**Dostupnost podataka bilo kad i bilo gdje se moze posmatrati kao preduslov za savremeno poslovanje, a osigurava se korištenjem mobilnih uredjaja cime se ostvaruju:**

* Uštede u poslovanju jer su cijene transakcija i interakcije mnogo manje
* Veće brzine reakcija na promjene i zahtjeve klijenata
* Redukcije grešaka nastalih prilikom obavljanja poslovnih aktivnosti

**XML**

1. XML se može posmatrati kao jezik za opisivanje podatak koji ne zavisi niti o hardveru niti o softveru.
2. XML **ne posjeduje** predefinisane tagove
3. Definiraš svoje tagove, dizajnirane specifično za svoje potrebe

**Web Servis**

1. Web servisi se mogu posmatrati kao **XML bazirani** sistemi za razmjenu podataka putem internet koristeći direktnu komunikaciju, sve u cilju osiguravanja **ponovne iskoristivosti** i **interoperabilnosti**
2. Također se mogu predstaviti kao dinamične, modularne i distribuirane aplikacije kojima se može pristupiti putem mreže u cilju dobijanja informacija o određenim resursima.
3. Ključne komponente Web servisa su:

* **SOAP** (Simple Object Acccess Protocol)
* **UDDI** (Universal Description, Discovery and Integration)
* **WSDL** (Web Services Description Language)

**WSDL (Web Services Description Language)**

1. WSDL je **XML bazirani** interface description language koji se koristi za opisivanje funkcionalnosti koje su ponuđene od strane web servisa.
2. WSDL akronim se također koristi za bilo koji specifičan WSDL opis web servisa (također često nazvanim WSDL fajlom), koji daje machine-readable opis o tome kako se servis može pozivati, kakve parametre očekiva, i kakve strukture podataka vraća.
3. Trenutna verzija je WSDL 2.0. Značenje akronima se promjenilo iz verzije 1.1 gdje je “D” stajalo za “Definition”

**SOAP**

1. Predstavlja **XML bazirani** protokol za razmjenu poruka, odnosno protokol za komunikaciju sa Web Servisom
2. Definiše set pravila o strukturi sadržaja poruke
3. Oslanja se najčesće na HTTP protokol
4. SOAP Building blocks:

* **Envelope** (Identificira XML dokument kao SOAP poruku) - **Obavezan**
* **Heade**r (Sadrži informacije o headeru) - **Nije obavezan**
* **Body** (Sadrži informacije o pozivima i odgovorima) - **Obavezan**
* **Fault** (Sadrži informacije o greškama nastalim tokom procesovanja poruke) - **Nije obavezan**

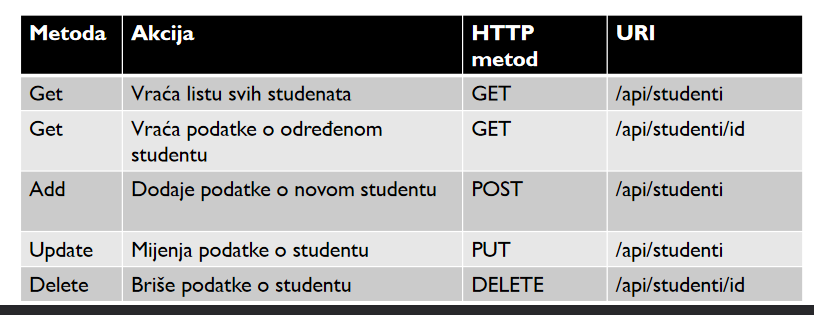
1. Soap ima tri glavne karakteristike:

* **Extensibility**
* **Neutrality** (SOAP can operate over any protocol such as [HTTP](https://en.wikipedia.org/wiki/HTTP), [SMTP](https://en.wikipedia.org/wiki/SMTP), [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol), [UDP](https://en.wikipedia.org/wiki/SOAP-over-UDP))
* **Independence** (SOAP allows for any [programming model](https://en.wikipedia.org/wiki/Programming_model))

**Web API**

1. API (Application Programming Interface) možemo predstaviti sa nekoliko stanovišta:

* Predstavlja okvir (Framework) za razvoj HTTP servisa kojima mogu pristupiti različite vrste klijenata
* Zasnovan je na **ASP.NET-u**
* Osnovna namjena je pružanje servisnog sloja za klijent-server, desktop, mobilne i druge vrste aplikacija
* Predstavlja posrednika u komunikaciji između dvije komponente/softvera
* HTTP podržava veliki broj uređaja i preglednika
* Ukoliko se resursima neke aplikacije može manipulisati korišenjem standardnih **HTTP GET; POST; PUT; DELETE** metoda, onda kažemo da ta aplikacija podržava API service
* Resursi se identificiraju koristeći odgovarajući **URI**



**JSON**

1. JSON definiše sintaksu za pohranu i razmjenu podataka
2. JSON je manje zahtjevan kada je u pitanju čitanje i pisanje podataka u odnosu na XML
3. JSON je sintaksa za serijalizovanje objekata, nizova, brojeva, stringova , booleana i null.
4. Primjeri svakog JSON Data Type-a:

* **STRING - { “Ime” : “Amer” }**
* **NUMBER - { “Iznos” : 540, “Postotak”: 85.47 }**
* **BOOLEAN - { “Primjer” : true }**
* **NULL - { “Primjer” : null }**
* **OBJECT - { “Objekat” : { “Ime”: “Petra”, “Godina”: 17, “Ocjena” : 10 } }**
* **ARRAY - { “Array” : [ “Prvi”, “Drugi”, “Treci” , “Cetvrti” ] }**
* **ARRAY (***Primjer sa key-pair***) -**  **{ “Array” : [ {“id” : 204, “Id2” : 305 } ] }**

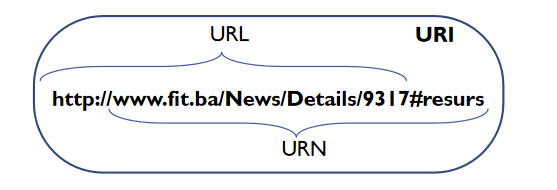
**--- PREDAVANJE - HTTP ---**

**Hypertext Transfer Protocol (HTTP)**

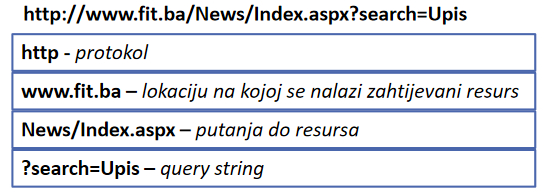
1. Protokol aplikacijskog sloja za prenos hypermedia dokumenata, kao što je HTML.
2. Dizajniran za komunikaciju između web browsera i web servera, ali može biti korišten i za druge svrhe
3. Prati klasični Client-server model, sa klijentom koji otvara konekciju da napravi zahtjev, i čekajući dok ne primi odgovor.
4. Stateless protokol, server ne zadržava podatke (state) između dva zahtjeva.

**URI = URL + URN**

1. **URI** = Uniform Resource Identifier
2. **URL** = Uniform Resource Locator
3. **URN** = Uniform Resource Name

****

**HTTP**

****

**QUERY STRING**

1. Posjeduje određena ograničenja:

* Ograničenje u dužini stringa
* Čitljivost sadržaja, ukoliko se ne implementiraju neki dodatni sigurnosni mehanizmi
* Predefinisano značenje pojedinih znakova

**HTTP METODE**

1. **GET** - Čitanje sadržaja - resursa
2. **POST** - Dodavanje novog resursa
3. **PUT** - Zamjena postojećeg resursa
4. **DELETE** - Uklanjanje resursa
5. **PATCH** - Promjena vrijednosti određenih atributa resursa
6. **HEAD** - Zahtijeva samo zaglavlje koje bi se dobilo GET upitom
7. **TRACE** - Zahtijeva više detalja o obavljenim akcijama
8. **OPTIONS** - Zahtijeva listu metoda koje podržava
9. **CONNECT** - Signal, najčešće proxy serveru, vezano za tuneliranje komunikacije (SSL)

**HTTP Status kodovi**

1. **1XX** - Poruke informativnog karaktera
2. **2XX** - Uspješna realizacija zahtjeva
3. **3XX** - Određena vrsta redirekcije
4. **4XX** - Greške uzrokovane klijentskim akcijama
5. **5XX** - Greške nastale na serverskoj strani

**Lista status kodova**

**100 - Continue** - Signalizira klijentu da nastavi slati zahtjeve

**200 - OK** - Zahtjev obrađen. Može vratiti resurs, al nije obavezno

**201 - Created** - Resurs je kreiran, najčešće se koristi kod POST metode. Vraća lokaciju na kome je kreirani resurs dostupan.

**204 - No Content** - Zahtjevani resurs ne postoji, a najčešće kod DELETE metode.

**301 -** **Moved permanently** - Zahtjevanom resursu je dodjeljen novi URI

**400 - Bad Request** - Zahtjevu nedostaju neke vrijednosti ili one nisu validne

**401 - Unauthorized** - Neovlašten pristup određenom resursu { Korisnik nije autorizovan ( tačnije authentifikovan) }

**403 - Forbidden** - Zabranjen pristup određenom resursu ( Korisnik je autorizovan (tačnije autentifikovan) , ne posjeduje privilegije za pristup)

**404 - Not Found** - Resurs nije pronađen

**500 - Internal Server Error** - Greška nastala na serveru je onemogućila obradu zahtjeva

**503 - Service unavailable** - Server je trenutno nedostupan zbog preopterećenja

**505 - HTTP Version not Supported** - Korištena verzija HTTP-a nije podržana od strane servera

Riješenje za problem čitljivosti sadržaja ?

1. HTTPS

* **SSL** - Secure Sockets Layer
* **TSL** - Transport Layer Security

**PREDAVANJE - RUTIRANJE/ANOTACIJE/VALIDACIJA**

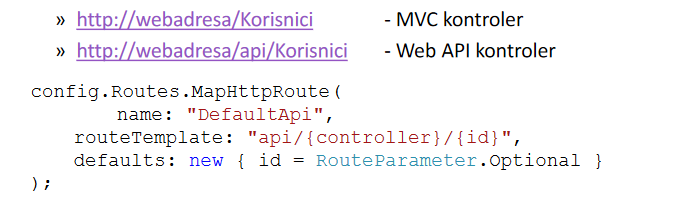
**ASP.NET Rutiranje**

1. Omogućava preslikavanje URI lokacije u pozive odgovarajućih metoda/akcija

* Web API kontroleri nasljeđuju **ApiController** klasu

1. Kontroleri predstavljaju klase koje obrađuju HTTP zahtjeve
2. Metoda **MapHttpRoute** omogućava dodavanje novog handler-a u infrastrukturu rutiranja

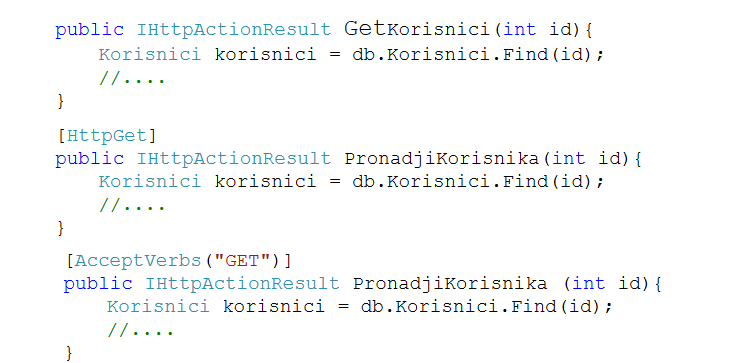
* Nakon prijema HTTP zahtjeva, Web API Framework pokušava mapirati URI sa nekom od putanja definisanih **routeTemplate-om**
* Riječ api u ruti ne predstavlja obavezni dio, ali predstavlja jedan od načina da se onemogući kolizija između Web API i uobičajenih MVC ruta.

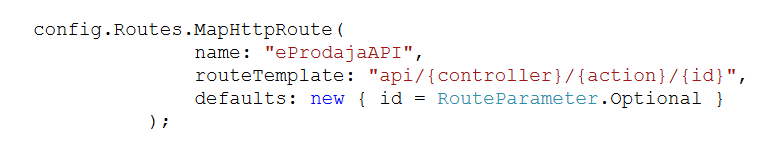


**U okviru ASP.NET Core-a, MVC i WebAPI su integrisan u jednu jedinstvenu cijelinu.**

1. Nakon prijema HTTP zahtjeva, Web API Framework pokušava mapirati URI sa nekim od zapisa u tabeli rutiranja
2. ASP.NET posjeduje ugrađenu konvenciju imenovanja metoda/akcija kontrolera

* GET zahtjev će se pokušati mapirati sa odgovarajućom metodom koja počinje sa riječju Get i sl.
* Pored ugrađenog mehanizma mapiranja na osnovu naziva metode, postoje i alternativni pristupi koji podrazumjevaju korištenje odgovarajućih atributa
* [HttpGet]
* [HttpPost]
* [AcceptedVerbs(“GET”)]
* [AcceptedVerbs(“GET”, “TRACE”)]

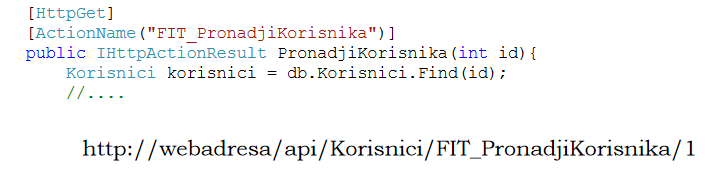
1. Koristeći URI <http://webadresa/api/Korisnici/1> moguće je pristupiti sljedećim akcijama:
2. U putanju se svakako može dodati i naziv akcije, ali je prethodno potrebno izvršiti njenu registraciju.



1. Nakon prethodne registracije, moguće je izvršiti direktni poziv akcija

* http://webadresa/api/Korisnici/PronadjiKorisnika/1

1. Korištenjem atributa **ActionName** akcijama se mogu dodjeliti i druga imena



1. U slučaju da je određenu metodu potrebno učiniti nedostupnom, na raspolaganju je **NonAction** atribut

**Pregovori**

1. WebAPI omogućava da se informacije o zahtijevanom resursu dobiju u različitim formatima:

* XML
* JSON
* BSON
* Korisnički definisani format

1. Odabir formata koji će se koristiti u komunikaciji se određuje u procesu pregovaranja (Content Negotiation), a podrazumjevani format je JSON

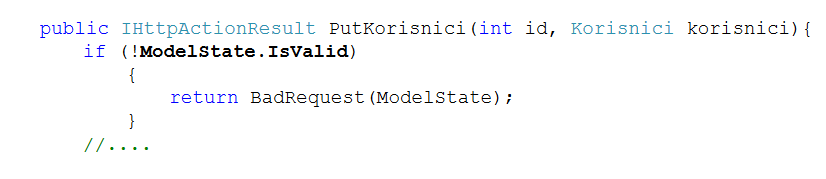
* Prilikom korištenja  **GET** metode, koristeći **Accept** polje u zaglavlju zahtjeva, moguće je navesti koji formati su prihvatljivi kao odgovor.
* Prilikom **dodavanja** resura, koristeći **Content-Type** polje u zaglavju zahtjeva se navodi format podataka koji se nalaze u tijelu zahtjeva.

**Validacija**

1. Validacija podataka se vrši prilikom svakog zahtjeva

* Izostavljanje obaveznih polja
* Prekoračenje opsega
* Definisanje ključnih polja

1. Eventualne greške se akumuliraju u okviru ModelState-a, te se na jednostavan način mogu provjeriti



1. Provjera validnosti se prije svega vrsi na osnovu anotacija koje su navedene unutar konkretnog modela
2. Anotacije u značajnoj mjeri olakšavaju proces provjere validnosti pojedinih vrijednosti unutar modela, a neke od značajnih su:

* CreditCard
* EmailAddress
* Min i Max
* Phone
* Range
* Required
* URL

1. Pored anotacija, na validnost modela se može direktno utjecati i unutar samih akcija korištenjem metode **AddModelError**

**---PREDAVANJE - HELPER METODE I RUKOVANJE IZUZECIMA---**

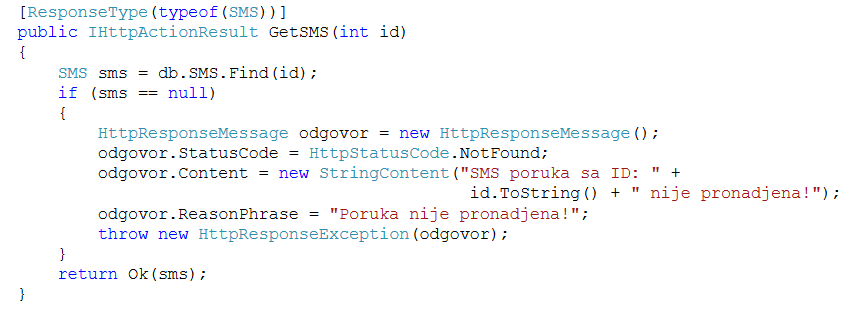
**Helper metode**

1. U odnosu na **ActionResult** koji je često korišten povratni tip akcija u ASP.Net MVC-u, kod WebAPI-a se kao povratni tip veoma često koristi IHttpActionResult čime je omogućeno korištenje *helper* metoda:

* BadRequest
* NotFound
* Ok
* Created
* CreatedAtRoute
* Conflict
* InternalServerError
* StatusCode
* Redirect

1. Zahvaljujući helper metodama moguće je, na vrlo jednostavan način, korisnika informisati o (ne)uspješnosti zahtjeva.

**Rukovanje Izuzecima**

1. Uobičajeno je da izuzeci nastali tokom izvršenja akcije rezultiraju vraćanjem HTTP poruke sa statusnim code-om 500.
2. Na rezultat izuzetaka se svakako može utjecati kreiranjem objekta **HttpResponseException**, te modifikovanja statusnog code-a.
3. Za potrebe **HttpResponseException-a** se može kreirati kompletan **HttpResponseMessage** objekat čime se detaljnije utječe na sadržaj odgovora.
4. Kada je riječ o izuzecima, objekti tipa HttpResponseException imaju poseban status zbog činjenice što su kreirani sa ciljem vraćanja odgovarajuće HTTP poruke.
5. Za potrebe rukovanja drugim izuzecima koji nisu obrađeni (API u takvim slučajevima obicno vraća odgovor InternalServerError koji ne daje baš dovoljno opisne poruke) na raspolaganju su filteri za rukovanje izuzecima.
6. Za kreiranje filtera potrebno je:

* Kreirati klasu koja će nasljediti **ExceptionFilterAttribut** klasu
* Implementirati (override) **OnException** metod
* Registrovati kreiranu filter klasu na nekom od sljedećih nivoa:
* Akcija
* Kontroler
* Globalno

1. Registracija filtera na globalnom nivou se vrši unutar WebApiApplication klase (Global.asax), odnosno njene Application\_Start metode.

**---PREDAVANJE - AUTENTIFIKACIJA & AUTORIZACIJA---**

1. Autentifikacija

* Identitet korisnika
* Ko si ti?

1. Autorizacija

* Privilegije korisnika
* Čemu korisnik može pristupiti?

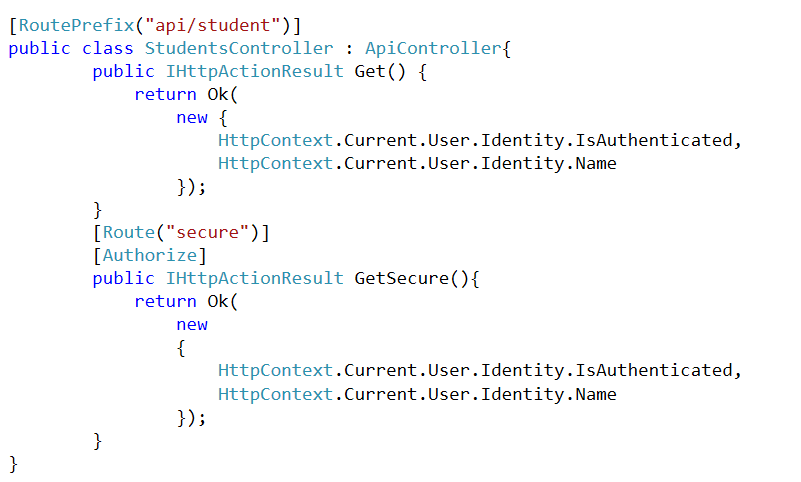
**Autentifikacija**

1. Autentifikacija se obično realizuje koristeći jedan od sljedećih pristupa:

* Individualni nalozi - prilikom pristupanja resursima koriste se nalozi pohranjeni unutar baze podataka aplikacije zadužene za autentifikaciju
* Organizacijski nalozi (poslovni ili školski nalozi) - prilikom pristupanja resursima koriste se nalozi pohranjeni u okviru lokalnog ili Cloud aktivnog direktorija
* Windows autentifikacija - kod ovog pristupa se konfiguriše Windows Authentication IIS modul koji se pokreće prilikom autentifikacija

1. U slučaju korištenja individualnih naloga vrlo jednostavno je moguće provjeriti da li je korisnik autentificiran. Na narednoj slici su prikazane dvije akcije, a pojašnjenje će se bazirati na pretpostavci da korisnik nije autentificiran.

* Prva akcija neće vratiti ime korisnika pošto ne postoje informacije koje bi mogla vratiti
* Druga akcija (na adresi: api/student/secure) će automatski izvršiti preusmjeravanje korisnika na dio aplikacije koji je zadužen za autentifikaciju



1. Autentifikaciju je moguće implementirati kreiranjem vlastite klase (npr. FitHandler), a tom prilikom je potrebno provesti sljedeće korake:

* **FitHandler** treba da nasljedi klasu **DelegatingHandler**
* Implementira vlastitu verziju **SendAsync** metode
* Unutar **SendAsync** metode se izvrši provjera pristupnih podataka koji su pristigli unutar zaglavlja **HttpRequestMessage-a**
* Ukoliko zaglavlje sadrži potrebne vrijednosti onda se za korisnika kreira odgovarajući Identity i Principal

1. Klasa **GenericIdentity** omogućava jednostavan način kreiranja identifikacije korisnika, te kao parametar obično upotrebljava korisničko ime

* **GenericIdentity** identitet = **new** **GenericIdentity**(“IB150051”);

1. Nakon identiteta moguće je kreirati principal (vlasnika) kome se šalje identitet i lista uloga

* **String**[] uloge = **new** **string**[] {“student”,”moderator”};
* **GenericPrincipal** principal = **new** **GenericPrincipal**(identitet,uloge);

1. Nakon kreiranja FitHandler klase potrebno je izvršiti njenju registraciju na jednoj od sljedećih lokacija:

* Global.asax fajl - GlobalConfiguration.Configuration.MessageHandlers.Add(new FitHandler());
* WebApiConfig fajl - config.MessageHandlers.Add(new FitHandler());

1. Često korišteni mehanizam autentifikacije se bazira na korištenju sigurnosnog token-a. Token bazirana autentifikacija posljeduje sljedeće karakteristike :

* Sigurnosni token se obično generiše od strane autorizacijskih tijela
* Sigurnosni token se generiše na osnovu podataka korisnika
* Sigurnosni token omogućava pristup određenim resursima
* Sigurnosni token se šalje u zaglavlju zahtjeva

**JSON Web Token**

1. Jedan od mogućih implementacija token bazirane autentifikacije je JWT - JSON Web Token
2. Sigurnosni token predstavlja strukturu podataka koja sadrži informacije o svom izdavaču, potpisu i vremenu trajanja
3. JSON Web Token je obično sastavljen od dva dijela:

* Zaglavlja (header)
  + Metapodaci
  + Korišteni algoritmi i ključevi
* Tvrdnje (claims)
  + Izdavač token-a ( issuer - iss)
  + Razlog izdavanja (audience - aud)
  + Vrijeme izdavanja (issuedAt - iat)
  + Trajanje (expiration - exp)

Struktura JSON Web Tokena :



**Zaglavlje i tvrdnje su base64 enkodirani stringovi, a potpis base64 enkodirani string zaglavlja i tvrdnji.**



**OAuth - Open Authorization**

1. Predstavlja protokol koji aplikacijama omogućava da koristeći jedan korisnički nalog dobiju pristup podacima koji se nalaze u okviru različitih sistema ili lokacija. Drugim riječima, korisnici su u mogućnosti pristupati različitim resursima koristeći svoj Facebook,Gmail,Twitter ili neki drugi račun.
2. U okviru OAuth protokola se mogu prepoznati sljedeći akteri:

* oAuth klijent - Aplikacija koja pristupa ili zahtijeva kredencijale korisnika
* oAuth provider - Facebook, Gmail, Twitter i sl.
* Korisnik - vlasnik kredencijala
* Resurs - resursi kojima korisnik želi pristupiti

**---PREDAVANJE 6 - XAMARIN / PCL ---**

**Tradicionalni Xamarin**

1. Neki od dijelova koda koji su kandidati za dijeljenje između projekata su:
   1. Poslovna logika
   2. Komunikacija sa bazom podataka
   3. Komunikacija sa web servisima
   4. Parsiranje podataka
   5. …
2. 60-90% dijeljenog koda
3. 10-40% koda specifičnog za platformu
4. Kako dijeliti UI kod zajedno sa poslovnom logikom? Odgovor je Xamarin.Forms.

**Xamarin Forms**

1. Osigurava set dijeljenih kontrola za kreiranje korisničkog interfejsa
2. Korištene kontrole se prevode u native kontrole platformi na kojima se izvršavaju

**Xamarin.Forms vs Xamarin.Android/IOS/UWP**

1. Forms
   1. Primarno unos podataka
   2. Prototipska aplikacija
2. IOS/Android/UWP
   1. Perfekcija u dizajnu interfejsa
   2. Korištenje funkcionalnosti vezanih za specifičnu platformu

**Dijeljeni (Shared) projekat**

1. Fajlovi koji su sadržani unutar dijeljenog projekta se kopiraju u projekat svake platforme (Android,IOS…), te kompajliraju za konkretno okruženje
2. Zahvaljujući mogućnosti zvanoj conditiona compilation, unutar dijeljenog projekta je omogućeno korištenje dijelova koda koji su specifični za pojedine platforme

**PCL (Portable Class Library)**

1. Predstavlja biblioteku koju mogu koristiti različite vrste aplikacija (windows, web, mobilne, konzole, itd)
2. Prilikom kreiranja PCL biblioteke vrši se odabir platformi na kojima će se biblioteka koristiti
3. Povećanjem broja odabranih platformi smanjuje se broj dostupnih API-ova jer se unutar PCL biblioteke može nalaziti samo onaj programski code koji je dostupan na svim odabranim platformama
4. U novijim verzijama, za dijeljenje programskog koda, je dostupna i opcija .Net Standard koji predstavlja set API-a dostupnih na svim .Net platformama

**Dijeljeni vs PCL pristup**

1. **Dijeljeni Prednosti**:
   1. Moguće je dodati dijelove koda koji su specifični za pojedine platforme
   2. Može sadržavati različite vrste fajlova
   3. Dostupni su svi API-ovi
2. **Dijeljeni Nedostaci**:
   1. Zbog načina kompajliranja isporučuje se u izvornom obliku
   2. Otežano izvršenje unit testova
   3. Može dovesti do kreiranja projekta koji je dosta teško održavati i nadograđivati
3. **PCL Prednosti**:
   1. Isporučuje se u binarnom formatu
   2. Pogodan je za unit testove
   3. Podržava postavljenu arhitekturu aplikacije
4. **PCL Nedostaci**:
   1. Nije pogodan za dodavanje dijelova koda koji su specifični za pojedine platforme
   2. Nije pogodan za dijeljenje fajlova koji ne sadrže programski kod
   3. Ograničen set dostupnih API-ova

**---PREDAVANJE 7 - DEVOPS---**

**Pregled stanja**

1. Klijenti očekuju da se u što kraćem roku isporuči određena vrijednost
   1. Brzina i kvalitet su ključ, a vrijeme je novac
2. Ispunjenje očekivanja je veoma često nemoguće realizovati zbog poteškoća sa isporukom
   1. Na određene servere se čeka danima
   2. Programeri ne mogu izvršiti isporuku posljednje razvijenih dijelova aplikacije
   3. Testeri ne mogu na vrijeme izvršiti potrebna testiranja, pa se propusti uočavaju u produkcijskom okruženju
   4. Kašnjenja i greške ne ulijevaju povrjerenje klijentima

**Neadekvatno trošenje resursa**

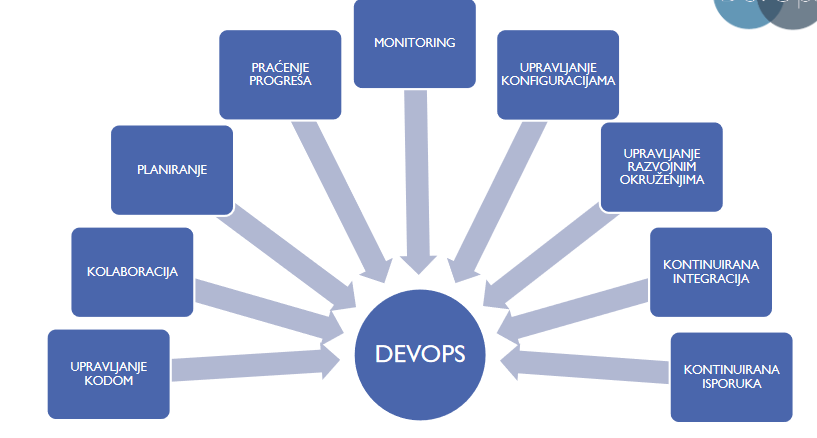
1. U okvir velikog broja kompanija se može indetifikovati trošenje velike količine resursa, a da tom prilikom ne dolazi do kreiranja nove vrijednosti. Kao najčešći uzroci neadekvatnog trošenja resursa mogu navesti:
   1. **Neadekvatna komunikacija** - neadekvatna ili čak nepostojeća komunikacija između pojedinaca, timova ili sektora (marketing, razvoj, isporuka itd.); česti i neproduktivni sastanci; iscrpni i nepregledni izvještaji
   2. **Neprihvatljiva čekanja** - ukoliko u procesu razvoja softvera bilo koji od timova ne realizuje svoje obaveze na kvalitetan način, to će zasigurno utjecati i na ostale učesnike
   3. **Nepotrebna alokacija resursa** - ukoliko svijesni da IT odjelu trebaju 2 dana da određenom timu učini dostupnim neki server, onda ćemo vjerovatno u startu zahtjevati više servera nego što nam je potrebno kako bismo izbjegli nepotrebna čekanja (mada postoji velika mogućnost da se zahtijevani serveri većinu vremena neće koristiti)
   4. **Veliki broj grešaka** - mnogo vremena se provodi na korigovanje grešaka koji su se pojavili u produkciji

**DevOps**

1. DevOps predstavlja koncept koji treba premostiti procjep koji evidentno postoji između Dev i Ops sektora kompanije
2. Rješavanjem prethodno identifikovanih problema moguće je u značajnoj mjeri unaprijediti performanse kompanije, te na taj način osigurati:
   1. 200 x češću isporuku softvera ( dodatnu vrijednost )
   2. 24 x brži oporavak u slučaju nepredviđenih zastoja u radu aplikacije
   3. 3 x manji neuspijeh prilikom korekcija na aplikaciji
   4. 22% manje vremena provedenog u okviru neplaniranih aktivnosti
   5. 50% manje vremena provedenog na sanaciji sigurnosnih propusta
3. DevOps **nije** softver ili tehnologija, već **način razmišljanja** o razvoju softvera koji se fokusira na **automatizaciji** svih procesa u cilju **eliminisanja** potrebe za manuelnim intervencijama
4. DevOps **zajedno sa** nekom od **agilnih metodologija** predstavlja idealnu kombinaciju za razvoj i isporuku softvera
5. Cilj DevOps-a je osigurati **kreiranje nove vrijednosti** kroz unapređenje postojećih procesa.
6. DevOps se zasniva na Lean pristupu koji zastupa sljedeće principe:
   1. Fokus na vrijednostima koje su značajne korisniku
   2. Eliminisanje otpada
   3. Reduciranje vremena potrebnog za određenu aktivnost
   4. Dijeljenje znanja
   5. Održanje kontinuiteta
7. Pitanje kojim je moguće saznati koliko je neka kompanija suštinski implementirala DevOps glasi:

* “Koliko je vremena potrebno da se određena modifikacija na softveru isporuči u produkcijsko okruženje”

1. Automatizaicja, odnosno eliminisanje intervencije pojedinaca ili grupa gdje god je to moguće, predstavlja osnovni koncept implementacije DevOps-a
2. Kompanije koje bi željele implementirati DevOps unutar poslovnih procesa, svoju pažnju bi trebale usmjeriti na unapređenje ključnih koncepata:
   1. Kulture
   2. Automatizacije
   3. Monitoringa
   4. Dijeljenja
3. Prilikom implementacije DevOps-a potrebno je prije svega napraviti reorganizaciju u kontekstu promjene pristupa ili kulture
   1. Značaj pojedinca - Problemi se prevazilaze zajedničkim radom, a ne identifikacijom pojedinaca pokazujući prstom na osobu koja treba podnijeti cijeli teret slučaja koji proživljava u tom momentu
   2. Odgovornost pojedinaca i tima - jedan tim svoje aktivnosti u potpunosti oslanja na odgovornost drugih timova i pojedinaca sa kojima radi na projektu
   3. Timski rad i povjerenje - slučajevi u kojima samo jedan tim, za razliku od ostalih timova, radi sinhronizovano nije neki pretjeran uspjeh. Timovi stalno trebaju komunicirati i sarađivati, te unapređivati jedni druge. U tom smislu veliki značaj imaju: Zajednički sastanci, druženja, ručkovi i sl.
   4. Stalno učenje - dijeljenje znanja i stalno izgrađivanje, kako pojedinačno tako i timski (konferencije, online kursevi, kursevi unutar kompanije i sl.), prezentovanje identifikovanih problema i pristupa njihovom riješavanju.
   5. Vrijednosti - potrebno je voditi računa o vrijednostima koje se prepoznaju i/ili nagrađuju (kašnjenje na posao i kvalitetan rad ili dolazak na vrijeme i mnogo nedostataka u radu)



**Upravljanje programskim kodom**

1. Korištenje repozitorija unutar kojih se objedinjuje programski kod svih učesnika u projektu predstavlja jedan od osnovnih preduslova razvoja savremenih aplikacija
   1. Čuvanje kopije programskog koda
   2. Historija svih promjena
   3. Vraćanje na neku od ranijih verzija
   4. Kreiranje branch-ova i njihovo ponovno spajanje (merge)
2. Generalnija podjela pomjenutih repozitorija je na:
   1. **Centralizirane** - TFS
   2. **Distribuirane** - Git

**Kolaboracija**

1. Kolaboraciju je moguće osigurati na veliki broj načina, a neki od njih su:
   1. Standup meetings
   2. Skype meetings
   3. Chat grupe (npr. Slack)
   4. Formiranje repozitorija znanja (blogovi, github wiki i dr.)

**Planiranje**

1. Transparentnost i informisanost svih učesnika projekta o budućim/planiranim aktivnostima osigurava kretanje informacija, najčešće u realnom vremenu, te podstiče na produktivnost i vizionarski rad.
2. Korištenje savremenih alata u procesu planiranja (Trello, VSO i dr.) je u potpunosti eliminisalo potrebu za dijeljenjem informacija ovog tipa putem email poruka ili dokumenata.

**Issue Tracking - Praćenje progresa**

1. Svi učesnici projekta bi trebali biti informisani o propustima na projektu koji se trenutno pokušavaju sanirati. Na taj način se spriječava degradacija i gubljenje, ponekad značajnih informacija o projektu.
2. Pored **VSO**, jedan od alata koji posjeduje mnogo širi spektar funkcionalnosti je **JIRA**.

**Monitoring**

1. Praćenje trenutnog stanja i analiza potencijalnih uzroka zastoja u radu softvera, veoma često nije jednostavan zadatak. Međutim, sve pomenute aktivnosti su u značajnoj mjeri olakšane korištenjem alata kao što su **Logstash i Kibana** ( prikupljanje, analiza i vizualizacija rezultata obrade log fajlova prikupljenih sa različitih lokacija)

**Upravljanje konfiguracijama**

1. Upravljanje konfiguracijama se veoma često posmatra kao sinonim za DevOps?
2. Različita okruženja veoma često zahtijevaju i različite postavke tj. Konfiguracije (postavke operativnog sistema, sigurnosne postavke i sl.).
3. Promjene napravljenje u jednom okruženju se veoma često ne propagiraju na druga okruženja što dovodi do zastoja u radu softvera, a neki od alata koji omogućavaju automatizaciju upravljanja konfiguracijama su:
   1. Chef
   2. Puppet
   3. Microsoft Powershell



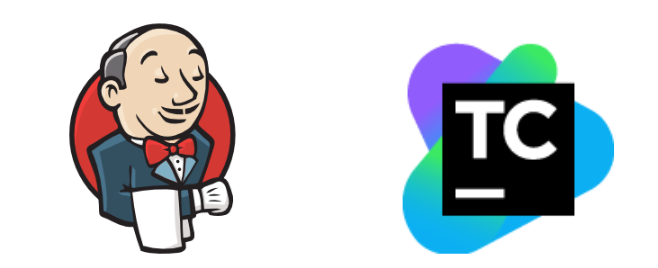
**Upravljanje razvojnim okruženjima**

1. Sinhronizacija postavki razvonjih okruženja svih članova tima koji učestvuju na određenom projektu u značajnoj mjeri eliminiše komentar: “Na mom računaru je radilo”
2. Jedan od alata koji omogućava upravljanje virtualnim okruženjima sa fokusom na automatizaciju je **Vagrant**

****

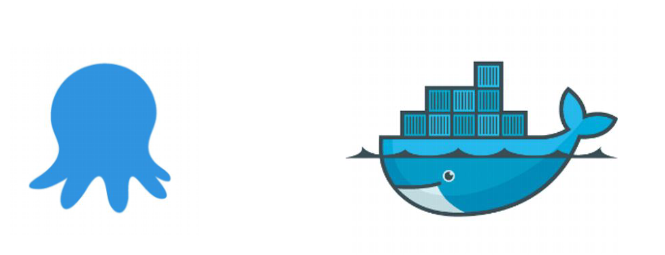
**Kontinuirana integracija**

1. U okviru ove aktivnosti se sve razvijene komponente povezuju u jednu cijelinu, te na taj način pripremaju za produkcijsko okruženje. Prilikom integracije značajno mjesto pripada i različitim vrstama testova. Neki od alata koji proces integracije značajno olakšavaju su:
   1. Jenkins
   2. TeamCity



**Kontinuirana isporuka**

1. U slučaju da je kompanija u potpunosti integrisala sve DevOps koncepte u svoje poslovne procese, promjene napravljene na softveru se mogu pustiti u produkciju u svega par minuta.
2. Kontinuiranu isporuku osiguravaju alati kao što je :
   1. Octopus
   2. Docker



**---PREDAVANJE 8 - DOCKER ---**

**Infrastruktura poslovanja ( Old School )**

1. Pristup : Jedna aplikacija, jedan server
2. Negativni aspekti ovog pristupa je neiskorištenost resursa jer se gotovo uvijek nabavlja konfiguracija koja može zadovoljiti mnogo veće potrebe od stvarnih ( često je iskoristivost kapaciteta iznosila najviše 30%)

**Infrastruktura poslovanja ( Hipvervizor )**

1. Pojava hipervizora koji je omogućio kreiranje i izvršavanje većeg broja virtuelnih mašina u okviru jednog fizičkog uređaja je označila novu eru u iskoristivosti infrastrukturnih kapaciteta
2. Pored velikog broja prednosti, ovaj pristup je imao određene nedostatke:
   1. Svaka virtualna mašina zahtijeva svoj operativni sistem (licence, hardverski resursi itd.)
   2. Svaka virtualna mašina zahtijeva održavanje (nadogradnju, sigurnosne postavke, monitoring itd.)

**Infrastruktura poslovanja ( Docker )**

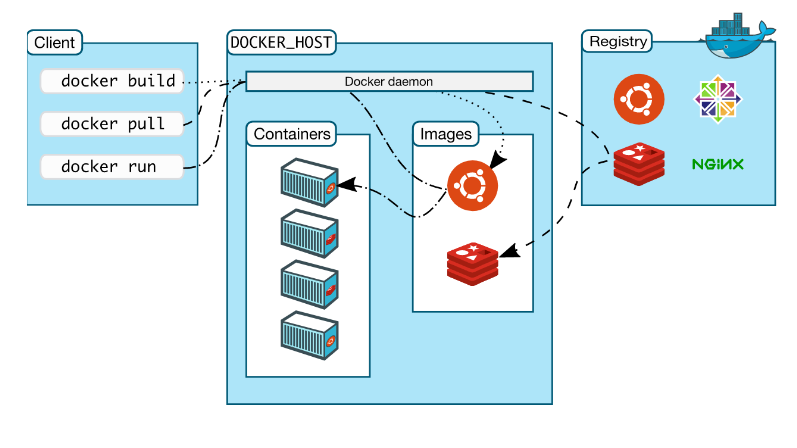
1. Docker je open-source projekt koji **automatizira** isporuku aplikacija unutar softverskih **kontejnera**, pružajući dodatni sloj apstrakcije i automatizacije operativnog sistema.
2. Analogija je preuzeta od transportnih kontejnera koji su uvijek isti, bez obzira na sadržaj, te na taj način odgovaraju svim kamionima, dizalicama, brodovima i drugim sredstvima i uređajima u procesu.
3. Neke od prednosti koje docker sa sobom donosi u područje razvoja softvera su:
   1. **Postavljanje razvojnih okruženja je dosta brzo i unificirano** - postavljanje svih potrebnih komponenti (baze, web servera, sigurnosnih postavki itd.) je jako teško uraditi za svakog programera, a posebno za članove tima koji rade na udaljenim lokacijama. Zahvaljujući docker-u, dovoljno je kreirati adekvatne kontejnere i učiniti ih dostupnim članovima tima.
   2. **Minimizira potencijalne konflikte između aplikacija** - u slučaju da ne želimo migrirati na novu verziju framework-a ili koristiti neke specifične postavke koje zahtijevaju pojedine aplikacije, dovoljno je samo da promjenimo kontejner
   3. **Jednostavno kretanje softvera kroz različita okruženja (razvojno, testno, produkcija)** - minimiziraju se komentari: “kod mene ne radi”
   4. **Brža isporuka kvalitetnog softvera**
4. **Kontejneri** su izolirani, ali **dijele operativni sistem** što omogućava:
   1. Eliminisanje velikog broja licenci koje zahtijevaju operativni sistem
   2. Eliminisanje potrebe za stalnim održavanjem i nadogradjnom velikog broja različitih operativnih sistema
   3. Jednostavnije izvršavanje na različitim platformama i serverskim infrastrukturama
   4. Jednostavniju migraciju - prelazak na novu verziju zahtjeva samo par koraka
   5. Ponovnu iskoristivost i prenosivost - kontejneri se na jednostavan način distribuiraju između pojedinaca, timova ili samih okruženja (razvojno, testno, produkcijsko)
   6. Samostalnost okruženja - kontejneri minimiziraju zavisnost između pojedinih komponenti i/ili verzija
   7. Jendostavno upravljanje - kontejnere na jednostavn način moguće povezati, grupisati, te na taj način njima upravljati

**Docker - ključni pojmovi**

1. **Image** predstavlja kolekciju fajlova i meta podataka koji sačinjavaju root fajl sistem kontejnera, a obično su sastavljeni od više slojeva od kojih se neki mogu modifikovati
2. **Kontejner** se može posmatrati kao set procesa koji se izvršavaju u okviru pomenutog fajl sistema. Drugim riječima, kontejner se može posmatrati kao pokrenuti/startani image.

**Docker komponente**

1. **Client -** Omogućava interakciju sa komponentom Host-a koja se naziva Server ili Daemon. Pomenuta komponenta je zadužena za upravljanje sa image-ima i kontejnerima, te interakciju sa Hub-om.
2. **Host -** U okviru ove komponente se izvršavaju kontejneri. Kada se preko klijenta zahtijeva pokretanje image-a, Server ili Daemon prvo provjerava da li taj image postoji u lokalnom repozitoriju, te ukoliko ne postoji vrši njegovo preuzimanje sa Hub-a (registry-a) i pokretanje u okviru kontejnera.
3. **Hub (hub.docker.com) -** Predstavlja centralni repozitorij za publikovanje image-a. Pored javnog, posjeduje i privatne repozitorije



**Docker**

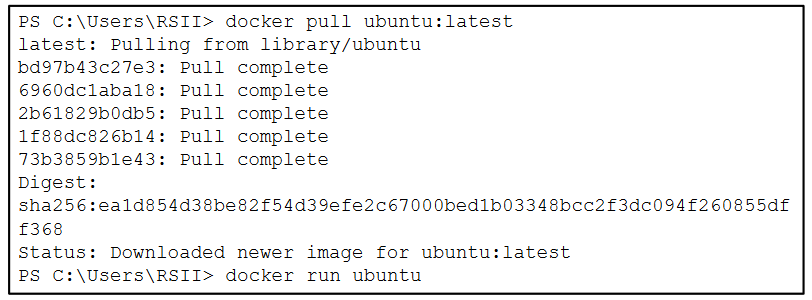
1. Sve komandne riječi počinju sa ključnom riječju *docker*

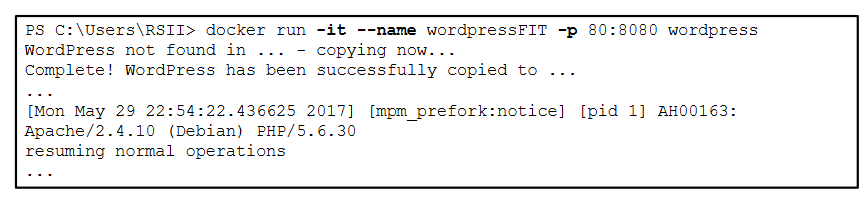
**Docker komande ( u zagradama je definicija sa oficijelne dokumentacije dockera )**

1. **Docker version -** informacije o klijentu i serveru (Show the Docker version information)
2. Docker info - informacije o broju image-a i kontejnera koji se trenutno izvršavaju ili su pauzirani ili stopirani, njihovim verzijama i dr. (Display system-wide information, Information displayed includes the kernel version, number of containers and images.)
3. Docker ps - informacije o kontejnerima koji se trenutno izvršavaju (List containers)
4. Docker ps - a - informacije o svim kontejnerima uključujući one koji su se ranije izvršavali (Show all containers (default shows just running))
5. Docker run - starta novi kontejner (Run a command in a new container)
6. Docker stop - stopira izvršenje kontejnera (Stop one or more running containers)
7. Docker pull - kopira image (određenog naziva npr. mysql) na docker host (Pull an image or a repository from a registry, Most of your images will be created on top of a base image from the [Docker Hub](https://hub.docker.com) registry.)
8. Docker images - ispisuje listu image-a na docker host-u (List images)
9. Docker rmi - uklanja image sa docker host-a (Remove one or more images, this does not remove images from a registry. )

**Docker**

1. Nakon naziva image-a može se navesti njegova konkretna verzija (npr. 1.0.8). U ovom slučaju je korištena ključna riječ latest koja označava posljednju verziju.
2. Kreiranjem naloga na sajtu hub.docker.com omogućen je pristup informacijama o svim verzijama image-a, te uočenim nedostacima za svaku od njih.





1. It - označava da interakciju sa kontejnerom želimo obavljati kroz komandnu liniju u kojoj se trenutno nalazimo. Postoji i opcija -d koja označava detached mode.
2. Name - omogućava definisanje naziva za izvršenje konkretnog kontejnera
3. P - omogućava mapiranje portova između trenutne mašine i kontejnera
4. Wordpress - naziv image-a kojeg želimo startati.

* Za sve one koji nisu ljubitelji konzolne interakcije, dostupan je alat **Kitematic** koji omogućava jednostavnu manipulaciju image-ima i kontejnerima

**Docker Swarm mode**

1. Docker engine je moguće grupisati u klaster, te se takav način rada naziva swarm i njime se osigurava skalabilnost
2. Swarm mode definiše postojanje menadžer članova koji su zaduženi za raspodjelu zadataka ostalim članovima swarm-a (advertise-addr se koristi za definisanje adrese za komunikaciju sa swarm-om)

**--- PREDAVANJE 9 - SISTEMI PREPORUKE---**

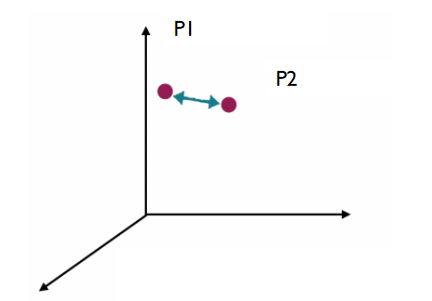
1. Sistemi preporuke imaju za cilj osigurati personalizaciju i kupcima ponuditi ono što odgovara njihovim potrebama. Postoji nekoliko vrsta podataka koji su potrebni za rad ovakvih sistema:
   1. Podaci o ponašanju kupca (ocjene proizvoda, broj pregleda, količina i učestalost kupovine i sl.)
   2. Demografski podaci (lokacija, starost i sl.)
   3. Atributi proizvoda (marka i snaga vozila, žanr knjige i sl.)
2. Na osnovu pomenutih podataka sistemi preporuke, koristeći odgovarajuće algoritme, mogu u značajnoj mjeri personalizirati okruženje i interakciju sa kupcima.

**Algoritmi**

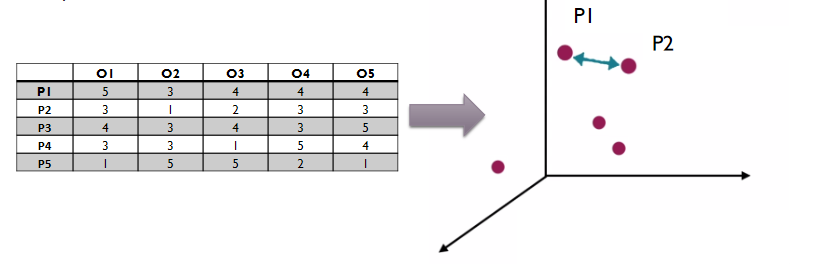
1. **Content-based filtering (CBF)**
   1. Omogućava preporuku proizvoda sličnih onima koje je kupac prethodno pregledao, ocjenio, komentarisao, kupovao (knjige istog žanra ili pisca, vozila iste snage ili proizvođača i sl.)
2. **Collaborative filtering (CF)**
   1. Omogućava preporuku proizvoda koje su drugi kupci, sa sličnim preferencijama, već ocjenili. Ukoliko su kupci slični, onda će vjerovatno imati i slične zahtjeve i potrebe (ako se kupcu zaninu sviđa marka vozila Hyundai, onda bi se ta marka vozila mogla sviđati i kupcu Jasminu sa kojim posjeduje veliku sličnost.
   2. Može se govoriti o dvije vrste kolaborativnog filtriranja : **User-based** i **Item-based**
3. **Association rules learning (ARL)**
   1. Omogućava preporuku proizvoda na osnovu učestalosti zajedničke kupovine (ako se prilikom kupovine mobilnog uređaja gotovo uvijek kupuje i zaštita za ekran, onda se može prepoznati neka asocijacija između njih).

**Content-Based filtering**

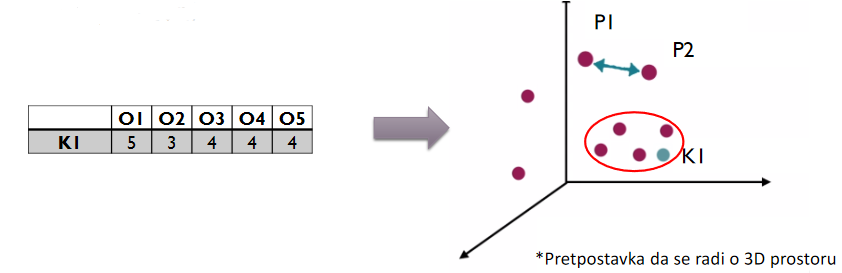
1. Posmatraju se vrijednosti atributa proizvoda, ali prvenstveno one koje imaju najveći značaj za korisnika (rijetko nam je bitno koji hemijski elementi ulaze u sastav šasije nekog vozila i sl.)
2. Svaka vrijednost atributa se posmatra kao koordinata u prostoru
3. Sličnost između proizvoda se može posmatrati kroz njihovu udaljenost u prostoru



1. Za određivanje sličnosti između proizvoda se mogu koristiti različite metrike a neke od njih su:
   1. Euclidean-ova
   2. Hamming-ova ili Correlation (korelacijska) distanca
2. U narednoj tabeli su prikazane ocjene osobina (Oi) za 5 različitih proizvoda, te su na osnovu njih pomenuti proizvodi predstavljeni u prostoru.

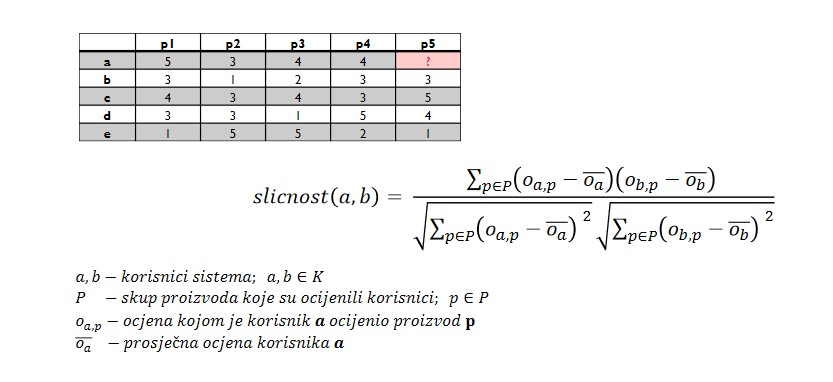


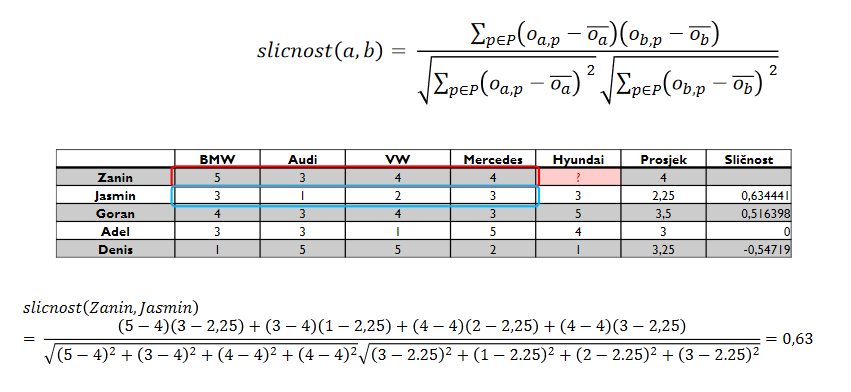
1. Da bismo znali koje proizvode preporučiti kupcu, potrebno je i kupca predstaviti u prostoru u kome se nalaze proizvodi. Vrijednosti, odnosno vektor, kojim će kupac biti predstavljen u prostoru mogu, ali ne moraju, biti prosječne vrijednosti ocjena kojim je kupac ocjenio osobine posmatrane kategorije proizvoda
2. Proizvodi koji budu najbliži kupcu su oni koji bi trebali biti preporučeni

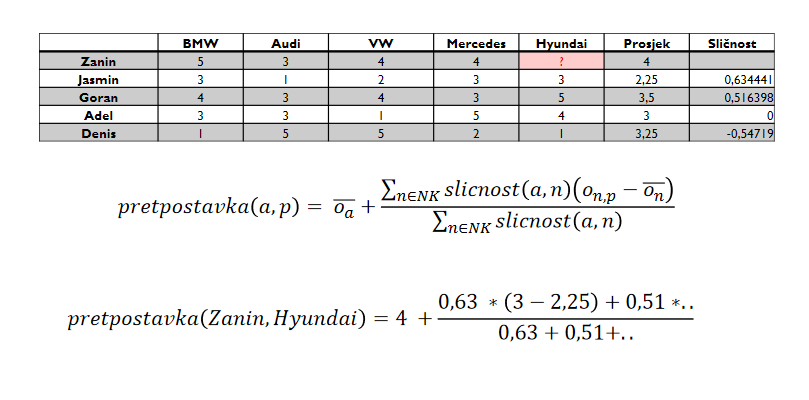


**User-Based CF**

1. Sličnost između dva korisnika se mjeri pomoću Preason korelacije.

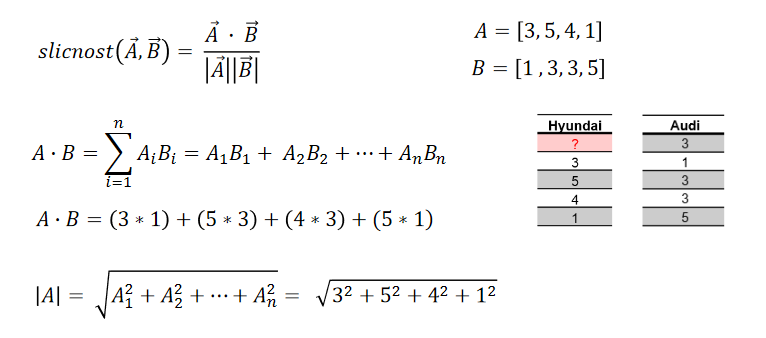




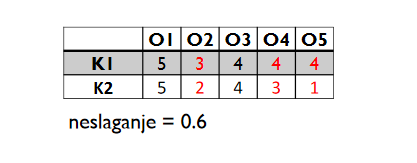


**Item-based CF**

1. Item-based CF računa preporuku na osnovu sličnosti između proizvoda.



1. Jedna od najjednostavnijih načina za procjenu sličnosti je Hamming-ova udaljenost kojom se izračunava pojam neslaganja
2. Hamming-ova udaljenost je pogodna prvenstveno u slučajevima kada se za vrijednost atributa proizvoda koriste diskretne vrijednosti
3. U narednom primjeru, neslaganje između kupca K1 i K2 je 60% ili 0.6



**Association rules learning**

1. Ukoliko se proizvodi često kupuju zajedno, onda će se na osnovu te relacije i bazirati preporuka
   1. Support - vjerovatnoća kupovine posmatranog proizvoda (mobitel)
   2. Confidance - vjerovatnoća kupovine drugog proizvoda nakon kupovine prvog (mobitel i zaštita za ekran)
   3. Lift - promjena u vjerovatnoći da će drugi proizvod biti kupljen nakon kupovine pr prvog
2. Tokom određivanja asocijacija se može koristiti Apriori algoritam koji u svakoj fazi eliminiše proizvode koji imaju vrijednost asocijacije ispod određenog praga.

