** o POST

**1.Navesti 3 ugrađena taska u ant-u** - javac, mkdir, delete

1. **Šta je Super POM?**
   * To je poseban POM fajl u kojem se nalazi default konfiguracija koja ce se koristiti ako u našem POM-u nije specificirano drugačije.
2. **Objasniti šta radi navedeni fragment Gradle skripta. dependencies { runtime files(dir:'libs', include: '\*.jar')**

**}**

* + Dodaje sve jar fajlove iz libsa u runtime dependency

**4.Da li je paket u Javi primer fizičke i/ ili logičke modularnosti (obrazložiti odgovor)?**

* + Logicke - odnosi se na vidljivost softverskog koda

1. **Kako se u OSGi definiše javni API komponenta?** o Kao set klasa i metoda koje mozemo koristiti od ostalih komponenti. Ukoliko komponenta koristi API druge komponente, ima dependency na drugu komponentu (Import-Package). Ovo radi samo u slucaju da je ta druga komponenta omogucila koriscenje odredjenih klasa sa Export-Package.
2. **Pored naziva build alata navesti jezik za pisanje script fajla koji koristi taj alat.**

**Gradle -** Groovy

**Apache Ivy -** XML

**Apache Maven -** XML

1. **Koje su osnovne prednosti gradle-a u odnosu na ant i maven?** o Koristi Groovy (lakse za odrzavanje), fleksibilan je, dodavanje custom ekstenzija maven-u je vrlo složeno za razliku od gradle-a.
2. **Šta radi maven goal package?** o Predstavlja specifičan zadatak plugina koji je deo procesa pakovanja aplikacije
3. **Objasniti šta radi slededi fragment ant build skripte.**

**<target name="compile" depends="clean">**

**<mkdir dir="${bin}"/>**

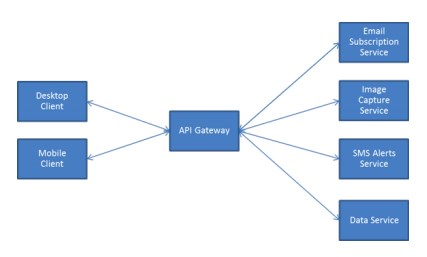
**<javac classpathref="compile.classpath" destdir="${bin}" srcdir="${src}" />**

**</target>** o Pravi task sa imenom "compile", koji ima dependency "clean"(koji mora da ide posle clean-a), pravi bin direktorijum, potom kompajluje klase iz source i stavlja ih u bin.

1. **Trenutna verzija našeg Java projekta ja 3.2.1, ako promenimo kod tako što dodamo tri public metode u jednu od klasa, koju verziju treba dodeliti tako izmenjenom projektu prema semantic versioning sistemu (obrazložiti odgovor)?** o Treca tacka su bug fix (bugs), druga tacka su nove promene, uz kompatabilnost sa proslom verzijom (minor), a prva tacka predstavlja release koji nije kompatabilan sa proslim.
   * Verzija bi bila: 3.3.1
2. **Objasniti razliku između javnog api-ja i jar fajla?**

Api predstavlja kako biblioteka izgleda spolja za program koji koristimo(interfejs je).Api je "lice" biblioteke drugim programima, predstavlja set public pristupa klasama, interfejsima i metodama iz biblioteke, dok je jar samo biblioteka koja se stavlja na class path.

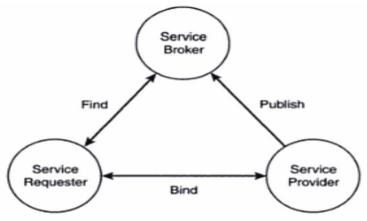
1. **Šta označava sledeci fragment u OSGi manifest fajlu? Import-Package: org.osgi.framework; version="(1.4.0,2.0.0]"** Import paketa koji se nalazi u range-u date verzije.
2. **Da li je Java klasa primer fizičke i/ ili logičke modularnosti (obrazložiti odgovor)?** 
   * Logicke - ogranicava vidljivost koda.
3. **Šta razlikuje executable jar od običnog jar fajla?** 
   * Executable jar sadrzi datoteku manifest.mf koja definise main klasu koja ce se izvrsiti kada se jar pokrene, dok je jar samo biblioteka koja se moze dodati u classpath.
4. **Objasniti šta je UDDI (Universal Description, Descovery and Integration) tehnologija.** 
   * UDDI koristi SOAP za komunikaciju i omogućava klijentima da pronađu servis, kao i serveru da ga objavi.
5. **Nacrtati primer mikroservisne arhitekture koja koristi API Gateway dizajn patern.**



1. **Kojoj konfiguraciji dependency-ja pripada sledeca specifikacija dependency-ja u Gradle-u? compile group: 'org.apache.commons', name: 'commons-csv', version: '1.3'** o konfiguracije compile, java plugina
2. **Čemu služi WSDL tehnologija?** 
   * WSDL je napisan XML-om i sadrži informacije gde se web servis nalazi i koji protokol za komunkaciju koristi. (Tehnologija koja se koristi za opis veb servisa.)Jedna od 4 tehnologije koja se koristi kombinovano sa xml,soap,uddi. Uloga mu je definisanje interfejsa.

**20. Objasniti osnovnu razliku između SOAP i REST tehnologije.** Rest otkriva resurse koji predstavljaju podatke, dok soap otkriva operacije koje predstavljalju logiku. Rest koristi HTTP (GET,POST,DELETE), dok soap koristi samo HTTP POST. Rest podrzava razlicite formate podataka, dok soap podrzava samo xml.

**21. Nacrtati osnovne elemente servisno-orijentisane arhitekture i objasniti komunikaciju između njih.**



|  |  |
| --- | --- |
| ▫ | Service provider - Razvija servis interfejs i implemenetira servis. Isporucuje servis tako da on postane dostupan za korišdenje preko mreže. Kreira opis servisa i objavljuje ga kod jednog ili više service broker-a. |
| ▫ | Service broker - registar servisa, obrađuje zahteve service provider-a da objave informacije o servisima koje obezbeđuju. Informacije koje se objavljuju uključuju tip servisa, opis servisa i lokaciju na kojoj se servis nalazi. |
| ▫ | Service requester - pretražuje registre da bi našao željeni servis (operacija find).Kada service requester pronađe servis, service broker mu više nije potreban. Koristi service broker da bi pronašao servise koji zadovoljavaju neki skup kriterijuma. |