- analiza,oblikovanje,ugradnja i provjera su (faze iterativnog Å¾ivotnog ciklusa)
-Å¡to nije svojstvo programske potpore (upravljanje ljudskim aktivnostima)
-pretvorba nekog vrijednosnog tipa podatka u referencijski tip object (boxing)
-izdvajanje vrijednosnog tipa podatka iz tipa object (unboxing)
- kako neki string s pospremiti u integer x
-s kojom rjeÄ\215ju se eksplicitno uvode sva imena sadrÅ¾ana u prostoru imena (using)
-1.predavanje slajd 61 slika
-osoba koja osigurava resurse za projekt (sponzor projekta)
-vrste resursa (resursi rada i resursi materijala)
-Ä\214emu sluÅ¾i prozor Locals (za praÄ\207enje stanja varijabli tijekom izvoÄ\221enja prog rama)
-Kako se sa TFS-a skinu verzije (to znate :P)
-Svaka pojava,pojam ili predmet (stvarni ili zamiÅ¡ljeni) o kojem se prate podaci i naÄ\215 in rukovanja podacima (objekt)
-Obrazac prema kojem se stvaraju instance razreda (razred)
-Modifikator pristupa public (neograniÄ\215en pristup)
-Operator koji stvara novu instancu (new)
-svojstva get i set (indekseri i joÅ¡ neÅ¡to)
-funkcionalni zahtjevi odgovaraju na pitanje (Å¡to)
-scenarij (niz koraka koji opisuje interakciju korisnik-sustav)
-Å¡to oznaÄ\215ava dijamant kod dijagrama razreda (agregaciju)
-programi koji se sastoje od slijeda naredbi i djeluju nad globalnim varijablama (nestruktu rirano programiranje)
-Bazni razred za iznimke koji generira CLR (SystemException)
- neki mali kod s try, catch i finally Å¡to Ä\207e se ispisati
- neki mali kod s nasljeÄ\221ivanjem i overloadom, Å¡to Ä\207e se ispisati
-IDataReader (odgovor je neÅ¡to s vraÄ\207anjem u prethodni redak, ne sjeÄ\207am se)
-Prikaz informacije o pogreÅ¡ci (ErrorProvider)
-Vrste DataSet skupova (Typed, UnTyped)
-SloÅ¾eno povezivanje (povezivanje sloÅ¾ene komponente s viÅ¡e redaka)
-Å¡to je zabranjeno za dogaÄ\221aj i bili ponuÄ\221eni neki izrazi s +=, -=, =

1) Za C# NE vrijedi (C# je proceduralni jezik)
2) Å to je od ovog type reference (array)
3) Å to je od ovog type value (enum)
4) Nabacana gomila teksta kao opis IDE-a, je li toÄ\215no/netoÄ\215no (toÄ\215no)
5) Jezik koji i onda opis u kojem se rijeÄ\215 'procedura' pojavljuje tri puta je (procedur alno orijentiran)
6) VoÄ\221a projekta je (osoba koja se brine da cilj projekta bude postignut)
7) Operator koji pokazuje na trenutnu instancu (this)
8) Å to radi ADO .NET (pribavlja podatke)
9) ADO .NET omoguÄ\207ava pristup podacima, ukoliko postoji provider, toÄ\215no /netoÄ\215n o (toÄ\215no)
10) Koji simbol od navedenih predstavlja izvoÄ\221enje (izduljena elipsa)
11) Definicija programske potpore je (najdulji odgovor, imate definiciju na slajdovima)
12) Definicija programskog inÅ¾enjerstva je (vidi iznad)
13) Za dobiti 10.8 bodova iz blica treba (proÄ\215itati prezentacije jednom)
14) Naziv za izvoÄ\221enjem dodatnih operacija ukoliko su zadovoljeni uvjeti (proÅ¡irenje)
15) Svi tipovi podataka u C# su izvedeni iz (System.Object)
16) Apstraktni razred se moÅ¾e instancirati, toÄ\215no/netoÄ\215no (netoÄ\215no)
17) Definicija apstrakcije, ne sjeÄ\207am se kako toÄ\215no ide (apstrakcija, oÄ\215ito)

1. nestrukturirano programiranje
2. IDbCommand
3.13-16 sati 200% radni utorak i srijeda.. koliko tjedno osoba radi?
4. OleDbDataReader i SqlDataReader - implementiraju System.Data.IDataReader
5. System.Data.IDataReader
6. using je trebalo odabrat - ono sa prostorom imena i to
7. boxing
8. unboxing
9.varijante vodopadnog modela
10. u kojoj fazi ciklusa je dizajn arhitekture i baze.. oblikovanje
11. nabroje ti faze ekstremnog programiranja pa ti zakljucis da je to ekstremno
12.ToString
13. iz stringa u int - int c = int.Parse(s)
14.simple binding Å¡to je

15. bila nacrtana neka strelica nekakvo povezivanje neznam..
16. kritiÄ\215ni put - niz zadataka bla bla na vrijeme
17. kako se salje datoteke na posluzitelj (check in)
18. definicija objekta - i onda ti kliknes objekt
19. konstruktori su se spominjali
20. gdje se moze koristit get i set (u postupcima i svojstvima)
21. def tvrdnje
22. preporuke za koriÅ¡tenje tvrdnji (sto je krivo)
23. nemodalni prikaz forme (show)
24. na koje pitanje odg poslovni zahtjevi (zaÅ¡to)
25. nes sa dijagramom razreda il tako nes (povezivanje)
26. izvedeni razred mora u potpunosti ugraditi nedovrÅ¡ene postupke - nes je bilo s tim nis am siguran tocno Å¡to (uglavnom apstraktni razredi)

200% = 2 osobe rade
utorak i srijeda = 2 dana tjedno
13-16 = 3 sata dnevno
2*2*3 = 12 ukupno

nesto kako prikazati nemodalnu formu(Show())
- plan iteracija, analiza, oblikovanje, izrada itd (aktivnosti iterativnog modela)
- u kojoj fazi se radi dizajn arhitekture, suÄ\215elja itd (oblikovanje)
- nesto kao sto je PropertyManager ili tak nekaj (odgovor je da u njemu se mogu namjeÅ¡tati i pregledavat svojstva objekta)
- ona strelica za generalizaciju u dijagramu razreda i pita kaj to oznaÄ\215ava(generalizac iju)
- pitanje toÄ\215no/netoÄ\215no za izgled strelice za include u obrascu upotrebe. Ne sjecam se tocno kak ide ali tocno je.

public â\200\223 pristup nije ograniÄ\215en
private - pristup ograniÄ\215en na razred u kojem je Ä\215lan definiran
protected â\200\223 pristup ograniÄ\215en na razred i nasljeÄ\221ene razrede

-napisano je
this.button1.Click += new System.EventHandler(this.button1_Click);
pitanje je Å¡to je this.button1.Click (event)

-neÅ¡to u smislu s Ä\215im se bavi plan projekta

-Å¡to se ne moÅ¾e izvesti
i sad su tu bili primjeri sa varom
var x=7;
var x= int.parse("123");
var x; // rjeÅ¡enje
var x=12.toString();

Å¡to je major versio.minor version.build.reviion(verzija asemblija)
Å¡to je asemblij
Å¡to se preporuÄ\215uje kod hvatanja iznimki
kako se otvara nemodalna forma(show)
koji se dogaÄ\221aj okine kada se pokreÄ\207e forma(load)
kada se stvara specifikacija zahtjeva
defincija spec. zahtjeva i treba odabrati s. z.
da li se poziva konstruktor klase koja je nasljeÄ\221ena (poziva se)
da li binding source ima enkapsuliran podatak i referencu na trenutni podatak
kada se moÅ¾e izvesti dio koda sa foreach petljom (bili su primjeri i toÄ\215an odgovor je kada se ne mijenja lista i kada se ne mijenja trenutni element)
bila su dva pitanja koliko osoba ima radnih sati ako radi 12-15 h utorak i srijedu
ako je 200% onda 12
ako je 50% onda 3

1. Action<object> (Å¡to je object)
2. Func<A,B,C> (Å¡to su A,B,C - a,b parametri, C povratna vrijednost)
3.ADO, sve o procedurama (i sve je bilo tocno)
4.Å¡to nastaje kad nastaje entity model(dbcontext bla bla dbset bla bla tak nes)
5.prednost generiÄ\215kih nad negeneriÄ\215kim kolekcija(perfomanse)
6.windows location, dekstop location, new point()
imamo vertikalni taskbar na lijevoj strani(900x80) na ekranu(1600x900)
forma 600x200, koja komanda da forma dodiruje taskbar i donji dio ekrana
7.ok, no , retry....(enum dialog neceg)
8.try using throw catch
overridan je dispose metoda da ispisiuje koji se objekt disposo

```
try {
objekt r1 = new objekt()
using ( r2 objekt = new r2()) {
r3 objekt = new r3();
throw exception;
}
r1.dispose();
catch {}
```

ispisat ce se Dispode : R2

9.try using catch sa datotekom, otvaranje datoteke, u slucaju greske
ispisuje se pogreska i zatvara datoteka
10.analiza
11. produkt oblikovanja (specifikacija sustava)
12.razred
13.Å¡to se moÅ¾e unijeti u bazu iz entita(tabice,view, stored procedure, funkcije)
14.value, reference, sta ce se ispisati
15.osoba radi pon-pet, 8-12 , 75% raspolozivosti , koliko je radno vriejme(15 sati)
16.protected internal
17.ADO baratanje sa vise rezultata(executereader() i nextresult())
18.najljepsi nacin da napisemo redni broj pitanja koji je atribut ?u bazi?(RbrPitanja)
19.najjepsi nacin ifovanja

```
bool ispravan = Fja();
if (ispravan) { ... }
```

20.kompozicija (jedno nemoÅ¾e postojati bez drugoga)
21.korisnk mora moci stornirat racun(korisnicki zahtjev)
22.stringbuilder (dodavanje elemenata u niz)
23.type safety (zaÅ¡tita od unosa podataka nedozvoljenog tipa)
24.undo pending changes
25.nulabilni tip (tip?)
26.sto je uml (jezik za sustave programske podrÅ¡ke)
27.barikada (razredi koji blokiraju neÄ\215iste razrede koji barataju ulaznim podacima)
28.povelja projekta (ovlast za provedbu)
29.modularnost (grupiranje u module koji mogu sadrÅ¾avati lokalne atribute)

Jedina nova kojih se sjeÄ\207am su neÅ¡to sa izdavaÄ\215em i pretplatnikom, ako Å¾elimo da aplikacija ima vlastitu kontrolu - odg. kontrola je izdavaÄ\215, a aplikacija pretplatnik onda neÅ¡to gdje su nabrojane kolekcije , pa je odgovor kolekcije
onda neki sa K1, K2 i K3 binding sourcovima, koji Ä\207e ubaciti neki element, a koji neÄ\207e - sjeÄ\207a li se itko toÄ\215nog zadatka da ostane za sljedeÄ\207e generacije?
par pitanja sa iterativnim modelom (neki odgovor je bio iterativni i inkrementalni)

Meni je doÅ¡ao ovaj s K1, K2, K3.
Pitanje je bilo neÅ¡to tipa:
Imate dva razreda A i B. B implementira INotifyPropertyChanged.
Imate formu s 3 kontrole. Na svaku je spojen binding source Ki.

```
K1 = new List<A>();
K2 = new BindingList<A>();
K3 = new BindingList<B>();
```

Pitanje je ako se programski doda novi element u listu, na kojoj kontroli Ä\207e se vidjeti promjena.
ToÄ\215an odgovor je na kontroli K3.

koja se notacija koristi kod prostora imena, i drugo pitanje isto to samo za postupke.
Pascal oba Ä\215ini mi se

1.Varijante vodopadnog modela (klasiÄ\215ni, pseudostrukturni, radikalni)
2.Kod iterativnog postupka razvoja svaka se iteracija obavlja (standardnim zivotnim cikluso m koju ukljucuje oblikovanje, ugradnju...)
3. Za koji je razvoj karakteristicno da se softver razvija i objavljuje po dijelovima, a gl avne faze se obavljaju kroz niz iteracija (iterativni i ikrementalni)
4. UgraÄ\221eni postupak za pretvorbu nekog objekta u tip string je (ToString())
5. Kojom kljucnom rjeci neki projekt ukljucuje prostor imena (using)
6. Plan projekta je dokument koji (opisuje sveukupnu organizaciju projekta)
7. Koje vrste projektnih zadataka postoje (primitivni, skupni i prekretnice)
8. Koja je tvrdnja istina (rjesenje (solution) se moze sastojati od vise projekata)
9. public modifikator oznacava (neogranicen pristup)
10. protected modifikator oznacava (pristup ogranicen na razred i nasljeÄ\221ene razrede)
11. Za automatsko oslobaÄ\221anje memorije koja je zauzeta referencama koje vise ne postoje , zaduzen je (garbage collector)
12. Å¡tapiÄ\207asti Ä\215ovjeÄ\215uljci u dijagramima sluÄ\215. koristenja predstavljaju (s udionike)
13. za vrijeme dokumentiranja sustava za unos narudzbi otkrijemo da netko tko nije kupac mo ze nazvati i napraviti narudzbu.
U ovom slucaju koristenja "Napravi narudzbu" ce koristiti sluacj koristenja "Naprav i kupca" kako bi prikupio podatke o kupcu.
O kojoj vezi izmedu slucajeva se radi (Prosirenje)
14. Koja tvrdnja *NIJE* tocna (Apstraktrni razred ne moze implementirati suÄ\215elje)
15. Razred koji implementira neki od ICollection, IEnumerator, IEnumerable,IDictionary, ILi st naziva se (kolekcija)
16. Skup prog. naredbi koje obavljaju jedan zadatak ili jedan dio zadatka, npr gl. program, potprogram zovu se (prog. cjelina)
17.Preporucena notacija za privatne varijable (camel case)
18. obrada iznimke najcesce je oblika (try catch finally)
19. Koja tvrdnja *NIJE* istina (obrada greske izaziva nepredvideni prekid izvodenja program a)
20. Å to je izdavac(publisher) a sto pretplatnik (subscriber) u modelu s app i vlastitom ko ntrolom (kontrola-izdavac, app - pretplatnik)
21. Nacelo oblikovanja korisnickog sucelja po kojem se prepoznaje da novi i iskusni korisni k imaju razlicit odnos prema sucelju je
(iskustvo korisnika (user experience))
22. Korisnicko sucelje: (definira interakciju s krajnjim korisnikom)
23. Kako se naziva forma koja je sadrzana u formi roditelj (Dijete(MDIChild))
24. Kod svakog postavljanja svojstva forme ili postavljanja kontrole na formu prog. kod unu tar funkcije InitializeComponent se: (generira automatski)
25. Osnovna namjena kontrole gumb-button:(pokretanje akcija)
26. Razred FileInfo sadrzi postupke za (stvaranje, premjestanje,...,brisanje i otvaranje da toteke)
27. Razred koji sluzi za dohvat postavki iz konfiguracijske datoteke je (ConfigurationManag er)
28. Tehnika izrade EF modela u koje se baza stvara kroz rucno pisane razrede te nema vizual nog modela (code first)
29.Tehnika izrade EF modela u koje se bazavec postoji, a EF model se pravi reverznim inzenj erstvom (database first)
30. Imate dva razreda A i B. B implementira INotifyPropertyChanged.
Imate formu s 3 kontrole. Na svaku je spojen binding source Ki.
K1 = new List<A>();
K2 = new BindingList<A>();
K3 = new BindingList<B>();

Pitanje je ako se programski doda novi element u listu, na kojoj kontroli Ä\207e se vidjeti promjena.

ToÄ\215an odgovor je na kontroli K3.

1)MDI
2) user experience
3) this.button1.Click += new System.EventHandler(this.button1_Click);
->Å¡to predstavlja system.eventhandler (delegat)

4) -> sto predstavlja this.button1_click (event handler - postupak)
5) gdje se odreduju spec. zahtjevi? -> u fazi analize
6) 7) nesto osnovno o iznimkama, ne mogu se sjetit al je bilo trivijalno
8) System.IO (Ä\215itanje i pisanje podataka)
9) niz imenovanih konstanti -> enum
10) nestrukturirano programiranje
11) OleDbDataReader i SqlDataReader -> implementiraju IDataReader
12) Simple binding (jedna stavka/redak)
13) sto je binding opcenito (mehanizam vezanja elemenata grafiÄ\215kog suÄ\215elja na podat ke)

suÄ\215elja se odreÄ\221uju za vrijeme (oblikovanja)
vrste vodopadnog modela (klasiÄ\215ni, pseudostrukturni i radikalni)
string NIJE (isto Å¡to i niz znakova u jeziku C#)
voditelj je (osoba imenovana kako bi ostvarila ciljeve projekta)
planprojekta (opisuje sveokupnu organizaciju projekta)
private (pristup ograniÄ\215en na razred u kojem je Ä\215lan definiran)
public (neograniÄ\215eni pristup)
scenarij (niz koraka koji opisuju interakciju sustav-korisnik)
NIJE toÄ\215no (private varijable i postupci se nasljeÄ\221uju)
nadjaÄ\215anom Ä\215lanu Postupak() se pristupa (instanca.Postupak())
vrsta programiranja gdje se radi s globalnim varijablama i ponavljanje postupka znaÄ\215i k opiranje koda (nestrukturirano programiranje)
skup programskih naredbi ili cijeli program se naziva (programska cjelina)
osnovni razred za iznimke koje geneira pogon programa (SystemException)
osnovni razred za iznimke aplikacije (ApplicationException)
pitanja o viÅ¡enitnosti, Å¡to je nit i pitanje o dijalogu
.... aplikacija okida dogaÄ\221aje vlastite kontrole (kontrola - izdavaÄ\215, aplikacija - pretplatnik)
CamelCase je programsko naÄ\215elo (Estetike)
brzi odabir na formi (Funkcijska tipka)
izvor podataka za povezivanje na kontrolu (Sve od navedenog)
simple binding (jedna vrijednost)
complex binding (sloÅ¾ena kontrola s viÅ¡e redaka)
pitanje Å¡to sadrÅ¾i DirectoryInfo - bazni razred za razrede
dohvat postavki iz konfiguracijske datoteke (ConfigurationManager)

1. Koje su vrste resursa? (resursi rada i materijala)
2. Koja operacija obavlja download datoteka u TFS-u? (get latest version)
3. Å to je od navedenih moguÄ\207i nadrazred od zaposlenika? (osoba)
4. Kako izgleda generalizacija u dijagramu strukture? (prazna strelica)
5. Å to simbolizira dijamant u dijagramu razreda? (agregacija)
6. Varijable deklarirane na vrhu razreda se zovu? (Ä\215lanske varijable ja mislim)
7. Å to znaÄ\215i da je razred parcijalan? (moÅ¾e se protezati u viÅ¡e datoteka)
8. Skup zajedniÄ\215kih tipova podataka platforme .NET se zovu...? (Common Type System)
9. Internal modifikator pristupa razredima i Ä\215lanovima razreda oznaÄ\215ava? (samo unut ar programa)
10. Internal protected modifikator pristupa razredima i Ä\215lanovima razreda oznaÄ\215ava? (unutar programa i naslijeÄ\221enih razreda)
11. Za neku osobu se evidentira radno vrijeme u utorak i srijedu 12-15h, 50% raspoloÅ¾ivost i radnika znaÄ\215i da radnik tjedno radi koliko? (3h)
12. Pojam koji se odnosi na suradnika koji radi na projektu je? (uloga)
13. Razred predloÅ¾ak koji definira porodicu razreda i ne ovisi o tipu podataka naziva se? (parametrizirani razred)
14. Å to je tipska sigurnost? (zaÅ¡tita od unosa podataka neÅ¾eljenog tipa)
15. Koji je ispravan poziv postupka Task<string> RunAsync(int a, int b)? int a = 4; int b = 3; string s = RunAsync (a, b);
16. ZnaÄ\215enje svojstva InvokeRequired? (dali je pozivatelj na drugoj niti)
17. Å to znaÄ\215i delegat tipa Action<object>? (postupak bez argumenata koji vraÄ\207a obj ekt tipa object)

1. Suradnik koji obavlja odreÄ\221eni posao, npr. voditelj projekta, arhitekt, programer? U loga
2. Prikupljanje informacija, modeliranje procesa, modeliranje podataka spada u SDLC fazu An aliza

3. Modeliranje sustava pripada fazi: Oblikovanje
4. Apstraktan opisP svojstava, varijabli i metoda objekata istog tipa: Razred
5. Kod i sto ce se ispisati (ovako nekako, znam da je bilo na hrv):

```
Class Razred
{
int broj;
}

int x1 = 1;
int x2 = x1;
x2 = 5;

Razred r1 = new Razred();
Razred r2 = r1;
r2.broj = 9;

Ispis(x1, x2, r1.broj, r2.broj)
```

"1 5 9 9"

6. int? num nazivamo Nulabilni tip
7. Ako kalendar evidentira radno vrijeme Pon-Pet od 8 do 12 sati. 75% raspolozivosti za nek og radnika znaci 15 sati rada tjedno
8. Povelja projekta je (ovo sam gadjao i pogodio, ne sjecam se tocno kako je islo, znam da je dokument i da nema veze s prvim sastankom): Dokument od strane sponzora i voditelja proj ekta
9. Otkazivanje lokalnih izmjena: Undo Pending Changes
10. Razlaganje slozenih sustava u lakse upravljive djelove: Modularnost
11. Protected Internal: Pristup dozvoljen nasljedjenim crazredima i svima iz programa u koj em je razred definiran
12. Dobra praksa pisanja ili tako nesto pa izabrati od ponudjenih djelova koda

```
condition = Provjeri(JMBG);
if( condition )
{
...
}
```

bitno je da ne smije funkcija pidi u if ()

13. Kako bi u bazi podataka oznacili atribut? RbrAtributa
14. Korisnik mora moci prijaviti gubitak kartice je: Korisnicki zahtjev
15. Objekt moze pripadati samo jednoj cjelini, elementi postoje tako dugo dok postoji cjeli na: Kompozicija
16. Prednosti generickih kolekcija na negenerickim: Bolje performanse
17. Tipska sigurnost je: Zastita od unosa podataka nezeljenog tipa
18. Konstrukcija sucelja kao granica prema sigurnim dijelovima koda je: Koncept barikada

1. Vrste resursa (resursi rada i resursi materijala)
2. Kako se dohvaÄ\207aju datoteke s TFS-a (Get Latest Version)
3. Kako se naziva postupak koji se koristi prilikom stvaranja instance (konstruktor)
4. Å to moÅ¾e biti nadrazred razreda zaposlenik (osoba)
5. Oznaka za generalizaciju (prazna strelica)
6. Å to znaÄ\215i dijamant u dijagramu razreda (agregacija)
7. Varijable definirane na vrhu razreda (Ä\215lanske varijable)
8. Å to znaÄ\215i da je razred parcijalan (znaÄ\215i da se pojedini dijelovi mogu pokrenuti u drugim datoteka)
9. Kako se zove naÄ\215elo oblikovanja korisniÄ\215kog suÄ\215elja prema kojemu se radi ogr aniÄ\215enje selekcije listama
   (lakoÄ\207a koriÅ¡tenja, minimalni napor korisnika)
10. Koliko najviÅ¡e moÅ¾e biti izbornika zavisnih o kontekstu (ContextMenuStrip) na formi/k ontroli? (1)
11. NeÅ¡to u stilu:

```
int a = 0; int b = 0;
try {
try {
Console.WriteLine("T1 ");
int c = b/a;
}
finally
{
```

```
Console.WriteLine("F1 ");
}
}
catch (Exception e)
{
Console.WriteLine("E1 ");
}
finally
{
Console.WriteLine("F2 ");
}
```

RJEÅ ENJE: T1 F1 E1 F2

12. NeÅ¡to u stilu:

```
int a = 0; int b = 0;
try {
try {
Console.WriteLine("T1 ");
}
finally
{
Console.WriteLine("F1 ");
}
}
catch (Exception e)
{
Console.WriteLine("E1 ");
}
finally
{
Console.WriteLine("F2 ");
}
```

RJEÅ ENJE: T1 F1 F2

13. Ono pitanje sa zajedniÄ\215kim i opÄ\207im tipovima podataka (odgovor je CTS)
14. Za internal modifikator definiran kod razreda i programa vrijedi sljedeÄ\207e (pristup je ograniÄ\215en na programe u kojem je razred definiran)
15. VeÄ\207 viÄ\221eno pitanje - ako netko radi utorkom i srijedom 12-15 s 50% raspodjelom, radi (3h tjedno)
16. Glavne faze za UDP/RUP (poÄ\215injanje, elaboracija, konstrukcija, prijelaz)
17. Metodologija razvoja softvera koja sadrÅ¾i niz predloÅ¾aka razvojnog procesa za razliÄ\215ite modele razvoja,
    a obavlja se kroz razvojne faze organizacijske iteracije (RUP)
18. Razlika raÄ\215unalne znanosti i programskog inÅ¾enjerstva (raÄ\215unalna znanost je teorijska, programsko inÅ¾enjerstvo se temelji na praktiÄ\215noj primjeni programske potpore, odgovor u nekom takvom stilu)
19. NeÅ¡to u stilu

```
public class Osoba
{
public Zaposlenik zap;
.....
}
```

Kako se naziva ovo? (kompozicija)

20. Razred-predloÅ¾ak koji definira porodicu razreda i ne ovisi o tipu podataka naziva se (parametrizirani razred)
21. ToÄ\215an zapis metode za ispisivanje adrese isporuke zaposlenika (IspisiAdresuIsporuke)
22. Elementi suÄ\215elja koje stvara programer, a moÅ¾e ih se koristiti u razliÄ\215itim programima se nazivaju (vlastite kontrole)
23. Postupak kojim se na osnovu spremljenih podataka objekt naknadno rekonstruira (deserijalizacija)
24. Koja je naredba naredbe SqlCommand() za ispis broja odreÄ\221enih zapisa? ExecuteNonQuery()
25. Å to se koristi za asinkrono izvÅ¡avanje od C# 5.0? (async i await)
26. Task<String> RunAsync(int a, int b) - ispravni poziv programa za asinkrono izvrÅ¡avanje? (string s = await RunAsync(2,5);)
27. ZnaÄ\215enje InvokeRequired (vraÄ\207a odgovor da li je pozivatelj na drugoj niti)
28. Ono s definiranjem DataSource:

DataSource=rppp.fer.hr\SQL2012; Initial Catalog=firma; UserId = rppp; Password = test;

29. NeÅ¡ u stilu:

```
using (var test = new SQLConnection(ne sjeÄ\207am se Å¡ta je tu bilo)
{
.....
}
```

Å to Ä\207e se dogoditi ako nije napisan test.close()? (Zatvorit Ä\207e se konekcija nakon izvrÅ¡avanja isjeÄ\215ka neovisno o tome je li se dogodila iznimka ili ne)

30. Kod ADO.NET upita s viÅ¡e rezultata se koristi (ExecuteReader i NextResult)

1) Å to je instanca razreda? Objekt
2) Razred zaduÅ¾en za iznimke? System.Exception
3) Bila je ponuÄ\221ena nekakva definicija, a odgovor je bio Nefunkcionalno ono neÅ¡t
4) this.button1.Click += new System.EventHandler(this.button1_Click) - Å¡to je Å¡to, event, delegat, event handler ja mislim.
5) Å¡to nije, ponuÄ\221eno je MDI, SDI, TDI i LDI? LDI nekakav long... to ne postoji
6) OleDbDataReader i SqlDataReader? implementiraju IDataReader
7) System.IO? Za datoteke
8) Dokument za sve moguÄ\207e, bla bla nekak definicija valjda? Specifikacija zahtjeva
9) Od kuda su dobiveni svi tipovi u C#? System.Object
10) Nekak opis nasljeÄ\221ivanja, deriviranja? NasljeÄ\221ivanje
11) Ado NETÄ\215emu sluÅ¾i? Za pristup podacima

Zavod za primijenjeno računarstvo

prof. dr. sc. Krešimir Fertalj
izv. prof. dr. sc. Boris Milašinović

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Razvoj primijenjene programske potpore

Zaštićeno Creative Commons licencom, http://creativecommons.org

O predmetu

- **Kratki opis**
  - Razrađuju se koncepti programskog inženjerstva i njihova primjena na razvoj programske potpore za krajnjeg korisnika. Obrađuju se elementi inženjerstva zahtjeva, oblikovanje i ugradnja programskih komponenti za različite tipove aplikacija, tehnike programiranja, dokumentiranje, uvođenje i održavanje aplikacija.
- **Kompetencije**
  - Predmet osposobljava za razvoj složenih interaktivnih aplikacija, naročito onih nad bazom podataka. Studentima se prenose znanja potrebna za uspješno oblikovanje, razvoj i primjenu programskih sustava. Studenti će biti sposobni formulirati zahtjeve na aplikaciju te na temelju njih razviti, ugraditi i održavati kvalitetnu programsku podršku za različite arhitekture sustava.
- **Obvezni predmet modula Programsko inženjerstvo**
    - izborni za druge module
  - ECTS = 4
  - Sati izravne nastave tjedno = 3
  - Sati laboratorijskih vježbi u semestru = 15

Izvedba

- **Nositelj kolegija**
  - prof. dr. sc. Krešimir Fertalj (D369)
- **Predavači & izvođači**
  - izv. prof. dr. sc. Boris Milašinović (D374)
  - Fran Tonković, mag. ing. comp. (D258)
  - Juraj Dončević, mag. ing. comp. (D373)
- **Demonstratori**
  - …
- **Administracija**
  - Zavod za primijenjeno računarstvo, III kat zgrada D
  - tajnica Sonja Majstorović, Tel: 6129-915
- **Obavijesti**
  - web-stranica predmeta http://www.fer.hr/predmet/rppp

BY NC SA

Nastavni materijali

- **Skripta (stara):**
  - http://bib.irb.hr/datoteka/807422.Razvoj_primijenjene_programske_potpore_-_skripta_FER_2016.pdf
  - nova u nastajanju
- **Folije s predavanja, prateći primjeri**
  - Git repozitorij https://rppp.fer.hr/tfs/studentcollection/_git/RPPP-FER2
    - Privatni repozitorij, prijava korištenjem domenskog računa
    - http://creativecommons.org/licenses/by-nc-sa/2.5/hr/
- **Dodatna literatura:**
  - Roger S. Pressman (2009). Software engineering: A Practitioner`s Approach, McGraw-Hill Science
  - Ian Sommerville (2010). Software engineering, Addison Wesley
  - Steve McConnell (2004). Code Complete: A Practical Handbook of Software Construction, Microsoft Press
  - Stephen R Schach (2010). Object-Oriented and Classical Software Engineering, McGraw-Hill Science
  - Andrew Troelsen, Philip Japikse (2017). Pro C# 7 with .NET and .NET Core, Apress

Konzultacije

- **Konzultacije kod izvoditelja nastave i demonstratora**
  - na laboratorijskim vježbama
    - raspored objavljen za svaku vježbu zasebno
  - prema dogovoru s nastavnikom
    - Zavod za primijenjeno računarstvo
      - zgrada D - 3. kat, sjeverozapadno krilo
      - zgrada D - 2. kat, jugozapadno krilo

- **Konzultacije i primjedbe elektronički**
  - Na e-mail adresu predavača/asistenata koji izvode nastavu
    - kresimir.fertalj@fer.hr, boris.milasinovic@fer.hr
    - fran.tonkovic@fer.hr, juraj.doncevic@fer.hr
  - Subject: [RPPP]
  - U potpisu navesti puno ime i prezime te grupu na predmetu (*RPPPnn)*

CC BY NC SA

Sadržaj

- **Teorije toliko da odredi praksu**
  - kompletan životni ciklus, od uspostave projekta te izrade plana i specifikacija do kontrole dijeljenog izvornog koda pri radu u grupi na izradi aplikacija
- **Prakse toliko da bude izvediva**
  - prikupljanje i evidentiranje zahtjeva, izrada specifikacije i modela podataka
  - izrada web aplikacije i uslojavanje aplikacije
  - izrada web servisa
  - izrada windows aplikacije
  - izrada programske i korisničke dokumentacije
- **Ekipni projekt razvoja aplikacije i izrade prateće dokumentacije**
  - Definiranje problema
  - Podjela posla
  - Dijeljenje izvornog koda i dokumentacije
  - Kolaboracija s nastavnikom i asistentima
- **Detaljnije na slajdu *Sadržaj po tjednima***
- **Sadržaj i termini domaćih zadaća nalaze se u dokumentu RPPP-Vrednovanje.pdf**

CC BY NC SA

Provedba nastave i vrednovanje rada

- **Sudjelovanje na predavanjima (5% bodova)**
  - aktivnost na predavanju i konstruktivni pristup rješavanju problema
  - ometanje nastave, čitanje dnevnog tiska ne donosi bodove
  - iznos bodova diskrecijsko pravo nastavnika
- **Ispiti (40%) – ocjenjivanje znanja "teorije"**
  - Međuispit (20%), završni ispit (20%)
- **Domaće zadaće (55% bodova)**
  - individualno rješavanje dijelova projekta (projektiranje, modeli ...)
  - izrada elemenata projektne dokumentacije (zapisnici, ...)
  - doprinos ekipnom radu (programiranje, komponente, ...)
  - rokovi isporuke zapisani u dokumentu RPPP-Vrednovanje.pdf
    - Kasnija rješenja se ne uzimaju o obzir!
  - isporučeno programsko rješenje mora se moći kompilirati na poslužitelju!
- **Laboratorijske vježbe**
  - rad s alatom za upravljanje verzijama izvornog koda (Git)
  - primopredaja rezultata DZ i diskusija o isporučenim modelima

CC BY NC SA

Ocjenjivanje

- **Uvjet za izlazak na ispit je barem 50,00 % (27,5) mogućih bodova iz domaćih zadaća**
  - Nemogućnost ispunjenja obveza treba dojaviti najkasnije tjedan dana od dana kada je nemogućnost nastupila
  - nema ispričnice "za cijeli semestar" i nema nadoknada zadaća nakon završetka predavanja

- **Za prolazak na ispitima potrebno je ostvariti barem 50% bodova**
  - 50% bodova sumarno na međuispitu i završnom ispitu

    ili
  - 50% bodova na ispitu na rokovima

- **Raspodjela ocjena:**
  - >=85 (5) , >=70 (4), >=60 (3), >=50 (2)

CC BY NC SA

Sadržaj po tjednima – 1. ciklus

1. P: Osnove programskog inženjerstva. Životni ciklus razvoja programske podrške. Projekt izrade aplikacije. Specifikacija zahtjeva
2. P: Intervju. Upravljanje verzijama (izvornog koda). Osnove platforme .NET i jezika C#.
3. L: Izrada konceptualnog modela temeljem zapisnika s intervjua i 4. P: Tehnike programiranja. OOP u C#-u. Životni vijek objekta
4. i 5. P: Povezivanje s bazom podataka, Entity Framework Core
6. P: Web aplikacije: ASP.NET Core - osnove
7. P: Web aplikacije: Tehnike validacije, sortiranja i straničenja

Sadržaj po tjednima – 2. ciklus

1. **Web aplikacije: Dinamičko ažuriranje unutar stranice. Praćenje traga rada.**
   Programska dokumentacija.

2. **L: Kontrola točka za web-aplikaciju**
   P: Web aplikacije: Specijalizacija i generalizacija. Rad sa slikama.

3. **P: Primjer zaglavlje-stavke. Izrada izvješća**

4. **P: Refleksija. Web servisi**

5. **P: Tehnike rada s podacima u Windows aplikacijama**

6. **L: Predaja i provjera web-aplikacije**
   P: Autentifikacija i autorizacija korisnika

BY NC SA

Ankete i komentari

- **Ankete**
  - FER-ova, sveučilišna, „Ocijeni svog nastavnika” ?
- **Nastavnik i ekipa cijene izravne komentare i kritiku !!!**
  - verbalno, mailom

Softver

- **Zajednička infrastruktura**
  - Microsoft Team Foundation Server 2015
    - privatni Git repozitoriji
    - automatska kompilacija i izgradnja programskih rješenja
  - Microsoft SQL Server 2014

- **Potrebno za rad**
  - Visual Studio 2019 Community + .NET Core
  - SQL Server Management Studio
  - Klijent za Git

- **„Ja bih radio pod Linuxom ili MacOs-om"**
  - .NET Core i ASP.NET Core su multiplatformski
  - SQL Server Management Studio → Azure Data Studio ili DBeaver
  - Visual Studio 2019 Community → Visual Studio Code
  - Windows aplikacija → Mono Framework?
    - nije dio domaćih zadaća

Preporuke

- Materijali su obimni jer sadrže i dijelove potrebne za daljnje istraživanje, učenje i izradu programskih projekata.
- Fakultativne folije su posebno označene.

- Korišteni primjeri služe objašnjenju koncepata i općeprihvaćenih tehnologija ili standarda. Ne zagovaraju konkretne proizvode koji samo služe kao alati koji podupiru koncepte.

- Primjeri mjestimično izgledaju nedorađeno, a mjestimično pretrpano jer su namijenjeni poduci i nisu gotova rješenja.
- Unatoč tomu što pokazuju "kako se to radi" ne predstavljaju kuharicu "tako treba uvijek i bezrezervno raditi".
- Preporuča se dolaziti na predavanja.

- Očekuje se dobra volja i kreativnost pri izradi vlastitih rješenja. ☺

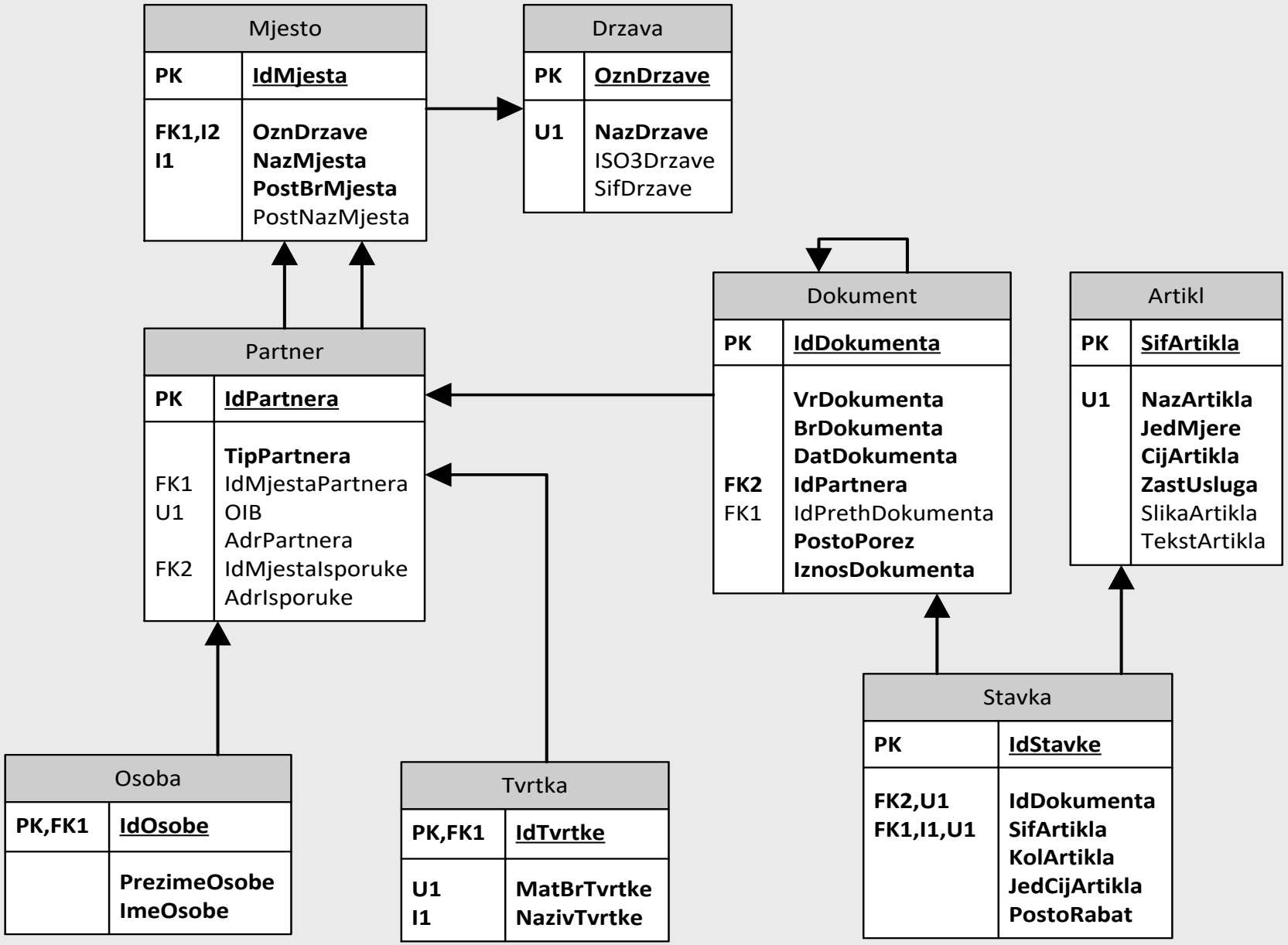
BY NC SA

?

Kako bi to moglo izgledati

CC BY NC SA

Primjer baze za neki reprezentativni slučaj



Primjer sučelja Windows i web aplikacije

Države
Mjesta
Artikli
Partneri
Dokumenti

Pregled svih dokumenata

Unos novog dokumenta

IdPartnera
Iznos 500,00 - 150000,00
Datum -

1 2 3 4 5 6 7 8 9 10 .. 16

Id dokumenta	Partner	Datum dokumenta	Iznos dokumenta
1968	Smirčić , Adrijan (2309953330197)	02.02.2014.	134.566,20 kn
1969	Srdelić, Nenad (2312955330101)	02.02.2014.	117.273,36 kn
1971	Stegić, Daniel (2409952335092)	02.02.2014.	68.435,77 kn
1972	Zakarija, Boris (3108960335399)	02.02.2014.	86.880,72 kn

Dokument br: 2543

Vrsta dokumenta
Porez (u %) 25
Datum 26.06.2015.
Broj 11242
Partner 572 FERAX (4875264)
Prethodni dokument
Iznos 1.

Artikl	Količina	Rabat [0-1]	Jedin
Knjiga "Photoshop 7 za 24 sata"	2,00000	0,00	
Knjiga "C #" ekspert	3,00000	0,50	
Knjiga "10 min. vodič kroz Outlook 2002" 19.4.2016. 15:16:3345454	5,60000	0,00	
knj			

Knjiga "10 min. vodič kroz Access 2002"
Knjiga "10 min. vodič kroz Excel 2002"
Knjiga "10 min. vodič kroz Front Page 2002"
Knjiga "10 min. vodič kroz Outlook 2002" 19.4.2016. 15:16:3345454

Firma Matični podaci Tablice (šifrarnici) Prozori

Partner: Lišnjić, Romina

K 1 - 31.12.2006 0:00:00

Id dokumenta 2407 Vrsta K Broj 1 Datum 31.12.2006
Partner SRCE (2259009) Porez 22.00%
Prethodni dok. P 2 - 2.1.2007 0:00:00 Iznos 403.916,44 kn

Naziv artikla	Mjera	Količina	Jedinična cijena	Rabat	Ukupna cijena
MP3/WMA player, DESTRUCTIVE MuVo TX S...	kom	3.00	780,00 kn	4.00%	2.246,40 kn
Knjiga "PC-i za sveznalice"	kom	10.00	169,00 kn	6.00%	1.588,60 kn
Stalak za notebook, univerzalni					
DVD/CD/MP3/DivX player, lini					
DVD RECORDER/VHS VIDEO					
Notebook PH Compaq 6720s (					
Bekin triPod Nano 3G Nano Ov					
Notebook PH C2710p (GP590					
Kućno kino DVD/DivX/CD/MP					
TV LCD 82 cm, SUNNY KDL-3					

Artikl: 4694, Mrežna kartica wireless CANYON, CN-7106PC, bež...

Šifra 4694
Naziv Mrežna kartica wireless CANYON, CN-7106PC, bežična mreža 11 Mbps, PCMCIA
Jed. mjere kom
Cijena 184.00
Usluga

Artikl: 4694, Mrežna ka Pregled
Korisnik: rppp

Statistika (1/2)

	Upisano	Aktivno	Prošlo	Prolaznost	5	4	3	2	1
2007/08	**106**	**101**	**86**	**85,15%**	**13**	**30**	**30**	**13**	**15**
					12,87%	29,70%	29,70%	12,87%	14,85%
				min bod:	84,65	70,85	61,9	56,1	
2008/09	**94**	**93**	**73**	**78,49%**	**12**	**26**	**18**	**17**	**20**
					12,90%	27,96%	19,35%	18,28%	21,51%
					77,83	67,58	58,17	50	
2009/10	**77**	**77**	**60**	**77,92%**	**9**	**22**	**16**	**13**	**17**
					11,69%	28,57%	20,78%	16,88%	22,08%
					77,33	67,88	56,79	50	
2010/11	**95**	**93**	**78**	**83,87%**	**14**	**25**	**18**	**20**	**16**
					15,05%	26,88%	19,35%	21,51%	17,20%
					77,53	66,42	57	50	

Statistika (2/2)

	Upisano	Aktivno	Prošlo	Prolaznost	5	4	3	2	1
2011/12	**89**	**87**	**69**	**79,31%**	**14**	**11**	**25**	**19**	**13+5**
			57+10+1+1		16,09%	12,64%	28,74%	21,84%	20,69%
					80	70	60	50	
2012/13	**92**	**91**	**84**	**92,31%**	**24**	**21**	**28**	**11**	**2+5**
			75+5+3+1		26,37%	23,08%	30,77%	12,09%	7,69%
			5/11, 3/6,1/1						
2013/14	**76**	**74**	**65**	**87,84%**	**22**	**23**	**15**	**5**	**6+3**
			57+8		29,73%	31,08%	20,27%	6,76%	12,16%
			8/10						
2014/15	**117**	**112**	**106**	**94,64%**	**31**	**35**	**30**	**10**	**10+1**
			102+4		26,5%	29,91%	25,64%	8,55%	9,4%
			4/5						
2015/16	**74**	**72**	**68**	**94,44%**	**6**	**31**	**22**	**7**	**5+1**
			62+5+1		8,1%	41,89%	29,73%	9,46%	10,81%
				Pragovi	85	70	60	50	
2016/17	**103**	**98**	**93**	**94,9%**	**10**	**37**	**29**	**17**	**8+2**
			85+7+1		9,7%	35,93%	28,16%	16,5%	9,71%
				Pragovi	85	70	60	50	
2017/18	**100**	**100**	**96**	**96%**	**5**	**17**	**36**	**38**	**4**
			84+9+3		5%	17%	36%	38%	9,71%
				Pragovi	85	70	60	50	
2018/19	**116**	**115**	**105**	**91,3%**	**2**	**21**	**62**	**20**	**9+2**
			104+1		1,8%	18,1%	53,4%	17,2%	9,5%
				Pragovi	85	70	60	50	

?

(još pitanja)

CC BY NC SA

Osnove programskog inženjerstva.
Životni ciklus razvoja programske podrške.

2019/20.01

CC BY NC SA

Programska potpora

- **Programska oprema/podrška/potpora, softver (software)**
  - dio računalnog sustava koji nema fizikalnih dimenzija
  - opći pojam za sve vrste programa, programskih jezika itd
  - Skup elemenata ili objekata u jedinstvenoj “konfiguraciji” koju čine računalni programi + podaci + dokumentacija
  - svojstva:
    - složenost, podložnost pogreškama,
    - ne troši se, teško mjerljiv,
    - stari, dugo se koristi,
    - lako se kopira (zajedno s pogreškama)
- **Primijenjena programska potpora = Računalna aplikacija (application)**
  - namjenski program, primjenska programska oprema
  - računalom podržano rješenje jednog ili više poslovnih problema ili potreba
- **Informacijski sustav = sustav aplikacija za upravljanje ljudskim aktivnostima**

CC BY NC SA

Vrste aplikacija

CC BY NC SA

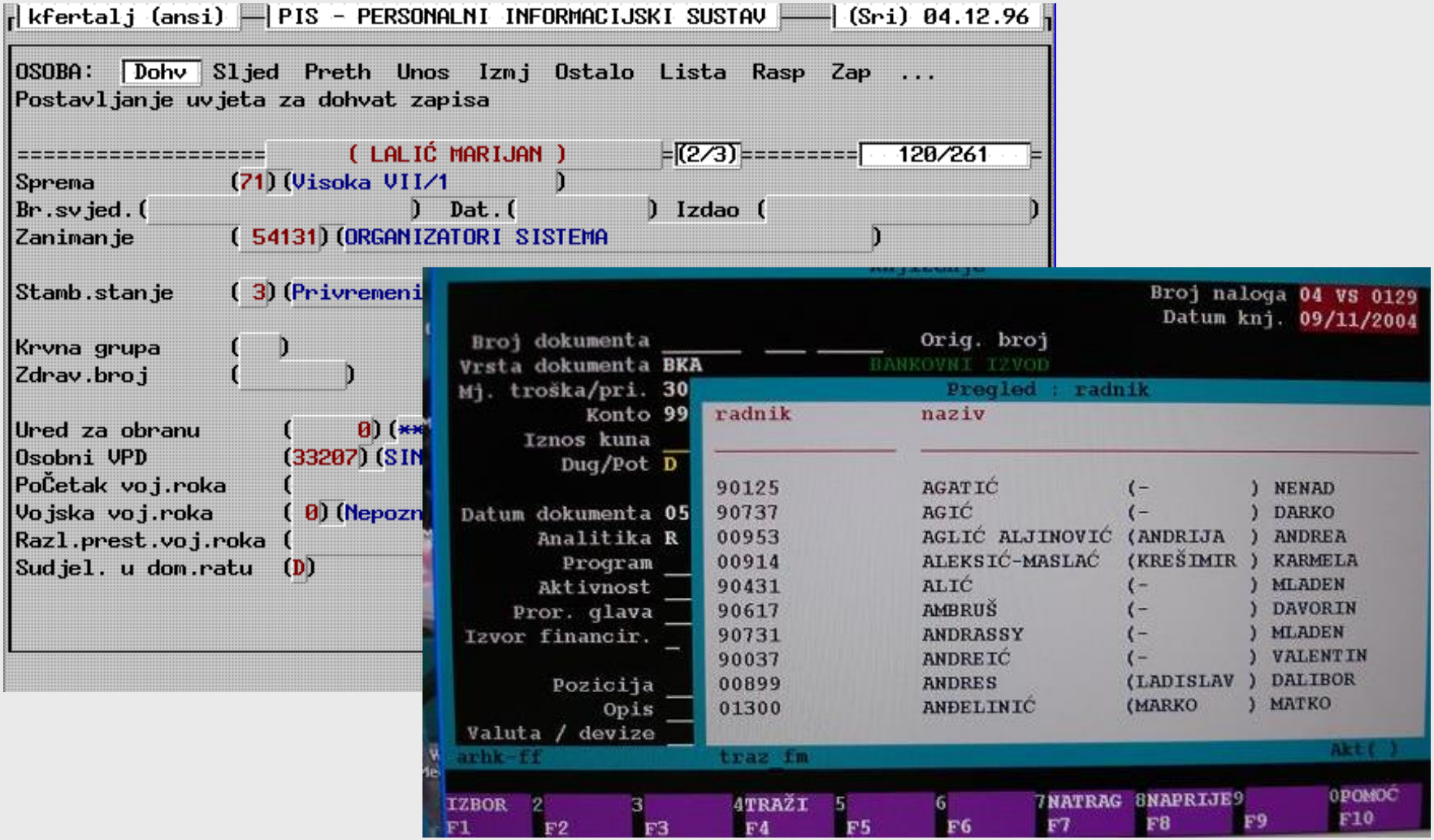
Jednokorisničke, samostalne aplikacije

- “standalone”, početkom 90-ih, dBase, Clipper, ZIM, ...

```
MR                                                    25.05.93

    MAIN MENU
Grad                                  Grad                  Add
Os
Ze                                Sifra zemlje     :  HR

                                  Naziv zem                    Zemlja                  1/1
Maticni broj
Prezime
Ime                               Postanski  Sifra zemlje     :  HR
Zemlja rodjenj
Mjesto rodjenj                    Naziv gra  Naziv zemlje     :  Republika Hrvatska·····
Naziv grada
Zemlja stanova                          F5  ENTER F2-Add F3-Delete F4-Change F8-Browse F9
Mjesto stanova
Naziv grada       :  Zagreb·····
Adresa            :  Srednjaci bb·····
                Zemlja rodjenja  : AAA
            F5-Zoom F6-Window F10-Save ESC
```

Poslužiteljske aplikacije

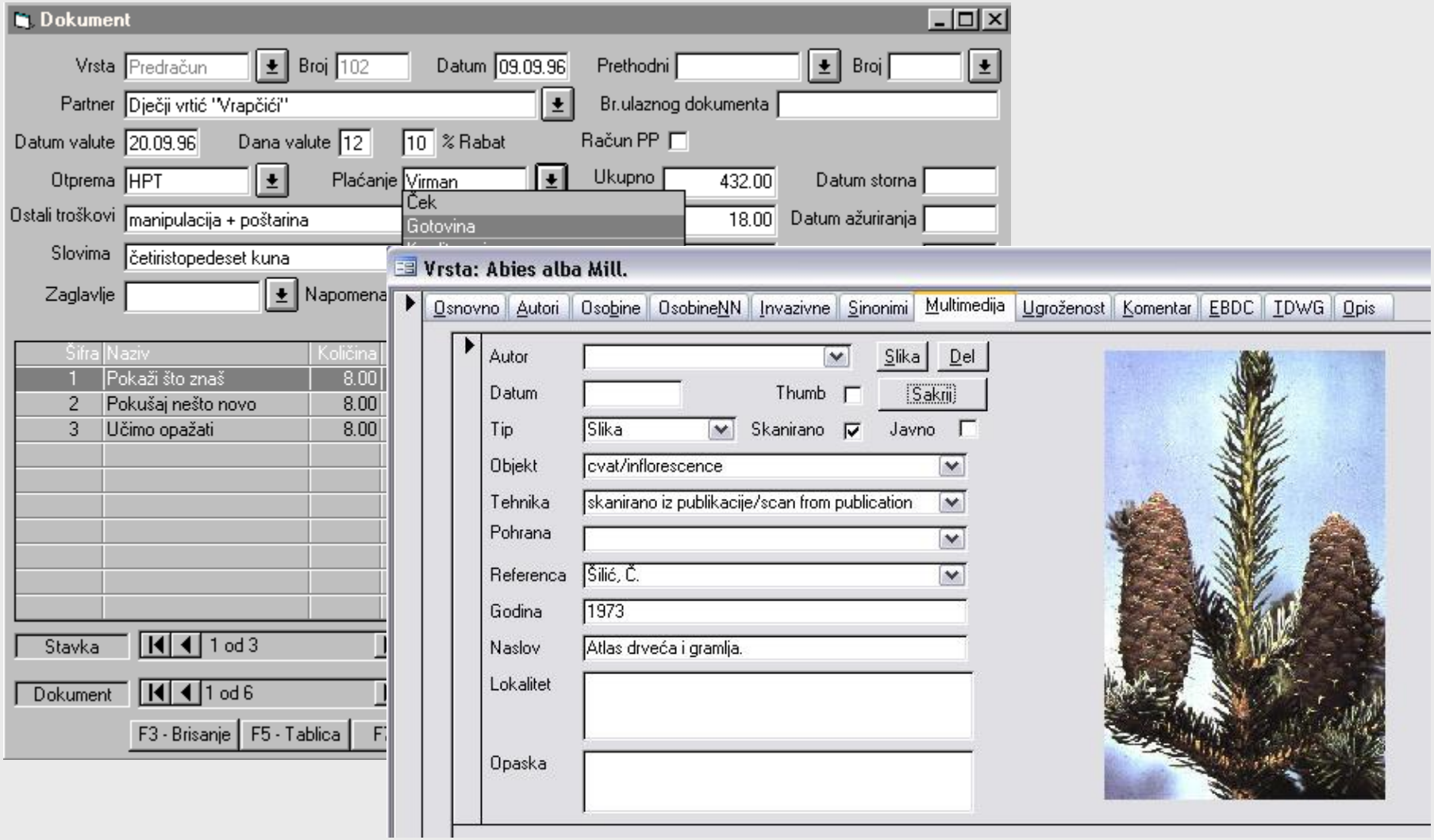
- serverske, 90-ih, Informix, Oracle, ...



CC BY NC SA

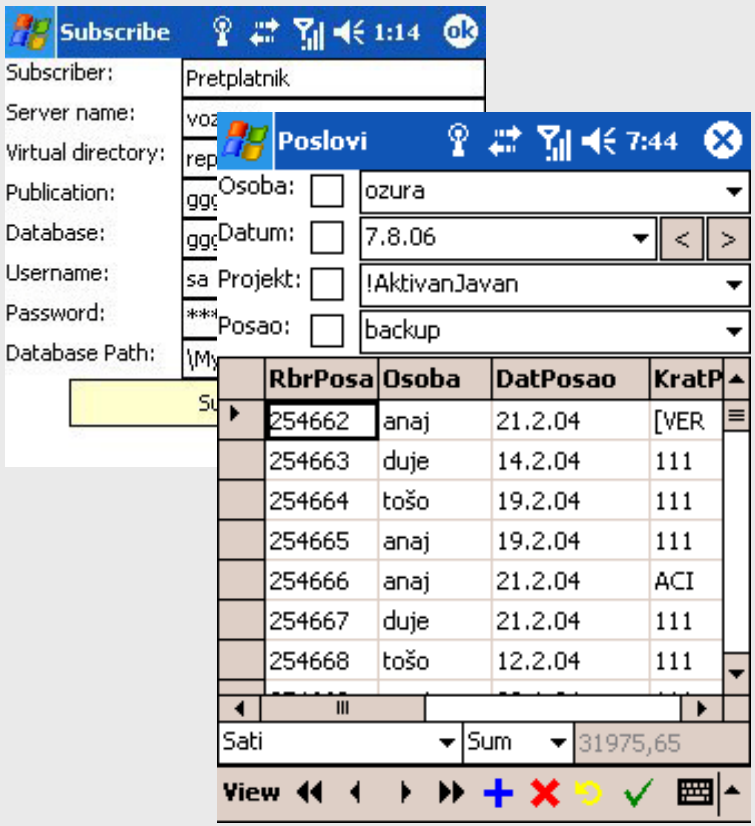
Klijentske aplikacije

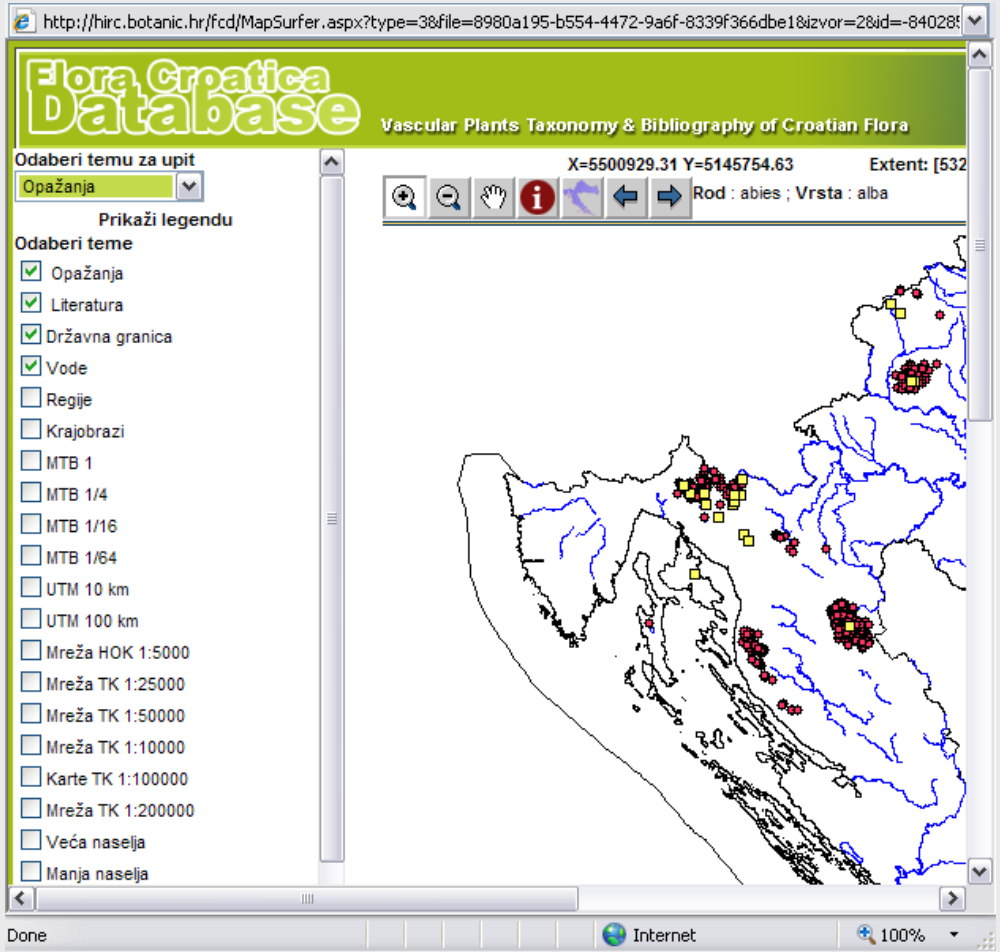
- “debeli” klijenti, kraj 90-ih, Microsoft Access, Visual Basic, Java, ...



Mobilne i distribuirane aplikacije (1)

- Internet, džepne, “tanki” klijenti, *remoting*, 2000-, .NET, J2EE





BY NC SA

Mobilne i distribuirane aplikacije (2)

- Web servisi + mobilne, web aplikacije, jQuery

Operations at http://hirc.botanic.hr/services/Herbar.svc

Count	GET	Vraća ukupni broj podataka u herbaru koji zadovoljavaju traženi filter. Za postavk
Count/{IdZbirke}	GET	Vraća ukupni broj podataka u herbarskoj zbirci koji zadovoljavaju traženi filter. Za
Sabiraci	GET	Vraća popis sabirača koje su zavedene u FCD-u koji počinju određenim nazivom (
Search	GET	Vraća popis herbara koji zadovoljavaju određeni filter (može biti prazan). Opciona imesvojte, datumsabiranja, inventarnibroj, zbirka, idslike, tdwg, porodica, tip. Filte godina, idherbara, idzbirke, inventarni broj (*), nagib opisnalazista (riječi odvojen Toni/saslikom=true/opisnalazista=otok Vis/idzbirke=14/godina=2011&sort=godina
SearchCollection/{IdZbirke}	GET	Pretražuje herbarsku zbirku. Vidi Search za detalje.
TDWG	GET	Vraća popis država po TDWG-u. Rezultat je lista parova (oznaka države, naziv
TipoviPrimjeraka	GET	Vraća popis tipova primjeraka herbara. Rezultat je lista parova (oznaka tipa, na
Zbirke	GET	Vraća popis zbirki koje su zavedene u FCD-u. Rezultat je lista parova (id zbirke

FCD®

Flora Croatica Database

Menu

Flora Croatica Database | Crvena knjiga | Bibliografija | Korisno bilje | Alohtone biljke | Galerija | Staništa

Opažanja | Herbar | Korisne poveznice | Prikaži praznu kartu

Kako koristiti baz

Prijava korisnika

1.3.2013. 12:22:3

Pretraga | Rezultati pretrage

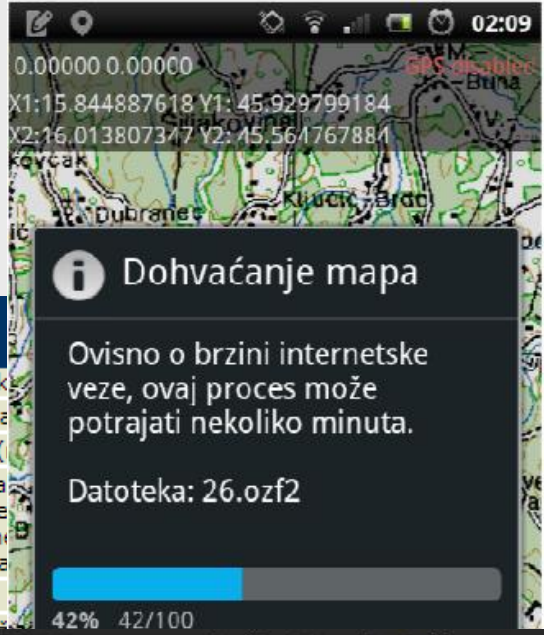
Kriterij pretrage: Godina sabiranja=2012, Naziv zbirke=CNHM Herbarium of Croatian Natural History Museum, TDWG=Croatia

1 2

Otisni herbarsku etiketu

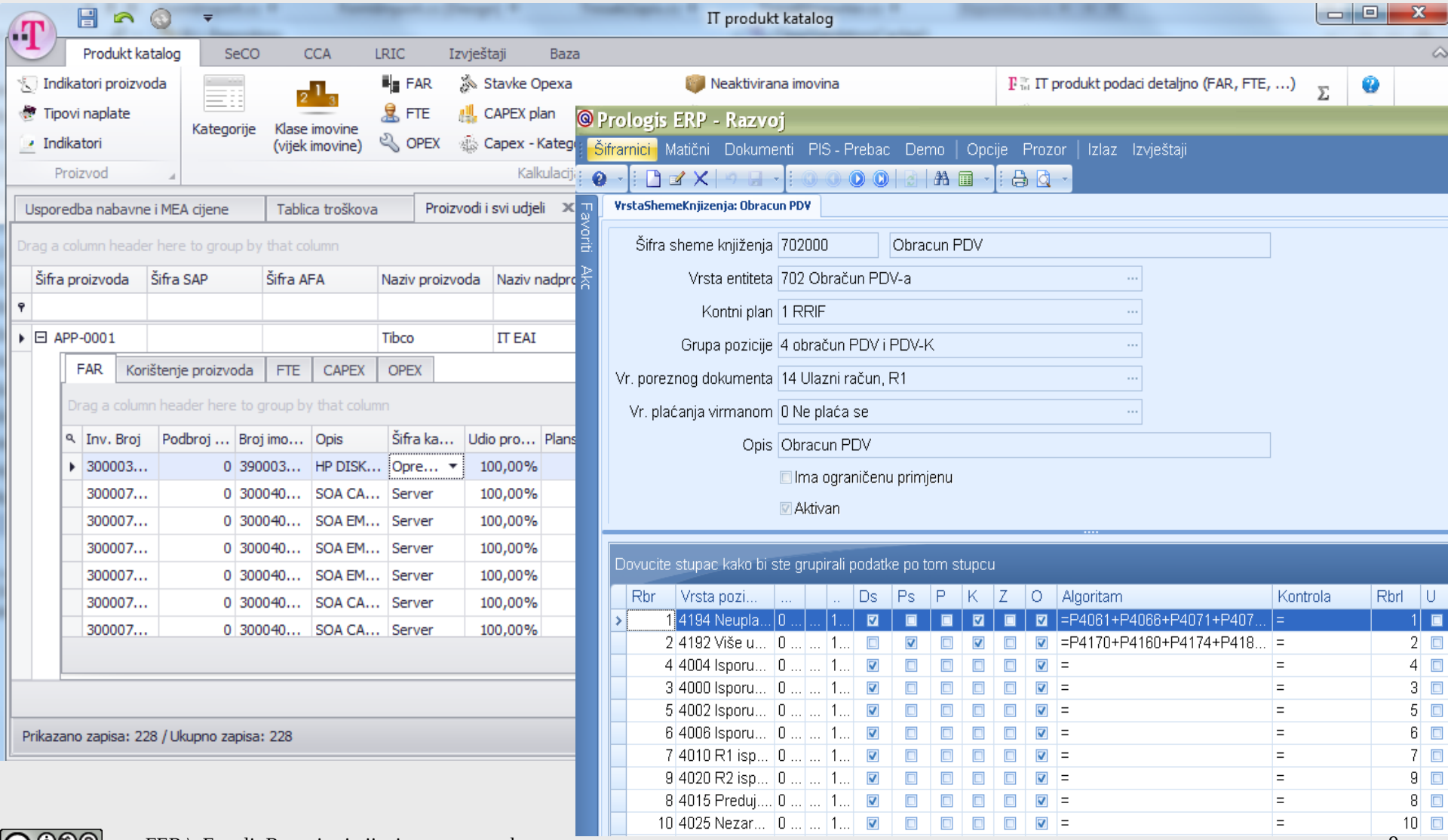
Zapisa po stranici 25 Ukupno rezultata: 4

	Id herbara	Slika	Porodica	Ime svojte	Sabirač	God.	Tip	Naziv zbirke	Država (TDWG)	Ozn koor
☐	31493		Rosaceae	Amelanchier ovalis Medik.	Vrbek, Mirjana	2012		CNHM Herbarium of Croatian Natural History Museum	Croatia	
☐	31492		Boraginaceae	Lithospermum purpurocaeruleum L.	Vrbek, Mirjana	2012		CNHM Herbarium of Croatian Natural History Museum	Croatia	



Debeli klijenti i višeslojne aplikacije (1)

- Ovisno o namjeni i debeli klijenti i višeslojne aplikacije



Debeli klijenti i višeslojne aplikacije (2)

- Aplikacije s različitim izvorima podataka (OPC server, baza podataka, konfiguracijske datoteke, …)

Control Recipe Editor (CRE)

01 : 005 - Milling

4 1055 Start line
Jump
6 1056 Milling - Local malt intake
7 1057 After run
8 1056 Milling - Silo malt intake
9 1057 After run
10 1058 Stop line

Batch ID: 0
Order ID: 0
Ctrl Num: 0
Batch spare: 0

Process cell: 03 Brewhouse
Recipe Category: 01 Production
Master Recipe: 01 Montel red

17:48:28 - Malt Quantity -3 → 1 OK
17:48:32 - Mill current - a : Value is not float
17:48:35 - Mill current - a : Value is not float

6-1056 Milling - Local malt intake

#	Id	Name level	Value	Unit	PasswordLevel	Min-Max
1	1	Control time	3.00	min	10	Unlimited
2	24	Mill current	a	A	10	Unlimited
3	32	Malt Quantity	1.00	kg	10	Unlimited
4	33	Amout of Malt in line	5.00	kg	10	Unlimited
5	35	Transfer Malt from:	1: Malt intake		10	
6	31	Fixed speed inlet rotary valve	12.00	%	10	Unlimited
7	37	Speed line rotary valve	54.00	%	10	Unlimited
8	43	Transfer from:	1: Second silo		10	

Version: **3.0.0.6218** User: **montel** Password level: **100** Group: **Development** Database: **MASTER** OPC Server: **opc.tcp://161.53.71.18:4845**

CC BY NC SA

Programsko inženjerstvo

- **Programsko inženjerstvo (software engineering)**
  - "Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is **the application of engineering to software**.
    - ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary
  - **sistematičan, discipliniran i mjerljiv pristup razvoju, primjeni i održavanju softvera**
  - **primjena inženjerskog pristupa na programsku opremu**
  - Programsko inženjerstvo je inženjerska disciplina koja obuhvaća sve aspekte izrade programske opreme. [Sommerville, 2004]
- **Područje programskog inženjerstva**
  - poslovi kojima se oblikuje i razvija programska oprema
  - sustavna primjena prikladnih alata i tehnika na čitav proces razvoja programske potpore

BY NC SA

Programsko inženjerstvo i srodna područja

"A scientist builds in order to learn; an engineer learns in order to build." (Fred Brooks)

- **Steve McConnell: „Software Engineering is not Computer Science"**
  - Professional Software Development
  - originalni isječak na https://www.gamasutra.com/view/feature/131817/software_engineering_is_not_.php
- **[Sommerville, 2004]**
  - računarska znanost fokusira se na teorijske osnove
  - programsko inženjerstvo orijentirano na praktičnu primjernu u razvoju i isporuci programske potpore
- **Petter J. Denning**
  - „Engineering has been marginalized by the unhealthy belief that engineering is the application of science".
    Communications of the ACM, Vol. 60 No. 12, Pages 20-23, 2017.

CC BY NC SA

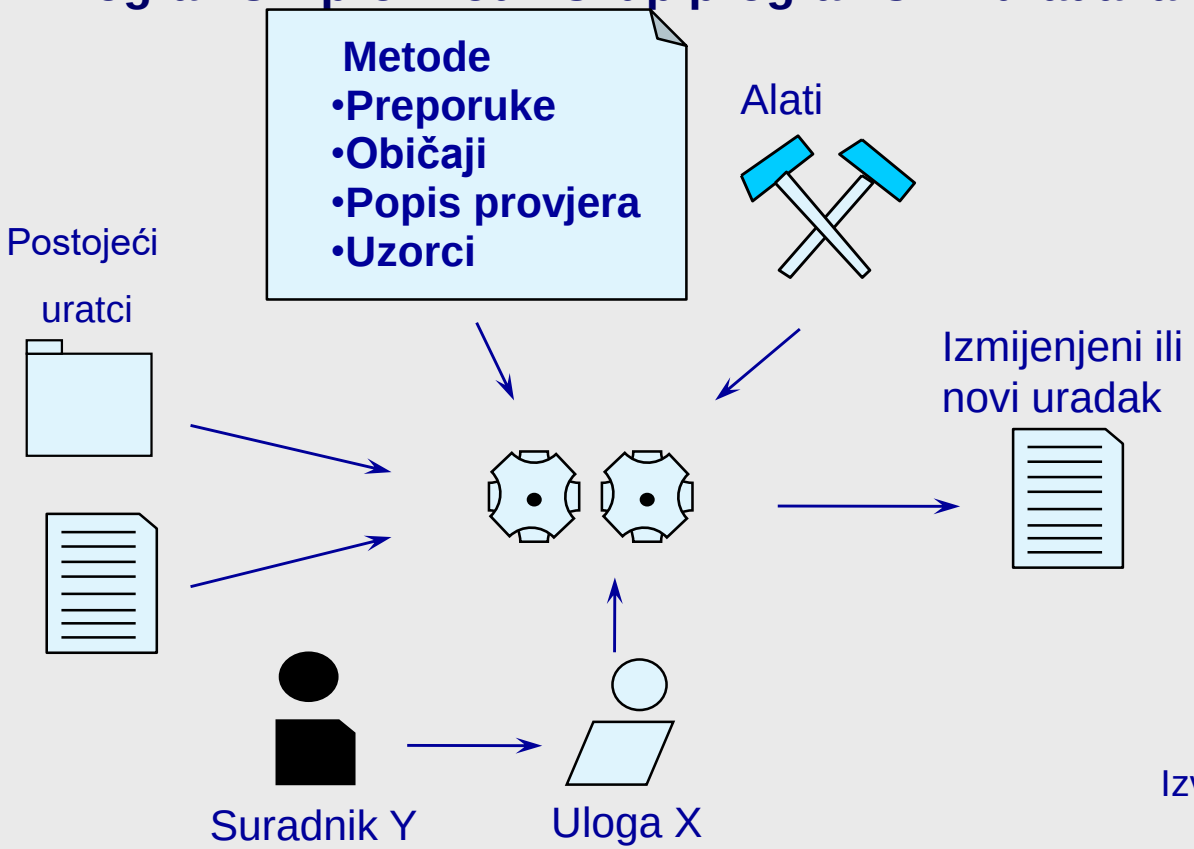
Modeli procesa razvoja

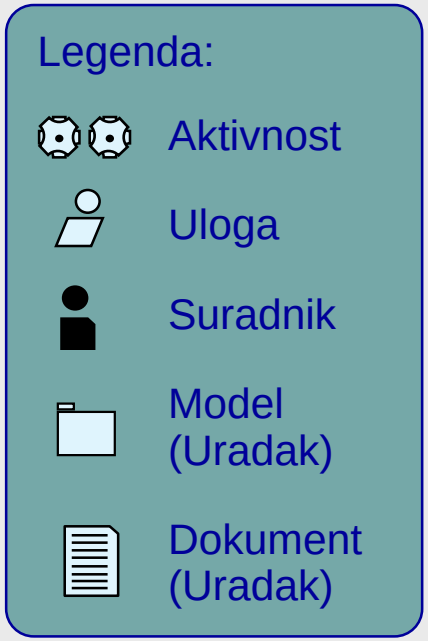
- **Model procesa**
  - Općenito: Plan razvoja, koji navodi opće postupke razvoja programskog proizvoda.
  - Preciznije: Definicija koja kaže koje aktivnosti treba obaviti, tko ih treba obaviti i u kojoj ulozi; kojim redoslijedom, koji će proizvodi biti razvijeni i kako ih vrednovati.

- **SDLC = software/systems development life-cycle**
  - model razvojnog procesa - unaprijed propisan proces razvoja
  - definira faze i zadatke (aktivnosti) koje treba obaviti tijekom razvoja
  - ciklus sigurava “kontrolne točke” za praćenje napretka, procjenu postignutih rezultata i donošenje odluka o daljnjim koracima

Aktivnosti procesa

- Aktvnost - podproces u modelu procesa
- Uloga - suradnik koji obavlja određeni posao, npr. voditelj projekta, arhitekt / specijalist za projektiranje, programer, administrator BP
- Programski uradak - dokument, model ili program
- Programski proizvod - skup programskih uradaka





Izvor: Balzert, vol. 1, p. 60

CC BY NC SA

Životni ciklus programske potpore

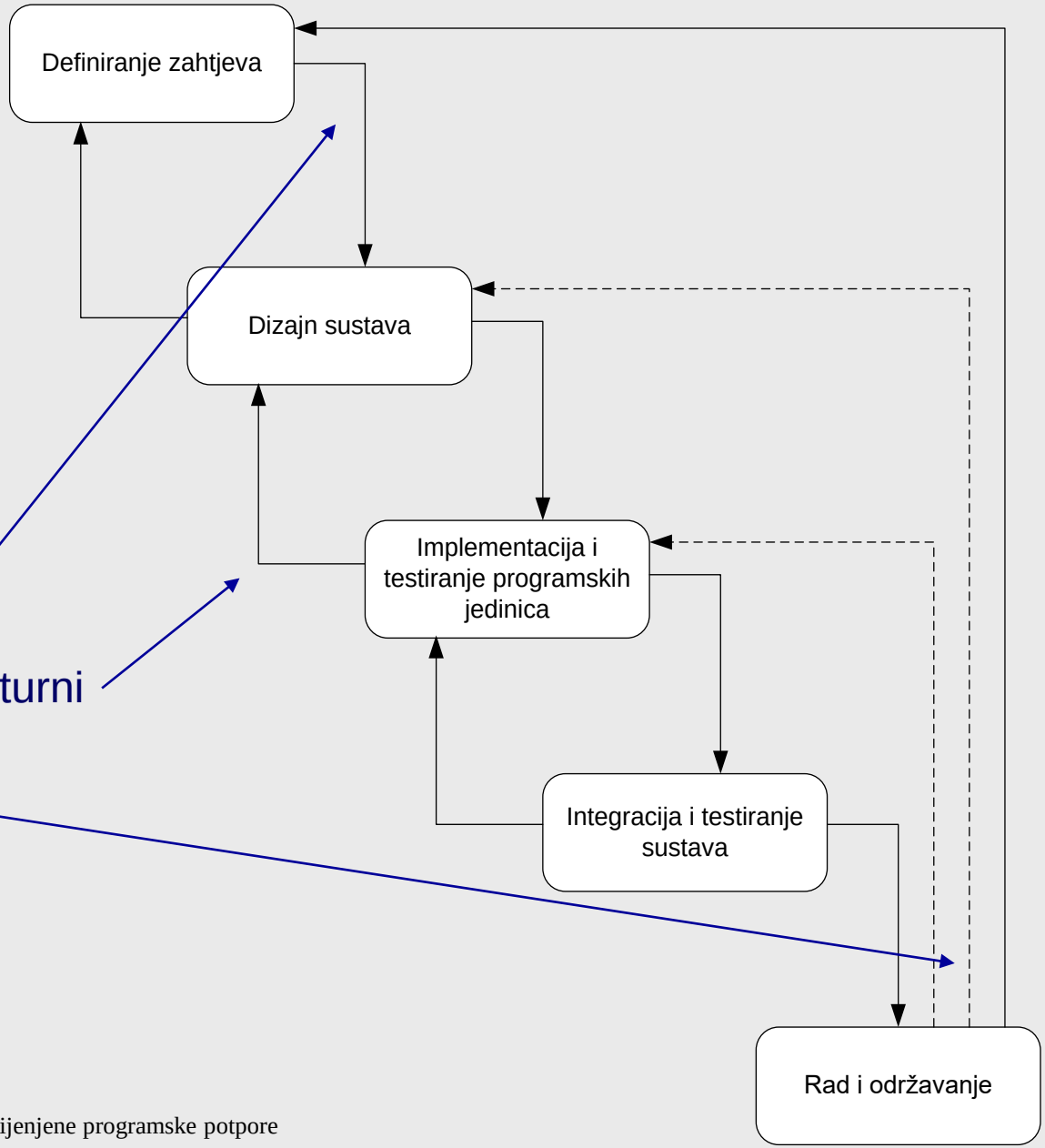
- **Planiranje (zašto)**
  - Zašto graditi sustav?
- **Analiza (tko, što, kada, gdje)**
  - Tko koristi sustav?
  - Što mora raditi?
  - Gdje i kada će se sustav koristiti?
- **Oblikovanje, projektiranje (kako)**
  - Kako sustav napraviti i kako će raditi?
- **Izrada**
  - ugradnja i provjera rješenja
- **Primjena**
  - isporuka, održavanje i poboljšavanje

CC BY NC SA

Faze životnog ciklusa

- **Planiranje**
  - Utvrđivanje ciljeva (poslovne koristi)
  - Analiza izvedivosti
  - Izrada plana rada
  - Ekipiranje projekta
  - Upravljanje projektom
- **Analiza**
  - Prikupljanje informacija
  - Modeliranje procesa
  - Modeliranje podataka
  - Specifikacija zahtjeva
- **Projektiranje, oblikovanje**
  - Dizajn arhitekture
  - Dizajn baze podataka i datoteka
  - Dizajn sučelja
  - Dizajn programa
- **Izrada, ugradnja (implementacija)**
  - Konstrukcija
  - Testiranje
  - Instalacija
- **Primjena**
  - Rad
  - Održavanje

Vodopadni model (waterfall)



- **Varijante**
  - klasični
  - pseudostrukturni
  - radikalni

BY NC SA

Vodopadni model razvoja

- **Definiranje zahtjeva (*requirements analysis and definition*)**
  - definira funkcionalnost programske potpore prema zahtjevima korisnika.
- **Dizajn sustava (*system and software design*)**
  - definira cjelokupnu arhitekturu programske podrške
  - grubi model sustava razrađuje se u detaljni opis izvedbe
- **Ugradnja i testiranje jedinica (*implementation and unit testing*)**
  - faza kodiranja tijekom koje se zahtjevi prevode u programski kod
  - programske jedinice zasebno se testiraju provjerom naspram specifikacije
- **Integracija i testiranje sustava (*integration and system testing*)**
  - programske jedinice povezuju se u cjelinu
  - provjerava se odgovara li programska potpora zahtjevima korisnika
- **Primjena i održavanje (*deployment/operation and maintenance*)**
  - započinje predajom sustava korisnicima na uporabu
  - tijekom održavanja uklanjaju se naknadno uočene neispravnosti te se sustav proširuje i poboljšava prema potrebama

CC BY NC SA

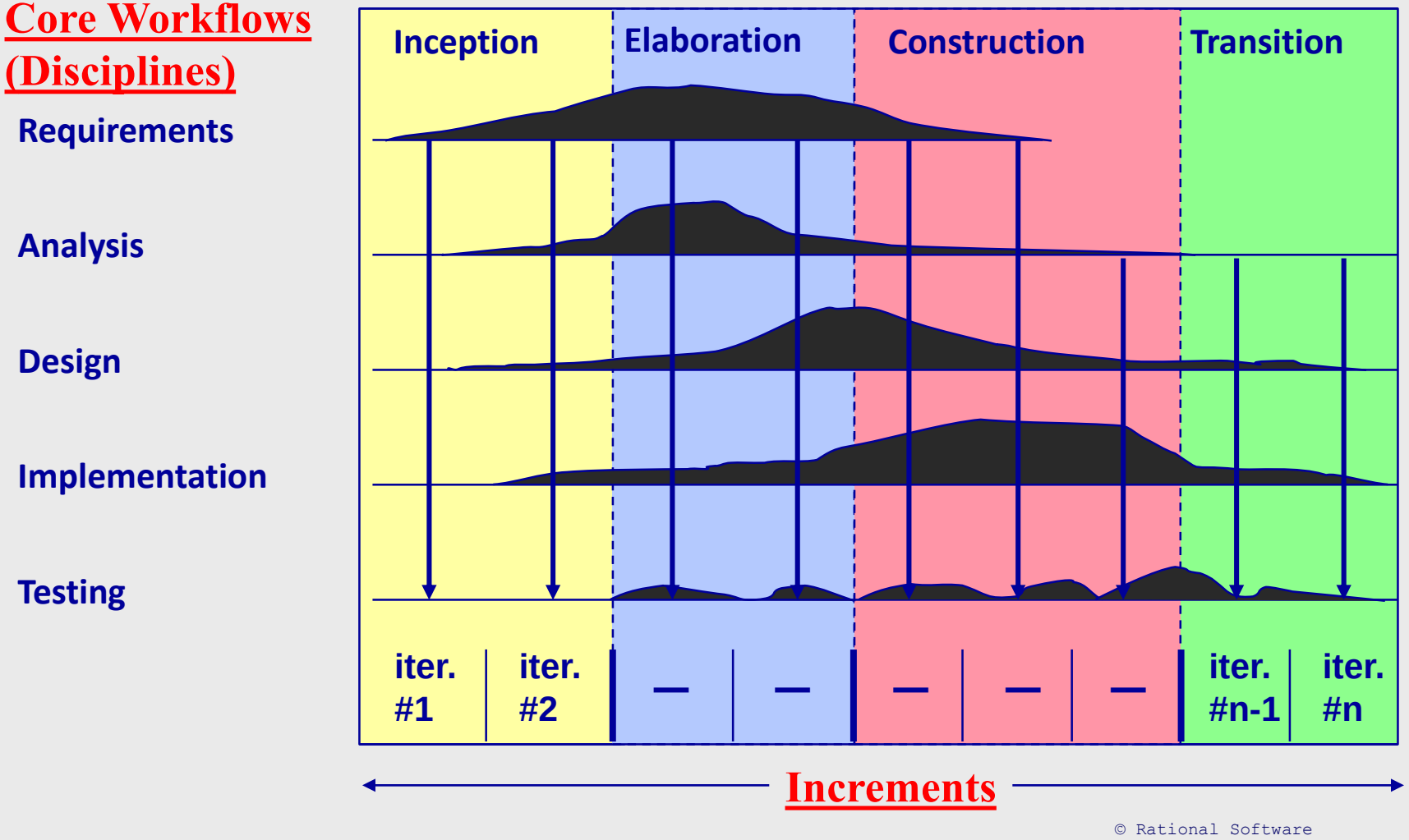
Iterativni postupak razvoja

- **Unified software development process (UDP)**
  - izvorno Objectory
  - danas IBM Rational Unified Process (RUP)

- **Iterativni i inkrementalni razvoj**
  - softver se razvija i objavljuje po dijelovima
  - glavne faze obavljaju se kroz niz iteracija
    - svaka iteracija obavlja se standardnim životnim ciklusom koji uključuje analizu, oblikovanje, ugradnju i provjeru
    - rezultat iteracije je proizvod završne kakvoće (production-quality), provjeren i integriran, koji zadovoljava podskup ukupnih zahtjeva
    - isporuke mogu biti interne ili prema korisnicima
- **RUP sadrži niz "predložaka" razvojnih procesa (roadmaps) za različite modele razvoja i tipove projekata**

BY NC SA

Faze i koraci razvoja



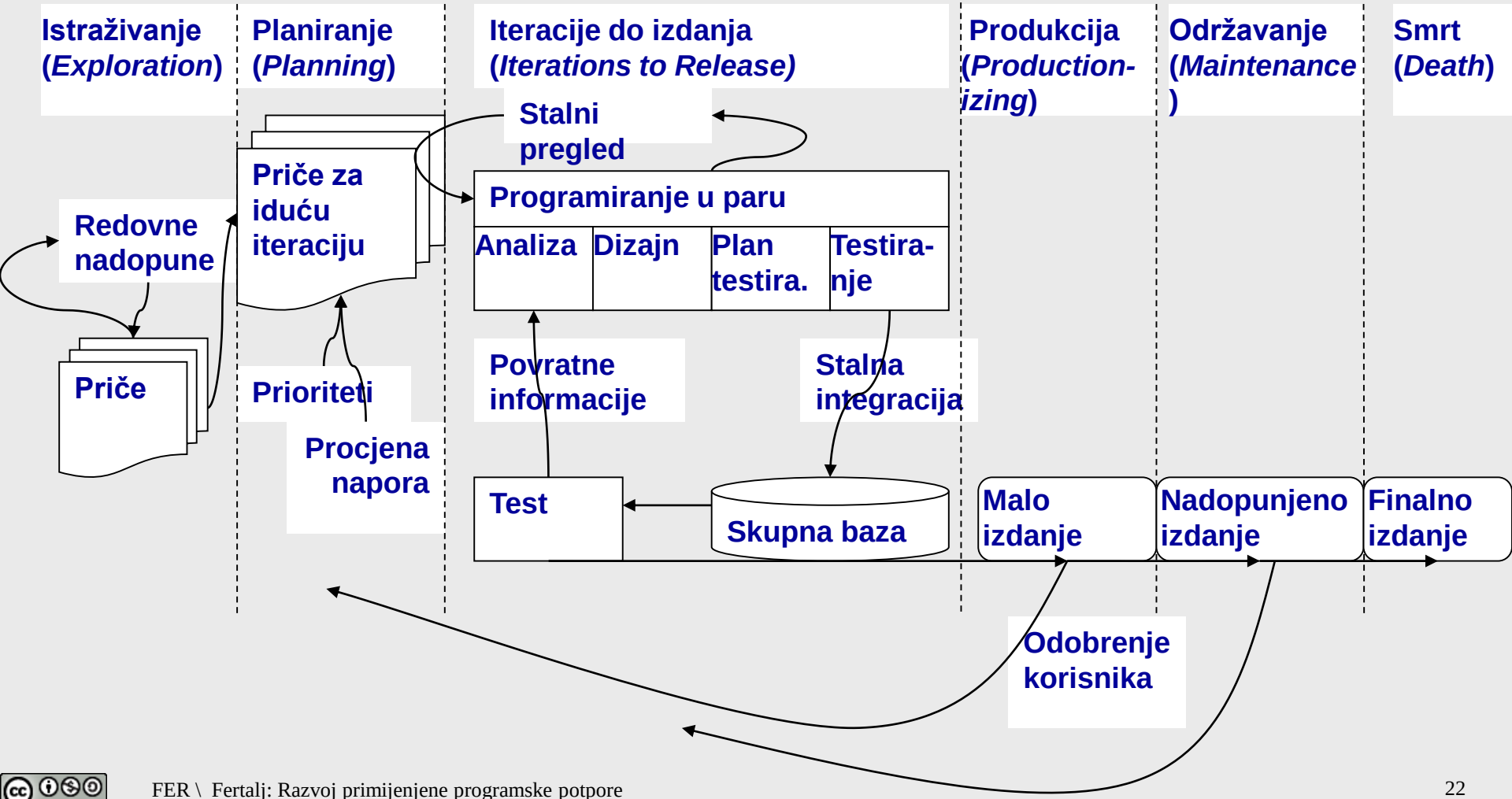
Glavne faze razvoja

- **Počinjanje (Inception)**
  - opravdanje razloga za pokretanje projekta
  - prikupljanje najvažnijih zahtjeva (10% detaljno)
  - određivanje dosega projekta
- **Elaboracija (Elaboration)**
  - prikupljanje detaljnih zahtjeva (80%)
  - globalna (high-level) analiza i dizajn
  - ustanovljavanje osnovne arhitekture
  - planiranje konstrukcije
- **Konstrukcija, gradnja (Construction)**
  - prikupljanje ostalih zahtjeva + promjene zahtjeva
  - razrada arhitekture i izrada sustava
  - kontinuirana integracija
- **Prijelaz (Transition)**
  - beta testiranje, podešavanje performansi, poduka korisnika
  - provjera prihvatljivosti i zadovoljstva korisnika
- **Post-implementacija (Post-deployment)**
  - nastavak evolucijskog razvoja
  - uz očuvanje integriteta aplikacije

cc BY NC SA

Ekstremno programiranje

- Agilni razvojni proces
- Vrijednosti (values): komunikacija, jednostavnost, povratne informacije, hrabrost, uvažavanje (respect)



CC BY NC SA

Faze ekstremnog programiranja

- **Istraživanje**
  - Korisnici bilježe svoje priče na kartice
  - Svaka kartica sadrži jednu mogućnost programa.
  - Projektni tim se pobliže upoznaje s alatima, tehnologijom i postupcima projekta.
  - Radi se prototip sustava za testiranje tehnologije i varijanti arhitekture sustava.
  - Faza istraživanja traje nekoliko tjedana do nekoliko mjeseci
- **Planiranje**
  - Postavlja prioritete na korisničke priče (tj. svojstva programskog rješenja)
  - Planira se doseg prvog malog izdanja i vrijeme za pojedinu karticu
  - Zatim se određuje cjelokupni vremenski raspored.
  - Rok za izdavanje prvog malog izdanja obično je unutar dva mjeseca.
  - Faza planiranja traje nekoliko dana.

BY NC SA

Faze ekstremnog programiranja

- **Iteracije do izdanja**
  - Uključuje nekoliko iteracija sustava prije prvog izdanja.
  - Vremenski raspored iz faze planiranja se razlaže u više iteracija
  - Pojedina iteracija traje jedan do četiri tjedna.
  - Prva iteracija stvara takav sustav koji obuhvaća cijelu arhitekturu ciljanog sustava.
  - Klijent određuje kartice koje će se koristiti pri svakoj narednoj iteraciji.
  - Testovi prihvatljivosti izvode se na kraju svake iteracije.
  - Na kraju posljednje iteracije, sustav je spreman za produkciju.
- **Produkcija**
  - Dodatno testiranje i provjera performansi sustava prije isporuke klijentu.
  - Razrješenje primjedbi na sustav te odlučivanje da li će se riješiti u ovom izdanju.
  - Iteracije trajanja tri do najviše tjedan dana.
  - Zakašnjele nove ideje i prijedlozi se dokumentiraju i njihova implementacija odgađa.

cc BY NC SA

Faze ekstremnog programiranja

- **Nakon što je prvo izdanje pušteno u produkciju,**
  - XP projekt mora istovremeno održavati softver u primjeni i proizvoditi nove iteracije
  - Zbog toga se brzina implementacije smanjuje
  - Održavanje može zahtijevati nove članove projektnog tima i promjenu strukture tima.

- **Faza smrti je blizu kada klijent nema više novih kartica s pričama**
  - Podrazumijeva se da sustav zadovoljava sve zahtjeve (npr. pouzdanost i stabilnost).
  - Vrijeme u XP projektu da se konačno napiše sva korisnička dokumentacija budući da više nema promjena na arhitekturi, dizajnu i kodu sustava.
  - Smrt može nastupiti i kada sustav ne ispunjava sva korisnička očekivanja, ili ako postane preskup za daljnji razvoj.

Primjer evidencija korisničkih priča

Backlogs Queries

You can now add more backlog levels to this team. Access this setting through the settings dialog.

Features
Backlog items

Past
Sprint 1
Sprint 2
Sprint 3
Bugovi i dorade 1

Current
Bugovi i dorade 2

Backlog items

Backlog Board

Forecast Off Mapping Off Parents Hide In progress items Show

New Create query Column options Filter

Order	Work Item Type	Title	State	Ef	Area Path	Iteration Path	Assigned
19	Bug	Klik na rezultat na počenoj stranici treba ažurirati rezult...	New		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
20	Bug	Promjena datuma za natjecanje u znanju 2x pita za dan,...	New		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
21	Bug	U postavke za sport dodati koliko bodova nosi pobjeda...	Committed		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrnj
22	Product Backlog Item	Rezultati krosa (M)	New		Elektrijada\SQL	Elektrijada	
23	Product Backlog Item	Rezultati krosa (Ž)	New		Elektrijada\SQL	Elektrijada	
24	Bug	Dodatne informacije se ne vidi u susretima po skupinama	Committed		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrnj
25	Bug	Dodati u tablici grupne faze broj bodova i po tome sorti...	Committed		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrnj
26	Bug	Dodati u postavke sporta tipove komparatora	Committed		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrnj
27	Bug	Ako je nešto u favoritima, onda zvjezdica mora biti oboj...	New		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
28	Bug	Gdje bi se jednostavno mogao vidjeti poredak u nekom...	New		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
29	Product Backlog Item	Rezultati veslanja	New		Elektrijada\SQL	Elektrijada	
30	Product Backlog Item	Prikaz rezultata znanja	New		Elektrijada\Web	Elektrijada\Sprint 3	Luka Juki
31	Product Backlog Item	Rezultati šaha	New		Elektrijada\SQL	Elektrijada	
32	Product Backlog Item	Skica Android aplikacije	Committed		Elektrijada\Andr...	Elektrijada\Sprint 2	Ivan Oreh
33	Bug	Nemogu ukloniti događaj iz favorita	New		Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
34	Product Backlog Item	Rezultati - Osnove elektrotehnike	New		Elektrijada\SQL	Elektrijada	
35	Product Backlog Item	Rezultati - Matematika 2	New		Elektrijada\SQL	Elektrijada\Sprint 2	Toni Baka
36	Product Backlog Item	Rezultati - Matematika 1	New		Elektrijada\SQL	Elektrijada\Sprint 2	Toni Baka

Primjer grafičkog pregleda statusa priča/zadataka

Backlog **Board**

New	Approved 0/5	Committed 21/5	Done
New item		Stranica s postavkama za izračun kako se boduje i organizira koje natjecanje Tomislav Maslač	Skratiti prikaz kod uspjeha fakulteta
Na početnoj stranici ne radi ažuriranje/brisanje događaja		Bluetooth prijenos Vjeran Hanžek	Dodati RadioButtone za prikaz bodova ili pozicije u rezultatima Toni Bakarčić
Definirati build i deploy za web aplikaciju Boris Milašinović		Bluetooth prijenos Toni Bakarčić	Prikaz pozicija u rezultati u znanju ne radi Tomislav Božurić
Login preko FER weba Luka Jukić		Brisati grupu koja je trenutno označena, a ne zadnju. Leon Hrnjak	Dodati mogućnost da se pojedinac u znanju natječe individualno Tomislav Božurić
Favoriti, sport i znanje prikazati na ekranu, a ne kroz opcije menija		Omogućiti unos pozicije tima u znanju	Sinkronizacija s webom sruši aplikaciju Leon Hrnjak
Obojane zvjezdice za favorite na random mjestima		Prikaz rezultata u sportovima Luka Jukić 0/3	Aktivirati formu za vijesti Leon Hrnjak
Uvoz podataka iz Excela			
Vizualni prikaz rezultata nekog sporta 0/1			

Reference

- https://www.computer.org/portal/web/swebok/swebokv3
- https://pascal.computer.org/sev_display/index.action
- http://www.rspa.com/
- https://www-01.ibm.com/software/hr/rational/
- http://www.extremeprogramming.org/
- http://www.agilemodeling.com/
- http://pm-training.net/agile-development-methodology-wiki/

Priprema za projekt

- **Formiranje projektnih ekipa**
  - RPPP-Popis.xls

- **Instalirati potreban softver i povezati se na odgovarajuće repozitorije**
  - Repozitorij s predavanjima
    - https://rppp.fer.hr/tfs/studentcollection/_git/RPPP-FER2
  - Repozitorij vlastite grupe
    - https://rppp.fer.hr/tfs/studentcollection/RPPP-FER2/_git/XX

- **Intervju i korisnička priča**
  - uvodno iznošenje zahtjeva korisnika na 2. predavanjima
  - po potrebi pojašnjenje zahtjeva na labosu i predavanjima u 3. tjednu
  - dodatne informacije na konzultacijama

CC BY NC SA

Projekt izrade aplikacije
Specifikacija zahtjeva

Projekt

- **Projekt**
  - Projekt je vremenski određeno nastojanje da se proizvede jedinstven proizvod, usluga ili rezultat. [PMBOK – Project Management Book of Knowledge, PMI]
  - Projekt je niz jedinstvenih, složenih i povezanih aktivnosti koje imaju određeni cilj i koji se mora postići u zadanom vremenskom roku, u okviru zadanog proračuna i u skladu sa specifikacijama. [Wisocky, Beck and Crane]

- **Vremenska određenost**
  - Svaki projekt mora imati jasno određen početak i kraj. Projekti mogu biti kratki ili trajati godinama, ali će svakako završiti.
  - Projekt završava u trenutku kada postane jasno da su ciljevi projekta dostignuti ili kada se zaključi da ciljevi projekta ne mogu ili neće biti dostignuti.
- **Jedinstvenost**
  - Projekt se odnosi na rad na nečemu što prije nije postojalo i što se razlikuje od rezultata nastalih sličnim projektima.

Upravljanje projektom

- **Upravljanje, rukovođenje projektom (Project management)**
  - Upravljanje projektima je primjena znanja, vještina, alata i tehnika u projektnim aktivnostima da bi se ispunili projektni zahtjevi. [PMI]
- **Upravljanje projektom uključuje**
  - Planiranje
    - Utvrđivanje zahtjeva
    - Postavljanje jasnih i ostvarivih ciljeva
    - Uravnoteženje zahtjeva na kvalitetu, doseg, vrijeme i trošak,
    - Prilagodbu interesima i očekivanjima zainteresiranih strana – dionika (eng. Stakeholders)
  - Organiziranje
    - Formiranje projektnog tima
  - Raspoređivanje obaveza
    - Tko što i kada treba napraviti
  - Usmjeravanje
    - Nadgledanje, omogućavanje izvršenja
  - Kontroliranje
    - Provjera učinka i rezultata

CC BY NC SA

Uloge na projektu

- **Korisnik, Korisnik usluga, Klijent (User, Customer, Client)**
  - osoba ili grupa, naručitelj ili krajnji korisnik
- **Sponzor projekta (project sponsor)**
  - Osoba ili grupa koja osigurava (financijske) resurse za projekt
- **Voditelj projekta (project manager)**
  - Osoba imenovana kako bi ostvarila ciljeve projekta
- **Resursi projekta**
  - Osobe, oprema, usluge, materijal, budžet ili druga sredstva.
- **Projektna ekipa**
  - Svi članovi ekipe, uključujući upravljačke, a u nekim slučajevima i sponzora
    - voditelj – upravljanje projektom
    - sistem analitičar – određivanje potreba, specifikacija zahtjeva i dizajna
    - projektant/arhitekt – uspostava osnovne arhitekture
    - razvojnik (developer, builder) – kodiranje, testiranje
    - administrator baza podataka – administriranje DBMS
    - sistem inženjer / sistem administrator – administriranje OS i mreže

BY NC SA

Dokumentiranje projekta

- **Povelja projekta (Project Charter)**
  - Dokument kojim pokretač projekta ili sponzor odobrava projekt i ovlašćuje voditelja za primjenu organizacijskih resursa u provedbi projekta.
- **Plan projekta = Plan upravljanja softverskim projektom**
  - IEEE Standard for Software Project Management Plans 1058-1998
  - dokument koji opisuje sveukupnu organizaciju projekta
- **Primjeri 🗁Prilozi**
  - PlanProjekta.dot
  - Firma-PlanProjekta.doc
- **Plan može sadržavati raspored**
  - općenito: SoftwareDevelopment.mpp, PlanRazvoja.mpp
  - konkretno: Firma-PlanProjekta.mpp

1. Introduction
   - 1.1 Project Overview
   - 1.2 Project Deliverables
   - 1.3 Evolution of the Software Project Management Plan
   - 1.4 Reference Materials
   - 1.5 Definitions and Acronyms
2. Project Organization
   - 2.1 Process Model
   - 2.2 Organizational Structure
   - 2.3 Organizational Boundaries and Interfaces
   - 2.4 Project Responsibilities
3. Managerial Process
   - 3.1 Management Objectives and Priorities
   - 3.2 Assumptions, Dependencies, and Constraints
   - 3.3 Risk Management
   - 3.4 Monitoring and Controlling Mechanisms
   - 3.5 Staffing Plan
4. Technical Process
   - 4.1 Methods, Tools, and Techniques
   - 4.2 Software Documentation
   - 4.3 Project Support Functions
5. Work Packages, Schedule, and Budget
   - 5.1 Work Packages
   - 5.2 Dependencies
   - 5.3 Resource Requirements
   - 5.4 Budget and Resource Allocation
   - 5.5 Schedule
6. Additional Components
7. Index
8. Appendices

cc BY NC SA

Izrada plana projekta

- **Koraci izrade plana projekta**
  - Izrada liste zadataka
  - Izrada hijerarhije zadataka (work breakdown structure)
  - Procjena trajanja zadataka
  - Izrada ovisnosti među zadacima
  - Dodjela resursa

	Zadatak	Trajanje (u danima)	Prethodnici	Naziv resursa
1	**Doseg**	**7**		
2	Određivanje dosega	5		Voditelj projekta
3	Određivanje sponzorstva	1,5	2	Sponzor projekta;Voditelj projekta
4	Određivanje resursa	2	2	Voditelj projekta
5	Dovršetak dosega	0	3;4	
6	**Analiza/Softverski zahtjevi**	**16**		
7	Analiza potreba	5	5	Sistem analitičar
8	Prikupljanje informacija	6	7	Sistem analitičar
9	Prijedlog izvedbe sustava	5	8	Sistem analitičar
10	Dovršetak analize	0	9	
11	**Dizajn**	**5**		
12	Razvoj funkcionalnih specifikacija	5	10	Sistem analitičar;Projektant
13	Izrada baze podataka	5	10	Administrator baze podataka
14	Dovršetak dizajna	0	13;12	
15	**Razvoj**	**35**		
16	Izrada formi korisničkog sučelja	4	12	Razvojnik
17	Izrada funkcija za pohranu podataka	12	14	Razvojnik
18	Izrada funkcija za ispis izvještaja	15	14	Razvojnik
19	Izrada izvještaja	10	18	Razvojnik
20	Izrada funkcija izračuna	12	16	Razvojnik
21	Razvojno testiranje (debugiranje)	10	16;17;18;19;20	Razvojnik
22	Dovršetak razvoja	0	21	
23	**Testiranje**	**12**		
24	Izrada testova programskig cjelina prema specifikacijama proizvoda	5	14	Tester
25	Izrada plana integracijskog testiranja prema specifikacijama proizvoda	5	14	Tester
26	Testiranje komponenti prema specifikacijama proizvoda	7	24	Tester
27	Provjera integracije modula	7	25	Tester
28	Dovršetak testiranja	0	26;27	
29	**Dokumentacija**	**5**		
30	Razvoj specifikacija i sustava pomoći	5	22	Razvojnik
31	Dovršetak dokumentacije	0	30	Razvojnik
32	**Uvođenje sustava i poduka korisnika**	**5**		
33	Ugradnja programske potpore	5	28	Razvojnik;Sistem administrator
34	Dovršetak uvođenja sustava i poduke korisnika	0	33	

BY NC SA

Izrada liste zadataka

- **Zadaci – osnovni gradbeni elementi svakog projekta**
  - predstavljaju posao koji se mora obaviti da bi se postigao cilj projekta
  - opisuju tijek događaja, trajanja i zahtjeva za resursima na projektu
- primitivni zadaci
  - zadaci koji se dekompozicijom ne mogu podijeliti na jednostavnije zadatke
- skupni zadaci (summary tasks)
  - zbrajaju trajanje i troškove primitivnih zadataka
  - trajanje, datum te izračunate vrijednosti se automatski izvode iz skupa primitivnih zadataka
- prekretnice ili miljokazi (milestones)
  - ključni događaj ili krajnji rok odnosno cilj koji treba postići
  - trajanja 0
  - služe za provjeru stupnja dovršenosti drugih zadataka
  - pomak ključnog događaja ima za posljedicu vremenski preraspored

cc BY NC SA

Izrada hijerarhije zadataka

- **Faza - grupa povezanih zadataka koji se odnose na fazu projekta**
  - Zbirni zadaci se odnose na faze

- **WBS (work breakdown structure)**
  - hijerarhijska lista faza, zadataka i prekretnica
  - osnova za pregledni raspored projekta

- **Dva su pristupa razvoju zadataka i faza:**
  - Planiranje s vrha prema dolje (Top-down)
    - pristup od općeg prema specifičnom
    - identificira glavne faze i rezultate projekta prije dodavanja zadataka potrebnih za završetak tih faza
    - složeni projekti mogu imati nekoliko slojeva faza
  - Planiranje s dna prema gore (Bottom-up)
    - pristup od specifičnog prema općem
    - identificira što više zadataka najnižeg sloja prije grupiranja u faze

cc BY NC SA

Procjena trajanja zadataka

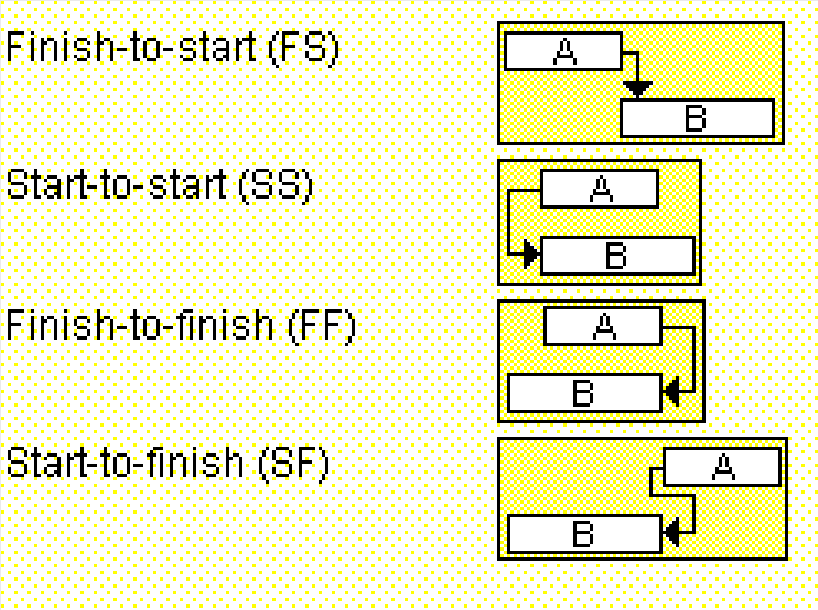
- **Trajanje zadatka**
  - očekivana količina vremena za završetak zadataka
    - minute (m), sati (h), dani (d), tjedni (w), mjeseci (mo)

	Zadatak	Trajanje (u danima)	Prethodnici	Naziv resursa
1	**Doseg**	**7**		
2	Određivanje dosega	5		Voditelj projekta
3	Određivanje sponzorstva	1,5	2	Sponzor projekta;Voditelj projekta
4	Određivanje resursa	2	2	Voditelj projekta
5	Dovršetak dosega	0	3;4	
6	**Analiza/Softverski zahtjevi**	**16**		
7	Analiza potreba	5	5	Sistem analitičar
8	Prikupljanje informacija	6	7	Sistem analitičar
9	Prijedlog izvedbe sustava	5	8	Sistem analitičar
10	Dovršetak analize	0	9	
11	**Dizajn**	**5**		
12	Razvoj funkcionalnih specifikacija	5	10	Sistem analitičar;Projektant
13	Izrada baze podataka	5	10	Administrator baze podataka
14	Dovršetak dizajna	0	13;12	

CC BY NC SA

Međuzavisnost zadataka

- **Projekt može zahtijevati da zadaci budu napravljeni u određenom redoslijedu**
  - Niz – iza jednog slijedi drugi zadatak
  - Zavisnost – sljedbenik (successor) može biti izvršen ako je dovršen prethodnik (predecessor)
  - Bilo koji zadatak može biti prethodnik jednom ili više sljedbenika

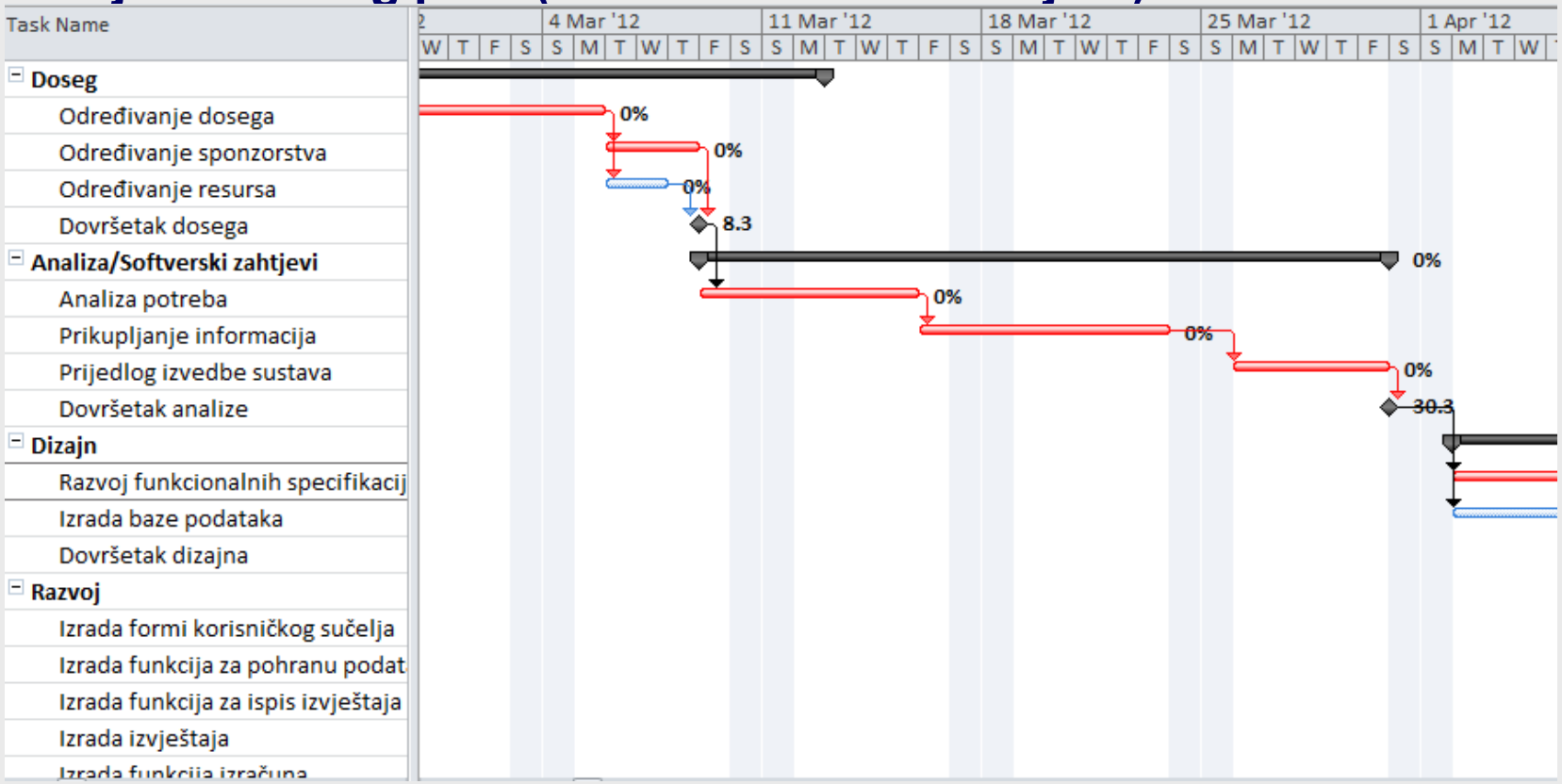


- Odnosi između zadataka:
  - Finish-to-start (FS) – završni datum prethodnika jest početni sljedbenika
  - Start-to-start (SS) - početni datum prethodnika utvrđuje početni sljedbenika
  - Finish-to-finish (FF) - završni datum prethodnika utvrđuje završni sljedbenika
  - Start-to-finish (SF) - početni datum prethodnika utvrđuje završni sljedbenika

CC BY NC SA

Kritični put

- **Kritični put**
  - niz zadataka koji moraju završiti na vrijeme da bi projekt završio na vrijeme
  - svaki zadatak na kritičnom putu je kritični zadatak
  - kašnjenje kritičnih zadataka uzrokuje kašnjenje projekta
- **Primjer kritičnog puta (koristeći alat MS Project)**



CC BY NC SA

Resursi

- **Resursi - sredstva**
  - ljudi, oprema i materijal potrebni za obavljanje zadataka
- **Vrste resursa:**
  - Resursi rada (work resources)
    - ljudi (ograničeno vrijeme rada)
    - oprema (neograničeno vrijeme rada)
  - Resursi materijala (material resources)
    - potrošni materijal koji predstavlja projektni utržak
    - daje informaciju o brzini konzumiranja resursa
- **Dva važna pogleda na resurse:**
  - Raspoloživost - u koje vrijeme određeni resurs može raditi na zadatku i koliko posla može obaviti
  - Trošak – koliko novca će biti potrošeno na resurse

BY NC SA

Raspodjela resursa

- **Unos resursa i pratećih podataka (dostupnost, trošak)**

- **Maksimalne jedinice (max. units*)***
  - prikazuju vrijednosti raspoloživosti resursa u postocima
  - 100% predstavlja jednog čovjeka punog radnog vremena
  - 300% predstavlja tri čovjeka punog radnog vremena

- **Osnova po kojoj je resursu dodijeljeno obavljanje posla**

- **Za pojedinačnu prilagodbu uvažavaju se radni i neradni dani resursa**
  - Primjer: ako kalendar evidentira radno vrijeme samo četvrtkom i petkom 13-17 sati, 100% raspoloživosti nekog resursa ne znači 40 satno tjedno radno vrijeme, nego 8 sati rada tjedno

Dodjela resursa

- **Dodjelom resursa zadatku, dodjeljuje mu se određeni posao**

- **Razlikovati posao (work) od trajanja (duration)**
  - Posao je stvarni rad i odnosi se na
    - zadatak - stvarni rad potreban za završetak zadatka
    - dodijeljeni zadatak - stvarni rad nekog resursa na nekom zadatku
    - resurs - ukupni rad neke osobe na svim zadacima
  - Trajanje = Posao / Jedinice (Duration = Work / Units)
    - Ovisi o kalendaru rada

- **Raspoređivanje temeljem napora (effort-driven scheduling)**
  - metoda planiranja koja se koristi kod ažuriranja resursa zadatka
  - trajanje obrnuto količini resursa

BY NC SA

Prekapacitirani resursi

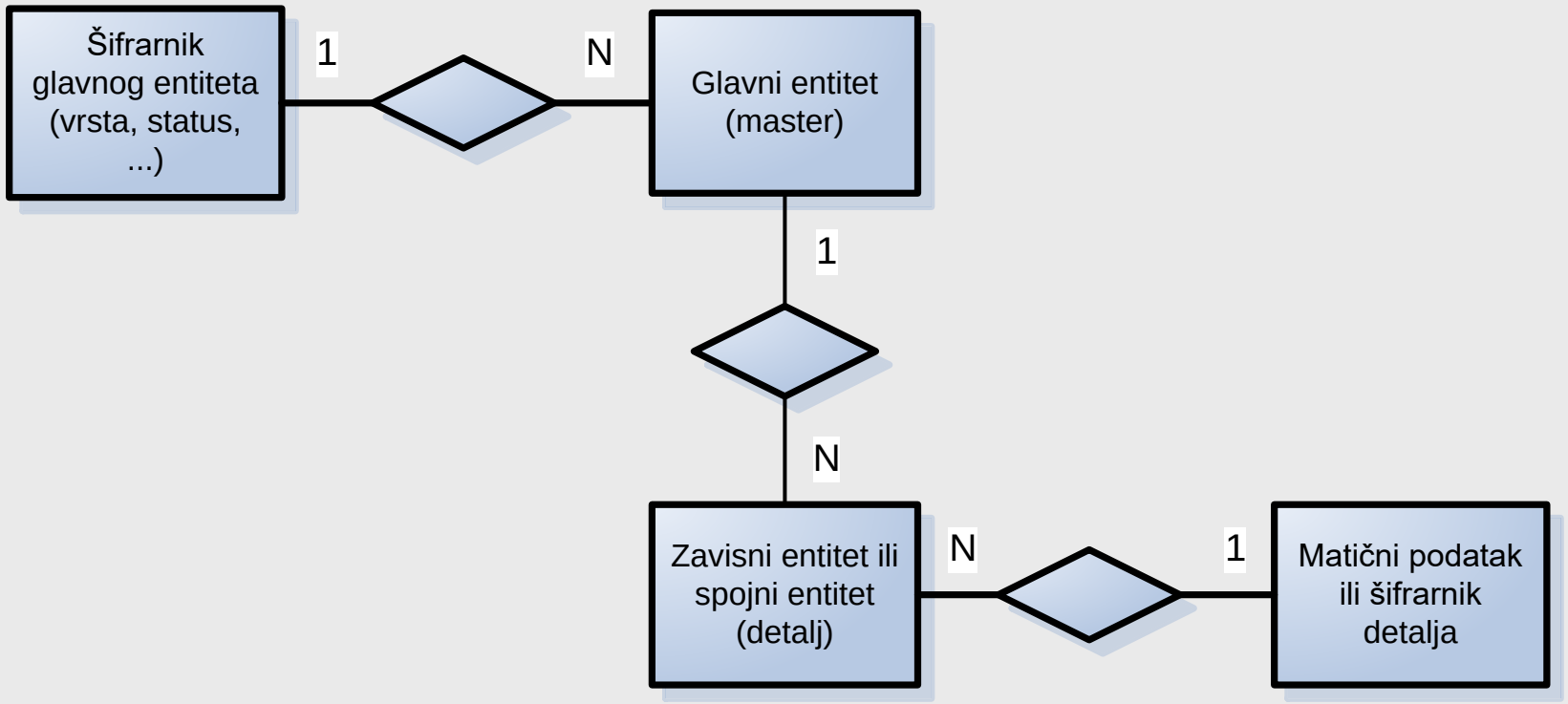
- **Rješavanje problema:**
  - Pomaknuti rokove (trajanje)
  - Dodati dodatne resurse
  - Produžiti radno vrijeme
  - Povećati jedinice posla
  - Smanjiti količinu posla
  - …

Zadaci

- **Projektni zadatak (za međuispit i završni ispit)**
  - Projektirati i izraditi sustav za podršku [bit će poznato na 2. predavanjima]
  - Svaki član mora napraviti barem po jednu reprezentativnu zaslonsku masku za web aplikaciju te web servis
  - Integraciju specifikacije posla, zajedničkih programskih knjižnica te ostale projektne dokumentacije radi ekipa zajednički.
- **1.DZ: Početni plan i raspored projekta – vidi Vrednovanje.pdf**
  - Kakav sustav ? Kako dobar sustav ? Kojim redom ? Kojim resursima ?
- **Realizacija**
  - Prikupiti i proučiti te provjeriti zahtjeve korisnika – intervjuiranjem, surfanjem, ...
  - Evidentirati problemska područja i zadatke
  - Uspostaviti životni ciklus razvoja na razini ekipe – plan faza razvoja
  - Popisati i pridijeliti nositelje zadataka (stvarne osobe, ne uloge)
  - Odrediti prekretnice s obzirom na realno stanje (cjeline posla, ciklusi, MI, ZI)
  - PlanProjekta - zajednički za ekipu
  - Nacrtati konceptualni model podataka

BY NC SA

Problemska područja



- **Matični podatak jednog problemskog područja smije biti glavni entitet u drugom problemskom području i obrnuto**

CC BY NC SA

Specifikacija zahtjeva

Zaštićeno Creative Commons licencom, http://creativecommons.org

Zahtjevi

- **ISO/IEC/IEEE 24765:2010 Systems and software engineering—Vocabulary:**
  - uvjet ili sposobnost koje korisnik treba da bi riješio problem ili ostvario cilj.
  - uvjet ili sposobnost koji mora posjedovati ili zadovoljiti sustav, komponenta sustava, proizvod ili usluga da bi zadovoljila ugovor, standard, specifikacije ili neki drugi ugovoreni dokument.
    - Dodatno, prema PMBOK zahtjevi uključuju nabrojane i dokumentirane potrebe, želje i očekivanja sponzora, korisnika i ostalih dionika u projektu
- **ISO/IEC/IEEE 29148:2011 Systems and software engineering--Life cycle processes--Requirements engineering**
  - izjava kojom se prevodi ili izražava potreba i potrebi pridružena ograničenja i uvjeti
- **Vrste zahtjeva**
  - poslovni
  - korisnički
  - funkcionalni
  - nefunkcionalni

CC BY NC SA

Poslovni zahtjevi

- **Odgovaraju na pitanje *zašto* (se radi neki sustav)**
  - predstavljaju ciljeve organizacije ili korisničke zahtjeve na višoj razini i ukratko opisuju problem koji treba riješiti
  - sadržani u dokumentima u kojima se opisuje vizija i opseg projekta

- **Primjeri:**
  - poboljšanje usluge postojećim klijentima tvrtke i pridobivanje novih
  - evidencija članstva i automatizacija postupka primanja novih članova neke udruge
  - praćenje financijskih podataka udruge i njenih članova
  - poboljšanje procesa prodaje
  - omogućavanje internetske prodaje
  - podrška organiziranju natjecanja i okupljanja

BY NC SA

Primjeri zahtjeva vlasnika sustava

- **Očekivana novčana ušteda**
  - Sustav mora biti tako koncipiran da prava na subvencioniranu prehranu može koristiti samo student koji ih je stekao i da ih može koristiti samo u svrhu prehrane.

- **Primjer: Sustav mora onemogućiti:**
  - korištenje subvencije od strane osoba koje nemaju na to pravo
  - zaradu ilegalnih posrednika
  - korištenje subvencije za druge svrhe osim prehrane
  - naplatu usluga koje nisu pružene

- **U idealnom slučaju zahtjevi vlasnika podudaraju se s poslovnim ciljevima!**

Korisnički zahtjevi

- **Zahtjevi krajnjih korisnika**
  - opisuju zadatke koje korisnik mora moći obaviti služeći se aplikacijama
  - sadržani u opisima slučajeva korištenja, tj. opisima scenarija rada
  - obično se izražavaju u obliku „Korisnik želi/treba/mora moći obaviti…“.

- **Primjeri:**
  - *Korisnik mora moći ostvariti pravo na prehranu kod bilo kojeg pružatelja usluge* - Novi sustav mora omogućiti da student ostvaruje svoje pravo kod bilo kojeg pružatelja usluge subvencionirane prehrane. Dosadašnja praksa je bila da svaki pružatelj usluga izdaje svoje bonove koji se mogu koristiti samo u određenim restoranima
  - *Korisnik treba plaćati obroke nakon korištenja pojedinog obroka.* - Treba izbjeći bilo kakvo plaćanje od strane studenata za potrebe ostvarivanja prava, a posebice unaprijed.
  - *Korisnik mora moći prijaviti gubitak kartice* – Potrebno je smanjiti rizik gubitka ostvarenih prava te sustav mora onemogućiti zloporabu stečenih prava.

cc BY NC SA

Funkcionalni zahtjevi

- **Odgovaraju na pitanje što (se može/mora napraviti koristeći sustav)**
  - definiraju softversku funkcionalnost (očekivano ponašanje i operacije koje sustav može izvoditi) koju treba ugraditi u proizvod da bi omogućio korisnicima obavljanje njihovih zadataka
  - posebno zanimljiva mogućnost programa (feature) – skup logički povezanih funkcionalnih zahtjeva koje korisniku omogućuju ispunjavanje poslovnih zahtjeva

- **Primjeri:**
  - Nakon što se studentu jednom zavedu prava na matičnoj ustanovi, sustav mora proslijediti informaciju svim pružateljima usluga, odnosno omogućiti distribuirane upite
  - Sustav na dnevnoj bazi mora kreirati izvještaje sa statistikom prehrane po pružateljima usluge i vrsti obroka.

Nefunkcionalni zahtjevi

- **Odgovaraju na pitanje kako (ili kako dobro sustav mora raditi)**
  - posljedica standarda, pravila i ugovora kojih se proizvod mora pridržavati
  - opisi vanjskih sučelja
  - zahtjevi na performanse
  - ograničenja na dizajn i implementaciju te svojstva kvalitete (preciziraju opis proizvoda navodeći karakteristike proizvoda u različitim dimenzija koja su važne ili korisniku, ili graditelju).

- **Primjer**
  - U sustavu prehrane nefunkcionalni zahtjevi mogu biti vezani za oblik korisničke kartice, protokol povezivanja,obvezu fiskalizacije itd…

Primjer nepotpunog zahtjeva

- **"Proizvod će dostaviti statusnu poruku u redovitim intervalima ne manjim od 60 sekundi."**
  - Što je statusna poruka i pod kojim uvjetima će biti dostavljena?
  - Koliko dugo ostaje vidljiva?
  - Koji dio proizvoda će dostaviti poruku?
  - Koliko dosljedni intervali moraju biti?
- **Zahtjev, preciznije i detaljnije**
  - Modul za nadzor će ispisivati statusnu poruku u za to određeni dio sučelja.
  - Poruka će se ažurirati svakih 60 sekundi (plus minus 10 sekundi) nakon što započne izvođenje pozadinskog zadatka i bit će vidljiva cijelo vrijeme.
  - Ukoliko se pozadinski zadatak izvodi normalno, modul za nadzor će ispisivati postotak obavljenog posla.
  - Modul za nadzor će ispisati "Zadatak obavljen." nakon što se zadatak obavi.
  - Modul će ispisati poruku o pogrešci ukoliko dođe do zastoja u izvođenju.
  - Problem je rastavljen u više zahtjeva jer će svaki zahtijevati posebno testiranje.
  - Ukoliko je više zahtjeva grupirano u jedan lakše je previdjeti neki od njih tijekom izrade ili testiranja.
- **Primijetiti da u zahtjevu nije detaljno opisano kako će se poruka i gdje ispisivati. To će biti odlučeno tijekom dizajna !**

Primjer neostvarivog zahtjeva

- **"Proizvod će se trenutno prebaciti između ispisivanja i skrivanja znakova koji se ne mogu tiskati."**
  - Računala ništa ne mogu napraviti trenutno te je ovaj zahtjev neostvariv.
  - Da li programska podrška sama odlučuje kad će se prebaciti iz jednog stanja u drugo ili je to inicirano akcijom korisnika?
  - Na koji dio teksta će se primijeniti promjena prikaza: da li samo označeni tekst, cijeli dokument ili nešto treće?
  - Nejednoznačnost: da li su "znakovi koji se ne mogu tiskati" skriveni znakovi, posebne oznake ili kontrolni znakovi?

- **Bolji zahtjev:**
  - "Korisnik će posebno dogovorenom akcijom, odabrati da li će se HTML oznake u trenutno otvorenom dokumentu prikazivati ili ne."
  - Sad je jasno da je riječ o HTML oznakama te da korisnik mora obaviti nekakvu akciju, ali nije točno navedeno kakvu (npr. kombinacija tipki), što se prepušta dizajnerima.

Primjer neodređenog zahtjeva

- **"Parser će brzo generirati izvješće o pogreškama HTML oznaka, koje omogućava brzi ispravak pogrešaka kada program koriste početnici u HTML-u."**
  - Riječ "brzo" je neodređena.
  - Nije definirano što i kada se tvori izvješće i to čini zahtjev nekompletnim.
  - Kako se ovjerava zahtjev? Pronaći nekoga tko se smatra početnikom u HTML-u i zatim vidjeti kako brzo će, uz pomoć izvješća, ispraviti pogreške?
  - Kada se generira izvješće?
- **Bolje:**
  - Nakon što je HTML analizator obradio datoteku generirat će izvješće koje sadrži broj linije i tekst pronađenih HTML pogrešaka, te opis svake pogreške.
  - Ukoliko nema pogrešaka prilikom analize, neće se generirati izvješće.

CC BY NC SA

Postavljanje prioriteta

- **Nužno svojstvo - Da li korisnik nešto stvarno mora imati?**
  - Postoji tendencija da se previše zahtjeva proglasi nužnim!
  - Po definiciji, ako sustav ne uključuje nužne zahtjeve, taj sustav ne može ispuniti svoju svrhu.
  - Treba testirati svaki zahtjev koji se smatra nužnim i probati ga rangirati.
    - Ako se zahtjev može rangirati onda nije obvezan!
    - Potpuno obvezni zahtjevi se ne mogu rangirati jer su nužni za prvu verziju sustava!
- **Poželjno svojstvo - Funkcije koje korisnik želi na kraju imati**
  - Ranije verzije sustava mogu pružiti (ne potpunu) funkcionalnost bez tih zahtjeva.
  - Poželjni zahtjevi mogu i trebaju biti rangirani.
- **Neobvezna svojstva - Proizvoljni zahtjevi**
  - Svojstva i mogućnosti bez kojih se može (npr. ostvarivanje ostalih prava iz studentskog standarda iz primjera zahtjeva krajnjih korisnika)
  - Iako bi ih lijepo bilo imati, to nisu pravi zahtjevi.
  - Ovi zahtjevi također mogu biti rangirani.

BY NC SA

Dokumentiranje analize (zahtjeva)

- **Definicija zahtjeva (Requirements Definition)**
  - izjava o stanju i ograničenjima sustava te potrebama
  - narativni dokument namijenjen korisniku ili ga piše korisnik
    - poslovni i korisnički zahtjevi te njihovi prioriteti
    - uočeni problemi, ključne pretpostavke i preporuke rješenja
- **Specifikacija zahtjeva (Requirements Specification)**
  - često se naziva i funkcionalnom specifikacijom
  - strukturirani dokument s detaljnim opisom očekivanog ponašanja sustava
  - namijenjen ugovarateljima i izvoditeljima razvoja
  - ugradbeno nezavisan pogled na sustav
    - funkcionalni i nefunkcionalni zahtjevi te njihovi prioriteti
    - model organizacijske strukture (strukturni dijagrami)
    - opis protoka dokumenata (dijagrami toka)
    - model procesa (dijagram toka podataka)
    - konceptualni model podataka (dijagram entiteti-veze)
- **Primjeri:**
  - 🗁Firma-Zahtjevi.doc (SpecifikacijaZahtjeva.dot)

BY NC SA

Predložak dokumenta specifikacije zahtjeva

1. Uvod
    - 1.1 Namjena
    - 1.2 Konvencije dokumenta
    - 1.3 Tko treba čitati dokument i savjeti za čitanje dokumenta
    - 1.4 Opseg proizvoda
    - 1.5 Reference
2. Sveobuhvatni pregled
    - 2.1 Kontekst proizvoda
    - 2.2 Funkcije proizvoda
    - 2.3 Kategorije korisnika i svojstva
    - 2.4 Okružje u kojem se izvodi proizvod
    - 2.5 Ograničenja dizajna i ugradnje
    - 2.6 Pretpostavke i ovisnosti
3. Zahtjevi za sučeljem
    - 3.1 Korisničko sučelje
    - 3.2 Hardversko sučelje
    - 3.3 Softversko sučelje
    - 3.4 Komunikacijsko sučelje
4. Svojstva sustava
    - 4.x Svojstvo X
    - 4.x.1 Opis i prioriteti
    - 4.x.2 Nizovi pobuda/odziv
    - 4.x.3 Funkcijski zahtjevi
5. Ostali nefunkcionalni zahtjevi
    - 5.1 Zahtjevi za performansama sustava
    - 5.2 Zahtjevi za sigurnošću korisnika
    - 5.3 Zahtjevi za sigurnošću podataka
    - 5.4 Kvaliteta programske podrške
    - 5.5 Poslovna pravila
    - 5.6 Korisnička dokumentacija
6. Ostali zahtjevi

Dodatak A: Rječnik

Dodatak B: Modeli i dijagrami

Dodatak C: Lista nedovršenih/neodređenih zahtjeva

BY NC SA

Dodatni materijali, reference

- **Upravljanje projektima**
  - http://www.pmi.org
  - http://www.pmi-croatia.hr

Upravljanje verzijama
Osnove .NET-a i C#-a

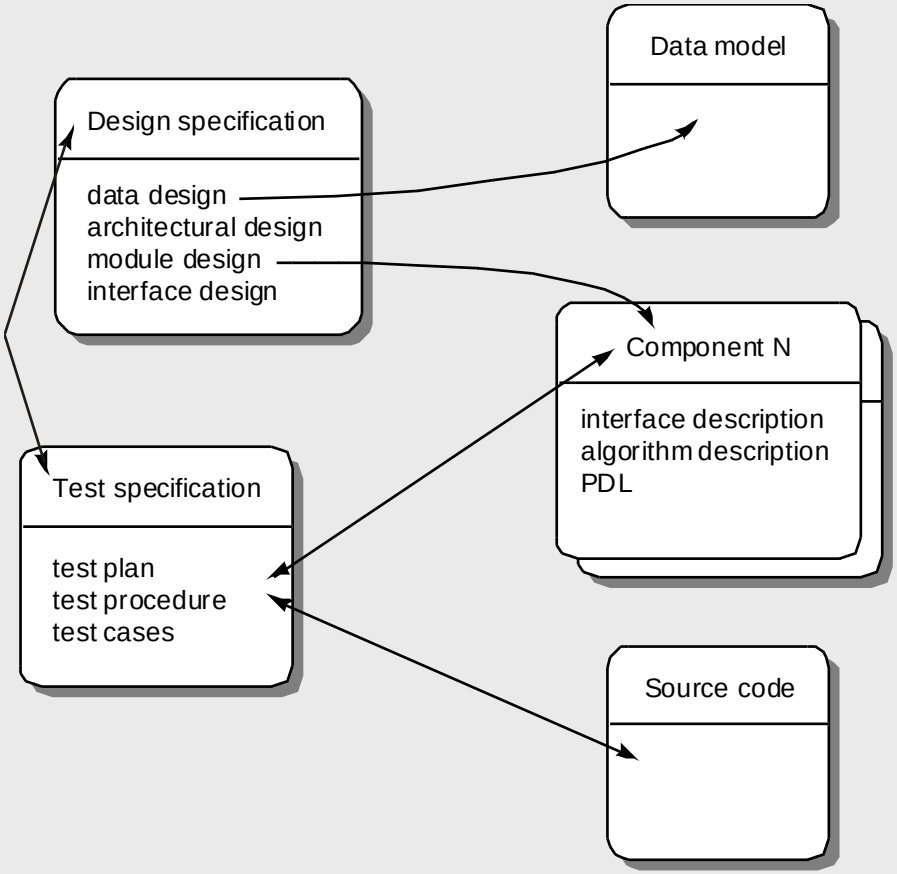
2019/20.02

Kontrola verzija

- **Kontrola verzija (Version control) = verzioniranje**
  - kombinira procedure i alate radi upravljanja različitim verzijama objekata konfiguracije, koji nastaju softverskim procesima
- **Mogućnosti sustava kontrole verzija**
  - baza projekata (project database) ili riznica (repository)
    - pohranjuje sve relevantne objekte konfiguracije
  - verzioniranje
    - razlikovanje pohranjenih inačica objekata konfiguracije
  - pomagalo za izradu (make facility)
    - prikuplja relevantne objekte i proizvodi određenu verziju softvera
  - praćenje problema (issues tracking), praćenje pogreški (bug tracking)
    - bilježenje i praćenje statusa tema koje se odnose na pojedine objekte konfiguracije

Upravljanje konfiguracijom

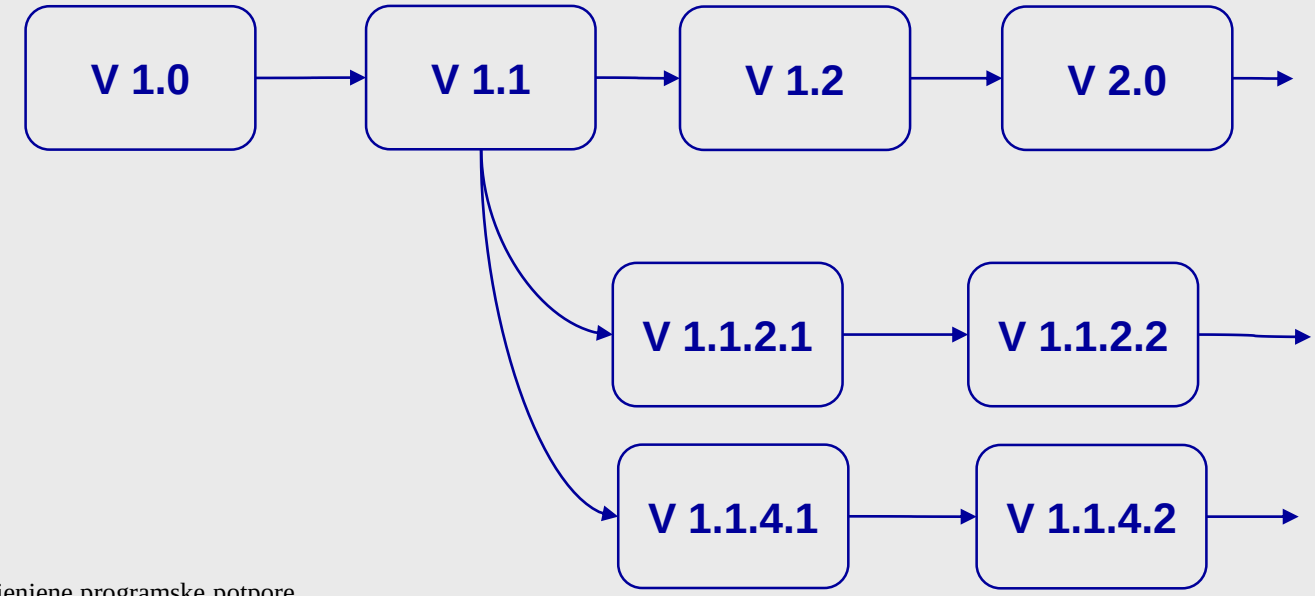
- **Konfiguracija**
  - imenovani skup konfiguracijskih elemenata u određenoj točki životnog ciklusa
- **Element konfiguracije (IEEE)**
  - agregacija hardvera i/ili softvera koja se tretira kao jedinka u procesu upravljanja konfiguracijom
- **Objekti konfiguracije**



BY NC SA

Verzije konfiguracije

- verzija, inačica (version) – određeno izdanje (issue, release) proizvoda
- objava, isporuka (release) – originalna verzija u primjeni, npr. zadnja v2.0
- revizija (revision) – ona koja se koristi umjesto originalne, podrazumijeva izmjene nastale kroz vrijeme (npr. zbog ispravljanja pogrešaka), npr. V1.2
- varijanta (variant) – alternativa originalu (hardverska platforma, različiti jezik), živi paralelno s njim, npr. v1.1.2.1
- osnovica (Baseline) – specifikacija proizvoda fomalno provjerena i usvojena, koja služi kao temelj razvoja i koja se mijenja samo kroz formalnu proceduru kontrole promjena, IEEE (IEEE Std. No. 610.12-1990)



BY NC SA

Označavanje konfiguracije

- **Verzija objektne datoteke u .NET Frameworku (assembly) određena je s četiri broja:**

  `<major version>.<minor version>.<build number>.<revision>`

  - major version - mijenja se prilikom znatne promjene u (npr. kod redizajna koji prekida vertikalnu kompatibilnost sa starijim verzijama)
  - minor version - mijenja se prilikom znatne promjene, ali uz zadržavanje kompatibilnosti s prethodnim verzijama
  - build number - predstavlja ponovno prevođenje istog koda (npr. prilikom promjene platforme, procesora i slično)
  - revision - primjenjuje se npr. prilikom izdavanja sigurnosnih zakrpa i sličnih manjih promjena
- **Primjer: Properties \ AssemblyInfo**
  - major.minor.* (ili major.minor.build.*) automatski određuje build number i revision
    - build number: broj dana od 1.1.2000.
    - revision: broj sekundi proteklih od ponoći aktualnog dana podijeljen s 2
- **.NET Core koristi oblik `major.minor.patch-sufix`**

BY NC SA

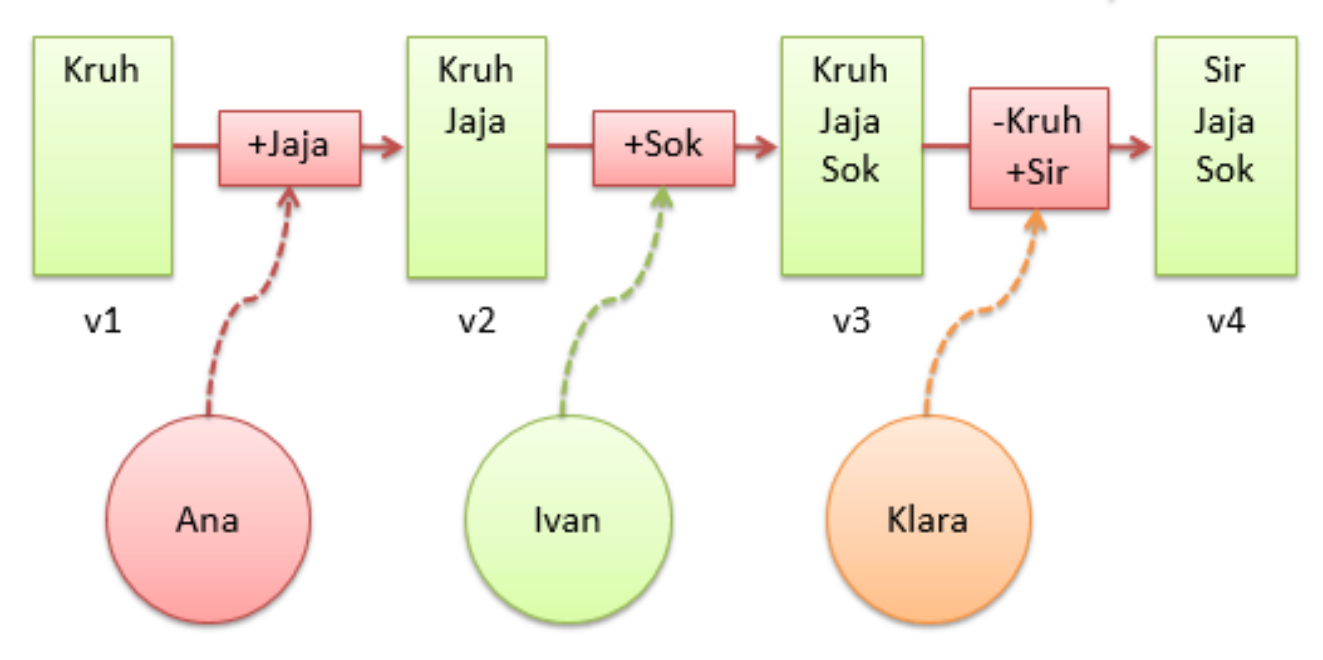
Automatsko i ručno verzioniranje

- **Automatsko označavanje**
  - prednosti:
    - eliminacija ručnog rada (npr. pisanja i izvedbe skripti)
    - ne postoje dvije inačice s istom oznakom
  - nedostaci:
    - oznaka elementa ne podudara se s oznakom cijelog sustava
    - novi brojevi ovise o danu i vremenu prevođenja
    - verzija se mijenja pri svakom prevođenju, neovisno o tome jesu li se dogodile promjene ili ne
- **Ručno verzioniranje**
  - prednosti:
    - potpuna kontrola nad brojevima verzije
    - moguća je sinkronizacija između verzije pojedinih komponenti i verzije cijelog sustava
  - nedostaci:
    - verzioniranje se mora raditi ručno
    - moguće je napraviti više različitih objektnih datoteka s istim oznakama

Kontrola programskog koda

Sustavi za upravljanje verzija datoteka

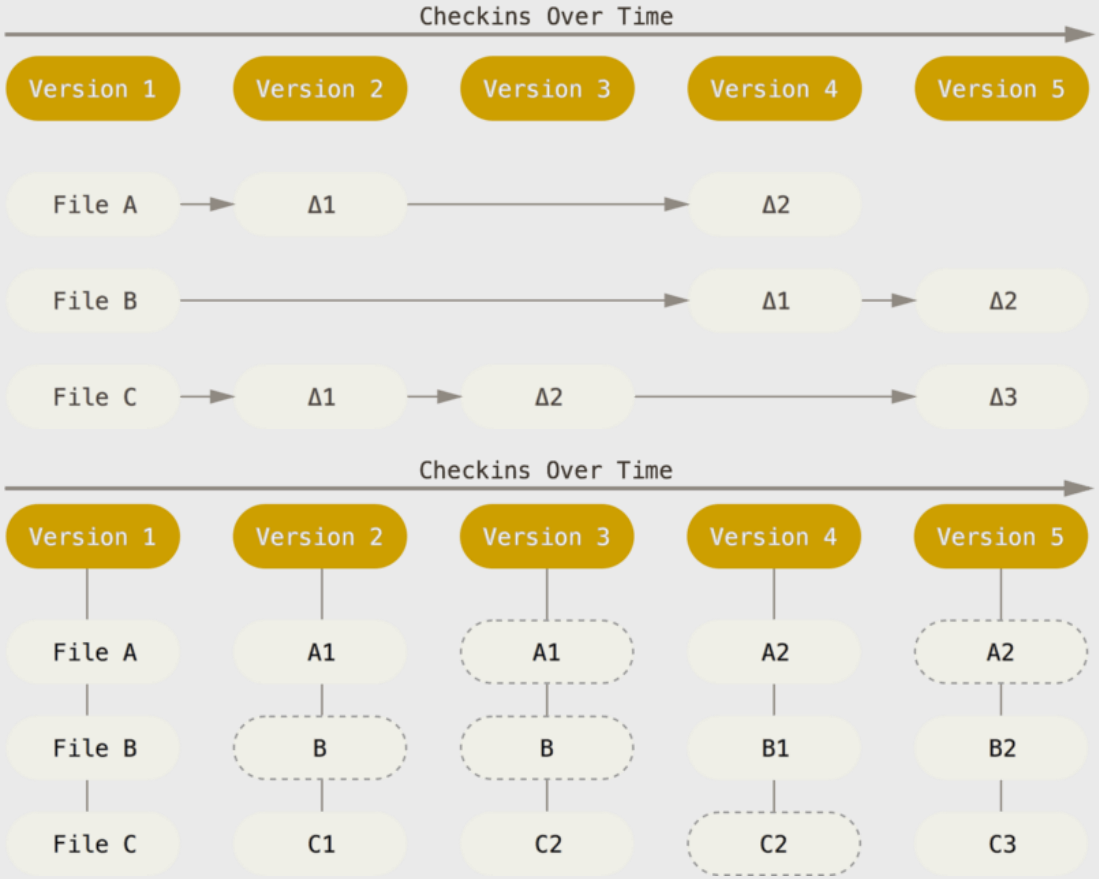
- **Najčešće kontrola promjena izvornog koda i konfiguracijskih datoteka**
  - ne evidentirati automatski generirane datoteke!
- **Verzionirane datoteke**
  - datoteke za koje se prati povijest promjena
- **Repozitorij**
  - sadrži zadnju verziju datoteka i povijest promjena (delte)
  - Moguće rekonstruirati bilo koju prethodnu verziju ili usporediti prethodne verzije



BY NC SA

Karakteristike sustava za upravljanje verzijama

- **Identifikacija verzija i izdanja**
  - Svaka verzija pohranjena u repozitoriju ima jedinstveni identifikator
- **Jedan skup promjena može sadržati promjene na više datoteka**
- **Kompaktna pohrana**
  - Umjesto kopije svake od verzija čuva se zadnja verzija te lista razlika između susjednih verzija



Chacon, Straub, ProGit 2nd Edition, Apress, 2014.

BY NC SA

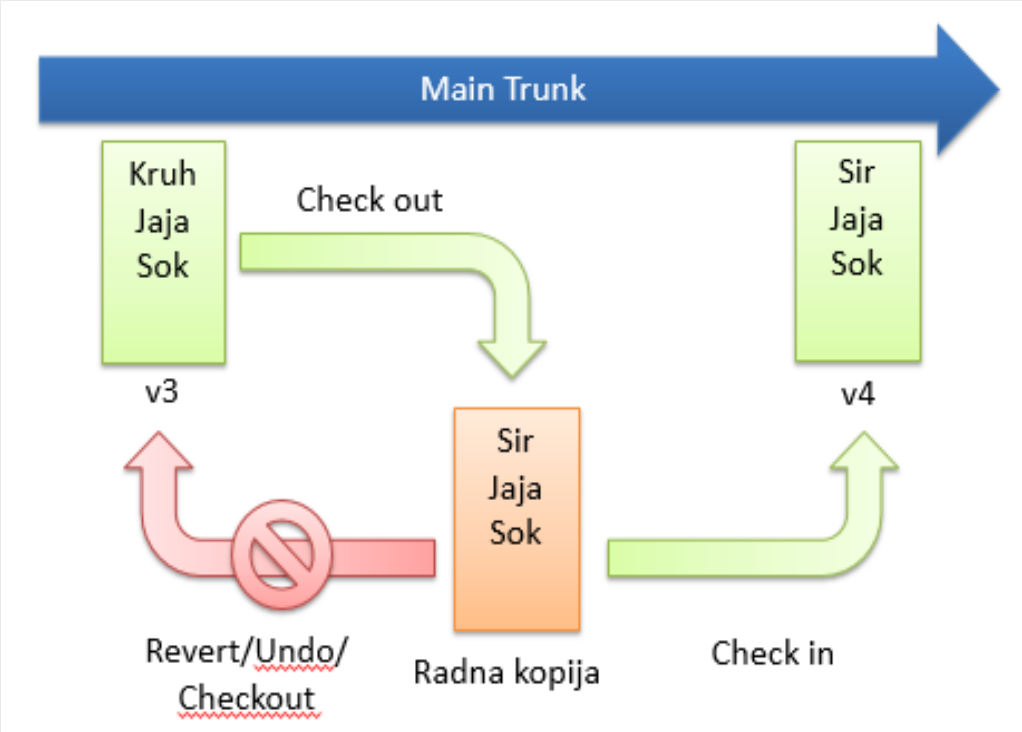
Centralizirani sustavi za upravljanje verzijama

- **Subversion, CVS, Team Foundation Server, …**
- **Samo jedan centralni repozitorij**
- **Korisnik ima radnu kopiju (working copy, workspace)**
  - Za svaku datoteku radna kopija sadrži određenu verziju iz centralnog repozitorija i informaciju o kojoj verziji se radi
  - Neki od alata (npr. TFS) dopuštaju da radna kopija sadrži samo dio centralnog repozitorija
- **Promjene na radnoj kopiji započinju checkout-om**
  - Najava izmjene datoteke
    - Može uključivati prethodni dohvat zadnje verzije datoteke i onemogućavanje ažuriranja te datoteke drugim korisnicima

BY NC SA

Promjene na radnoj kopiji

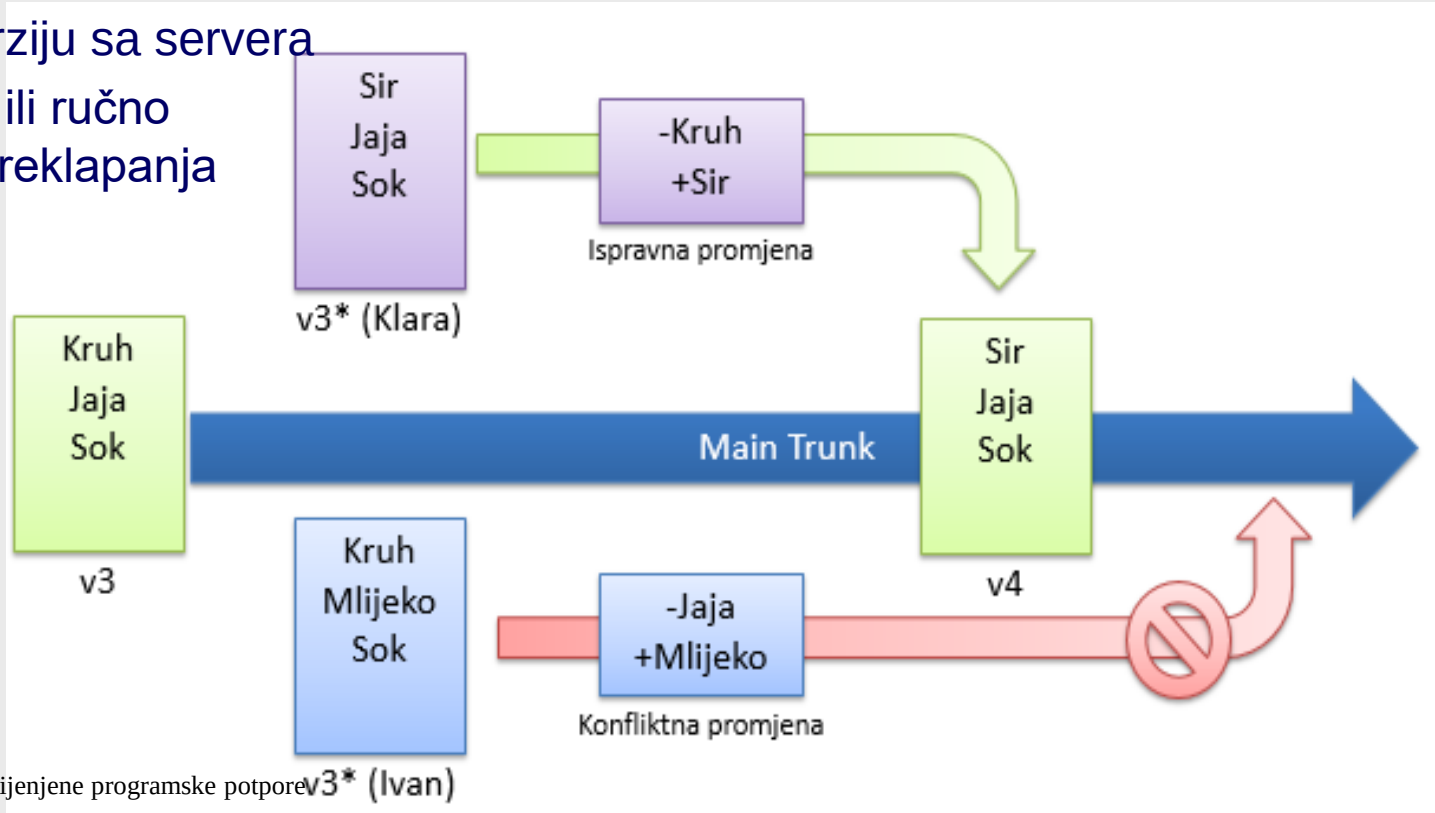
- **Korisnik promjene na radnoj kopiji može**
  - odbaciti (undo, revert, checkout)
  - potvrditi slanjem na centralni repozitorij (check-in, commit)
- **Skup poslanih promjena naziva se *changeset***
- **Svaki skup promjena dobiva jedinstveni identifikator**
  - Kod centraliziranih sustava identifikator je redni broj koji se povećava za 1



CC BY NC SA

Konfliktne promjene

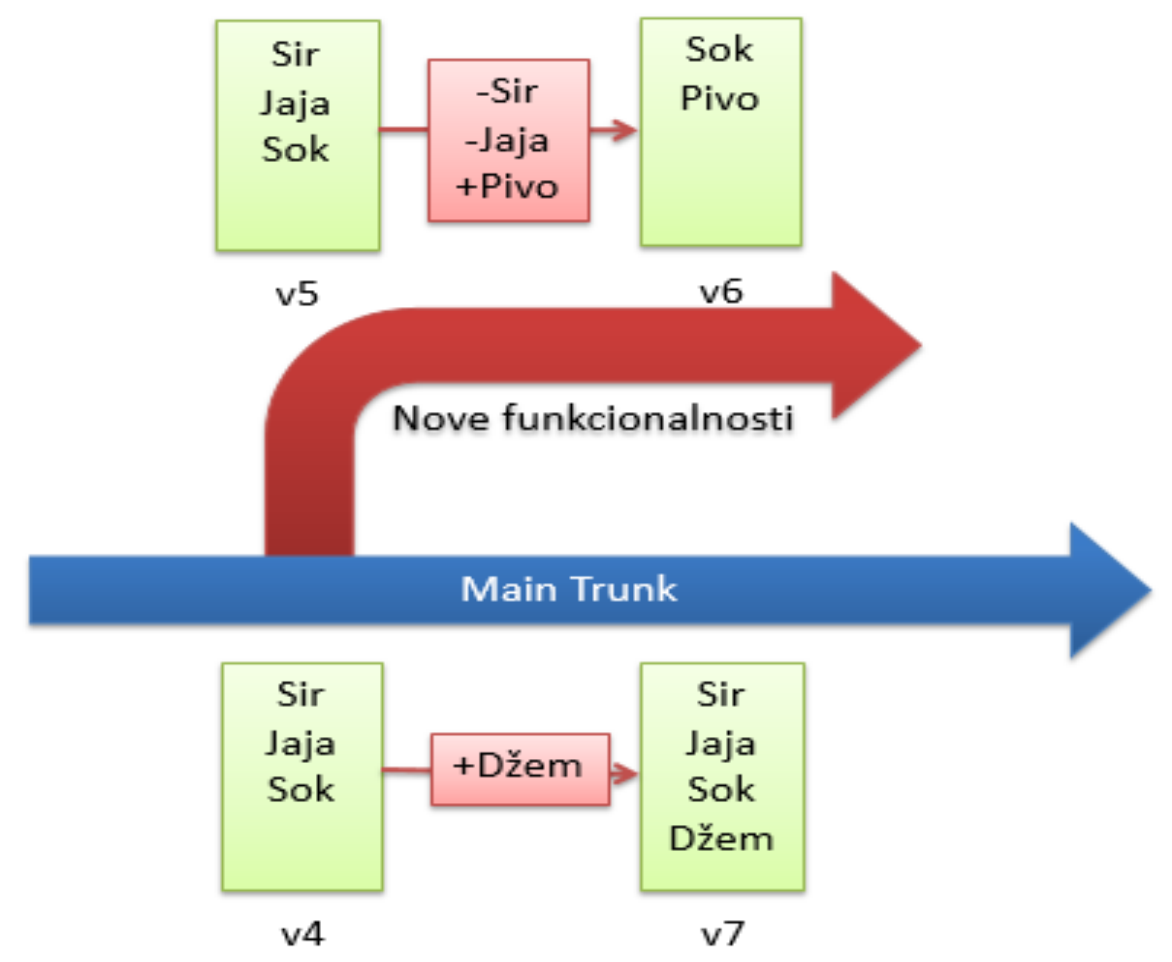
- **Istovremene promjene mogu voditi do konflikta**
  - Prvi korisnik će uspješno potvrdi promjene
  - Drugom korisniku će sustav dojaviti da su promjene napravljene u odnosu na verziju koja više nije najsvježija
- **Rješenja**
  - Odbaciti vlastitu verziju
  - Ignorirati verziju sa servera
  - Automatsko ili ručno rješavanje preklapanja



BY NC SA

Paralelne grane

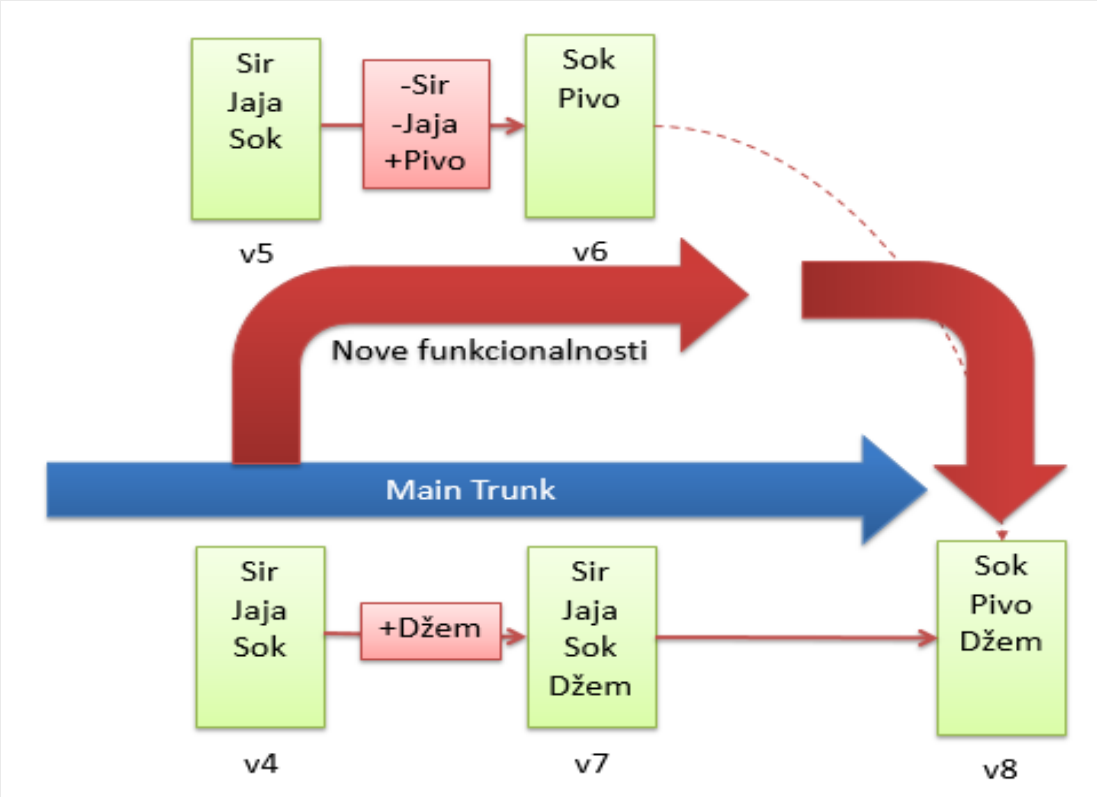
- **Omogućava istovremeni rad na novim funkcionalnostima uz očuvanje ispravnosti i održavanje glavne (isporučene verzije).**
  - Ovisno o sustavu može se raditi o fizički odvojenim kopijama datoteka



BY NC SA

Spajanje grana

- **Nije nužno – paralelna grana ne mora se nikad spojiti natrag**
- **Ugradnja promjena iz paralelne grane s promjenama u međuvremenu obavljenim na glavnoj grani**
  - paralelna grana se može, ali i ne mora obrisati
- **Git dodatno omogućava reproduciranje samo dijela promjena iz paralelne grane (cherry-pick)**



CC BY NC SA

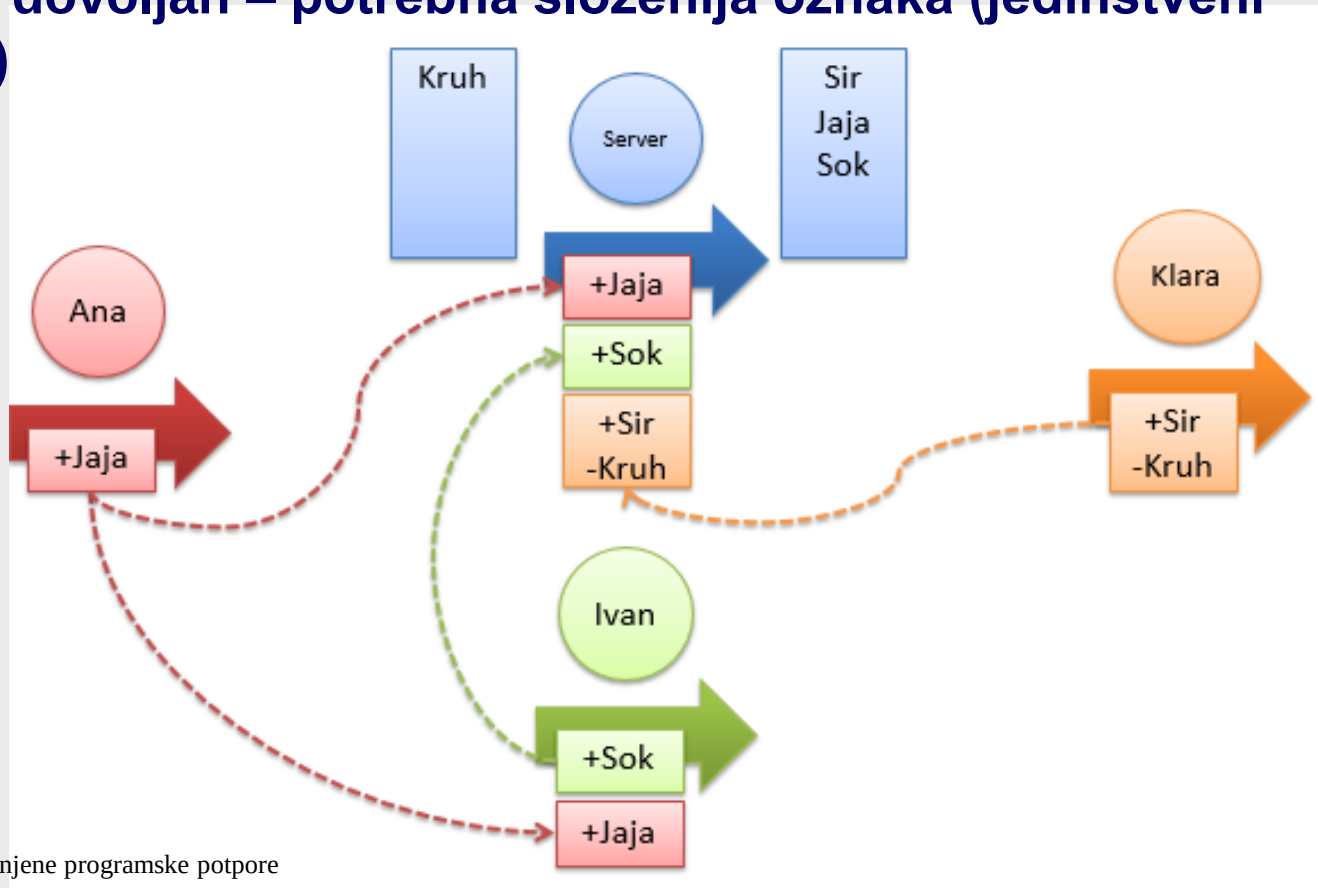
Distriburani sustavi

- **Git, Mercurial**
- **Svaki korisnik ima kompletnu kopiju repozitorija**
  - sadržaj verzioniranih datoteka, ali i cijela povijest promjena
  - lakše kopiranje repozitorija na neko drugo mjesto
- **Korisnik može evidentirati promjene u izvanmrežnom načinu rada**
  - naknadno prenosi svoju povijest na centralni server (push)
  - osvježava svoju verziju s verzijom na serveru (fetch i/ili pull)
    - pull = fetch + merge
- **Inicijalno stvaranje kopije centralnog repozitorija naziva se kloniranje (engl. clone)**
- **Napomena: centralni repozitorij ne mora postojati!**
- **Ostali koncepti akcija i problema (konflikata) su isti ili slični kao kod centraliziranih sustava.**

CC BY NC SA

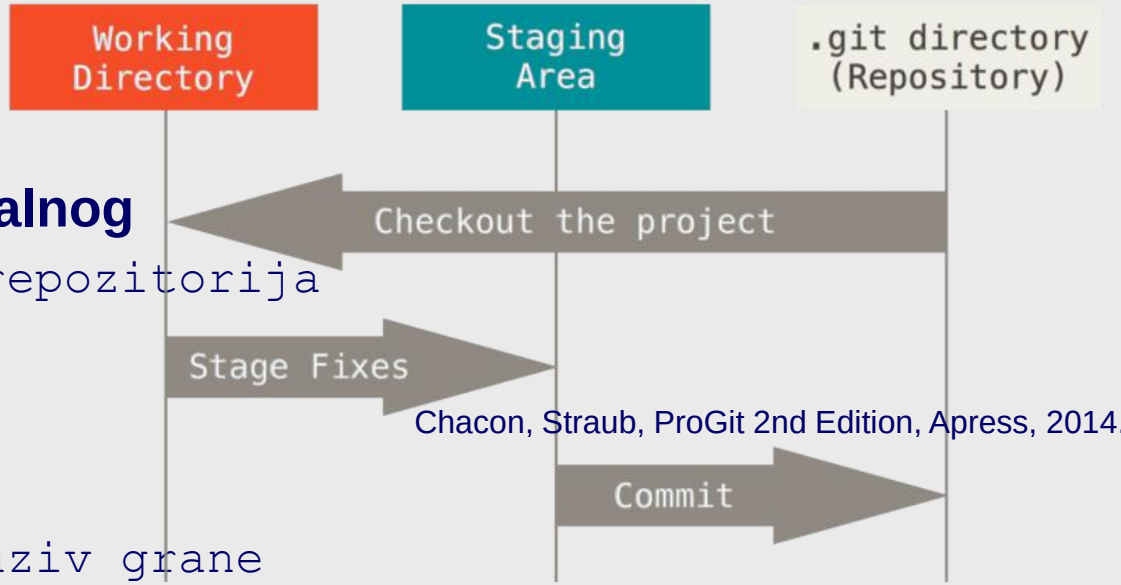
Pojava konflikta u distribuiranim sustavima

- **Kod distribuiranih sustava češće dolazi do konflikta**
  - Različiti sadržaji repozitorija kod pojedinih korisnika + neovisne promjene
  - Potrebno spajanje promjena
- **Svaka promjena mora imati jedinstveni identifikator**
- **Redni broj nije dovoljan – potrebna složenija oznaka (jedinstveni kod, npr. hash)**



BY NC SA

Tipični scenarij rada s Git-om



Chacon, Straub, ProGit 2nd Edition, Apress, 2014.

- **Kloniranje centralnog repozitorij ili inicijalizacija samostalnog**
  - `git clone adresa_repozitorija`
  - `git init`
- **Dohvat zadnje verzije centralnog repozitorija**
  - `git pull origin naziv grane`
  - origin označava udaljeni repozitorij, naziv glavne grane je master
- **Datoteke čije se promjene žele evidentirati dodaju se u međuspremnik (Staging Area)**
  - `git add naziv datoteke ili putanja do više datoteka`
- **Korisnik**
  - potvrđuje `git commit -m "opis promjena"`
  - Ili odbacuje `git checkout naziv datoteke ili putanja`
- **Korisnik šalje svoje promjene na server**
  - `git push origin naziv grane`

CC BY NC SA

Neki od ostalih pojmova

- **Stash i Unstash**
  - Privremeno spremanje izmjena, ali bez potvrđivanja promjena (tj. bez commita)
  - Slično kao Shelve/Unshelve kod TFS-a

Osnove platformi .NET Framework, .NET Core i jezika C#

.NET Framework

- **Microsoft .NET Framework**
  - nastao s idejom iste osnovice za izradu lokalnih i Internet aplikacija
    - začetak krajem 90-tih, prva verzija 2002. g
    - trenutna verzija 4.8.
  - neovisnost o jeziku
    - jezici Visual Basic .NET, Visual C++ .NET, C# i drugi
    - zajednička knjižnica osnovnih razreda Base Class Library (BCL) ili Framework Class Library (FCL)
    - zajednički, opći tipovi podataka Common Type System (CTS)
      - CLS (Common Language Specification) – podskup CTS-a zajednički za sve .NET jezike
    - zajednički pogon programa - Common Language Runtime (CLR)
  - neovisnost o platformi ... na kojoj postoji CLR
    - iako tako zamišljen, .NET je namijenjen uglavnom Windows platformi
    - Mono – (djelomična) implementacija .NET Frameworka za Linux

CC BY NC SA

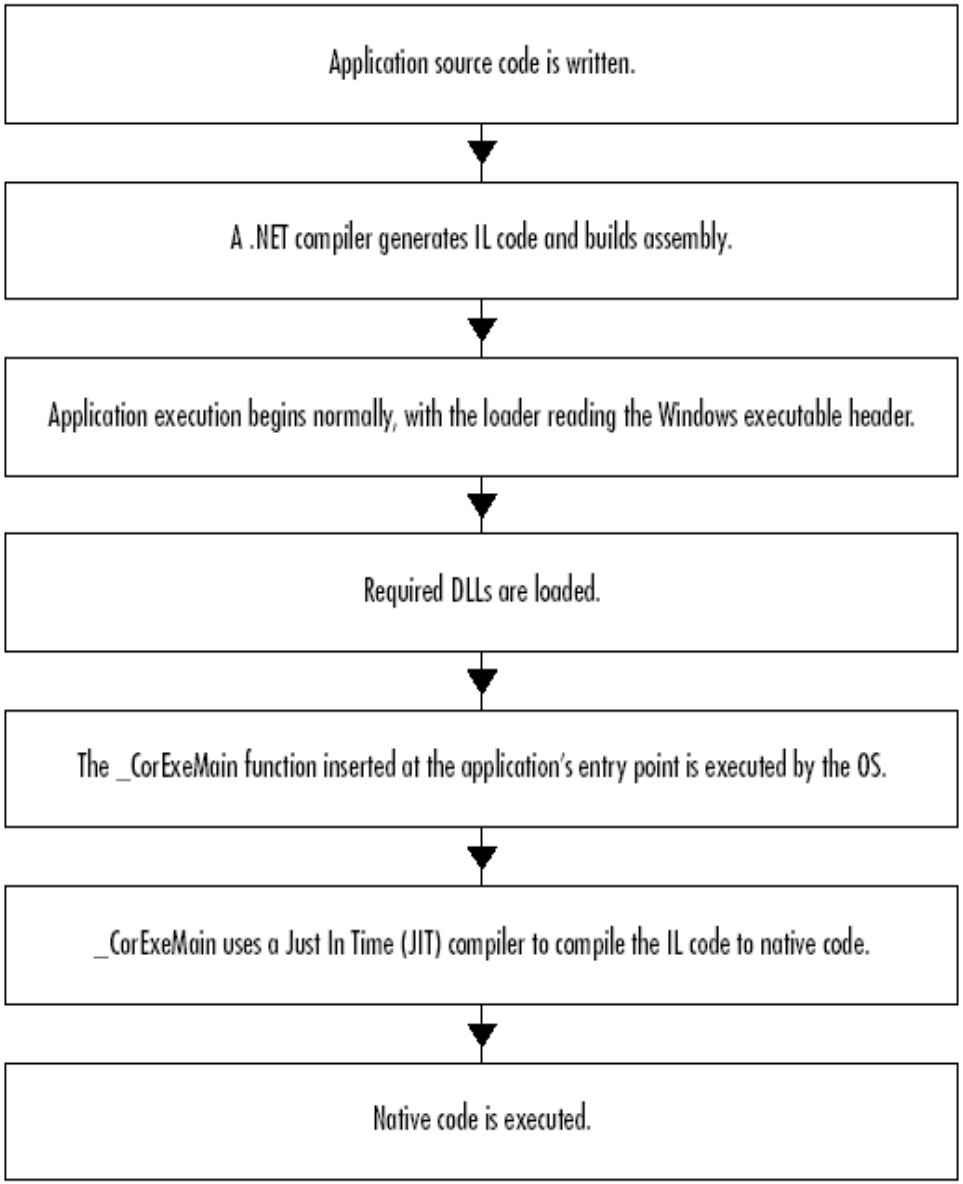
.NET Core

- **Nije podskup .NET Frameworka, ali dijeli dio funkcionalnosti**
  - Standard propisan kao .NET Standard (trenutno 2.1)
  - Podjela na modularne pakete dohvatljive korištenjem alata NuGet
- **Dostupan za Windowse, OSX i razne (ne nužno sve) distribucije Linuxa**
  - .NET Core 3.1 (preciznije 3.1.2, SDK 3.1.102), veljača 2020.
  - CoreCLR

- **Što trenutno može .NET Core?**
  - konzolne aplikacije
  - web aplikacije i web servisi pisani u ASP.NET Core-u
  - objektno-relacijsko preslikavanje prema nekoliko tipova sustava za upravljanjem bazama podataka korištenjem alata Entity Framework Core
- **Što ne može .NET Core?**
  - ne postoji multiplatformska podrška za samostalne (*desktop*) aplikacije
    - podrška za Windows desktop aplikacije (Desktop Runtime)

Microsoft .NET Core/Framework

- **C# prevoditelj**
  - prevodi izvornik kod u C#-u (ili nekom drugom jeziku) u poseban međujezik MSIL
- ***MS Intermediate Language* (MSIL)**
  - MSIL se izvodi u virtualnom stroju a ne izravno na procesoru računala
- ***Common Language Runtime* (CLR)**
  - zajednički pogon programa
  - interpretira MSIL naredbe
  - koristi JIT prevodilac
- **Just-In-Time compiler**
  - pri prvom pokretanju programa prevodi MSIL u strojni kod
- **Asemblij (Assembly)**
  - skup prevedenih razreda



BY NC SA

Jezik C# i *.NET Framework*

- **Jezik C# - Anders Hejlsberg et al.**
  - vođen događajima (event-driven),
  - objektno usmjeren (object-oriented),
  - otvoren mreži (network-aware),
  - vizualan (visual) – interaktivno sučelje razvojne okoline i aplikacija

- **Trenutna verzija C# 8.0 (rujan 2019.)**

CC BY NC SA

Stvaranje prvog programa (.NET Core)

- **Komanda linija**
  - u nekoj mapi pokrenuti dotnet new console –n Naziv
  - programski kod urediti u proizvoljnom uređivaču teksta
  - dotnet restore (dovlači pakete uključene unutar projekta)
  - dotnet run

- **Visual Studio Code**
  - u nekoj mapi pokrenuti dotnet new console –n Naziv
  - Visual Studio Code → Open Folder
  - Potvrdi dohvat paketa, a zatim F5

- **Visual Studio 2019**
  - File → New project → C# Console Application (NET Core)
  - Unijeti naziv i lokaciju projekta
  - F5 (Debug) ili CTRL+F5 (Execute)

CC BY NC SA

Elementi jezika

- **Identifikatori**
  - nazivi razreda, varijabli, ... koje određuje programer
  - sastoje se od slova, znamenki i podcrte, prvi znak mora biti slovo
- **Ključne riječi (keywords)**
  - rezervirane riječi jezika, posebni identifikatori koji u jeziku imaju predefinirano značenje
  - definicije objekata, naredbe i direktive za prevođenje
- **Operatori slični kao u C-u i Javi: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/**
- **Programske strukture**
  - odvajanje naredbi s ;
  - blokovi programa - { }
- **C# razlikuje velika i mala slova!**
- **Komentari**
  - Retkovni komentar – tekst od znakova "//" do kraja retka
  - Blok komentar – tekst oblika /* … */, može se rasprostirati u više redaka

Struktura C# programa

- **Program**
  - sadrži jedan ili više razreda
- **Razred (klasa, *class*)**
  - sadrži članove – svojstva (varijable) i metode tj. postupke (potprograme)
- **Ne postoje slobodni potprogrami ni globalne varijable**
  - sve se zbiva unutar tijela razreda
- **Ulazna točka (entry point)**
  - postupak `static void Main`
- **Korištenje knjižnica (using)**
  - svaki prevedeni razred ima manifest - definiciju sučelja koju mogu koristiti drugi programi, uključujući one napisane u nekom drugom jeziku
- **Primjer 🗁Osnove\Pozdrav**

```
// korištenje knjižnice
using System;

class Pozdrav// zaglavlje razreda
// tijelo razreda
{
  // metoda Main
  static void Main(string[] args)

  // blok naredbi
  {
    // naredba WriteLine

Console.WriteLine("Pozdrav!");
  }
}
```

Prostor imena (namespace)

- **Izvorni .NET kod je organiziran u prostore imena (namespaces)**
  - C# program se može sastojati od više datoteka
  - Svaka datoteka sadrži jedan ili više prostora imena
  - Isti prostor imena može se protezati (deklarirati) u više datoteka.
- **Prostor imena**
  - Prostor imena sadrži definicije razreda, struktura, sučelja, pobrojanih tipova, delegata i deklaraciju drugih prostora imena
  - Prostor imena u kojem su deklarirani svi ostali prostori imena i tipovi podataka u .NET Frameworku je System
  - Ako se u programu eksplicitno ne definira prostor imena, C# kod je sadržan u globalnom imeniku (*global namespace*)
- **Koncept prostora imena omogućuje postojanje istih imena u različitim prostorima imena**
  - Jedinstvenost imena tipova u prostoru imena i samog prostora imena osigurana je preko tzv. potpunih imena (*fully qualified names*)

CC BY NC SA

Definiranje prostora imena

- **Prostor imena N1 je član globalnog prostora imena**
  - Njegovo puno ime je N1
- **Prostor imena N2 je član prostora N1**
  - Njegovo puno ime je N1.N2
- **Razred C1 je član od N1**
  - Njegovo puno ime je N1.C1
- **Ime razreda C2 se pojavljuje dva puta, ali je njegovo puno ime jedinstveno**
  - N1.C1.C2 ili
  - N1.N2.C2.

```
namespace N1{      // N1
  class C1{       // N1.C1
    class C2{    // N1.C1.C2
    }
  }
  namespace N2{  // N1.N2
      class C2{   // N1.N2.C2
      }
   }
}
```

BY NC SA

Korištenje prostora imena

- **Referenciranje koda u prostoru imena obavlja se**
  - kvalificiranim imenom, od naziva prostora imena i identifikatora razreda
    - npr. System.Console
  - direktivom `using` ili `using static`
    - eksplicitno se uvode sva imena sadržana u prostoru imena ili svi potupci nekog razred
      - using System; referencira imenik u kojem je definiran razred Console, pa se Console može direktno koristiti
  - zamjenskim imenom – koristi se za definiranje skraćenih naziva i uklanjanje neodređenosti kad u različitim imenicima postoje razredi istog imena.

```
using System; // navod imenika
using stdout = System.Console; // alias razreda
using static System.Console;
class Pozdrav {
  static void Main(string[] args) {
    System.Console.WriteLine("Kvalificirano!");
    Console.WriteLine("Izravno!"); // radi using System
    stdout.WriteLine("Zamjenski!"); // radi using stdout =
    WriteLine("Zbog static using System.Console");
```

CC BY NC SA

Ključne riječi

- **Sve ključne riječi definirane su malim slovima !!**

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	get
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	set	short	sizeof	stackalloc
static	string	struct	switch	this
throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
virtual	void	volatile	while	

- **https://msdn.microsoft.com/en-us/library/x53a06bb(v=vs.140).aspx**

Ključne riječi ovisno o kontekstu

add	alias	ascending	async	await
descending	dynamic	from	get	global
group	into	join	let	orderby
partial	remove	select	set	value
var	where	yield		

- **https://msdn.microsoft.com/en-us/library/x53a06bb(v=vs.140).aspx**

CC BY NC SA

Standardni ulaz/izlaz

- **Čitanje standardnog ulaza**
  - public static string ReadLine();
    - `Console.ReadLine()` – čita niz znakova
  - public static int Read();
    - `Console.Read()` – čita znak

- **Pisanje na standardni izlaz**
  - public static void Write(...);
    - piše argument(e) i ostaje u istom retku
    - `Console.Write("Pozdrav!");`
  - public static void WriteLine(...);
    - piše argument(e) i znak za skok u novi redak
    - `Console.WriteLine("Pozdrav!");`

- **Primjeri primjene slijede u narednim programima**
- **Za više informacija pogledati dokumentaciju**

CC BY NC SA

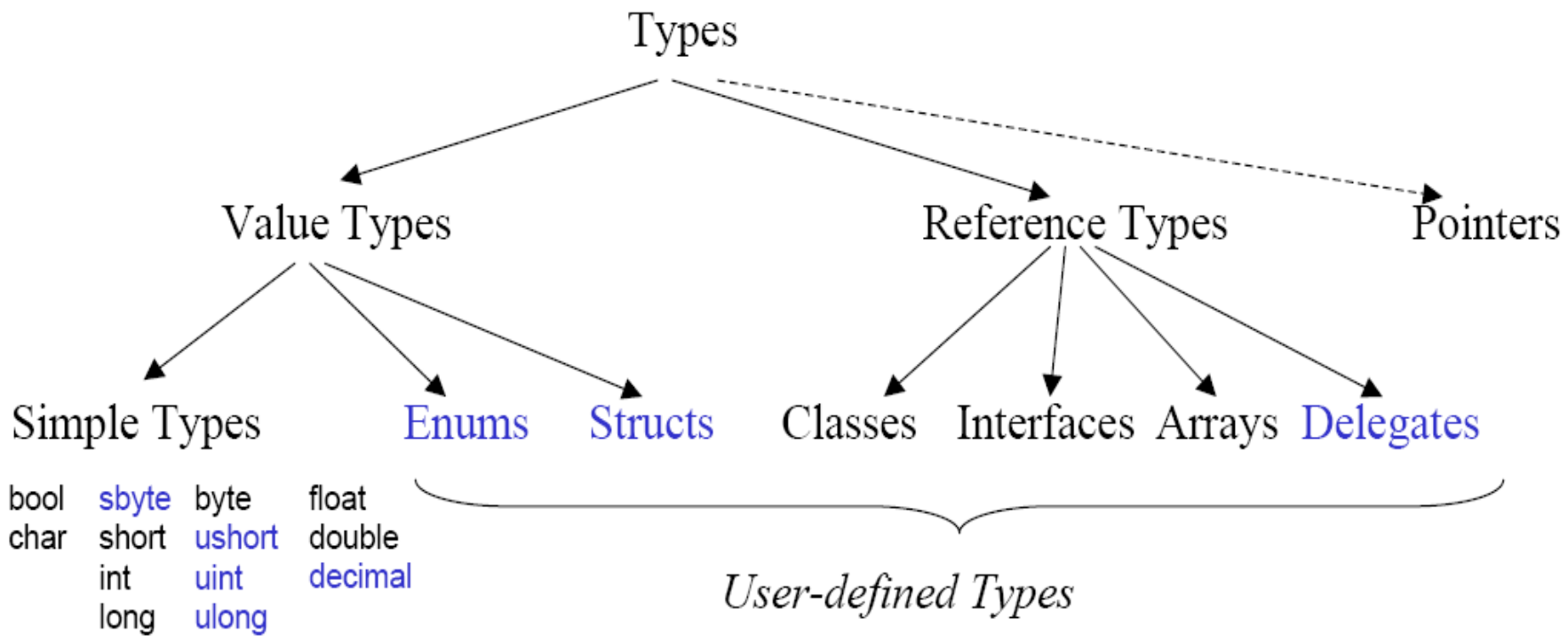
Formatirani ispis, numerički formati

Character	Description	Examples	Output
C or c	Currency	Console.Write("{0:C}", 2.5); Console.Write("{0:C}", -2.5);	$2.50 ($2.50)
D or d	Decimal	Console.Write("{0:D5}", 25);	00025
E or e	Scientific	Console.Write("{0:E}", 250000);	2.500000E+005
F or f	Fixed-point	Console.Write("{0:F2}", 25); Console.Write("{0:F0}", 25);	25.00 25
G or g	General	Console.Write("{0:G}", 2.5);	2.5
N or n	Number	Console.Write("{0:N}", 2500000);	2,500,000.00
X or x	Hexadecimal	Console.Write("{0:X}", 250); Console.Write("{0:X}", 0xffff);	FA FFFF

BY NC SA

Tipovi podataka

- **Svi tipovi izvode se iz osnovnog tipa System.Object**
  - "sve je objekt"
  - primjeri: `3.ToString(); "abc".ToUpper();`



Tipovi podataka

- **Osnovne kategorije tipova podataka:**
  - Vrijednosti (value types) – sadrže podatke
    - osnovni tipovi (int, float, char...), enum, struct
  - Reference (reference types) – pokazivači (pointers)
    - razredi (class types),
    - sučelja (interface types),
    - delegati (delegate types),
    - polja (array types) ,
    - znakovni nizovi (strings)

- **Varijable i pridruživanje vrijednosti**
  - varijabla - imenovana memorijska lokacija određenog tipa i poznate veličine te vrijednosti koja se u njoj nalazi a koja se može promijeniti
  - prije uporabe (zadavanja ili korištenja vrijednosti) varijablu treba deklarirati

```
int broj1;    //  <type> <variable-name>;
broj1 = 45;   //  <variable-name> = <exp>
```

BY NC SA

Ugrađeni tipovi podataka

C# Type	.Net Framework (System) type	Broj bajtova	Interval brojeva
sbyte	System.Sbyte	1	-128 do 127
short	System.Int16	2	-32768 do 32767
int	System.Int32	4	-2147483648 do 2147483647
long	System.Int64	8	-9223372036854775808 do 9223372036854775807
byte	System.Byte	1	0 do 255
ushort	System.Uint16	2	0 do 65535
uint	System.UInt32	4	0 do 4294967295
ulong	System.Uint64	8	0 do 18446744073709551615
float	System.Single	4	$\pm 1.5 \times 10^{-45}$ do $\pm 3.4 \times 10^{38}$ 7 točnih znamenki
double	System.Double	8	$\pm 5.0 \times 10^{-324}$ do $\pm 1.7 \times 10^{308}$ 15 točnih znamenki
decimal	System.Decimal	12	$\pm 1.0 \times 10^{-28}$ do $\pm 7.9 \times 10^{28}$ 28 točnih znamenki
char	System.Char	2	Unicode znak (16 bit)
bool	System.Boolean	1 | 2	true ili false

BY NC SA

Konverzija tipova

- **Implicitna konverzija – bez navođenja operatora konverzije**
  - Standardno int->double, int->long, ...

- **Eksplicitna konverzija – Primjer: 🗁Osnove\Konverzija**
  - operator konverzije tipa `(cast)`

```
int c = (int) 4.5;
```

  - ToString()

```
int a = 154;
string s = a.ToString();
```

  - Parse()

```
int c = Int32.Parse(s);
```

  - razred Convert

```
decimal d = Convert.ToDecimal(c);
```

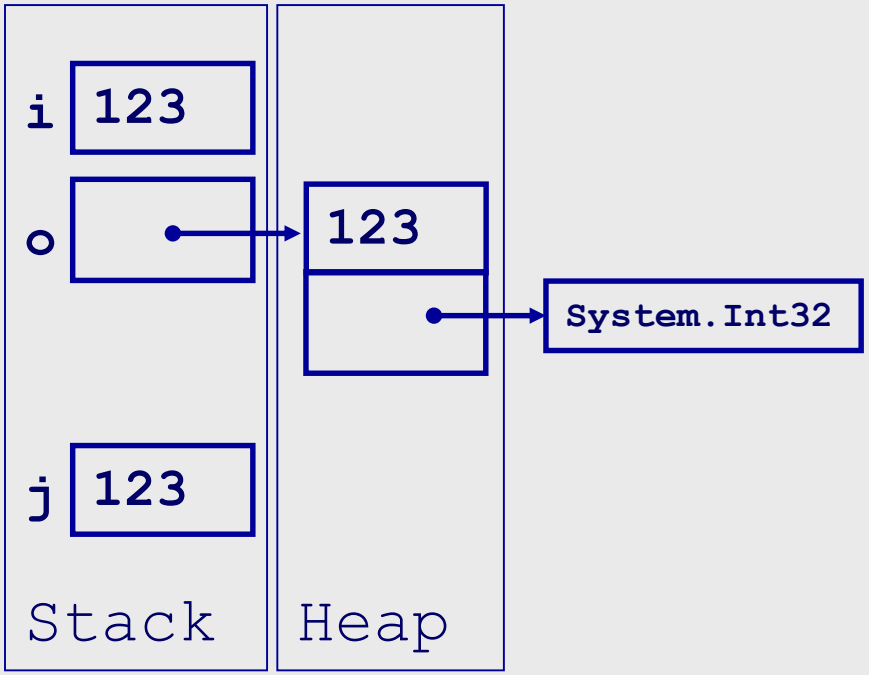
BY NC SA

Boxing/Unboxing

- ***Boxing***
  - pretvorba nekog vrijednosnog tipa podataka u referencijski tip `object`
  - alocira instancu i kopira vrijednost u novostvoreni objekt
- ***Unboxing***
  - izdvajanje vrijednosnog tipa podataka iz tipa object
  - kopiranje vrijednosti (objekta) s reference

- **Primjer 🗁Osnove\Boxing**

```
int i = 123; // A value type

object o = i; // Boxing


int j = (int) o; // Unboxing
```



BY NC SA

Aritmetički operatori

Operator	Značenje
+	zbrajanje
-	oduzimanje
*	množenje
/	dijeljenje
%	ostatak (modulo) pri dijeljenju (radi i s realnim operandima)
++	operator uvećanja za 1
--	operator smanjenja za 1

- **Znak plus (+) također predstavlja operator ulančavanja (konkatenacije) nizova znakova**

BY NC SA

Operatori usporedbe

- **Usporedbeni (relacijski) operatori**
  - Operatori == i != mogu se koristiti sa svim tipovima podataka
  - ostali usporedbeni operatori mogu koristiti samo s tipovima podataka koji podržavaju usporedbene operatore.
  - Npr. `bool` vrijednosti ne mogu se uspoređivati operatorima <, <=, >, >=
  - Pri usporedbi referenci uspoređuju se adrese (pokazivači) na objekte
    - Izuzetak su nizovi znakova.

Operator	Značenje
==	jednako
!=	različito
>	veće
>=	veće ili jednako
<	manje
<=	manje ili jednako

BY NC SA

Logički operatori

- **Logički operatori (bool i bitovni)**
  - Operandi moraju biti tipa bool i rezultat logičke operacije je tipa bool.
  - U ovu grupu mogu se pribrojiti i unarni *true* i *false*
- **Operatori `&` i `|` uvijek ispituju vrijednost svih operanada u izrazu.**
- **Reducirani operatori `&&` i `||` uvijek ispituju logičku vrijednost samo prvog operanda a ostalih po potrebi.**

Operator	Značenje
&	AND
|	OR
^	XOR (isključivi OR)
&&	reducirani AND
||	reducirani OR
!	NOT
~	komplement

cc BY NC SA

Ostali operatori

- **Operator pridruživanja: =**
- **Skraćeno pridruživanje:**
  - `+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
- **Posmak: `<<`, `>>`**
- **Uvjetni operator: (`?:`)** ***// cond-expr ? expr1 : expr2***
- **a ?? b**
  - a == null ? b : a
- **Veličina tipa: sizeof(*type*)**
- **Tip objekta:**
  - npr. `System.Type type = typeof(int);`

```
int n = 3;
if (n is int)
...
```

- **Operator is - expression is type**
  - provjerava da li je izraz određenog tipa
- **Pristup članu: . (točka)** ***// name1 . name2***
- **Indeksiranje: `[]`**
- **Indirekcija i adresa: `*` `->` `[]` `&`**

CC BY NC SA

Prioritet operatora

Category	Operators
Primary	(x), x.y, f(x), a[x], x++, x--, new, typeof, sizeof, checked, unchecked
Unary	+, -, !, ~, ++x, --x, (T)x
Multiplicative	*, /, %
Additive	+, -
Shift	<<, >>
Relational	<, >, <=, >=, is
Equality	==
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||
Conditional	?:
Assignment	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

BY NC SA

Naredbe za upravljanje programskim tokom

- **Selekcija**
  - if (expression) statement1 [else statement2]
  - switch (*expression*) { case *constant-expression*: *statement jump-statement* [default: *statement jump-statement*] }
- **Petlje**
  - while (expression) statement
  - for ([initializers]; [expression]; [iterators]) statement
  - foreach (*type identifier* in *expression*) statement
    - *expression* se ne smije mijenjati za vrijeme *foreach* petlje
    - Nije moguće: identifier = …
  - do statement while (expression);
- **Skokovi**
  - break , continue, goto
    - goto *identifier*;
    - goto case *constant-expression*;
    - goto default;
  - return [*expression*];

```
int[] numbers =
  {4, 5, 6, 1, 2, 3, -2, -1, 0};
for(int=0;i<numbers.length;i++){
  Console.WriteLine(numbers[i]);
}
foreach (int i in numbers){
  Console.WriteLine(i);
}
```

CC BY NC SA

Razredi

```
class Tocka
{
    public int x;
    public int y;
    public Tocka(int x, int y){
        this.x = x;
        this.y = y;
    }
    public int Kvadrant(){
        if (x > 0){
          if (y > 0) return 1;
          else return 4;
        } else {
          if (y > 0) return 2;
          else return 3;
        }
    }
}
```

- Primjer 🗁Osnove\RazredTocka

Javne varijable

Konstruktor

Javni postupak

Instanciranje objekta

```
Tocka t = new Tocka(1,-2);
Console.WriteLine("Točka ({0},{1})
        je u {2}.kvadrantu.", t.x,
        t.y, t.Kvadrant());
```

Poziv postupka

Pristup javnoj varijabli

CC BY NC SA

Razredi i članovi razreda

- **U jeziku C# razredi se sastoje od članova (members):**
  - Konstante (constants)
  - Atributi (fields)
  - Konstruktori (constructors)
  - Finalizatori (finalizers)
  - Postupci (methods)
  - Svojstva (properties)
  - Indekseri (indexers)
  - Operatori (operators)
  - Događaji (events)
  - Statički konstruktori (static constructors)
  - Ugniježđeni tipovi (nested types)
- **Razred je obrazac prema kojem se stvaraju instance razreda - objekti**
  - `new` – operator koji stvara novu instancu
  - `this` – referenca na trenutnu instancu razreda
  - razred s početnim postupkom (`Main`) ne mora se eksplicitno instancirati

CC BY NC SA

Modifikatori tipova i članova

- **Modifikatori pristupa razredima i članovima**
  - `public` – pristup nije ograničen
  - `private` - pristup ograničen na razred u kojem je član definiran
  - `protected` – pristup ograničen na razred i naslijeđene razrede
  - `private protected` – slično kao protected, ali samo za naslijeđene razrede u istom programu/projektu
  - `internal` – pristup ograničen na program u kojem je razred definiran
  - `protected internal` - pristup dozvoljen naslijeđenim razredima (bez obzira gdje su definirani) i svima iz programa u kojem je razred definiran
- **Neki od značajnijih modifikatora**
  - `abstract` – razred može biti samo osnovni razred koji će drugi nasljeđivati
  - `const` – atribut (polja) ili lokalna varijabla je konstanta
  - `new` – modifikator koji skriva naslijeđenog člana od člana osnovnog razreda
  - `readonly` – polje poprima vrijednost samo u deklaraciji ili pri instanciranju
  - `sealed` – razred ne može biti naslijeđen
  - `static` – član definiran na razini razreda, a ne instance
  - `virtual` – postupak ili dostupni član koji može biti nadjačan u naslijeđenom razredu – (prilikom nadjačavanja dodaje se modifikator `override`)

CC BY NC SA

Pretpostavljeni modifikatori

- **Ako modifikator nije naveden onda se smatra da je**
  - `internal` za razrede, sučelja, delegate i događaje
  - `private` za članske varijable, svojstva i postupke i ugniježđene razrede
  - `public` za postupke sučelja i članove enumeracija (nije ni dozvoljen drugačiji modifikator)
- **Izvedeni razred ne može imati veću dostupnost od baznog razreda**

BY NC SA

Statički članove, konstante i atributi (fields)

- **`Static` se može primijeniti na atribute, postupke, svojstva, operatore i konstruktore**
  - Statički član pripada tipu, a ne instanci.
- **Modifikator `const`**
  - Vrijednost konstante određena je pri prevođenju i ne može se mijenjati
  - Konstante su po prirodi statičke, pa ih se ne može eksplicitno proglasiti statičkim ni pristupati im preko reference na instancu objekta
- **Atributi**
  - Varijable kojima se može promijeniti vrijednost pristupom članu
  - Nepromjenjivi atributi – modifikator `readonly`
    - ne dopuštamo promjene, a vrijednosti nisu poznate u vrijeme prevođenja
- **Primjer 🗁Osnove\Razred**

```
public int var = 0; // atribut, varijabla objekta
public const int konst = 13; // konstanta razreda
public readonly int neDiraj; // postavlja se u konstruktoru
public static int trajna = 0; // dijeljena između instanci
```

BY NC SA

Konstruktori

- **Konstruktor je postupak koji se obavlja pri stvaranju instance**
  - Standardni (default) konstruktor i preopterećenje konstruktora s argumentima
  - Ne vraća vrijednost, služi za inicijalizaciju instance
  - Ime mu je jednako imenu razreda
- **Primjer 🗁Osnove\Razred**

```
class Razred
{
  //atributi
  public readonly int neDiraj;
  public int var = 0;
...
  //konstruktori
  public Razred(){
    neDiraj = 0;
  }
  public Razred(int neDiraj0, int var0)
  { neDiraj = neDiraj0;
    var = var0;
  }
```

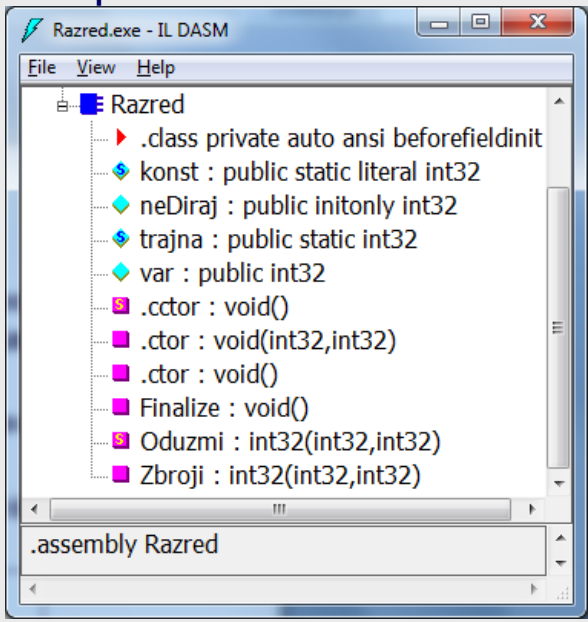
```
Razred r1 =
  new Razred();
Razred r2 =
  new Razred(13,666);
Razred r2 =
  new Razred() {var=4};
```

BY NC SA

Finalizatori (Destruktori)

- **Finalizator je postupak (Finalize) koji se automatski poziva neposredno prije uništenja objekta od strane sakupljača smeća (Garbage Collector).**
  - Piše se u obliku destruktora - jednak je nazivu razreda s predznakom '~'
  - Ne vraća vrijednost
  - Koristi se za brisanje tzv. "unmanaged" resursa (ako su korišteni)
- **Primjer 🗁Osnove\Razred**

```
//finalizator se piše u obliku destruktora
  ~Razred(){
    Console.WriteLine("Finalizer");
  }
}
```



- **Sakupljač smeća (garbage Collector)**
  - Oslobađa memoriju koja je bila zauzeta referencama koje više ne postoje.
  - Sakupljanje smeća obavlja tijekom izvođenja programa.
  - Ne može se unaprijed točno odrediti kada će se memorija osloboditi, to jest kada će se finalizator pozvati, ali ga se može potaknuti (metodom Collect).

CC BY NC SA

Postupci (metode)

- **Postupak je programska funkcija pridružena razredu.**
  - Skrivanje člana (učahurivanje - enkapsulacija)
    - Pristup skrivenom članu javnim postupcima
- **Primjer: 🗁Osnove\Postupci**

```
class Postupak{
  private int brOp = 0; // brojač pozvanih operacija
  public int Zbroji(int x, int y) {
    ++brOp; return x + y;
  }
  public int Razlika (int x, int y) { ++brOp; return x-y; }
  public int GetBrOp() { return brOp; }
...
```

```
Postupak p = new Postupak();
int zbroj = p.Zbroji(13, 15);
int razlika = p.Razlika(13, 15);
int brojOperacija =  p.GetBrOp();
```

CC BY NC SA

Postupci (2)

- **Argumenti:**
  - standardno, bez modifikatora – "call by value“

```
public static void SwapByVal (int a, int b);
SwapByVal (a, b);
```

  - ref modifikator – argumenti su reference ("call by reference")
    - Prije poziva postupka argumenti MORAJU biti inicijalizirani
    - ref parametru argument mora eksplicitno biti predan navođenjem ref.

```
public static void SwapByRef (ref int a, ref int b);
SwapByRef (ref a, ref b);
```

  - out modifikator – izlazni argumenti
    - U trenutku poziva out postupka argumenti ne moraju biti inicijalizirani.
    - Pri izlasku iz postupka out argumenti MORAJU biti postavljeni.

```
public static void TestOut(out int val, int defval);
TestOut(out i, 13);
```

- **Primjer 🗁Osnove\Postupci**

cc BY NC SA

Postupci (3)

- **Prijenos varijabilnog broja argumenata:**
  - Ključna riječ `params`
    - koristi se kad broj argumenata nije unaprijed poznat
    - polje argumenata može biti bilo kojeg tipa
- **Primjer: 🗁Osnove\Postupci**

```
public static void TestParams(params object[] args){
    Console.WriteLine("Params: ");
    for(int i= 0; i< args.Length; i++)
      Console.WriteLine("{0}:{1}", i, args[i])
}
```

```
TestParams("jedan", "dva", 3);
TestParams();
```

- **`Main` je postupak koji preuzima argumente pri pozivu programa**
  - `static void Main( string[] args )`
  - Primjer `TestParams(args);`

BY NC SA

Postupci (4)

- **Opcionalni argumenti kojima se zadaje pretpostavljena vrijednost**
  - Navode se zadnji po redu
- **Primjer: 🗁Osnove\Postupci**

```
public static void TestDefault(int a, int b = 99999,
       string s = "default string") {
 Console.WriteLine("a = {0}, b = {1}, s = {2}", a, b, s);
}
```

```
TestDefault(1, 2, "RPPP");
TestDefault(2, 15);
TestDefault(3);
```

- **Imenovani argumenti:**
  - Priliko poziva navodi se naziv argumenta i vrijednost odvojeni dvotočkom
  - Imenovani argumenti se navode zadnji

```
TestDefault(2, s:"moj string");
TestDefault(s: "moj string", b:55, a:4);
```

CC BY NC SA

Svojstva (properties) (1)

- **Svojstvo je postupak pristupa zaštićenim varijablama instance**
- **Primjer 🗁Osnove\Postupci\Postupak_v2.cs**
  - umjesto postupka za pristup skrivenom članu koristi se tzv. Svojstvo

```
class Postupak_v2{
  private int brOp;
  public int BrOp {
      get { //dohvat vrijednosti člana
        return brOp;
      }
      //set { //omogućava promjenu vrijednosti
      //  brOp = value;
      //}
   }
```

```
Postupak_v2 p2 = new Postupak_v2();
zbroj = p2.Zbroji(a, b);
razlika = p2.Oduzmi(a, b);
brojOperacija = p2.BrOp;
```

CC BY NC SA

Svojstva (2)

- **Nije nužno da svojstvo samo vraća ili postavlja vrijednost varijabli**
  - Može sadržavati i složeniji kod
  - Primjer 🗁Osnove\SvojstvaIndekseri

```
class Temperatura{
  private float T;
  public float Celsius{
    get { return T - 273.16f; }
    set { T = value + 273.16f; }
  }
  public float Fahrenheit{
    get { return 9f / 5 * Celsius + 32; }
    set { Celsius = (5f / 9) * (value - 32); }
  }
}
```

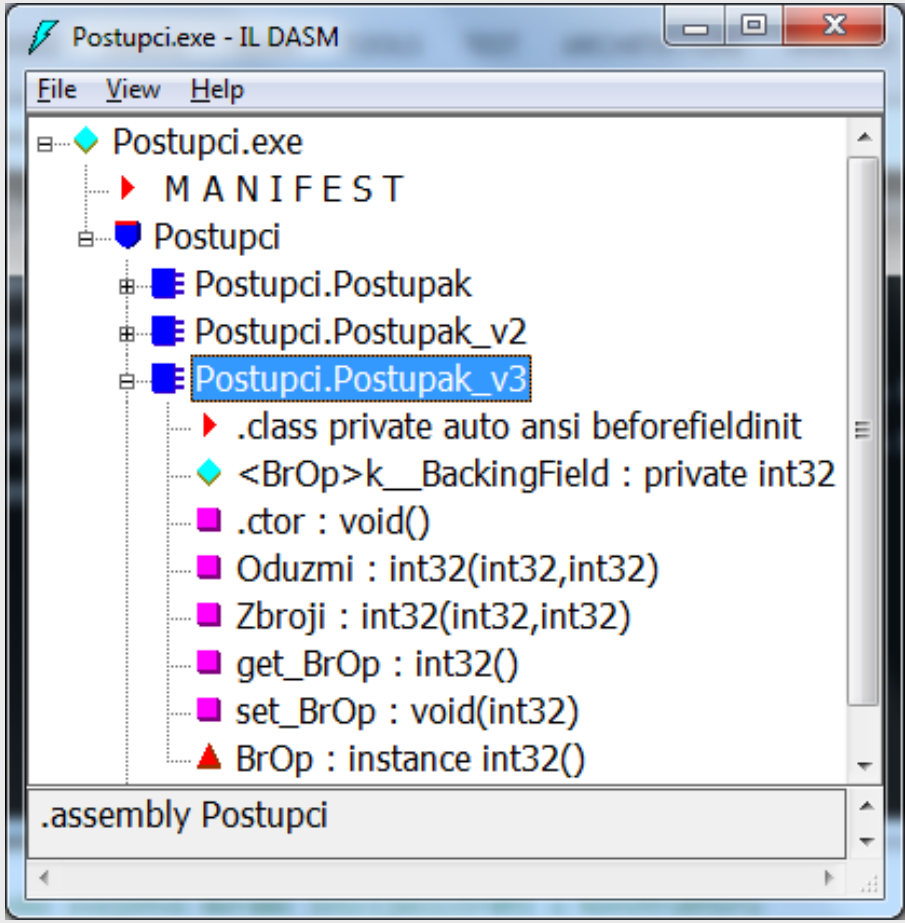
```
Temperatura X = new Temperatura();
X.Fahrenheit = 70;
Console.WriteLine("{0} = {1}", X.Fahrenheit, X.Celsius);
```

CC BY NC SA

Automatska svojstva

- Koristi se u slučaju kad svojstvo služi samo kao omotač oko privatne varijable
- Može se postaviti modifikator pristupa (npr. private) za get i/ili set
- Interno se stvara varijabla za pohranu i kod za dohvat i pridruživanje
- Početna vrijednost se može postaviti od C# 6.0
- Primjer 🗁Osnove \ Postupci \ Postupak_v3.cs

```
class Postupak_v3  {
   public int BrOp {
     get;
     private set;
   } = 0;
...
```



CC BY NC SA

Indekseri

- **Indekseri**
  - omogućavaju korištenje objekta kao niza (pristup elementima operatorom [ ])
  - sintaksa uobičajena za svojstva (get i set)
- **Primjer: 🗁Osnove\SvojstvaIndekseri**

```
public Temperatura this[int index]   // Indekser
{
  set{
    if (index >= 0 && index < nizTemperatura.Length)
      nizTemperatura[index] = value;
    else throw new Exception("Pogreška!");
  }
  get{
    if (index >= 0 && index < nizTemperatura.Length)
      return nizTemperatura[index];
    else throw new Exception("Pogreška!");
  }
}
```

CC BY NC SA

Preopterećenje postupaka (method overloading)

- **Postupci jednakog imena definirani u istom razredu**
  - moraju imati različitu signaturu (prototip), tj. različiti tip ili redoslijed argumenata
  - uobičajeno obavljaju isti posao – nad različitim tipovima podataka
- **Primjer: 🗁Osnove\PreopterecenjePostupaka**

```
static double Maximum(double x, double y, double z){
  Console.WriteLine("double Maximum( double x, double y,
   double z )");
   return Math.Max(x, Math.Max(y, z));
}
static int Maximum(int x, int y, int z){
   Console.WriteLine("int Maximum( int x, int y, int z )");
   return Math.Max(x, Math.Max(y, z));
}
```

```
// koji Maximum se poziva (double ili int)?
Console.WriteLine("maximum1: " + Maximum(1, 3, 2));
Console.WriteLine("maximum2: " + Maximum(1.0, 3, 2));
```

CC BY NC SA

Preopterećenje operatora (operator overloading)

- **Operatori koji se mogu preopteretiti**
  - `unarni: + - ! ~ ++ -- true false`
  - `binarni: + - * / % & | ^ << >> == != > < >= <=`

- **Primjer: 🗁Osnove\PreopterecenjeOperatora**

```
public static ComplexNumber operator +(ComplexNumber x,
  ComplexNumber y){
   return new ComplexNumber(x.Re + y.Re, x.Im + y.Im);
}
public static ComplexNumber operator *(ComplexNumber x,
  ComplexNumber y){
   return new ComplexNumber(x.Re * y.Re - x.Im * y.Im, x.Re *
        y.Im + y.Re * x.Im);
}
```

```
Console.WriteLine(x + " + " + y + " = " + (x + y));
Console.WriteLine(x + " - " + y + " = " + (x - y));
Console.WriteLine(x + " * " + y + " = " + (x * y));
```

CC BY NC SA

Zadaci za vježbu

- **Implementirati razred `Planet`**
  - Konstruktor neka prima radijus, gravitaciju i ime planeta.
  - Razred sadrži i statičku varijablu za brojanje instanciranih planeta, svojstva `Ime, Radijus, Gravitacija` i postupak `GetCount()`.
  - Demonstrirati rad implementiranog razreda.
- **Implementirati razred `PlanetCollection`**
  - Razred predstavlja listu planeta.
  - Iskoristiti razred iz prethodnog zadatka s planetima.
  - Napisati postupak `Add` za dodavanje planeta.
  - Planetima se može pristupati preko indeksa.
- **Implementirati razred `Nebo` koji**
  - sadrži `Planet` i njegovu udaljenost od njegovog sunca.
  - Dodati postupak sortiranja planeta po udaljenosti od sunca,
  - te postupak za računanje “udaljenosti” dvaju planeta.

Vrijednosti i reference

- **Koju vrijednost imaju varijable `val1` i `val2` nakon sljedećeg programskog odsječka?**

```
int val1 = 0;
int val2 = val1;
val2 = 5;
```

- **Koju vrijednost imaju varijable `ref1` i `ref2` nakon sljedećeg programskog odsječka?**

```
class Razred
{
    public int Value = 0;
}
Razred ref1 = new Razred();
Razred ref2 = ref1;
ref2.Value = 123;
```

`new` operator stvara novu instancu razreda `Razred` i vraća pokazivač na nju

ref2 pokazuje gdje i ref1

- **Primjer 🗁Osnove\VrijednostiReference**

CC BY NC SA

Nabrajanje (Enum)

- **Pobrojani tip (enumerator)**
  - Korisnički tip vrijednosti koji nasljeđuje `System.Enum`
  - Sastoji se od imenovanih konstanti
  - Temeljni tip podataka pobrojanog tipa je int

- **Primjer 🗁Osnove\Enum**

```
enum Dani
{
  Ponedjeljak, Utorak, Srijeda,
  Cetvrtak, Petak, Subota,Nedjelja
}
```

```
int x = (int) Dani.Ponedjeljak;
int y = (int) Dani.Utorak;
Console.WriteLine("Pon={0}", x);
Console.WriteLine("Uto={0}", y);
```

Dani počinju od 0.

- **Ako želimo da dani idu od 1. nadalje:**

```
enum DaniPoRedu
{
  Ponedjeljak=1, Utorak, Srijeda, Cetvrtak,
        Petak, Subota,Nedjelja
}
```

CC BY NC SA

Polja

- **Polja određene duljine:**

```
int[] a = new int[10]; // inicijalno vrijednosti 0
int[] b = new int[] { 1, 2, 3};
int[] c = { 1, 2, 3};
```

- **Za duljinu polja moguće je navesti i varijablu**

```
int n = 3;
int[] d = new int[n];
```

- **Dohvat elementa na mjestu i: b[i]**
- **Broj elemenata u polju: b.Length**
- **Primjer 🗁Osnove\Polja**

```
for (int i = 0; i < b.Length; i++)
{
  Console.Write(b[i]);
}
```

CC BY NC SA

Rad s poljima

- **Neki od statičkih postupaka razreda `System.Array` su:**
  - `Sort` – sortira polje ili dio polja, opcionalno uz vlastiti usporednik
  - `BinarySearch` – binarno pretražuje polje i vraća indeks traženog elementa
    - bitovni komplement indeksa prvog elementa većeg od traženog, kad traženi nije u polju a vrijednost traženog je manja od preostalih elemenata polja
    - bitovni komplement broja elemenata polja, kad traženi nije u polju a vrijednost traženog je veća od vrijednosti najvećeg u polju
  - `Copy` – kopira čitavo polje ili dio polja polja u novo polje
  - `Reverse` – obrće redoslijed članova polja

```
int[] z = new int[] { 3, 1, 2 };
System.Array.Sort(z);
Console.WriteLine("Tražim 3 na ind.{0}",
      Array.BinarySearch(z, 3));
…
Array o = new int[3];
Array.Copy(z, o, 2);
Array.Reverse(o);
```

BY NC SA

Pravokutna višedimenzionalna polja

- **Pravokutna (eng. rectangular)**

```
int[,] a = new int[3,5]; //inicijalno vrijednosti 0

int[,] b = new int[2,3] { {1, 2, 3}, {4, 5, 6}};
//int[,] b = new int[,] { { 1, 2, 3 }, { 4, 5, 6 } };

int[,] c = { {1, 2, 3}, {4, 5, 6}};

int[,,,] d = new int[5,6,7,8];

for (int i = 0; i < 2; i++)
   for (int j = 0; j < 3; j++)
     Console.Write (b[i, j]);
```

BY NC SA

Nazubljena višedimenzionalna polja

- **Nazubljena (eng. jagged) ili *polja polja***

```
int[][] a = new int[][] {
              new int[] {2,3,4},
              new int[] {5,6,7,8,9}
            };
int[][] b = { new int[] {2,3,4}, new int[] {5,6,7,8}};

for (int i = 0; i < b.Length; i++){
    for (int j = 0; j < b[i].Length; j++)
        Console.Write(b[i][j]);
    Console.WriteLine();
}

int[][] c;
c = new int[5][];
c[0] = new int[3]; c[1] = new int[7]; ...
```

Struktura Char

- **Tip vrijednosti koji služi za pohranu jednog znaka**
  - `char znak = '٣'; // arapski broj 3`
  - `char znak = '\n'; // novi red`
  - `char znak = '\u0663'; // kod u Unicode tablici za arapski br. 3`

- **Neki atributi i postupci**
  - `ToUpper` pretvara znak u odgovarajuće *Unicode* veliko slovo
    - npr. System.Char.Toupper(znak);
  - `ToLower` pretvara znak u odgovarajuće *Unicode* malo slovo
  - `IsDigit` pripada li znak skupini Unicode decimalnih znamenaka
  - `IsLetter` pripada li znak skupini Unicode slova
  - `MaxValue` najveća moguća vrijednost za `Char`
  - `MinValue` najmanja moguća vrijednost za `Char`

BY NC SA

Razred String

- **`System.String` (isto što i `string`) – niz znakova**
  - jednom postavljena vrijednost ne može se promijeniti.
  - postupci na *stringu* stvaraju novi (promijenjeni) *string*.
  - Općenito, objekti koji nisu string imaju postupak `ToString()`, npr. `123.ToString()`
- **Primjer 🗁Osnove\Stringovi**
  - `string a = "Primjer za String";`
  - Duljina: `a.Length // 17`
  - Uklanjanje znakova: `a.Remove(7,4) // PrimjerString`
  - Velika slova: `a.ToUpper() // PRIMJER ZA STRING`
  - Mala slova: `a.ToLower()  // primjer za string`
  - Podniz: `a.Substring(11)  // String`
  - Uklanjanje vodećih i pratećih praznina `a.Trim()`
  - Rastavljanje stringa na dijelove:
    ```
    string[] stringovi = a.Split(new char[] { ' ' });
     // za niz s početka daje {"Primjer", "za", "String"}
    ```
  - Spajanje u string:
    ```
    string.Join("-", "spoji", "u", 1, "string");
    // vraća spoji-u-1-string
    ```

CC BY NC SA

Razred StringBuilder

- **`System.Text.StringBuilder` – promjenjiv niz znakova**
  - Sve promjene primjenjuju se na izvornom nizu.

- **Primjer 🗁Osnove\Stringovi**
  - `StringBuilder a= new StringBuilder("Primjer za String");`
  - Umetanje: `a.Insert(7,'i') // Primjeri za String`
  - Dodavanje: `a.Append("Builder") //Primjeri za StringBuilder`

CC BY NC SA

Nulabilni tipovi

- Varijable mogu imati nedefiniranu vrijednost
- Deklaracija standardnog tipa sadrži upitnik (?)

```
int? num = null;
if (num.HasValue) // isto što i if (!(num == null))
  Console.WriteLine("num = " + num.Value);
else
  Console.WriteLine("num = Null");
```

- `int?` je pokrata za `Nullable<int>`, ali o Genericsima naknadno
- `?` s ovim označenjem može stajati kod vrijednosnih tipova (struktura)

BY NC SA

Nulabilne reference

- **C# < 8.0**
  - String se ne deklarira kao nulabilan
    - String može biti prazan (tzv. null-string) ali to ne znači da je null
    - `string s = string.Empty // isto što i s = ""`
    - postavljanje na null znači da nije ni prazan
    - `string s = null; // vrijedi s != "", s == null`
- **C# 8.0 uvodi koncept nulabilnih referenci, ali ga je potrebno eksplicitno uključiti u projektu**
  - https://docs.microsoft.com/en-us/dotnet/csharp/nullable-references
  - `string?` bi na taj način predstavljao referencu koja smije biti *null*, a `string` bi bila referenca koja nije *null* i za koju nije potrebno provjeravati je li *null*

BY NC SA

Tipovi podataka *var* i *dynamic*

- **Varijable definirane kao tip *var***
  - Deklaracija je moguća samo unutar određenog postupka
  - Stvarni tip podatka se određuje prilikom kompilacije

```
var sb = new StringBuilder();
StringBuilder sb = new StringBuilder();
```

- **Varijable definirane kao *Dynamic* varijable**
  - Stvarni tip podatka se određuje prilikom izvršavanja

```
dynamic s;
s = "Neki tekst";
Console.WriteLine(s.GetType());
s = 12;
Console.WriteLine(s.GetType());
```

  - Zaobilazi se provjera prilikom kompilacije (većinom nepoželjno osim u iznimnim slučajevima) te se pretpostavlja da podržava bilo koju operaciju (metodu, varijablu)
  - Može biti argument funkcije

cc BY NC SA

Zadaci za vježbu

- **Napisati program koji iz polja stringova ispisuje one koji sadržavaju zadani podniz.**

- **Napisati program koji**
  - učitava tekst sa standardnog ulaza
  - mijenja ga u velika slova
  - ispisuje tekst u obrnutom poretku.

- **Demonstrirati generator pseudoslučajnih brojeva**
  - Upotrijebiti razred `System.Random`

BY NC SA

Dodatni materijali, reference

- **Objects, Classes, and Structs (C# Programming Guide)**
  - Objects, Classes, Structs, Access modifiers, ...
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers

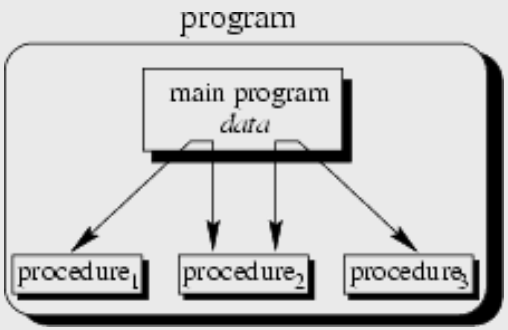
Tehnike programiranja

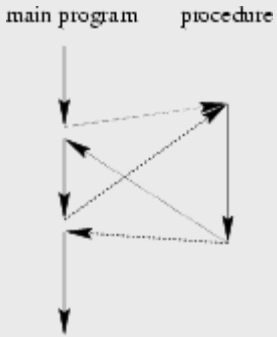
2019/20.03

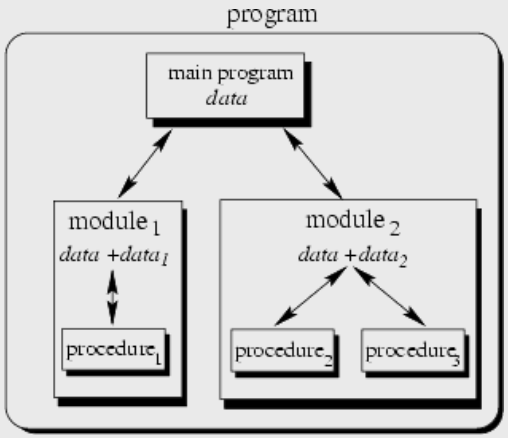
Zaštićeno Creative Commons licencom, http://creativecommons.org

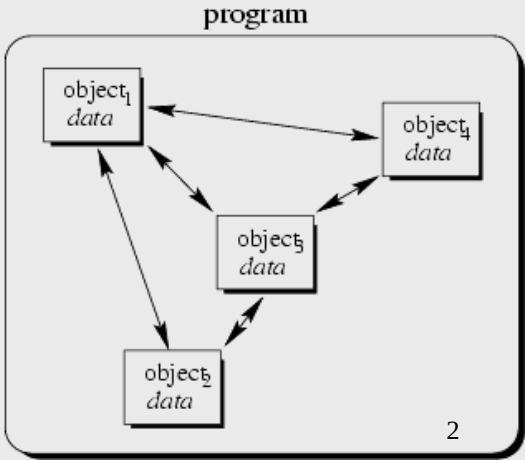
Tehnike programiranja

- **Nestrukturirano**
  - Programi koji se sastoje od slijeda naredbi i djeluju nad zajedničkim skupom podataka (globalnim varijablama)
  - Ponavljanje nekog posla znači i kopiranje naredbi.
- **Proceduralno**
  - Izdvajanjem naredbi u procedure,
  - Program postaje slijed poziva procedura.
- **Modularno**
  - Procedure zajedničke funkcionalnosti grupiraju se u module
  - Dijelovi komuniciraju pozivima, moduli mogu imati lokalne podatke
- **Objektno**
  - Objekti komuniciraju slanjem poruka









CC BY NC SA

Dijelovi programa

- **programska cjelina (program unit)**
  - skup programskih naredbi koje obavljaju jedan zadatak ili jedan dio zadatka, npr. glavni program, potprogrami (procedure, funkcije)

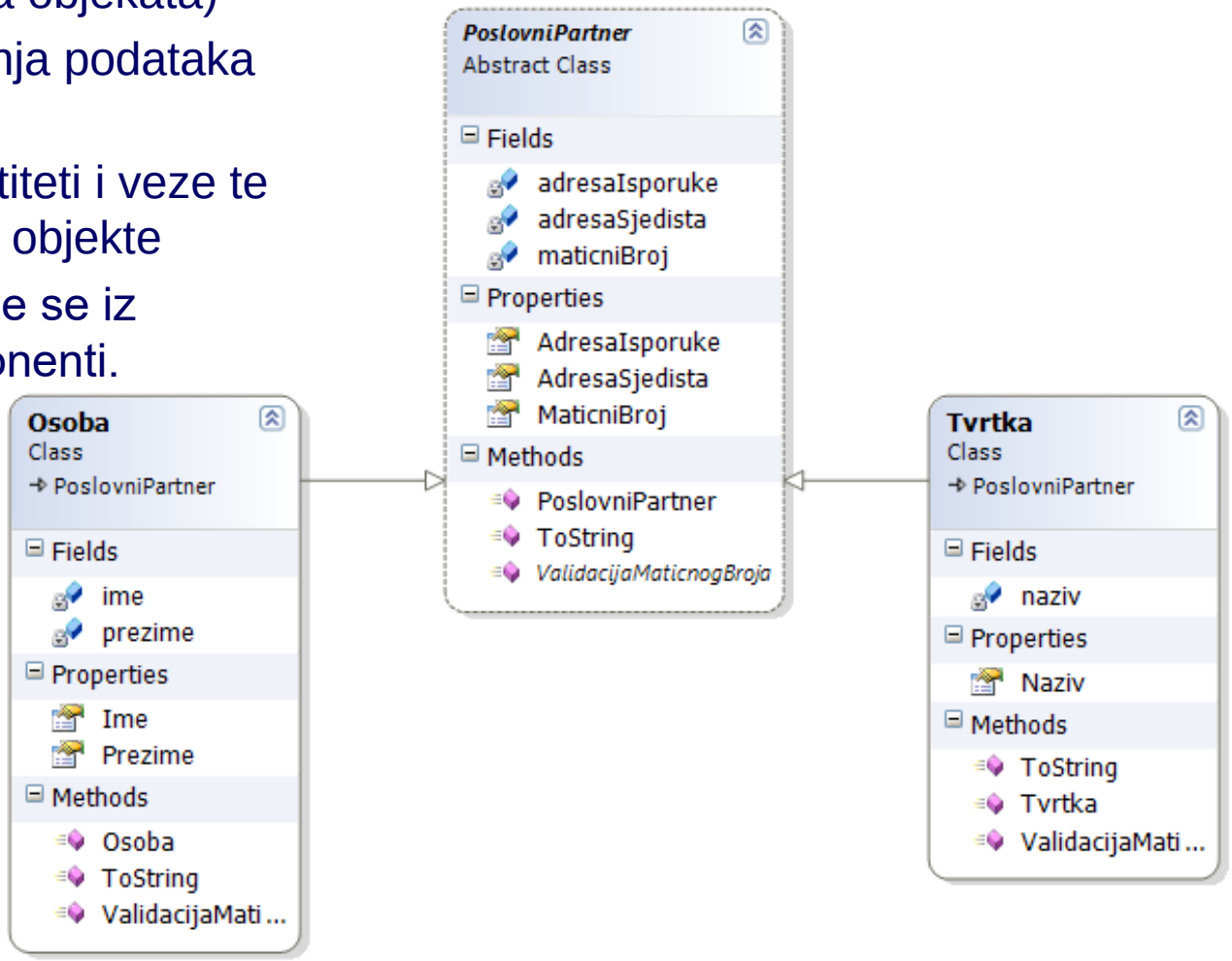
- **programski modul**
  - skup logički povezanih programskih cjelina → modularno programiranje
  - npr. C datoteka sa statičkim varijablama

- **komponenta**
  - bilo koji sastavni dio softvera, uobičajeno podrazumijeva fizičke cjeline
  - npr. assembly (skup, sklop)
    - DLL ili EXE (izvršna) datoteka uključujući pripadajuće resursne (datoteka.resx) ili sigurnosne elemente

CC BY NC SA

Osnove objektno orijentiranog programiranja

- **Objektno usmjerena paradigma**
  - entiteti iz stvarnog svijeta opisuju se apstraktnim objektima (razredima, klasama objekata)
  - integracija oblikovanja podataka i procesa
  - tokovi podataka, entiteti i veze te procesi integrirani u objekte
  - složeni sustavi grade se iz pojedinačnih komponenti.



CC BY NC SA

Koncepti objektnog pristupa

- **Objekt (Object)**
  - objekt - svaka pojava, pojam ili predmet (stvarni ili zamišljeni) o kojem se prate podaci i načini rukovanja podatcima (procesi)
  - pojava objekta (object instance) – pojedinačna pojava nekog tipa objekta, s vlastitim stanjem i ponašanjem
- **Apstrakcija (Abstraction)**
  - Obuhvat glavnih aspekata uz ispuštanje nevažnih detalja
- **Učahurivanje (Encapsulation)**
  - skrivanje ugradnje od korisnika objekta
  - objekti su “crne kutije” čijim se podacima rukuje putem ugrađenih metoda
- **Modularnost (Modularity)**
  - Razlaganje složenih sustava u (lakše) upravljive dijelove.
- **Hijerarhija (Hierarchy)**
  - Hijerarhijsko uređivanje apstrakcija

BY NC SA

Koncepti objektnog pristupa (2)

- **Razred (Class)**
  - tip objekta (abstract data type) – generalizacija grupe objekata, zajednički naziv za sve objekte sa zajedničkim svojstvima (property) i načinima rukovanja (process, method), npr. Proizvod
  - opis grupe objekata koji imaju jednake atribute, operacije, veze i semantiku
  - objekt je instanca razreda, s vlastitim vrijednostima svojstava
  - razred je apstraktna definicija objekta
- **Svojstvo (Property), atribut (Attribute)**
  - definira skup vrijednosti koje instanca može poprimiti za neku karakteristiku
  - može biti nekog tipa, npr. integer, Boolean, Adresa
- **Postupak (Method), Operacija (Operation)**
  - implementacija usluge koja može biti zatražena od objekta razreda
  - ima prototip, „potpis” (naziv, parametre, povratnu vrijednost)
- **Višeobličje (Polymorphism)**
  - mogućnost "skrivanja" različitih ugradnji iza zajedničkog sučelja
  - postupci koje objekti različitih razreda tumače zavisno o specijalizaciji

CC BY NC SA

Koncepti objektnog pristupa (3)

- **Nasljeđivanje (Inheritance)**
  - podrazred (subclass) nasljeđuje svojstva i metode nadrazreda (superclass), npr. Igla i Avion nasljeđuju Proizvod
- **Podsustav (Subsystem)**
  - kombinacija paketa razreda
  - realizacija jednog ili više sučelja koje definiraju ponašanje razreda
- **Komponenta (Component)**
  - nezavisan, zamjenjiv, ne-trivijalan dio sustava
  - može biti izvorni kod, pogonska inačica ili izvršni program
- **Veza, asocijacija (Association)**
  - smislena poveznica između dva ili više dijelova

Nasljeđivanje, višeobličje ...

BY NC SA

Nasljeđivanje

- **Omogućuje izvođenje (deriviranje, ne pokretanje) novog razreda na temelju postojećeg razreda**
  - `class DerivedClass: BaseClass { }`
- **Generalizacija (i pripadna specijalizacija)**
  - osnovni razred – bazni razred (base class), superrazred (superclass), roditelj
  - izvedeni razred – derivirani (derived), podrazred (subclass), dijete
- **Dijete nasljeđuje članove roditelja i definira vlastite članove**

- **Višeobličje (polymorphism)**
  - Operacija deklarirana u osnovnom razredu biva poništena u korist operacije jednako deklarirane u naslijeđenom razredu.
  - Podtip se ponaša drukčije odnosu na nadtip ili neki drugi podtip.
- **Zamjenjivost (supstitutability)**
  - Razred je podtip osnovnog razreda ako je u aplikaciji moguće zamijeniti osnovni razred podtipom, a da aplikacija normalno nastavi s radom.
  - Drugim riječima, ako postoji kôd koji se odnosi na nadtip `Kupac`, umjesto njega može se supstituirati bilo koji njegov podtip

CC BY NC SA

Primjer nasljeđivanja

```
class Kupac {
  public void PisiAdresu() { … }
}
class PravOsoba: Kupac {
  public string VrstaRacuna() { return "R-1"; }
}
class FizOsoba : Kupac {
  public string VrstaRacuna() { return "R-2"; }
}
…
FizOsoba f = new FizOsoba();
f.pisiAdresu();       // poziv naslijeđene metode

string s = f.VrstaRacuna();  // metoda naslijeđenog razreda

Kupac k = f;          // podrazred kao osnovni razred
k.pisiAdresu();
```

BY NC SA

Nasljeđivanje u programskom jeziku C#

- U programskom jeziku C# moguće je naslijediti samo jedan razred, a implementirati više sučelja.

```
class Osoba : Partner, IAdresa, IRazred { ... }
```

- `public` i `protected` varijable i postupci se nasljeđuju
- `private` varijable i postupci ne mogu biti naslijeđeni
- `sealed` razred ne može biti naslijeđen

```
sealed class SealedPoint { ... }
class MyPoint : SealedPoint // pogreška prevođenja
```

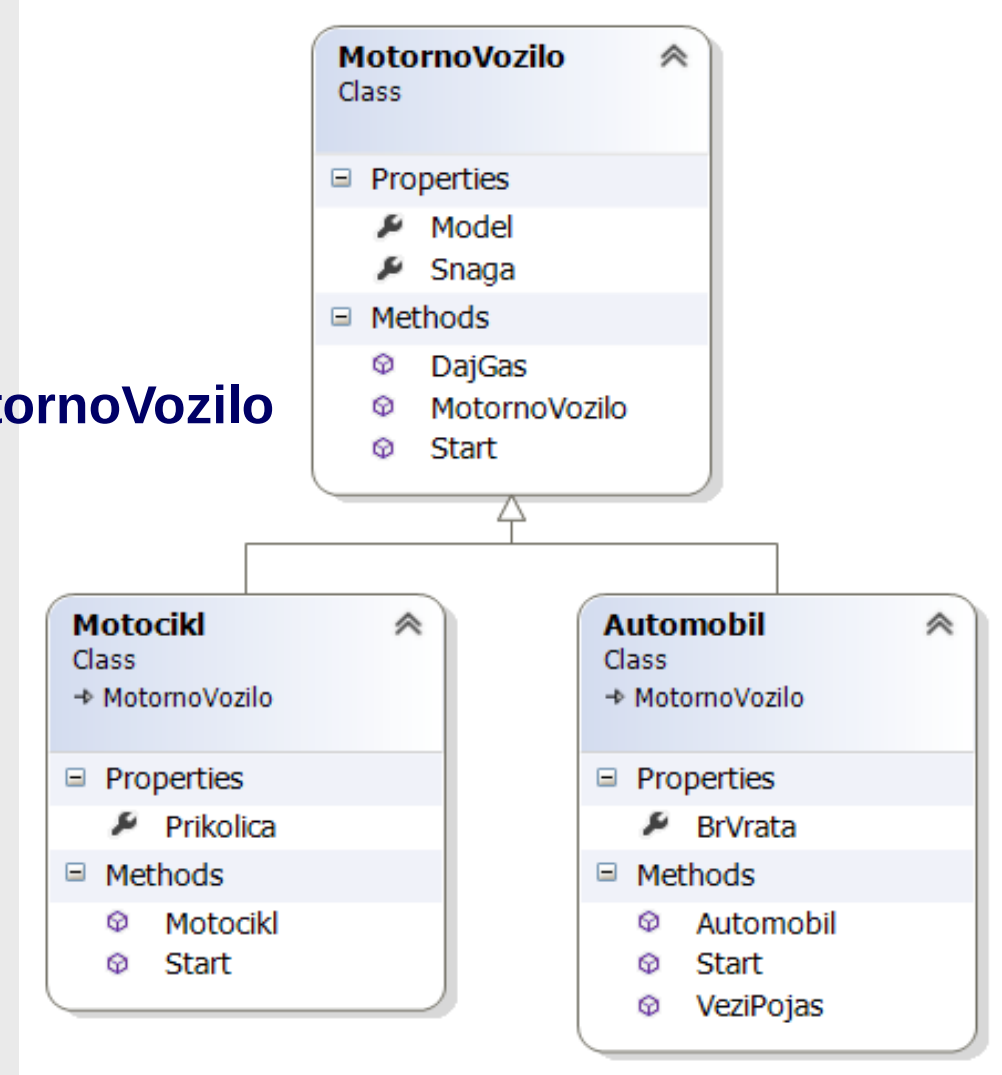
- `base.member` pristup nadjačanim članovima
  - Npr. `base.F();`
- Instanciranje objekta izvedenog razreda izaziva poziv konstruktora osnovnog razreda
- Pri uništenju objekta prvo se poziva finalizator izvedenog razreda, a zatim finalizator osnovnog razreda

CC BY NC SA

Nadjačavanje postupaka (method overriding)

- `virtual` deklarira virtualni postupak roditelja koji može biti nadjačan
- `override` deklarira postupak djeteta koji nadjačava, a može koristiti nadjačani postupak

- **Primjer 🗁Razredi\MotornoVozilo**



CC BY NC SA

Primjer nasljeđivanja – osnovni razred

- **Primjer 🗁Razredi \ MotornoVozilo**

```
class MotornoVozilo
{
  // atributi
  public string Model{ get; set; } // svojstvo
  ...

  // konstruktor
  public MotornoVozilo(string model, double snaga) { ... }

  // postupak (zajednicki svim vozilima)
  public void DajGas() { ... }

  // postupak koji izvedeni razredi implementiraju zasebno
  public virtual void Start() { ... }
}
```

Primjer nasljeđivanja

```
class Automobil : MotornoVozilo
{
    // dodatni atribut
    public int BrVrata { get; set; }

    // konstruktor
    public Automobil(string model, double snaga, int brVrata)
        : base(model, snaga)  // poziv osnovnog konstruktora
    { ... }

    // nadjačavanje baznog postupka Start()
    public override void Start() { ... }

    // dodatni postupak za Automobil
    public void VeziPojas() { ... }
}
```

- Tko što nasljeđuje?
- Može li se „sakriti” bazno svojstvo ili metoda (public pretvoriti u private) ?
- Što bi bilo da metoda *Start* nije virtualna?

BY NC SA

Podsjetnik: modifikatori tipova i članova

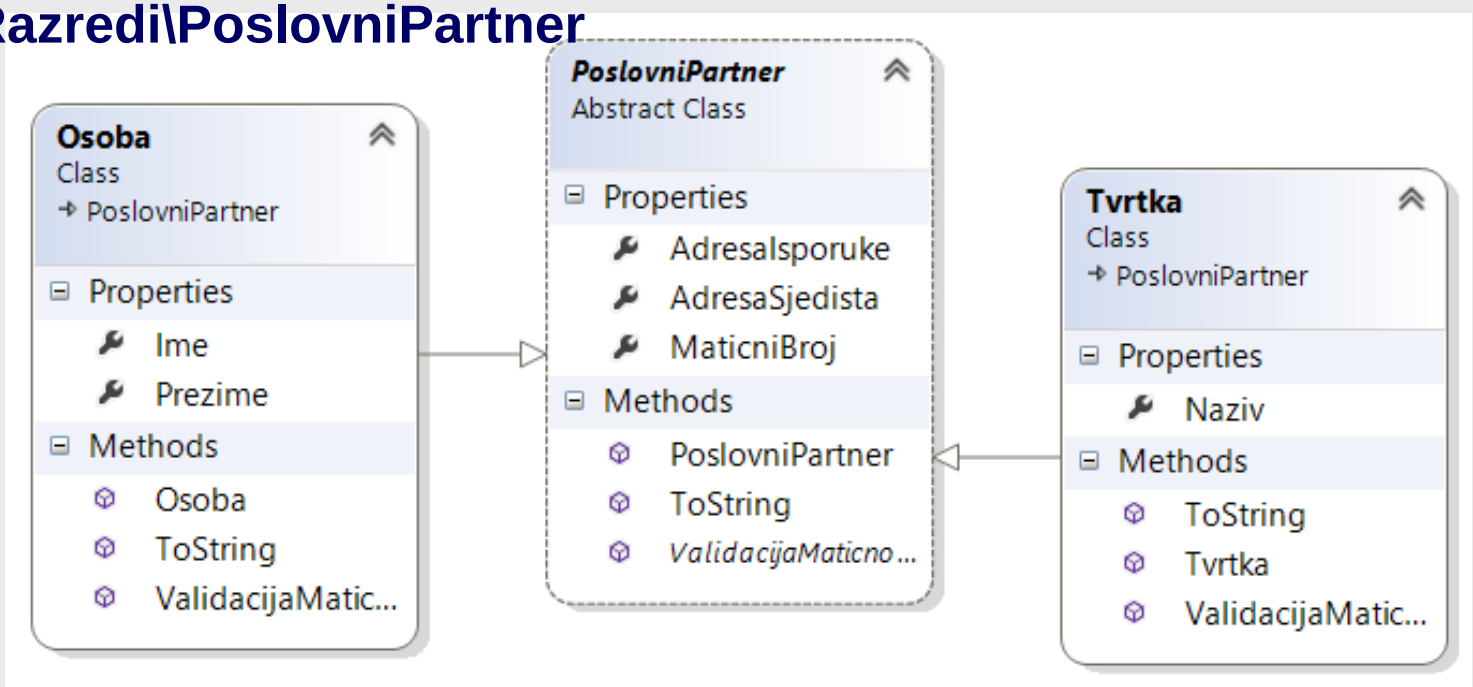
- **Modifikatori pristupa razredima i članovima**
  - `public` – pristup nije ograničen
  - `private` - pristup ograničen na razred u kojem je član definiran
  - `protected` – pristup ograničen na razred i naslijeđene razrede
  - `private internal` – ogramničenje prethodnog samo na nasljeđene razrede unutar istog projekta
  - `internal` – pristup ograničen na program u kojem je razred definiran
  - `protected internal` - pristup dozvoljen naslijeđenim razredima (bez obzira gdje su definirani) i svima iz programa u kojem je razred definiran
- **Neki od značajnijih modifikatora**
  - `abstract` – razred može biti samo osnovni razred koji će drugi nasljeđivati
  - `const` – atribut (polja) ili lokalna varijabla je konstanta
  - `new` – modifikator koji skriva naslijeđenog člana od člana osnovnog razreda
  - `readonly` – polje poprima vrijednost samo u deklaraciji ili pri instanciranju
  - `sealed` – razred ne može biti naslijeđen
  - `static` – jedini, zajednički član svih instanci razreda (ne kopija nastala s instancom)
  - `virtual` – postupak ili dostupni član koji može biti nadjačan u naslijeđenom razredu – (prilikom nadjačavanja dodaje se modifikator `override`)

BY NC SA

Apstraktni razredi

```
abstract class A {
   protected A(){...}
   public abstract int F();
}
class B: A {
   public override int F() { ... }
}
```

- **`abstract` – apstraktni razred**
  - Osnovni razred je nedovršen
  - Apstraktni postupci nisu ugrađeni
  - Izvedeni razred **mora** u potpunosti ugraditi nedovršene postupke
    - osim ako i sam nije apstraktan
  - Ne može se instancirati objekt apstraktnog razreda, ali apstraktni razred može imati konstruktor!
- **Primjer 🗁Razredi\PoslovniPartner**



CC BY NC SA

Primjer nasljeđivanja apstraktnog razreda

- **Razred Osoba nasljeđuje razred PoslovniPartner**
  - Konstruktor

```
public Osoba(string maticniBroj, string adresaSjedista, string
adresaIsporuke, string ime, string prezime)
  : base(maticniBroj, adresaSjedista, adresaIsporuke){
        this.Ime = ime; this.Prezime = prezime;
}
```

  - Implementacija apstraktnog postupka

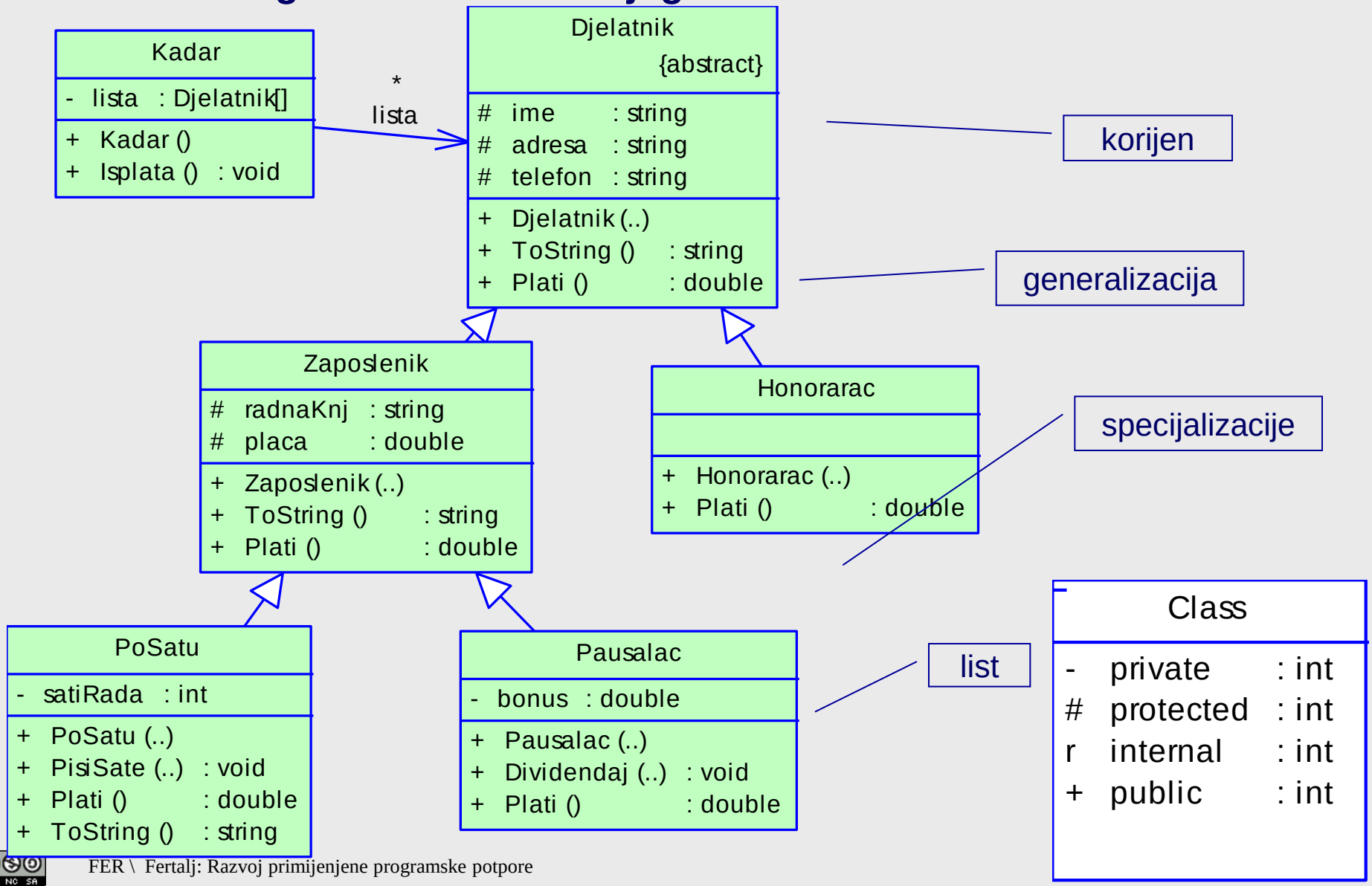
```
abstract class PoslovniPartner{
  public abstract bool ValidacijaMaticnogBroja();
  // zbog abstract, ovaj postupak potrebno je implementirati u
  // izvedenom razredu
}
```

```
class Osoba : PoslovniPartner{
  public override bool ValidacijaMaticnogBroja()
  { ...implementacija u izvedenom razredu }
}
```

CC BY NC SA

Hijerarhija nasljeđivanja

- **Definirati i ugraditi razrede s dijagrama - 🗁Razredi\Kadrovska**



BY NC SA

Dinamička klasifikacija

- **Dinamičko povezivanje- primjer 🗁Razredi\Kadrovska**
  - Referenca može pokazivati na različite objekte
  - tip trenutnog objekta određuje postupak koji se obavlja

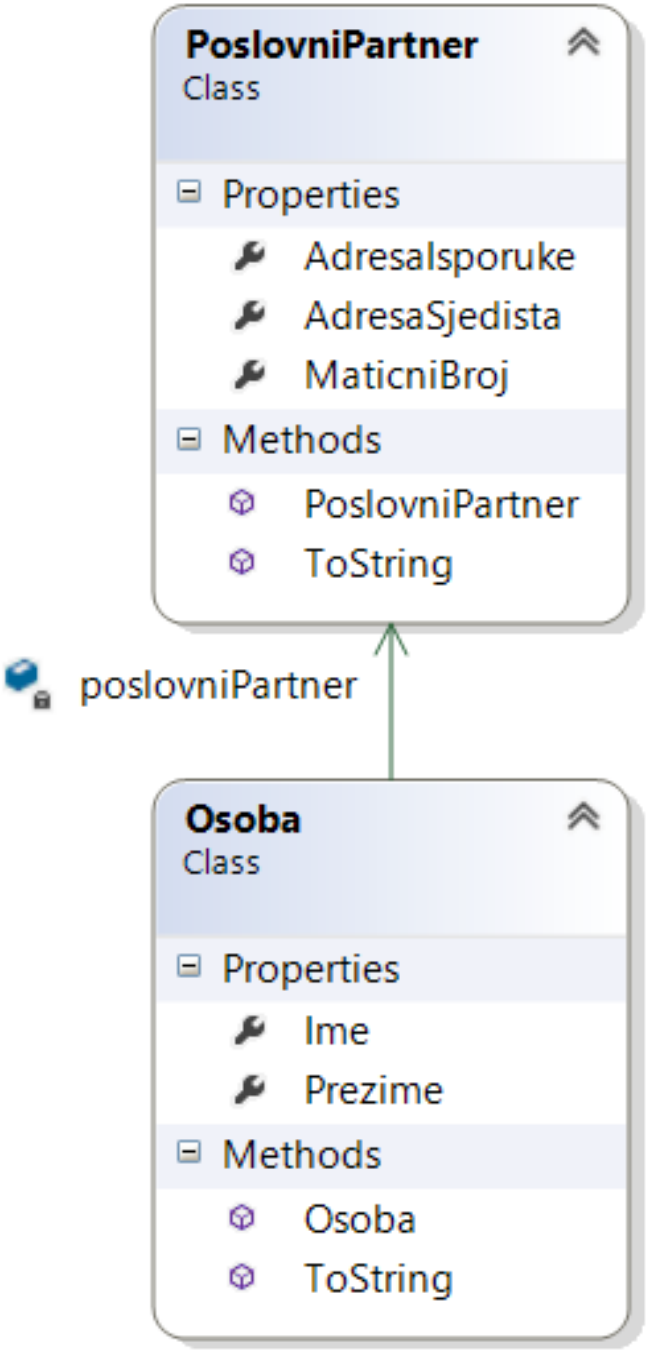
```
public class Kadar {
  private Djelatnik[] lista;

  public Kadar () {
    lista = new Djelatnik[6];
    lista[0] = new Pausalac ("Bobi", "Bobinje bb",
      "555-0469", "123-45-6789", 2423.07);
    ...
    ((Pausalac)lista[0]).Dividendaj (500.00); // cast
    ((PoSatu)lista[3]).PisiSate (40); // cast
  }
  public void Isplata() {
    double iznos;
    for ( int count=0; count < lista.Length; count++ ) {
      Console.WriteLine( lista[count] );
      iznos += lista[count].Plati();  // polymorphic
  }
}
```

Kompozicija umjesto nasljeđivanja

- Umjesto nasljeđivanja, dijete može "učahuriti roditelja"

- Primjer 🗁Razredi\Kompozicija

```
class Osoba
{
    PoslovniPartner poslovniPartner;
    private string ime;
    private string prezime;
    ...
}
```



CC BY NC SA

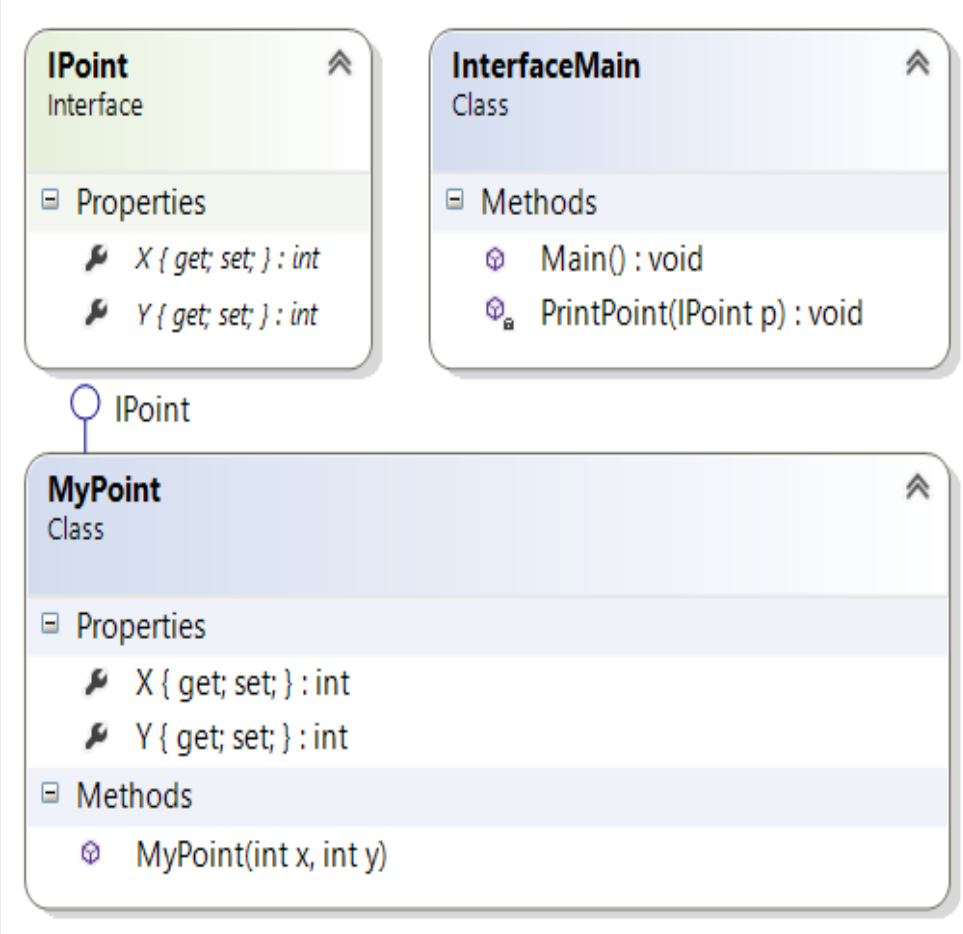
Sučelje (interface)

- **Sučelje (interface) - specifikacija članova razreda bez implementacije**
  - Sadrži samo deklaracije operacija
  - Može sadržavati postupke, svojstva, indeksere i događaje
  - Svaki postupak je apstraktan (nema implementaciju)
- **Notacija**
  - Standard za naziv: *INaziv*
  - Proširena i skraćena (*lollipop*) notacija
- **Sučelje definira obvezu ugradnje**
  - Razred koji nasljeđuje sučelje, mora implementirati sve postupke
  - Sučelje može implementirati i apstraktni razred
- **Realizacija (realisation)**
  - veza izvornog razreda i sučelja
  - sučelje može biti argument nekog postupka čime se postiže veća općenitost

BY NC SA

Primjer sučelja i realizacije

🗁Razredi\Sucelje




```
interface IPoint {
  // Property signatures:
  int X { get; set; }
  int Y { get; set; }
}
```

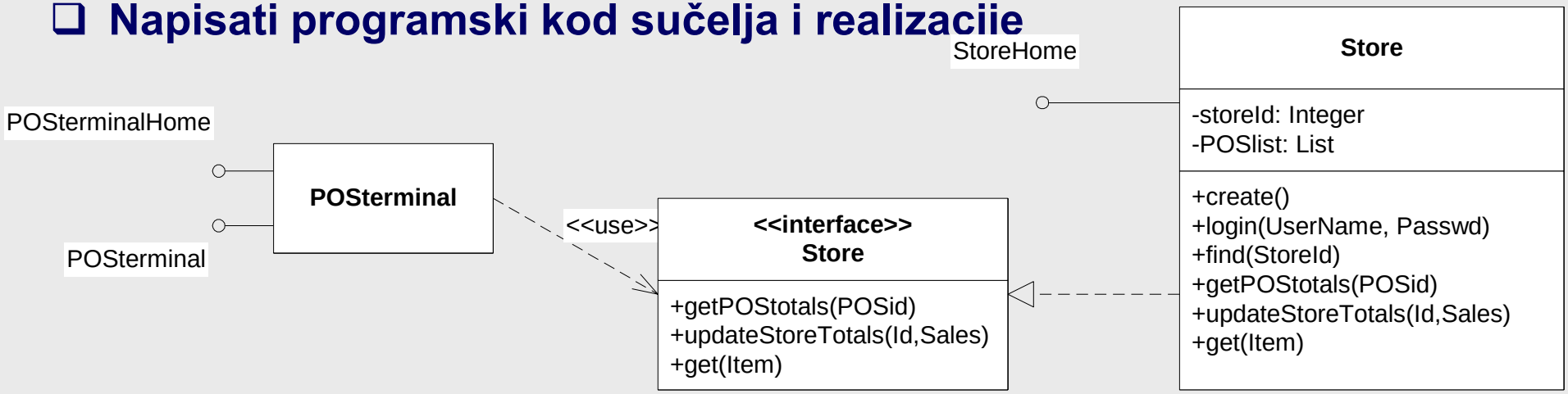
```
class MyPoint : IPoint {
  //automatsko svojstvo
  public int X{
        get ; set ;
  }
...
}
```

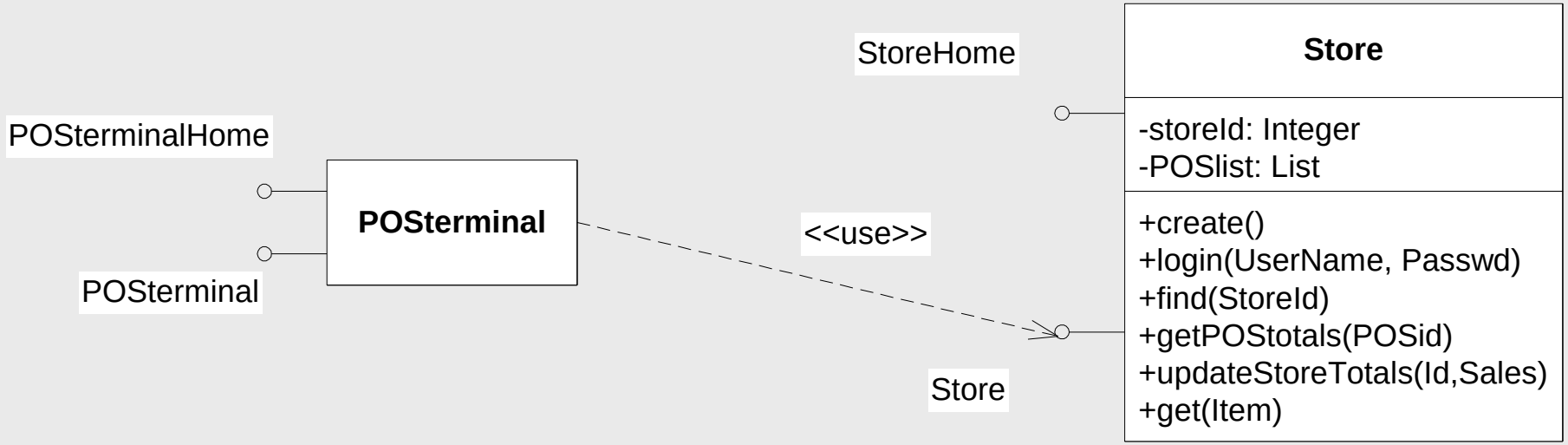
```
class InterfaceMain
{
  private static void
    PrintPoint(IPoint p) {
...
```

CC BY NC SA

Zadatak za vježbu

- Ujedno primjer za različite notacije sučelja
- Napisati programski kod sučelja i realizacije





System.Collections

- **Sadrži sučelja i razrede za rad s kolekcijama**
  - Kolekcija je skup povezanih objekata.
- **Sučelja: ICollection, IEnumerator, IEnumerable, IDictionary i IList**
  - Određuju osnovne funkcionalnosti kolekcija
  - Razred koji implementira jedno ili više tih sučelja naziva se kolekcija.
- **Razredi:**

`ArrayList`	**implementira *IList* sučelje pomoću polja za pohranu objekata čija se veličina po potrebi povećava**
`BitArray`	**Radi s poljem bitova koji su predstavljeni kao *Boolean* vrijednosti (*true* predstavlja 1 a *false* 0)**
`Hashtable`	**Predstavlja kolekciju vrijednosti ključ-i-vrijednost parova (vrijednost je objekt pohranjen u kolekciji) organiziranih pomoću *hash* vrijednosti ključa**
`Queue`	**Predstavlja first-in-first-out kolekciju objekata**
`SortedList`	**Predstavlja kolekciju ključ-i-vrijednost parova poredanih (sortiranih) po ključu, uz moguć dohvat po ključu i indeksu**
`Stack`	**Predstavlja last-in-first-out kolekciju objekata**

CC BY NC SA

System.Collections.Generic

- **Prostor imena `System.Collection.Generics`**
  - Sadrži sučelja i razrede za generičke kolekcije
  - Funkcionalnosti negeneričkih kolekcija
  - Bolja tipska sigurnost (type safety) i bolje performanse od negeneričkih

- **Neki razredi prostora imena `System.Collections.Generics`**
  - `public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>, IList, ICollection, IEnumerable`
  - `public class Queue<T> : IEnumerable<T>, ICollection, IEnumerable`
  - `public class Stack<T> : IEnumerable<T>, ICollection, IEnumerable`
  - `public class Dictionary<TKey,TValue> : IDictionary<TKey,TValue>, ICollection<KeyValuePair<TKey,TValue>>, IEnumerable<KeyValuePair<TKey,TValue>>, IDictionary, ICollection, IEnumerable, ISerializable, IDeserializationCallback`

CC BY NC SA

System.Collections.Generic

- **`List(T)` – parametrizirana kolekcija objekata kojima se može pristupati preko indeksa u listi**
  - Npr `List<int>, List<string>, List<Osoba>`
  - Funkcionalnost razreda `ArrayList`

```
List<string> listaString = new List<string>();
listaString.Add("jedan");
...
string trazeni = ...
if (listaString.Contains(trazeni))
  Console.WriteLine("Element postoji u listi na " +
      listaString.IndexOf(trazeni) + ". mjestu.");
else
  Console.WriteLine("Element ne postoji u listi!");
```

BY NC SA

Prednosti generičkih kolekcija

- **Negeneričke kolekcije objekte pohranjuju kao `System.Object`**
  - Takva pohrana je prednost sa stanovišta fleksibilnosti
  - Nedostatak je obavljanje *boxinga* prilikom dodavanja `value` tipa podataka (dobivanje reference na `value` podatak da bi se s tim podatkom moglo postupati kao sa `System.Object`)
    - pad performansi za veće količine podataka
    - potrebna konverzija (*unboxing*) da se dobije smisleni podatak

- **Generičke kolekcije su tipizirane**
  - Bolje performanse – ne mora se obavljati *boxing/unboxing*
  - Tipska sigurnost (*type safety)* – zaštita od unosa podataka nepoželjnog tipa i pogrešaka koje pritom mogu nastati (npr. pokušaj izvršavanja naredbe neprikladne za dani tip podataka)

BY NC SA

Primjer izrade parametriziranog razreda

- **Primjer 🗁Razredi\Generics**

```
using System.Collections.Generic;

public class Stog<T>
{
  T[] elementi;
  public void Stavi(T element) {...}
  public T Skini() {...}
}

Stog<int> stogInt = new Stog<int>();
Stog<string> stogString = new Stog<string>();
Stog<Automobil> stogAutomobil = new Stog<Automobil>();
```

CC BY NC SA

Ograničenja na tip parametriziranog razreda

- **Kontekstualna ključna riječ *where***
- **Moguća ograničenja nekog tipa T:**
  - where T:struct – tip T mora biti *value type* (*Nullable* također isključen)
  - where T:class – tip T mora biti *reference type*
  - where T:new() – tip T mora imati prazni konstruktor
  - where T:naziv baznog razreda– tip T mora biti navedeni razred ili razred koji nasljeđuje taj razred
  - where T:naziv sučelja – tip T mora implementirati navedeno sučelje
  - where T:U – tip T mora tip U ili izveden iz tipa U pri čemu je U drugi tip po kojem se vrši parametrizacija
- ***Primjer:***

```
public class GenRazred<T, U>
  where T:Stog<U> where U:IPoint, new(){
...
```

  - U navedenom primjeru *GenRazred* je određen s dva tipa pri čemu prvi tip mora biti stog iz prethodnog primjera određen tipom koji implementira sučelja IPoint i ima prazni konstruktor

CC BY NC SA

Tablice s raspršenim adresiranjem i rječnici

- **Dictionary<TKey, TValue>**
  - System.Collection.Generics
  - Parovi ključ, vrijednosti
  - Ključ i vrijednost su tipizirani
  - Razlikovati postupak Add i indekser [ključ]

```
Dictionary<int, string> dict =
 new Dictionary<int, string>(10);

dict.Add(100, "Jedan");
dict[200] = "Dva";

foreach(KeyValuePair<int,string> entry in dict){
  Console.WriteLine(entry.Value);
}
```

CC BY NC SA

Proširenja (eng. extensions)

- **Razred za koji se piše proširenje je naveden kao prvi parametar statičke metode u statičkom razredu, prefiksiran s *this***
  - Pozivaju se kao da se radi o postupku unutar tog razreda (iako to nije)
- **Primjer 🗁Razredi\Generics**

```
public static class Extensions{
  public static V DohvatiIliStvori<K, V>
   (this Dictionary<K, V> dict, K key) where V : new(){
      if (!dict.ContainsKey(key)){
        V val = new V();
        dict[key] = val;
      }
      return dict[key];
    }
  }
}
```

```
var dict = new Dictionary<int, Stog<string>>();
var stog = dict.DohvatiIliStvori(1);
```

CC BY NC SA

Odnosi razreda u složenijim tipovima

- **Razred `Automobil` je podrazred razreda `MotornoVozilo`.**

- **U kojem su odnosu `List<Automobil>` i `List<MotornoVozilo>` ?**
  - Nisu hijerarhijski povezani

- **U kojem su odnosu**
  `IEnumerable<Automobil>` i `IEnumerable<MotornoVozilo>`
  `IComparer<Automobil>` i `IComparer<MotornoVozilo>`

  - Odgovor nije očit (jednostavan) !!

BY NC SA

Invarijantnost, kovarijantnost i kontravarijantnost

- **Invarijantnost (engl. invariance)**
  - Mora se koristiti samo specificirani tip
- **Kovarijantnost (engl. covariance)**
  - Mogućnost korištenja nekog izvedenog tipa umjesto originalno navedenog
  - Neki tip je kovarijantan ako zadržava postojeće odnose među tipovima
- **Kontravarijantnost (engl. contravariance)**
  - Mogućnost korištenja nekog općenitijeg tipa umjesto originalno navedenog
  - Neki tip je kontravarijantan ako stvara suprotan odnos među postojećim tipovima

Kovarijantnost

- **Primjer 🗁Razredi \ CovarianceContravariance**

```
void IspisiVozila(IEnumerable<MotornoVozilo> vozila) {
  for(var vozilo in vozila)
    Console.WriteLine(vozilo.Model);
}
```

- **Kao argument moguće je poslati `List<MotornoVozilo>`, jer `List<T>` implementira sučelje `IEnumerable<T>`, ali moguće je poslati i `List<Automobil>`**
  - *List<Automobil>* se može pretvoriti u *IEnumerable<Automobil>*, a <u>sučelje *IEnumerable<T>* je kovarijantno</u>

```
public interface IEnumerable<out T>
```

```
IEnumerable<Automobil> auti = new List<Automobil>();
...
IspisiVozila(auti)
```

BY NC SA

Kontravarijantnost

- Primjer 🗁Razredi \ CovarianceContravariance

```
void IspisiBoljiMotor(Motocikl a, Motocikl b,
  IComparer<Motocikl> comparer)
{
  Console.Writeline(comparer.Compare(a, b) < 0 ? a : b);
}
```

- Kao komparator moguće je poslati objekt tipa `IComparer<MotornoVozilo>`, jer je sučelje <u>`IComparer<T>` kontavarijantno</u>

```
public interface IComparer<in T>
```

Delegati

- **Delegati su objekti koje sadrže reference na postupke**
  - Omogućavaju metodama da budu argumenti neke druge metode
  - Nalik pokazivačima na funkcije u C-u
- **Delegati koji sadrže reference na više postupaka nazivaju se *MultiCastDelegate***
  - Pozivom delegata redom se pozivaju referencirane metode
- **Delegat se definira kao varijabla određenog tipa delegata**
- **Tip delegata definira se sljedećom sintaksom**

```
public delegate PovratniTip NazivTipaDelegata (argumenti)
```

  - Npr. public delegate double Tip(int a, string b) bi definirao tip delegata koji bi omogućio pohranu referenci na sve postupke kojima imaju dva argumenta tipa int i string, a vraćaju double
  - Interno se stvara novi razred *NazivTipaDelegata* koji nasljeđuje razred *MultiCastDelegate*
  - Nakon toga bi se mogao definirati delegat na sljedeći način `Tip nazivdelegata`;
  - Delegati imaju definirane operacije =, +=, -=

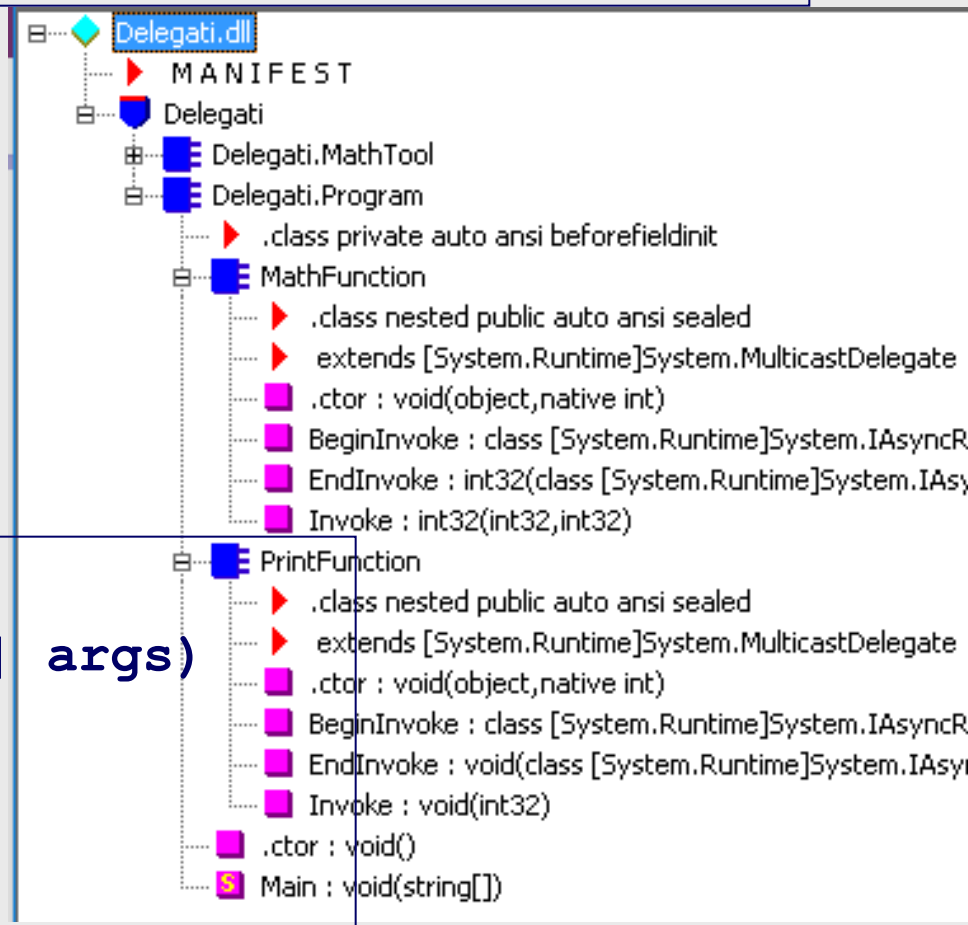
Primjer tipa delegata

- Primjer 🗁Razredi \ Delegati \ Program.cs

```
class Program {
    public delegate int MathFunction(int a, int b);
    public delegate void PrintFunction(int n);
```

- Definirana su 2 tipa delegata
  - MathFunction za postupke koji primaju dva cjelobrojna argumenta i vraćaju cijeli broj
  - PrintFunction za postupke koji primaju cijeli broj i ne vraćaju ništa
- U glavnom programu definirane dvije varijable (delegata)

```
class Program {
  static void Main(string[] args)
  {
    MathFunction mf = ...
    PrintFunction pf = ...
```



Primjer pridruživanja postupka delegatu (1)

- **Primjer 🗁Razredi \ Delegati \ MathTool.cs**
  - Vlastiti razred MathTool sadrži nekoliko postupaka koji svojim potpisom odgovaraju tipova delegata

```
public class MathTool {
   public static int sum(int x, int y) {
      return x + y;
   }
   public static int diff(int x, int y) {
      return x - y;
   }
   public static void printSquare(int x) {
     Console.WriteLine("x^2 = " + x * x);
   }
   public static void printSquareRoot(int x) {
     Console.WriteLine("sqrt(x) = " + Math.Sqrt(x));
   }
}
```

BY NC SA

Primjer pridruživanja postupka delegatu (2)

- **Primjer 🗁Razredi \ Delegati \ Program.cs**
  - Varijabla mf je delegat koji sadrži reference na postupke koje primaju 2 cijela broja (kao što je npr. postupak sum iz razreda MathTool

```
class Program {
  public delegate int MathFunction(int a, int b);
  public delegate void PrintFunction(int n);
  static void Main(string[] args) {
    int x = 16, y = 2;
    MathFunction mf = MathTool.sum;
    Console.WriteLine("mf({0}, {1}) = {2}",
                                x,   y,     mf(x, y));
    mf = MathTool.diff;
    Console.WriteLine("mf({0}, {1}) = {2}",
                                x,   y,     mf(x, y));
    ...
```

Primjer pridruživanja postupka delegatu (3)

- **Primjer 🗁Razredi \ Delegati \ Program.cs**
  - Delegatu se može pridružiti više postupaka (mogu se naknadno ukloniti)
    - Obično ima smisla za postupke koje ne vraćaju nikakvu vrijednost

```
class Program {
  public delegate int MathFunction(int a, int b);
  public delegate void PrintFunction(int n);
  static void Main(string[] args) {
    int x = 16, y = 2;
    ...
    PrintFunction pf = MathTool.printSquare;
    pf += MathTool.printSquareRoot;
    pf(x);
    pf -= MathTool.printSquare;
    Console.WriteLine();
    pf(y);
```

Primjeri postojećih tipova delegata

- **Func i Action kao dva najpoznatija tipa delegata**
- **`Action<in T1>, Action<in T1, in T2>, Action<in T1, in T2, in T3>, …, Action<in T1,…, in T16>`**
  - Referenca na postupke koji ne vraćaju ništa, a primaju 1, 2, 3, …, ili16 argumenata
  - Argumenti su kontravarijantni (vidi sljedeći slajd)
- **`Func<in T1, out TResult>, Func <in T1, in T2, out TResult>, …, Func<in T1, in T2, … in T16, out TResult>`**
  - Referenca na postupke primaju do 16 argumenata i vraća vrijednost tipa TResult
  - TResult je kovarijantan, a ostali su kontravarijantni (vidi sljedeći slajd)
- **U prethodnim primjerima se umjesto *MathFunction* mogao koristiti *Func<int, int, int>* , a umjesto *PrintFunction Action<int>***

CC BY NC SA

Func i varijantnost

- **Povratni tip je kovarijantan → postupak koji se pridružuje delegatu može vraćati izvedeni tip onog koji je predviđen pri parametrizaciji**
- **Ulazni tipovi su kontravarijantni → postupak koji se pridružuje za ulazne argumente može imati traženi tip ili njemu nadređene.**
- **Npr. Ako je definiran delegat tipa *Func<Automobil, Zaposlenik>* tada se delegatu tog tipa mogu pridružiti reference na postupke**
  - `Zaposlenik Test(Automobil a){ … }`
  - `PoSatu Test(Automobil a){ … }`
  - `Pausalac Test(MotornoVozilo a) { … }`

ali ne i npr.

  - `Djelatnik Test(Automobil a){ … }`
  - `Zaposlenik Test(ElektricniAutomobil a){ … }`

- **Navedeno ima smisla, jer ako je**

  Func<Automobil, Zaposlenik> f = nešto od navedenog

  tada kasnije u programu slijedi *Zaposlenik z = f(neki automobil)* pa je potpuno svejedno je li povratna vrijednost *Zaposlenik* ili nešto izvedeno iz njega, odnosno prima li pridruženi postupak *Automobil* ili nešto općenitije.

BY NC SA

Tehnike obrade pogrešaka

BY NC SA

Defenzivno programiranje

- **Zaštita programa od neispravnog unosa podataka**
  - Defenzivna vožnja automobila temelji se na načelu da vozač nikad ne može biti siguran što će učiniti drugi vozači, pa unaprijed nastoji izbjeći nezgodu za slučaj pogreške drugih vozača
  - U defenzivnom programiranju ideja vodilja je da će potprogram s neispravnim podacima “opstati” i onda kada su pogreškom pozivajuće procedure predani neispravni argumenti
  - Pristup “smeće unutra, smeće van” (“garbage in, garbage out”) treba zamijeniti sa: “smeće unutra, ništa van”, “smeće unutra, poruka o pogrešci van” ili “smeću zabranjen ulaz”

- **Osnovna pravila kojih se treba držati:**
  - Provjeriti ispravnost svih vrijednosti podataka iz vanjskih izvora (datoteka, korisnik, mreža, ...)
  - Provjeriti ispravnost svih vrijednosti ulaznih parametara
  - Odlučiti kako postupiti u slučaju neispravnih podataka

cc BY NC SA

Jednostavan primjer defenzivnog programiranja

- **Nedefenzivno programiranje rekurzivne funkcije faktorijel**

```
int faktorijel( int N )
{
  if ( N == 0 ) return 1;
  else return N * faktorijel( N-1 ) ;
}
```

  - Za negativne vrijednosti nastupa (teoretski) beskonačna rekurzija.

- **Defenzivno programiranje rekurzivne funkcije faktorijel**

```
int faktorijel( int N )
{
  if ( N <= 0 ) return 1;
  else return N * faktorijel( N-1 ) ;
}
```

  - Matematički neprecizno, ali sprječava beskonačnu rekurziju u slučaju preljeva

BY NC SA

Tehnike obrade pogrešaka

- **Tehnike obrade pogrešaka**
  - vratiti neutralnu vrijednost (0, "", NULL)
  - zamijeniti neispravnu vrijednost sljedećom, moguće ispravnom
    - Npr. while (GPSfix != OK) sleep(1/100s) ...
  - vratiti vrijednost vraćenu pri prethodnom pozivu
  - zamijeniti neispravnu vrijednost najbližom ispravnom, npr. min(max(kut, 0),360)
  - zapisati poruku o pogrešci u datoteku, kombinirano s ostalim tehnikama
  - vratiti kôd pogreške – pogrešku obrađuje neki drugi dio koda – dojava drugom dijelu:
    - postavljanjem "globalne" varijable statusa
      - extern int errno; char *sys_errlist[] ;
    - preko imena funkcije (negativna vrijednost, 0)
    - bacanjem iznimki (u nastavku)
  - pozvati "globalnu" metodu za obradu pogreške, npr. perror()
  - bezuvjetni završetak programa, npr. Application.Exit (CancelEventArgs)

- **Robustnost i ispravnost (programa)**
  - robustnost - u slučaju pogreške omogućen je daljnji rad programa, iako to ponekad znači vratiti neispravan rezultat
  - ispravnost - nikad ne vratiti neispravan rezultat, iako to značilo ne vratiti ništa

CC BY NC SA

Iznimke (Exceptions)

- **Iznimka predstavlja problem ili promjenu stanja koja prekida normalan tijek izvođenja naredbi programa**

- **U programskom jeziku C#, iznimka je objekt instanciran iz razreda koji nasljeđuje *System.Exception***

- **System.Exception – osnovni razred za iznimke**
  - StackTrace – sadrži popis poziva postupaka koji su doveli do pogreške
  - Message – sadrži opis pogreške
  - Source – sadrži ime aplikacije koja je odgovorna za pogrešku
  - TargetSite – sadrži naziv postupka koji je proizveo pogrešku

CC BY NC SA

Neke ugrađene iznimke

- Neke sistemske iznimke izvedene iz razreda `SystemException`

`ArrayTypeMismatchException`	tip vrijednosti koji se pohranjuje u polje je različit od tipa polja i implicitna konverzija se ne može obaviti
`DivideByZeroException`	pokušaj dijeljenja s 0
`IndexOutOfRangeException`	indeks polja je izvan deklarirane veličine polja
`InvalidCastException`	nedozvoljena konverzija tipa
`OutOfMemoryException`	nedostatak memorije za alociranje objekta
`OverflowException`	preljev pri izračunavanju aritmetičkog izraza
`NullReferenceException`	referenci nije pridružen objekt
`StackOverflowException`	stog je prepunjen

BY NC SA

Obrada iznimki

- **Obrada iznimki sprječava nepredviđeni prekid izvođenja programa**
- **Iznimka se obrađuju tzv. rukovateljem iznimki (exception handler)**
  - Obrada pogreške sastoji se razdvajanju kôda u blokove *try*, *catch* i *finally*

```
try {
  //dio kôda koji može dovesti do iznimke
}
catch (ExcepType1 exOb){
  //kôd koji se obavlja u slučaju iznimke tipa ExcepType1
}
catch (ExcepType2 exOb) {
  //kôd koji se obavlja u slučaju iznimke tipa ExcepType2
}
...// ostali catch blokovi
finally {
//kôd koji se obavlja nakon izvođenja try, odnosno catch bloka
```

BY NC SA

Postupak obrade iznimki

- **Varijante obrade**
  - Za jedan try blok može postojati jedan ili više catch blokova koji obrađuju različite vrste pogrešaka.
  - Kada dođe do pogreške u try bloku, a postoji više catch blokova, obavlja se onaj catch blok koji obrađuje nastali tip iznimke. Ostali catch blokovi neće se obaviti.
  - Ako postoji više catch blokova, posljednji se navodi blok koji obrađuje općenite iznimke (tipa System.Exception)
- **Primjer: 🗁Kodiranje\TryCatch**

```
try  {
      int x = new Random().Next(2); //0 ili 1
      int y = 10 / x;
      int[] a = { 1, 2, 3 };
      Console.WriteLine(a[3].ToString());
}
catch(DivideByZeroException e) { ... }
catch(IndexOutOfRangeException e) { ... }
finally { ... }
```

Generiranje (bacanje) iznimke naredbom throw

- **Sintaksa**
  - `throw exceptOb;`
    - `exceptOb` – objekt tipa `Exception` ili iz njega izvedenih razreda
    - npr. `throw new Exception("Nova iznimka");`
  - throw;
    - prosljeđivanje iznimke na vanjski blok (rethrow)
- **Primjer, 🗁Kodiranje\ExceptionInfo**
  - bacanje iznimki i prikaz informacije o iznimci

```
 throw new Exception("poruka");
...
 Console.WriteLine( "Izvor: {0}", e.Source );
 Console.WriteLine( "Postupak: {0}", e.TargetSite );
 Console.WriteLine( "Poruka: {0}", e.Message );
 Console.WriteLine( "Trag: {0}", e.StackTrace );
```

- **Primjer: 🗁Kodiranje\Rethrow**
  - prosljeđivanje uhvaćene iznimke

CC BY NC SA

Kreiranje vlastitih iznimki

- **Definiranjem razreda izvedenog iz razreda Exception**
  - Izvorna ideja je bila da su sistemske iznimke izvedene iz SystemException, a aplikacijske iz ApplicationException, ali se odustalo od ApplicationException te se vlastite iznimke izvode direktno iz Exception
- **Primjer: 🗁Kodiranje\CustomException**

```
class Iznimka : Exception
{
  private string val;
  public Iznimka() {}
  public Iznimka(string str) : base(str) {
    val = str;
  }
  public override string Message {
    get {
      return "Nije neparan " + val;
    }
  }
}
```

CC BY NC SA

Preporuke za korištenje iznimki

- Izbjegavati prazne blokove za hvatanje iznimki (“catch { }”)
- Koristiti specifične iznimke (ne samo osnovne iznimke `Exception`) znajući što bacaju vlastite knjižnice
- Zapisivati trag bačenih iznimki (log)
- U poruci iznimke uključiti sve informacije o kontekstu nastanka iznimke
- Razne tipove pogrešaka obrađivati na konzistentan način kroz čitav kod
- Razmotriti izradu centraliziranog sustava za dojavu iznimki u kodu
- Prosljeđivanje iznimki raditi samo kada želimo specijalizirati iznimku
- Koristiti iznimke za obavijest drugim dijelovima programa o pogreškama koje se ne smiju zanemariti
- Bacati iznimke samo u stanjima koja su stvarno iznimna
- Ne bacati iznimke za pogreške koje se mogu obraditi lokalno
- Izbjegavati bacanje iznimki u konstruktorima i finalizatorima, osim ako ih na istom mjestu i ne hvatamo

Primjer centralizirane obrade iznimki

- **Primjer: 🗁Kodiranje \ CentraliziraniSustav**

```
public static void ReportError(Exception iznimka) {
  Console.WriteLine("Neocekivana iznimka:  " + iznimka.Message);
  Console.WriteLine("Mjesto iznimke:");
  Console.WriteLine(iznimka.StackTrace);
  Console.WriteLine("\nPritisni tipku...");
  Console.ReadLine();
}
...
try {
  Exception ex = new Exception("\"Poruka iznimke...\"");
  throw ex;
}
catch (Exception exc)
{
  ReportError(exc);
}
```

BY NC SA

Unutarnje iznimke

- **`Exception.InnerException` – dobavlja instancu razreda *Exception* koja je izazvala aktualnu iznimku**
- **Primjer: 🗁Kodiranje\InnerException**

```
  class Primjer
  {
    public void F()
...
      throw new Exception
                ("Iznimka u Primjer.F() :", e);
…
  class Program
  {
    static void Main(string[] args){
      try{ new Primjer().F(); }
      catch (Exception e)
      {
        Console.WriteLine("Iznimka u Main: "
          + "{0}\nInnerException: {1}",
             e.Message, e.InnerException.Message);
```

Ostale mogućnosti obrade iznimki

- **Eksplicitno postavljanje provjere preljeva za operacije s cijelim brojevima (integral-type arithmetic operations and conversions)**
  - checked context – aritmetički preljev podiže iznimku
  - unchecked context – aritmetički preljev bude ignoriran, a rezultat "odrezan" (truncated)
  - Ako nije ništa navedeno, pretpostavlja se da je *unchecked*

- **Primjer: 🗁Kodiranje\CheckedUnchecked**

```
int deset = 10;

// bez provjere
i = unchecked (2147483647 + deset); // negativno

// s provjerom
i = checked (2147483647 + deset); // iznimka
```

BY NC SA

Tvrdnje (Assert)

- Naredba kojom se program testira tako da određeni izraz mora biti istinit, a inače se izvršavanje programa zaustavlja
- Tvrdnje se koriste za uklanjanje pogrešaka (debugging) i dokumentiranje ispravnog rada programa
- Koriste se u fazi kodiranja, naročito u razvoju velikih, kompliciranih programa te u razvoju programa od kojih se zahtijeva visoka pouzdanost
- Pišemo ih na mjestima gdje se pogreške ne očekuju (tj. ne smiju se pojaviti)

Tvrdnje u C#-u

- **Prostor imena System.Diagnostics, naredba Debug.Assert**
  - Logički izraz za koji se pretpostavlja (tvrdi) da je istinit i poruka koja se ispisuje ako izraz nije istinit
  - U .NET Core-u neistinit uvjet uzrokuje trenutno završavanje programa (bez izvođenja eventualnog finally bloka
  - Automatski se uklanjaju iz *Release* verzije
- **Primjer: 🗁Kodiranje \ Barikade \ Predmet.cs**

```
public double ProsjecnaOcjena() {
  int sum = 0;
  foreach(var par in ocjene) {
    int ocjena = par.Value;
    Debug.Assert(ocjena >= 1 && ocjena <= 5,
       $"Pohranjena neispravna ocjena ${par.Key}={par.Value}");
    sum += ocjena;
  }
  return ocjene.Count > 0 ? (double) sum / ocjene.Count : 0;
}
```

CC BY NC SA

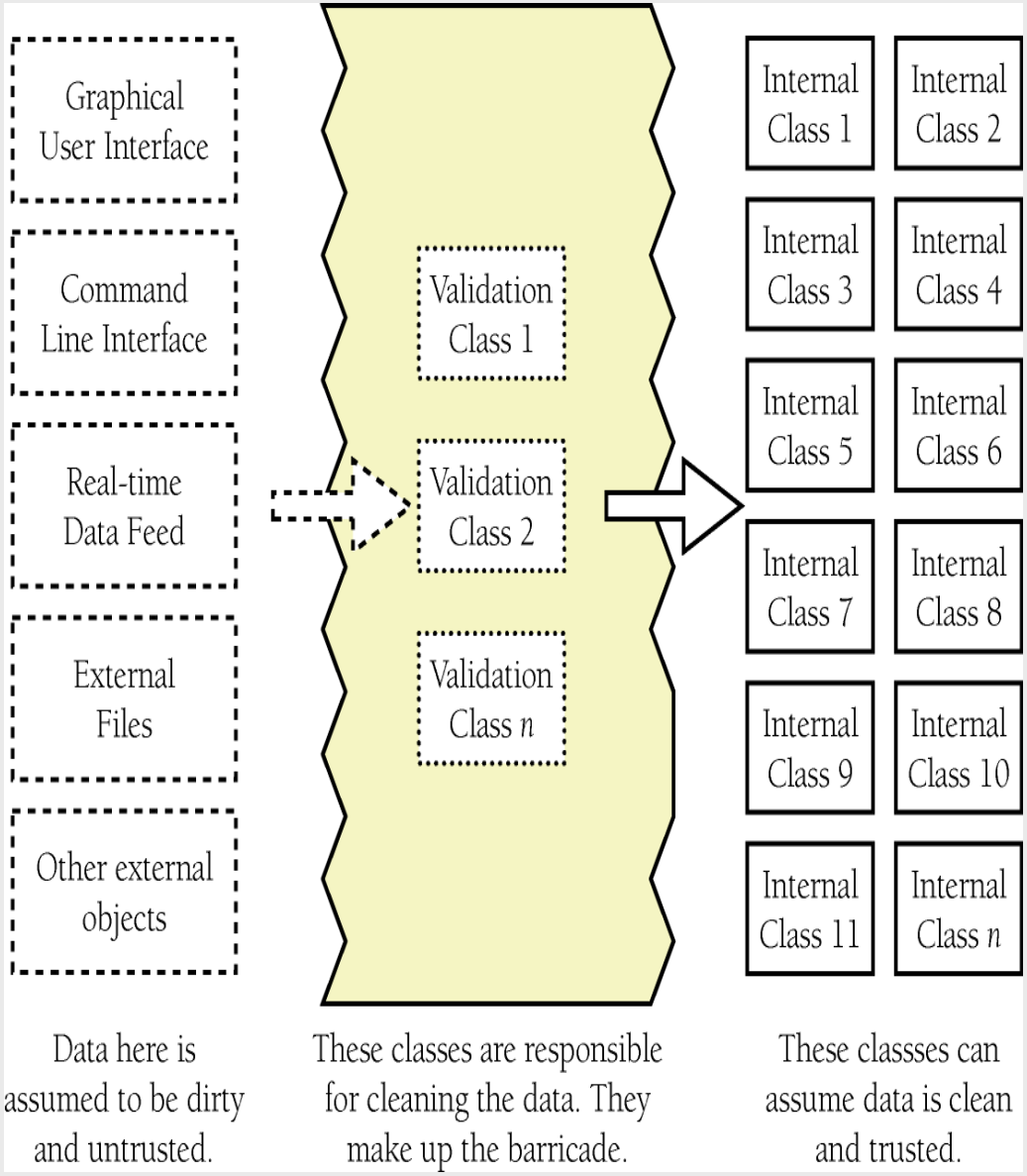
Preporuke za korištenje tvrdnji

- **Obradu pogreške (iznimke) pisati tamo gdje očekujemo pogreške**
  - Iznimke su informacije o pogrešnom uvjetu ili neočekivanom događaju
  - Mogu se uhvatiti
- **Koristiti tvrdnje tamo gdje nikad ne očekujemo pogreške**
  - Predstavljaju fatalne pogreške, ne mogu se uhvatiti i ukazuju na logičku pogrešku u programskom kodu
- **Za jako robustan kod koristiti tvrdnje + kod za obradu pogreške**
- **Koristiti tvrdnje**
  - za dokumentiranje i verificiranje uvjeta koji moraju vrijediti prije pozivanja metode ili instanciranja razreda (“preconditions”), te
  - uvjeta koji moraju vrijediti poslije djelovanja metode ili rada s razredom ("postconditions”).
- **Izbjegavati poziv metoda u izrazima tvrdnji**
  - npr. `Debug.Assert (Obavi(), "Neobavljeno");`
  - u produkciji (s isključenim tvrdnjama) *Obavi* se neće nikada izvršiti

BY NC SA

Koncept barikada

- Konstrukcija sučelja kao granica prema "sigurnim" dijelovima koda
- Definiranje dijelova softvera koji će rukovati "prljavim" (nesigurnim) podacima i drugih koji rukuju samo s "čistim" podacima
- Validacijski razredi koji su odgovorni za provjeru ispravnosti podataka sačinjavaju barikadu prema internim razredima koji rukuju s podacima za koje se pretpostavlja da su provjereni i ispravni



Koncept barikada

- **Primjer: 🗁Kodiranje \ Barikade \ Predmet.cs**

```
class Predmet {
  //privatna kolekcija iza barikade
  Dictionary<string, int> ocjene = new Dictionary<string, int>();
  public int this[string ime] {
    get {
      //što ako netko traži studenta koji nije evidentiran?
      bool postoji = ocjene.TryGetValue(ime, out int ocjena);
      return postoji ? ocjena : -1;
    }
    set {
      if (value < 1 || value > 5)
        throw new ArgumentOutOfRangeException(
              $"Neispravna ocjena {value}.
                                  Ocjena mora biti od 1 do 5");
        ocjene[ime] = value;
    ...
```

Preporuke za korištenje barikada

- **Barikade naglašavaju razliku između tvrdnji i obrade iznimaka**
  - Metode s vanjske strane barikade trebaju koristiti kôd za obradu pogreške
  - Unutarnje metode mogu koristiti tvrdnje jer se ovdje pogreške ne očekuju!

- **Na razini razreda**
  - javne metode rukuju s „prljavim” podatcima i „čiste” ih
  - privatne metode rukuju samo s “čistim” podatcima.
  - pojava „prljavog“ podatka u privatnoj metodi nije iznimka koja se očekuje, već neispravnost tvrdnje koja ukazuje na pogrešku u kôdiranju

- **Pretvarati podatke u ispravan tip odmah pri unosu**

Otkrivanje pogrešaka

- **Uobičajena zabluda programera je da se ograničenja koja se odnose na konačnu verziju softvera odnose i na razvojnu verziju**
  - Treba biti spreman žrtvovati brzinu i resurse tokom razvoja u zamjenu za olakšani razvoj
- **Ofenzivno programiranje – učiniti pogreške u fazi razvoja toliko očitim i bolnim da ih je nemoguće zanemariti**
  - osigurati da `assert` naredbe uzrokuju prekid izvođenja pri pogrešci
  - popuniti bilo koju alociranu memoriju prije upotrebe radi detektiranja eventualnih problema s njenom alokacijom
  - popuniti alocirane datoteke ili tokove podataka prije upotrebe radi detektiranja eventualnih grešaka u formatu datoteka ili podataka
  - osigurati da svaka case naredba koja propagira do `default` slučaja uzrokuje pogrešku koju nije moguće zanemariti
  - napuniti objekt “smećem” (junk data) neposredno prije njegovog brisanja

Otkrivanje pogrešaka (2)

- **Planirati uklanjanje dijelova programa koji služe kao pomoć u otkrivanju pogrešaka u konačnoj verziji softvera**
  - koristiti alate za upravljanje verzijama
  - koristiti ugrađene predprocesore za uključivanje/isključivanje dijelova koda u pojedinoj verziji
  - korištenje vlastitog (samostalno napisanog) predprocesora
  - zamjena metoda za otkrivanje pogrešaka u konačnoj verziji "praznim" metodama koje samo vraćaju kontrolu pozivatelju
- **Primjer: 🗁Kodiranje\Conditional**

```
#define RAZVOJ //definiramo simbol
...
#if (RAZVOJ)
  // kod za debugiranje
  Console.WriteLine("Poruka UNUTAR koda za debugiranje!");
#endif
  ...
  Console.WriteLine("Poruka IZVAN koda za debugiranje!");
```

CC BY NC SA

Otkrivanje pogrešaka (3)

- **Umjesto #if i #endif koristiti *Conditional* (iz *System.Diagnostics*)**
- **Kod za testiranje odvojiti u posebni postupak i iznad postupka navesti atribut *Conditional***
  - U slučaju da simbol nije definiran, u kompiliranoj verziji nije uključen poziv označenog postupka
  - Simbol se može definirati u kodu, ali i kao parametar prilikom kompiliranja
    - Properties → Build → Conditional compilation symbols
- **Primjer: 🗁Kodiranje\Debug**

```
#define RAZVOJ //definiramo simbol
...
Test();
...
[Conditional("RAZVOJ")]
static void Test()
{
Console.WriteLine("Poruka iz postupka Test");
}
```

CC BY NC SA

Količina defenzivnog koda u završnoj verziji

- **Ostaviti kôd koji radi provjere na opasne pogreške**
- **Ukloniti kôd koji provjerava pogreške s trivijalnim posljedicama**
  - Ukloniti pretprocesorskim naredbama, a ne fizički
- **Ukloniti kôd koji može uzrokovati pad programa**
  - U konačnoj verziji treba omogućiti korisnicima da sačuvaju svoj rad prije nego se program sruši.
- **Ostaviti kôd koji u slučaju pogreške omogućava "elegantno" rušenje programa**
- **Ostaviti kôd koji zapisuje pogreške koje se događaju pri izvođenju**
  - Zapisivati poruke o pogreškama u datoteku.
- **Treba biti siguran da su sve poruke o pogreškama koje softver dojavljuje "prijateljske"**
  - Obavijestiti korisnika o "unutarnjoj pogrešci" i navesti e-mail ili broj telefona tako da korisnik ima mogućnost prijaviti pogrešku

Zadaci za vježbu

- **Obraditi iznimku kada se ostvari Debug.Assert**

- **Implementirati razred Niz koji osim niza cijelih brojeva sadržava gornju i donju granicu niza.**
  - Baciti iznimku ako je gornja granica manja od donje, te ako se preko indeksera pristupa članu izvan gornje odnosno donje granice.
  - Napisati vlastiti razred za obradu navedenih iznimki.

- **Implementirati razred Temperature koji prima razred Niz. Razred treba imati metodu za računanje prosječne temperature.**
  - Ubaciti provjere da niz temperatura nije prazan prije izračuna.
  - Provjeriti i da je dobiveni prosjek unutar granica definirane gornje i donje vrijednosti niza.

Životni vijek objekta

- **Primjer: 🗁Kodiranje\Using**
- **Instancirani objekt postoji dok ga sakupljač smeća (GarbageCollector) ne uništi**
  - GC će ga obrisati ako na njega ne pokazuje niti jedna referenca*
- **Što ako ne možemo čekati GC?**
  - Implementirati sučelje IDisposable (postupak Dispose)

```
public class Razred : IDisposable {
   public void Dispose() {
     //zatvaranje datoteke, konekcije
     // i sličnih "dragocjenih" resursa
     ...
   }
}
```

  - Ako neki razred implementira IDisposable, preporuka je da se za objekte tog razreda Dispose uvijek pozove nakon što objekt više ne bude potreban.

CC BY NC SA

IDisposable, *using blok* i iznimke

- **Dispose se može pozvati eksplicitno**
- **Što ako se dogodi iznimka prije poziva postupka Dispose?**
  - Koristiti tzv. *using blokove*
  - Za objekt stvoren unutar *using bloka*, Dispose se automatski poziva nakon napuštanja bloka (bez obzira na razlog izlaska iz bloka)

```
Razred r1 = new Razred("A1");
using (Razred r2 = new Razred("B2")) {
   Razred r3 = new Razred("C3");
   throw new ApplicationException("Poruka");
}
r1.Dispose();
```

  - Using blok se može koristiti samo za one razrede koji implementiraju IDisposable

CC BY NC SA

Materijali i reference

- **Samostalno pogledati**
  - materijale u RPPP03-smjernice.pdf
- **Objects, Classes, and Structs (C# Programming Guide)**
  - http://msdn2.microsoft.com/en-us/library/ms173109.aspx
- **Generics (C# Programming Guide)**
  - http://msdn.microsoft.com/en-us/library/0x6a29h6.aspx
- **Kovarijantnost i kontravarijantnost**
  - https://msdn.microsoft.com/en-us/library/mt654055.aspx
  - https://msdn.microsoft.com/en-us/library/dd799517.aspx
  - http://tomasp.net/blog/variance-explained.aspx/

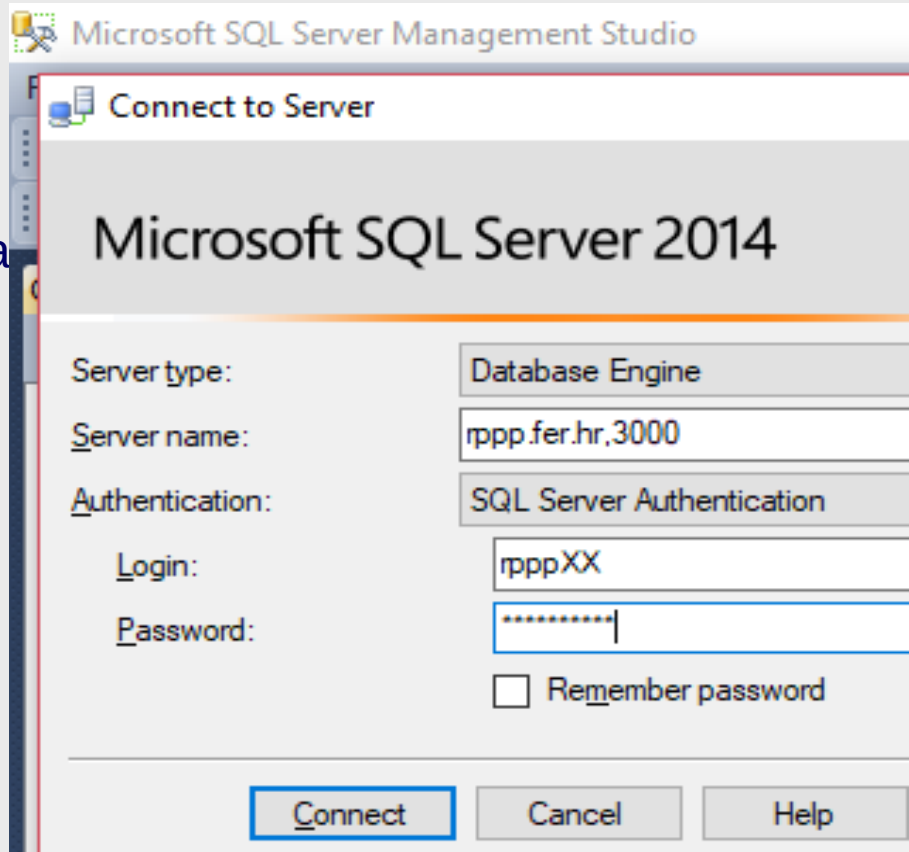
Rad s bazom podataka

2019/20.04

Zaštićeno Creative Commons licencom, http://creativecommons.org

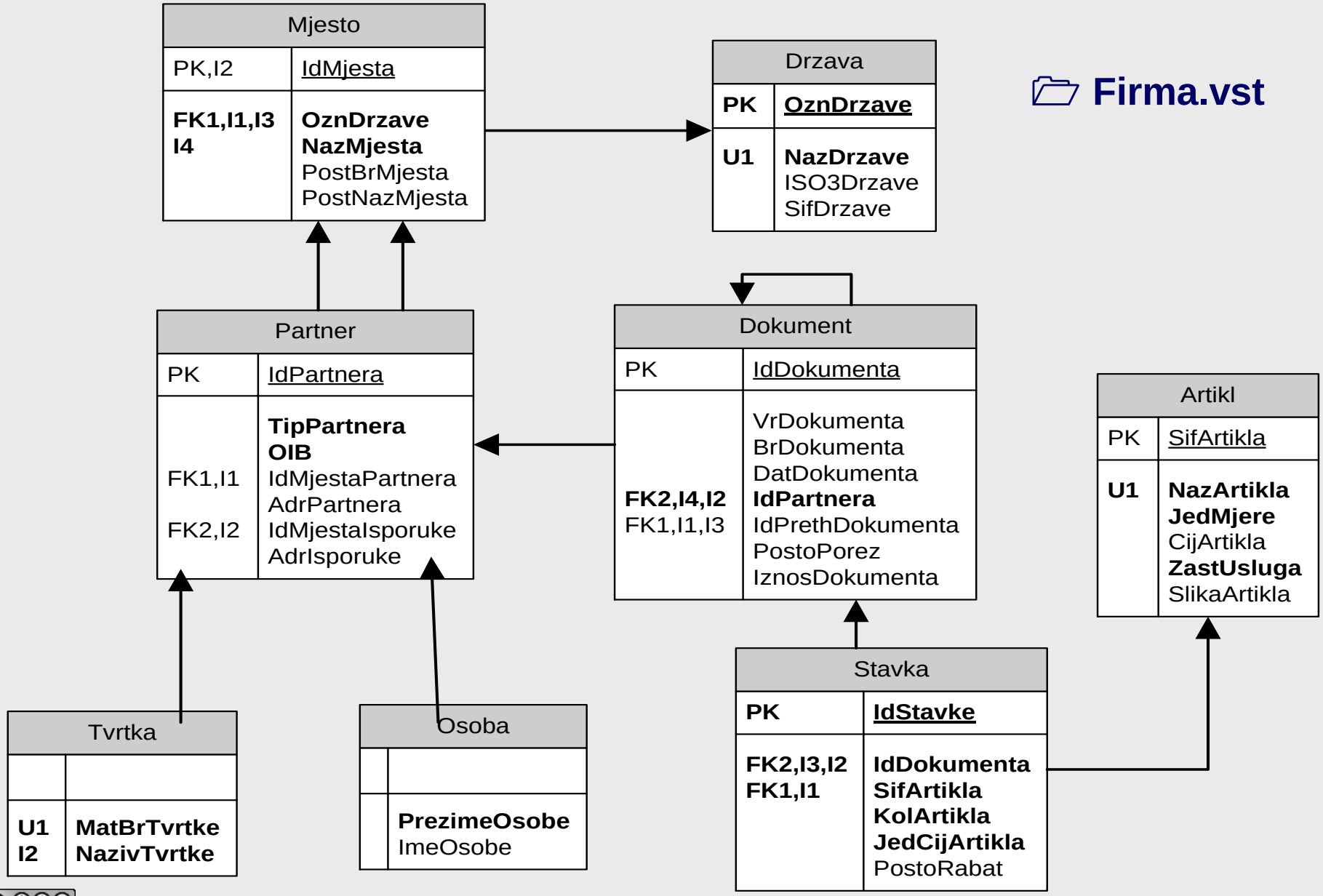
Ogledna baza podataka

- **SQL Server: rppp.fer.hr,3000**
- **Ogledna baza podataka: Firma**
  - SQL Server Authentication: rppp/zaporka se nalazi u primjerima na TFS-u
  - Moguće mijenjati podatke u svrhu testiranja
- **Datoteka BazePodataka.zip na TFS-u sadrži**
  - Firma.bak
    - backup baze podataka Firma
  - Firma.vst
    - MS Visio dijagram baze podataka
- **Baze podataka oblika RPPPxx:**
  - Baza podataka za grupu XX
  - Korisničko ime rpppXX
    - zaporka predana na labosima



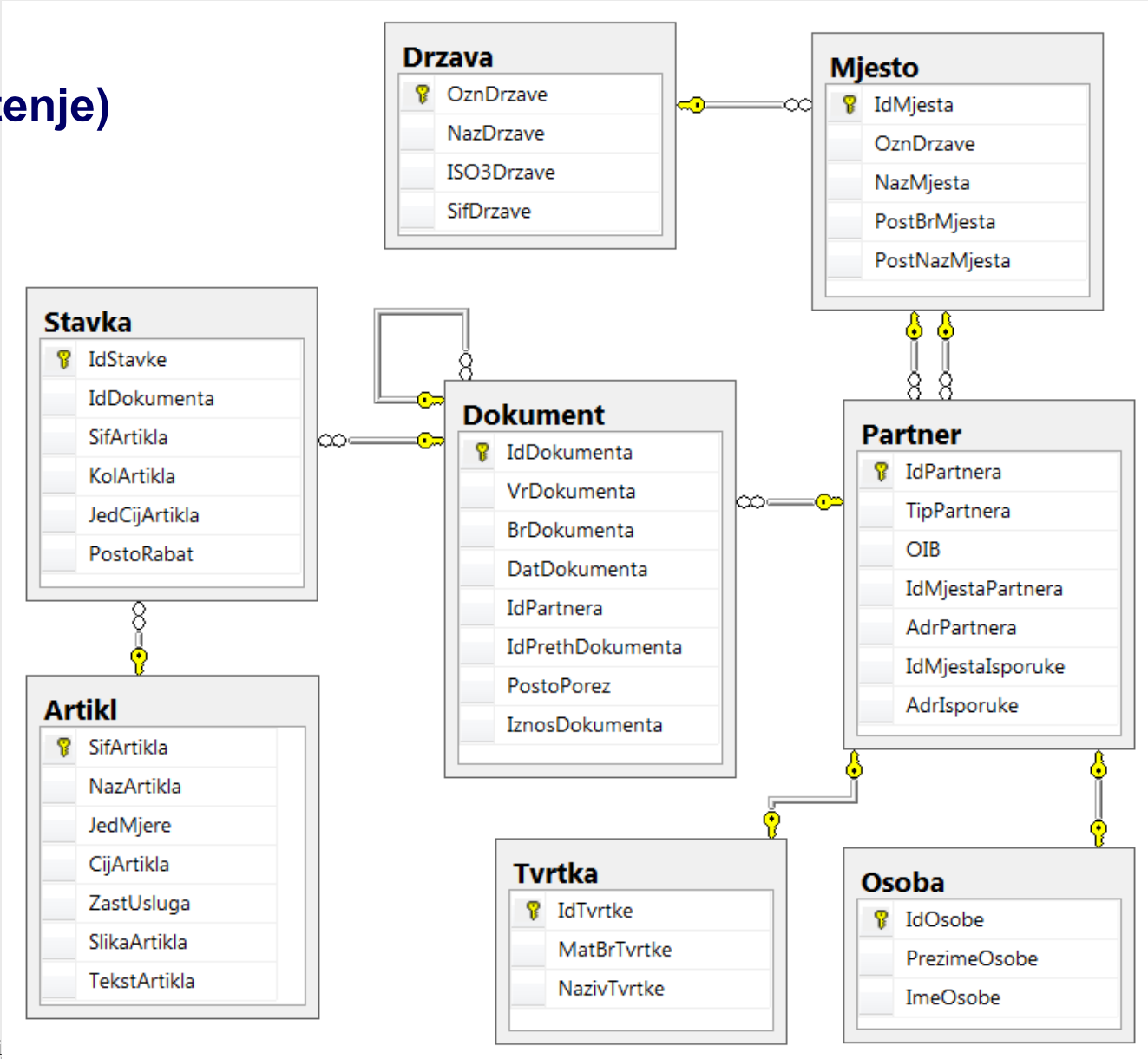
CC BY NC SA

Ogledna baza podataka (MS Visio)



Ogledna baza podataka

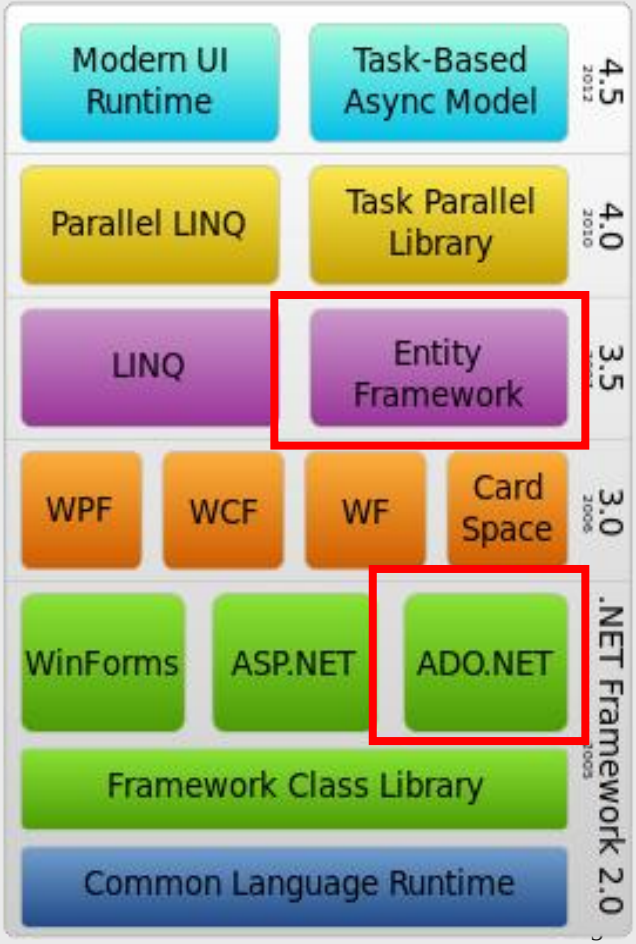
🗁 **Firma.bak**
(za lokalno korištenje)



CC BY NC SA

.NET Framework i ADO.NET

- **ActiveX Data Objects .NET (ADO.NET) je tehnologija za rukovanje podacima unutar .NET Frameworka**
  - Omogućuje pristup bazama podataka, ali i drugim spremištima podataka, za koje postoji odgovarajući opskrbljivač podacima (provider)
    - sinonimi za opskrbljivač: davatelj, pružatelj, poslužitelj
- **Podrška različitim tipovima spremišta**
  - Strukturirani, nehijerarhijski podaci
    - Comma Separated Value (CSV) datoteke,
    - Microsoft Excel proračunske tablice, …
  - Hijerarhijski podaci (npr. XML dokumenti)
  - Relacijske baze podataka
    - SQL Server, Oracle, MS Access, …
- **Entity Framework za objektno-relacijsko preslikavanje**
  - Izvorno dio .NET-a, kasnije *Open Source* paket
  - U .NET Coreu razdvojeno u manje pakete



BY NC SA

Opskrbljivači (davatelji) podataka

- **Davatelji za različite tehnologije (SQL Server, PostgreSQL, SQLite, MongoDB, …)**
  - https://blogs.msdn.microsoft.com/dotnet/2016/11/09/net-core-data-access
  - direktni pristup ili tehnologije s određenom razinom apstrakcije (npr. ORM alati)

- **System.Data.SqlClient**
  - optimiran za rad s MS SQL Server-om
  - razredi: *SqlCommand*, *SqlConnection*, *SqlDataReader*, *SqlDataAdapter*
- **Za ostale relacijske baze podataka razredi sličnih naziva**
  - npr. *NpgsqlConnection*, *NpgsqlCommand*, *SqliteConnection*, …

- **Navedeni razredi implementiraju zajednička sučelja pa imaju članove jednakih naziva**
  - neovisnost aplikacije o fizičkom smještaju podataka

CC BY NC SA

Osnovni pojmovi u pristupu bazi podataka

- **`Connection`**
  - Priključak (veza) s izvorom podataka
- **`Command`**
  - naredba nad izvorom podataka
  - izvršava se nad nekim otvorenim priključkom
- **`DataReader`**
  - Rezultat upita nad podacima (forward-only, read-only connected result set)
- **`ParameterCollection`**
  - Parametri `Command` objekta
- **`Parameter`**
  - Parametar parametrizirane SQL naredbe ili pohranjene procedure
- **`Transaction`**
  - Nedjeljiva grupa naredbi nad podacima

BY NC SA

Priključak na bazu podataka

- **Priključak, veza (Connection)**
  - otvara i zatvara vezu s fizičkim izvorom podataka
  - omogućuje transakcije i izvršavanje upita nad bazom podataka
- **Sučelje *System.Data.IDbConnection* i apstraktni razred *System.Data.DbConnection***
- **Implementacije: *NpgsqlConnection*, *SqlConnection*, …**
- **Važnija svojstva**
  - *ConnectionString* – string koji se sastoji od parova postavki oblika naziv=vrijednost odvojenih točka-zarezom
  - *State* – oznaka stanja priključka (enumeracija *ConnectionState*)
    - *Broken, Closed, Connecting, Executing, Fetching, Open*
- **Važniji postupci**
  - *Open* – prikapčanje na izvor podataka
  - *Close* - otkapčanje s izvora podataka

BY NC SA

Primjeri postavki priključka na bazu

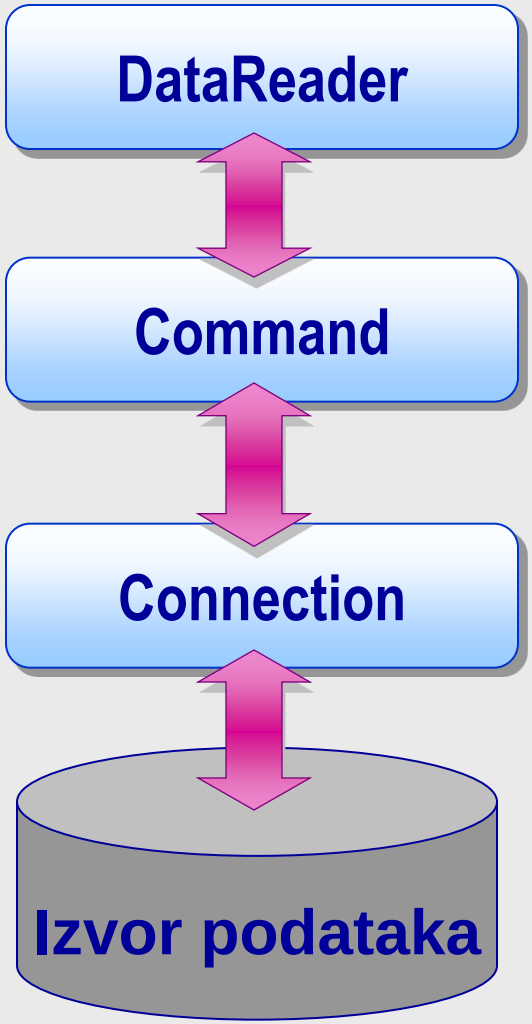
- **Microsoft SQL Server**
  - `Data Source=.;Initial Catalog=Firma;Integrated Security=True`
  - `Data Source=rppp.fer.hr,3000;Initial Catalog=Firma;User Id=rppp;Password=šifra`
- **PostgreSQL**
  - `User ID=rppp;Password=**;Host=localhost;Port=5432;Database=firma;Pooling=true;`
- **Više primjera na https://www.connectionstrings.com**

BY NC SA

Izravna obrada podataka

CC BY NC SA

Izravna obrada podataka na poslužitelju

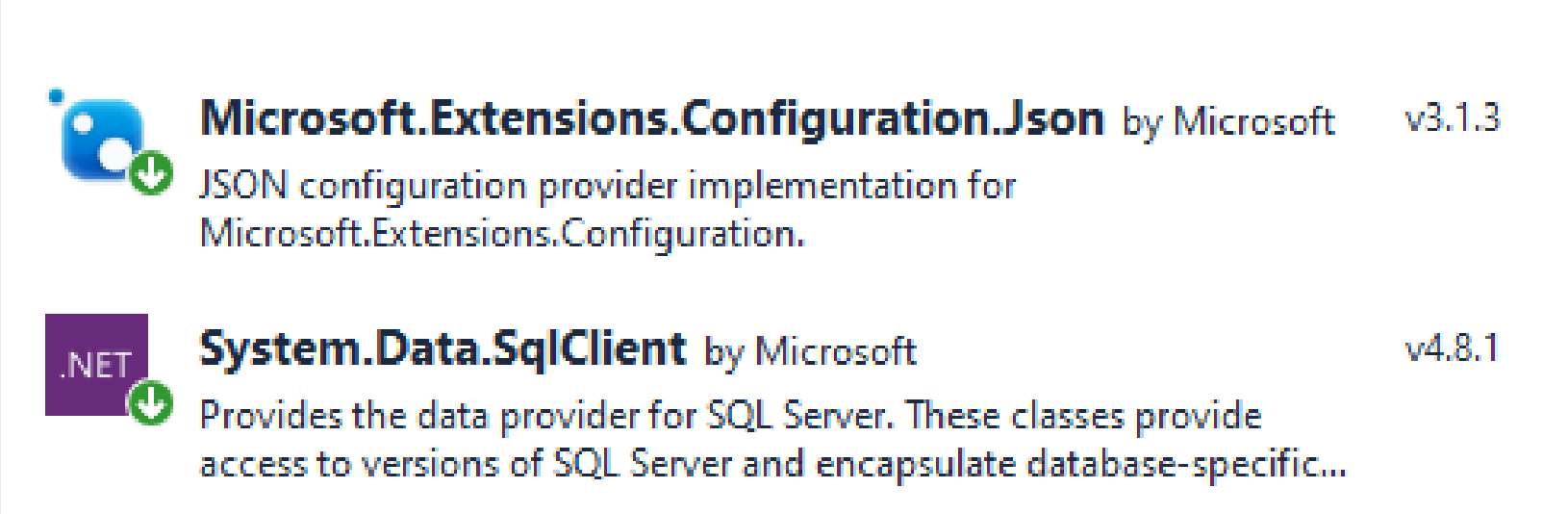


1. Otvori priključak
2. Izvrši naredbu
3. Obradi podatke u čitaču
4. Zatvori čitač
5. Zatvori priključak

- Za vrijeme obrade (čitanja) podataka priključak na izvor podataka je otvoren!

Dodavanje NuGet paketa

- **U primjeru koji slijedi bit će potrebno uključiti dodatne biblioteke**
  - ručnim ažuriranjem csproj datoteke ili odabirom opcije Manage NuGet Packages
- **U primjeru koji slijedi ti paketi su**
  - Microsoft.Extensions.Configuration.Json
  - System.Data.SqlClient



CC BY NC SA

Skica rješenja izravne obrade podataka

- **Primjer: 🗁 DataAccess / DataReader / Program.cs : PopisArtikala**

```
string connString = ... ;
IDbConnection conn = new SqlConnection(connString);
IDbCommand command = new SqlCommand();
command.CommandText = "SELECT TOP 3 * FROM Artikl";
command.Connection = conn;
conn.Open();
IDataReader reader = command.ExecuteReader();
while (reader.Read())
{
   object NazivArtikla = reader["NazArtikla"];
   ...
}
reader.Close();
conn.Close();
```

BY NC SA

Neovisnost o konkretnoj implementaciji

- **Primjeri se mogu poopćiti na način da se za tip reference umjesto konkretnih implementacija koriste sučelja illi apstraktni razredi**
  - *IDbConnection* ili *DbConnection*
  - *IDbCommand* ili *DbCommand*
  - *IDataReader* ili *DbDataReader*

- **Alternativno definirati reference s ključnom riječi *var*.**

- ***DBProviderFactory kao „tvornica“***
  - Omogućava stvaranje priključaka i naredbi bez navođenja konkretnih implementacija
  - Postupci *CreateConnection*, *CreateCommand, …* kao rezultat vraćaju instance konkretnih implementacija, ali promatrane kroz odgovarajuće apstraktne razrede
  - Primjer slijedi uskoro

Sučelje *IDbCommand*

- **Reprezentira SQL naredbe koje se obavljaju nad izvorom podataka**
  - upit može biti SQL naredba ili pohranjena procedura

- **Važnija svojstva**
  - `Connection`: priključak na izvor podataka
  - `CommandText`: SQL naredba, ime pohranjene procedure ili ime tablice
  - `CommandType`: tumačenje teksta naredbe, standardno `Text`
    - `enum CommandType { Text, StoredProcedure, TableDirect }`

- **Važniji postupci**
  - `ExecuteReader` – izvršava naredbu i vraća `DataReader`
  - `ExecuteNonQuery` – izvršava naredbu koja vraća broj obrađenih zapisa, npr. neka od naredbi `UPDATE, DELETE ili INSERT`.
  - `ExecuteScalar` – izvršava naredbu koja vraća jednu vrijednost, npr. rezultat agregatne funkcije

BY NC SA

Sučelje *IDataReader*

- **Sučelje za iteriranje nad rezultatom upita.**
- **Važnija svojstva**
  - `Item` – vrijednost stupca u izvornom obliku
    - `public virtual object this[int] {get;}`
    - `public virtual object this[string] {get;}`
  - `FieldCount` - broj stupaca u rezultatu upita
- **Važniji postupci**
  - `Read` – prelazi na sljedeći redak rezultata i vraća true ako takav postoji
  - `Close` – zatvara `DataReader` objekt (ne nužno i priključak s kojeg čita)
  - `GetName` – vraća naziv za zadani redni broj stupca
  - `GetOrdinal` – vraća redni broj za zadano ime stupca
  - `GetValue` – dohvaća vrijednost zadanog stupca za aktualni redak

    ```
    public virtual object GetValue( int ordinal );
    ```
  - `GetValues` – dohvaća aktualni redak i sprema u polje objekata

    ```
    public virtual int GetValues( object[] values );
    ```
  - `GetXX (GetInt32, GetChar, …)` – dohvaća vrijednost zadanog stupca pretpostavljajući određeni tip

CC BY NC SA

Postavke priključka na bazu podataka

- **Izbjegavati pisanje postavki priključka unutar koda**
  - Promjena postavki zahtijeva ponovnu izgradnju programa (nova verzija?)
  - potencijalni sigurnosni problem (npr. ispis u tragu stoga kod iznimke)
- **Postavke staviti u konfiguracijsku datoteku**
- **.NET Framework (app.config ili web.config)**

```
<configuration>
  <connectionStrings>
     <add name="Firma" connectionString="Data
       Source=rppp.fer.hr,3000;Initial Catalog=Firma;
       User Id=rppp;Password=šifra;"/>
  </connectionStrings>
</configuration>
```

  - Dohvatljivo iz koda pomoću razreda *ConfigurationManager*

```
string connString =
ConfigurationManager.ConnectionStrings["Firma"].ConnectionString;
```

BY NC SA

Konfiguracijska datoteka za .NET Core

- **Datoteka u JSON formatu proizvoljnog imena**
- **Primjer: 🗁DataAcess / DataReader / appsettings.json**
  - Desni klik → Copy To Output Directory : Copy if newer

```
{
  "ConnectionStrings": {
    "Firma": "Data Source=rppp.fer.hr,3000;Initial
Catalog=Firma;User Id=rppp;Password=sifra"
  }
}
```

- **Dohvatljivo iz koda pomoću razreda *ConfigurationBuilder***
  - Potrebno uključiti paket Microsoft.Extensions.Configuration.Json
  - Primjer: 🗁DataAcess / DataReader / Program.cs

```
var config = new ConfigurationBuilder()
              .SetBasePath(Directory.GetCurrentDirectory())
              .AddJsonFile("appsettings.json")
              .Build();
string connString = config["ConnectionStrings:Firma"];
```

BY NC SA

Zatvaranje priključka

- **Svaku otvorenu vezu prema bazi podataka treba zatvoriti!**
- **Što ako se dogodi iznimka u prethodnom primjeru?**
  - `Conn.Close()` nije izvršen – veza ostaje otvorena i ne može se ponovo iskoristiti
  - Staviti `Conn.Close()` unutar *finally* blocka
    - 🗁DataAccess \ DataReader \ Programs.cs - PopisArtikalaTryCatch
  - Priključak implementira sučelje *IDisposable* → *Dispose* je (u ovom slučaju) ekvivalentan *Close* → koristiti *using*
    - 🗁 DataAccess \ DataReader \ Programs.cs - PopisArtikalaUsing

```
using (var conn = new SqlConnection(connString)){
  using (var command = conn.CreateCommand()){
   ...
    using (var reader = command.ExecuteReader()){
      ...
```

- **Napomena: rješenje s *using* nije uvijek moguće – npr. ako će povezivanje/čitanje podataka naknadno nastupiti**
  - Automatsko zatvaranje priključka kad se zatvori čitač

```
command.ExecuteReader(System.Data.CommandBehavior.CloseConnection)
```

CC BY NC SA

DbProviderFactory / DbProviderFactories

- **.NET Framework:**
  - Statički postupak GetFactory u razredu DbProviderFactories
- **.NET Core:**
  - [*]ClientFactory.Instance
    - Ili prethodno registrirati s
      DbProviderFactories.RegisterFactory(
      "System.Data.SqlClient", SqlClientFactory.Instance);
- **Primjer: 🗁 DataAccess / DataReaderParamProc / Program.cs**

```
DbProviderFactory factory = SqlClientFactory.Instance;

using (DbConnection conn = factory.CreateConnection())
{
   conn.ConnectionString = ...
   using (DbCommand command = factory.CreateCommand()){
       command.Connection = conn;
       ...
```

Parametrizirani upiti

- **Dijelovi upita s parametrima oblika @NazivParametra**
  - Olakšava pisanje upita
  - Brže izvršavanje u slučaju višestrukih izvršavanja
  - Zaštita od SQL injection napada
  - Parametar se kreira s *new [Sql]Parameter* ili pozivom postupka CreateParameter na nekoj naredbi
- **Primjer: 🗁DataAccess \ DataReaderParamProc \ Program.cs – ParamUpiti**

```
command.CommandText = "SELECT TOP 3 * FROM Artikl WHERE JedMjere =
        @JedMjere ORDER BY CijArtikla DESC;" +
        "SELECT TOP 3 * FROM Artikl WHERE JedMjere = @JedMjere AND
        CijArtikla > @Cijena ORDER BY CijArtikla";
DbParameter param = command.CreateParameter();
param.ParameterName = "JedMjere"; param.DbType = DbType.String;
param.Value = "kom"; command.Parameters.Add(param);

param = command.CreateParameter();
param.ParameterName = "Cijena"; param.DbType = DbType.Decimal;
param.Value = 100m; command.Parameters.Add(param);
```

Svojstva parametra

- ***DbType*** **– vrijednost iz enumeracije *System.Data.DbType***
  - Predstavlja tip podatka koji se prenosi parametrom.
- ***Direction*** **– vrijednost iz enumeracije *System.Data.ParameterDirection***
  - Određuje da li je parametar ulazni, izlazni, ulazno-izlazni ili rezultat poziva pohranjene procedure. Ako se ne navede, pretpostavlja se da je ulazni.
- ***IsNullable*** **– Određuje može li parametar imati null vrijednost**
- ***ParameterName*** **– Naziv parametra**
- ***Size*** **– Maksimalna veličina parametra u bajtovima**
  - Upotrebljava se kod prijenosa tekstualnih podataka.
- ***Value*** **– Vrijednost parametra**
  - Vrijednost izlaznog argumenta se može dobiti i preko instance naredbe `command.Parameters["Naziv parametra"].Value`

BY NC SA

Upit s više skupova rezultata

- U slučaju da rezultat upita vraća više skupova rezultata, svaki sljedeći dohvaća se postupkom *NextResult* na čitaču podataka

- Primjer: 🗁DataAccess \ DataReaderParamProc \ Program.cs – ParamUpiti

```
command.CommandText = "SELECT TOP 3 * FROM Artikl WHERE JedMjere =
        @JedMjere ORDER BY CijArtikla DESC;" +
        "SELECT TOP 3 * FROM Artikl WHERE JedMjere = @JedMjere AND
        CijArtikla > @Cijena ORDER BY CijArtikla";
...
using (DbDataReader reader = command.ExecuteReader()){
    do{
        while (reader.Read()){
                ...
        }
    }
    while (reader.NextResult());
}
```

Pozivi pohranjenih procedura

- **Primjer: 🗁DataAccess \ DataReaderParamProc \ Program.cs – Procedura**
  - Parametri procedure navode se kao i kod parametriziranih upita
  - Svojstvo *CommandType* na naredbi potrebno je postaviti na *System.Data.CommandType.StoredProcedure*
  - Ako procedura ne vraća skup podataka, koristi se postupak *ExecuteNonQuery*
    - Očekuje li se skup podataka kao rezultat koristi se *ExecuteReader*.
    - Vrijednosti izlaznih parametara mogu se dobiti <u>tek po zatvaranju čitača</u>

```
command.CommandText = "ap_ArtikliSkupljiOd";
command.CommandType = System.Data.CommandType.StoredProcedure;
...
param = command.CreateParameter();
param.ParameterName = "BrojJeftinijih"; param.DbType = DbType.Int32;
param.Direction = System.Data.ParameterDirection.Output;
command.Parameters.Add(param);
...
using (DbDataReader reader = command.ExecuteReader()){ ... }
int brJef = command.Parameters["BrojJeftinijih"].Value
```

CC BY NC SA

Zaštita postavki za spajanje na bazu

- **Što ako prilikom razvoja treba javno dijeliti izvorni kod?**
- **Mehanizam „korisničkih tajni” (engl. user secrets)**
- **U projektnim datotekama stoji ključ (<userSecretsId>), a vrijednost spremljena u korisnikovom profilu**
  - Windows:
    %APPDATA%\microsoft\UserSecrets\<userSecretsId>\secrets.json
  - Linux:
    ~/.microsoft/usersecrets/<userSecretsId>/secrets.json
  - Mac:
    ~/.microsoft/usersecrets/<userSecretsId>/secrets.json

- **Samo za razvoj! Datoteke nisu kriptirane**

CC BY NC SA

Postavljanje datoteke s „tajnim” vrijednostima

- **Izmijeniti projektnu datoteku i dodati odgovarajuće pakete**
  - Primjer: 🗁 DataAccess / DataReaderSecret / DataReader.csproj

```
<Project Sdk="Microsoft.NET.Sdk"> ...
  <PropertyGroup>
   <UserSecretsId>Firma</UserSecretsId>
  </PropertyGroup>
  <ItemGroup>
     <PackageReference
Include="Microsoft.Extensions.Configuration.UserSecrets"
Version="2.2.0" />
  ...
  </ItemGroup>
```

- **U naredbenom retku u mapi projekta pokrenuti**
  `dotnet user-secrets set FirmaSqlPassword šifra`
  - Stvara …/Firma/secrets.json s ključem FirmaSqlPassword

CC BY NC SA

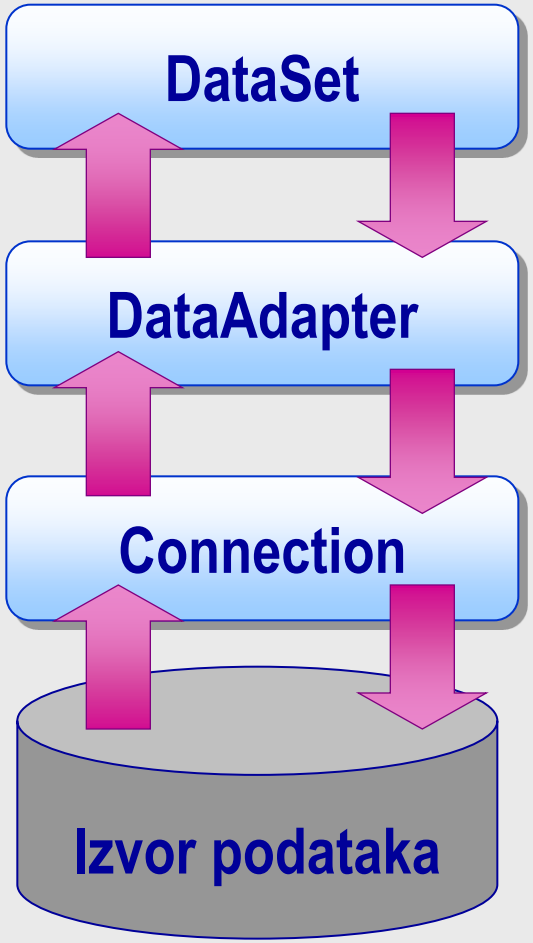
Dohvat „tajnih” vrijednosti

- **Primjer: 🗁 DataAccess / DataReader / Program.cs**
  - U *appsettings.json* zapisano sve osim prave lozinke korisnika
  - *AddUserSecrets("Firma”)* uključuje datoteku …/Firma/secrets.json koja sadrži ključ FirmaSqlPassword

```
var config = new ConfigurationBuilder()
                 .AddUserSecrets("Firma")
                 .SetBasePath(Directory.GetCurrentDirectory())
                 .AddJsonFile("appsettings.json")
                 .Build();

 string connString = config["ConnectionStrings:Firma"];

 connString = connString.Replace("sifra",
                                 config["FirmaSqlPassword"]);
```

BY NC SA

Lokalna obrada podataka



- **Podaci se obrađuju lokalno**
  - DataSet reprezentira stvarne podatke pohranjene u memoriju

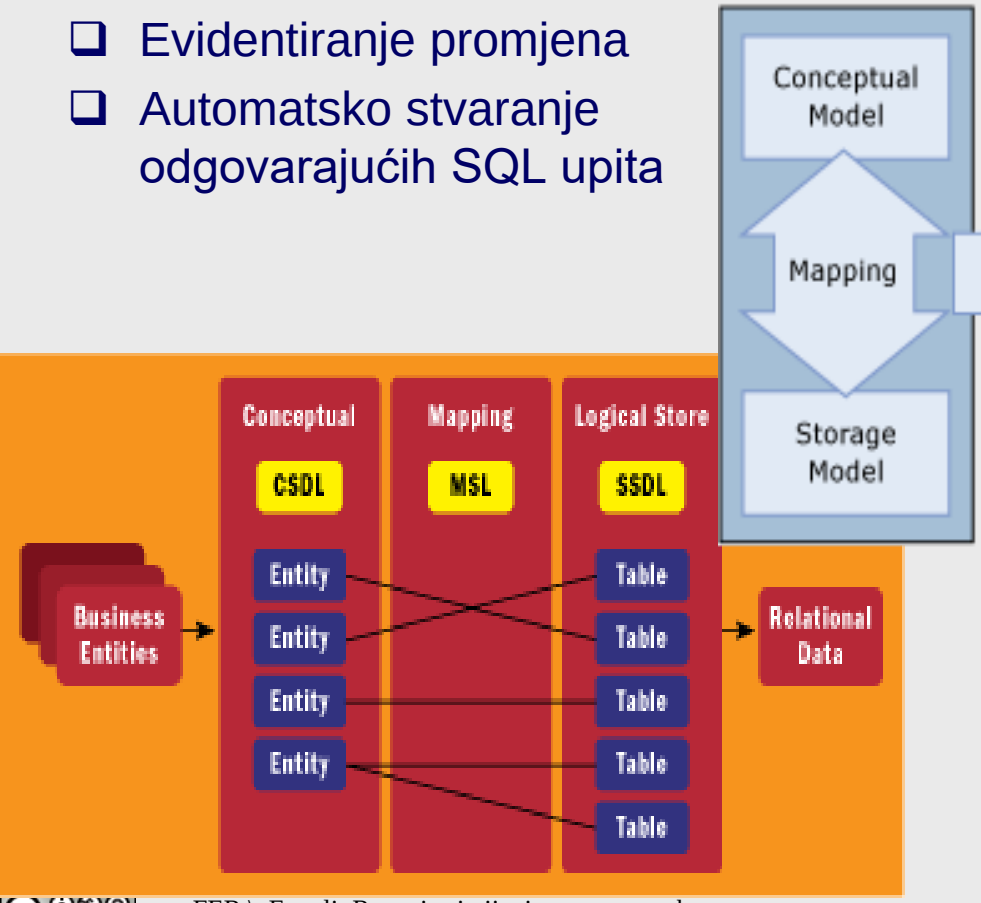
1. **Otvori priključak**
2. **Napuni *DataSet***
3. **Zatvori priključak**
4. **Obradi *DataSet***
5. **Otvori priključak**
6. **Ažuriraj izvor podataka**
7. **Zatvori priključak**

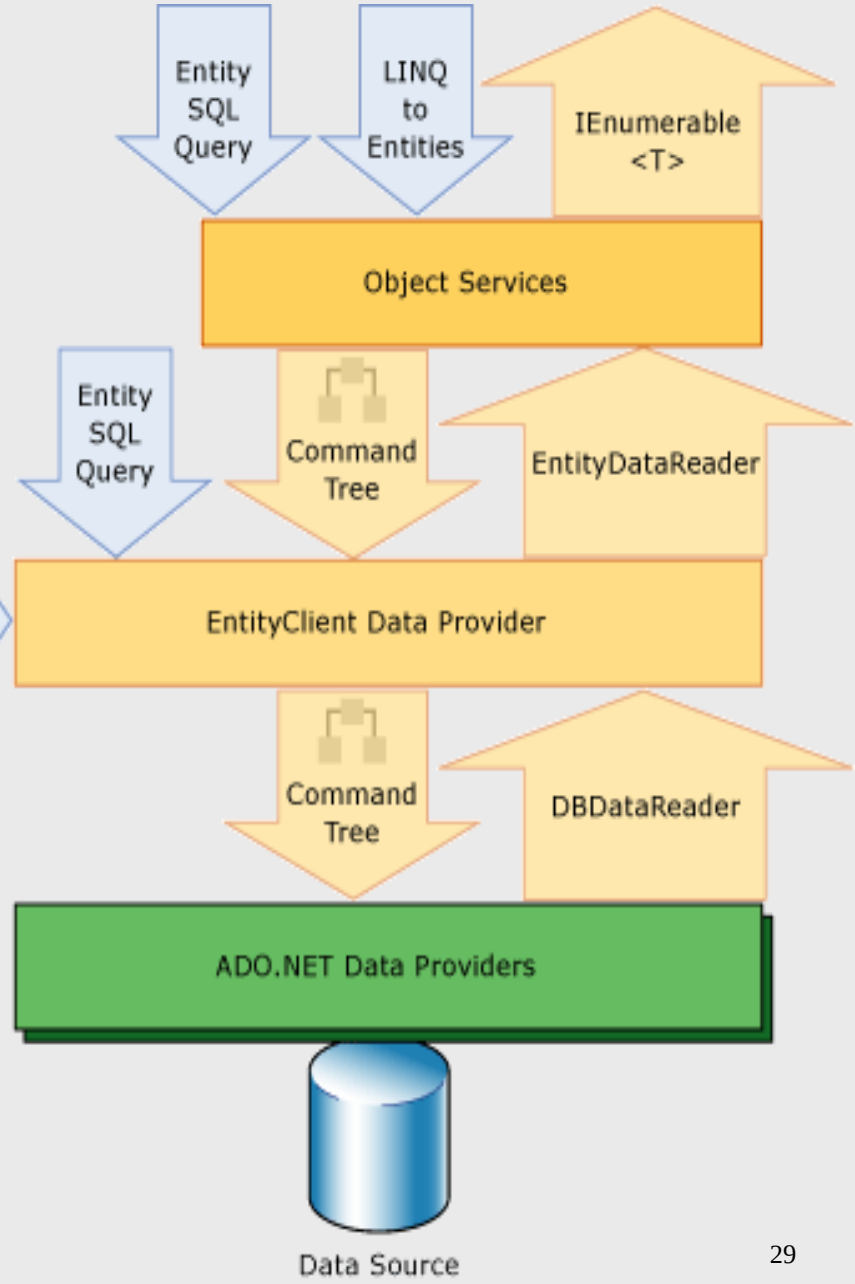
- **NET sadržavao razred DataSet koji je bio preslika relacijske baze podataka (nije ORM)**
  - Ideja lokalne obrade podataka *DataSetom* „prenesena” na EntityFramework

CC BY NC SA

Inicijalna ideja Entity Frameworka

- Nadgradnja nad ADO.NET-om
- Rad s BP na višoj razini putem objektnog modela
  - preslikavanje između modela i podataka u BP)
- Evidentiranje promjena
- Automatsko stvaranje odgovarajućih SQL upita





Entity Framework Core

BY NC SA

Načini kreiranja EF modela

- ***Database First***
  - Baza podataka već postoji i model nastaje reverznim inženjerstvom BP
- ***Model First***
  - Model se dizajnira kroz grafičko sučelje, a BP nastaje na osnovu modela.
- ***Code First***
  - Model opisan kroz ručno napisane razrede te nema vizualnog modela
  - BP se stvara na osnovu napisanih razreda. Izgled BP određen nazivima razreda, nazivima i vrstama asocijacija između razreda te dodatnim atributima.
- ***Code First from existing database***
  - Slično kao Code First, ali za postojeću bazu podataka
  - Baza podataka opisuje se razredima, ali ne uzrokuje stvaranje nove baze podataka.
  - Razredi se mogu stvoriti ručno ili nekim od generatora.
- ***Code First with Migrations***
  - Izvršava skup radnji (migracija) definiranih u posebnim postupcima (Up/Down)
- **Entity Framework Core podržava samo Code First varijante**
  - U primjerima koristimo Code First from existing database

BY NC SA

Stvaranje modela na osnovu postojeće BP (1)

1. **Instalirati dotnet-ef na računalu**

```
dotnet tool install --global dotnet-ef
```

2. **U mapi ciljanog projekta izvršiti sljedeće naredbe**

```
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

- ili dodati koristeći opciju Manage NuGet Packages

3. **U naredbenom retku izvršiti sljedeće dvije naredbe**

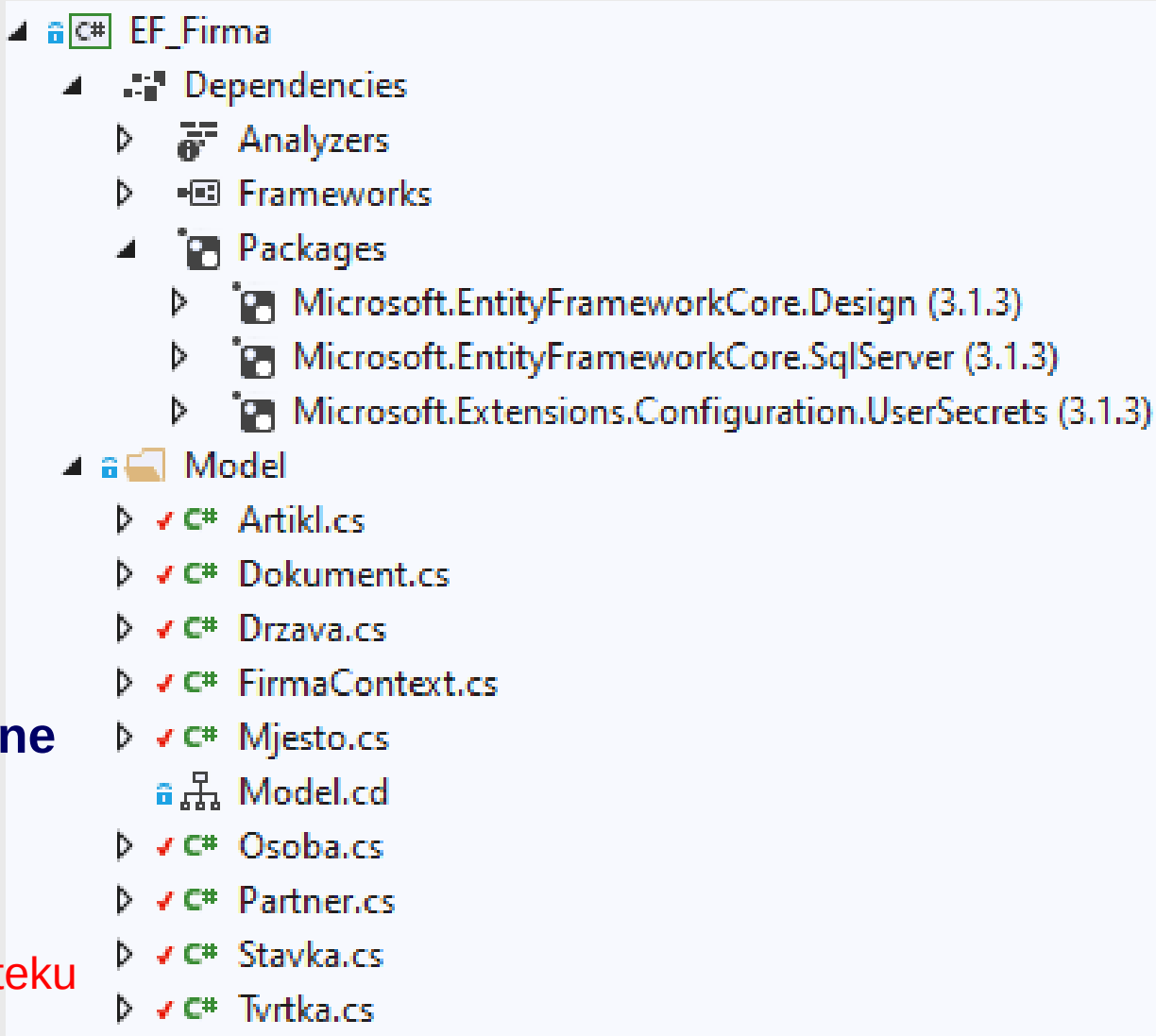
```
dotnet restore


dotnet-ef dbcontext scaffold
"Server=rppp.fer.hr,3000;Database=Firma;User
Id=rppp;Password=*"
Microsoft.EntityFrameworkCore.SqlServer -o Model -t
Artikl -t Dokument -t Drzava -t Mjesto -t Osoba -t
Partner -t Stavka -t Tvrtka
```

Stvaranje modela na osnovu postojeće BP (2)

- **Na osnovu postojećih stranih ključeva EF automatski stvara asocijacije između stvorenih razreda**

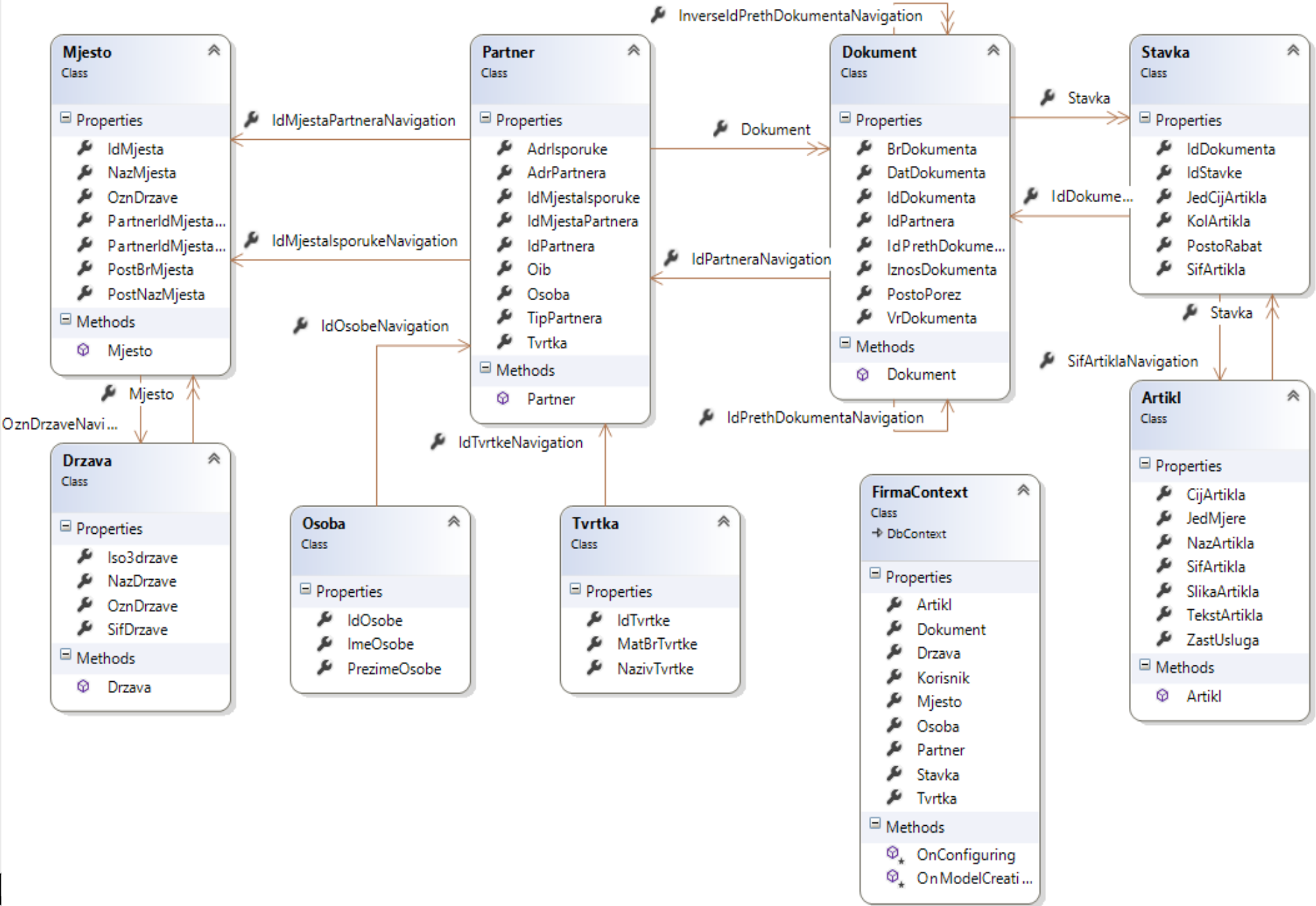
- **Za naš primjer stvaraju se:**
  - Firma.Context.cs
  - Po jedna cs datoteka za svaku tablicu

- **Postavke spajanja inicijalno tvrdo kodirane u FirmaContext.cs**
  - Potrebno ukloniti i prebaciti u konfiguracijsku datoteku



CC BY NC SA

Dijagram razreda generiranog modela



Postavke spajanja na BP korištenjem EF-a

- **Generirani model sadrži tvrdo kodirane postavke za spajanje na BP**
  - Primjer: 🗁 Bilo koji [Naziv]Context.cs stvoren prema prethodnim uputama

```
void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
   optionsBuilder.UseSqlServer(@"Server=...sword=*");
```

- **Može se zamijeniti odsječkom za dohvat postavki iz konfiguracijske datoteke**
  - Nije idealno rješenje, jer model određuje naziv konfiguracijske datoteke i naziv ključa, ali trenutno bolje od postojećeg – bit će unaprijeđeno naknadno
    - Primjer 🗁 DataAccess \ EF_Firma\ Models \ FirmaContext.cs

```
void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
 var config = new ConfigurationBuilder()
                .AddUserSecrets("Firma")
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json").Build();
 string connString = config["ConnectionStrings:Firma"];
 connString = connString.Replace("sifra",
                                 config["FirmaSqlPassword"]);
 optionsBuilder.UseSqlServer(connString);
```

BY NC SA

Elementi EF modela

- ***FirmaContext* – naslijeđen iz razreda *DbContext***
  - predstavlja kontekst za pristup bazi podataka
  - podaci pohranjeni unutar konteksta u skupu entiteta tipa DbSet<T>, gdje je T tip entiteta
  - Definiran u 🗁 **DataAccess \ EF_Firma \ Model \ FirmaContext.cs**

- **Svaki entitet predstavljen parcijalnim razredom**
  - Asocijacije kao virtualna svojstva (ICollection<T> za agregacije)
  - Omogućava stvaranje *proxy* razreda koji pruža vlastitu implementaciju virtualnih svojstava

- **"Korisnički" definiran dio parcijalnih razreda smješta se unutar projekta po volji**
  - Generirani razredi su parcijalni, pa se njihova definicija može nalaziti u više datoteka
  - Parcijalni razredi moraju biti definirani unutar istog prostora imena (npr. namespace EF_Firma.Model)

CC BY NC SA

Preslikavanje između EF modela i BP (1)

- **Primjer: 🗁 DataAccess \ EF_Firma \ Model \ FirmaContext.cs**

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    modelBuilder.Entity<Artikl>(entity =>
    {
        entity.HasKey(e => e.SifArtikla).HasName("pk_Artikl");
        entity.HasIndex(e => e.NazArtikla)
              .HasName("ix_Artikl_NazArtikla")
              .IsUnique();
        entity.Property(e => e.SifArtikla)
              .HasDefaultValueSql("0");
        entity.Property(e => e.CijArtikla)
              .HasColumnType("money")
              .HasDefaultValueSql("0");
```

Preslikavanje između EF modela i BP (2)

- Entiteti ostaju „čisti”, a preslikavanje je u jednom postupku
- Olakšava promjenu naziva atributa u bazi podataka
- Npr. PostgreSQL ima drugačiji stil imenovanja i tipove, pa bi tada isječak prilagođen za PostgreSQL bio

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    ...
    modelBuilder.Entity<Artikl>(entity =>
    {
       entity.Property(e => e.JedMjere)
              .IsRequired()
              .HasColumnName("jed_mjere")
              .HasColumnType("varchar")
              .HasMaxLength(5)
              .HasDefaultValueSql("'kom'::character varying");
```

Važnija svojstva i postupci razreda *DbContext* i *DbSet*

- ***DbContext***
  - *SaveChanges[Async]* – spremanje promjena u bazi podataka
  - *Database* – svojstvo koje omogućava direktni rad s BP (npr. kreiranje i brisanje BP, izvršavanje vlastitih SQL upita i procedura)
  - *ChangeTracker* – pristup do razreda koji prati promjene na objektima u kontekstu
  - *Set* i *Set<T>* - vraćaju *DbSet* za konkretni tip entiteta (Koristi se ako se želi napisati općeniti postupak, inače je svaki entitet već sadržan u kontekstu kao svojstvo)
  - *Entry* i *Entry<T>* - služi za dohvat informacije o nekom entitetu u kontekstu i promjenu njegovog stanja (npr. otkazivanje promjena)
- ***DbSet<T>***
  - *Add* – dodavanje objekta u skup
  - *Remove* – označavanje objekta za brisanje
  - *Local* – kolekcija svih trenutno učitanih podataka (koristi se za povezivanje na forme)
  - *Find [Async]* - Dohvat objekta unutar konteksta na osnovu primarnog ključa
  - *AsNoTracking* – Dohvat podataka za koje se ne evidentiraju promjene

CC BY NC SA

Dodavanje novog zapisa

- **Primjer: 🗁 DataAccess \ EF_Firma \ Program.cs - DodajArtikl**
  - Stvoriti novi objekt konstruktorom te ga dodati u kolekciju nekom od mogućih varijanti
    - `context.Artikl.Add(artikl);`
    - `context.Add(artikl);`
    - `context.Set<Artikl>().Add(artikl);`
  - Pohraniti promjene u kontekstu (jednom za sve promjene)

```
using (var context = new FirmaContext()){
        Artikl artikl = new Artikl{ // (1)
                SifArtikla = 12345678, CijArtikla = 10m,
                JedMjere = "kom", NazArtikla = "Burek sa sirom"
        };
        context.Artikl.Add(artikl);
        context.SaveChanges(); // pohrani sve promjene
}
```

BY NC SA

Ažuriranje postojećeg zapisa

- **Primjer: 🗁 DataAccess \ EF_Firma \ Program.cs - IzmijeniCijenuArtikla**
  - Dohvatiti entitet
    - korištenjem postupka Find ili FindAsync na DbSetu – traži zapis na osnovu vrijednosti primarnog ključa
      - Pretražuje unutar već učitanog konteksta, a ako ga ne pronađe obavlja se upit na bazu. Vraća *null* ako traženi zapis ne postoji
    - Ili postavljanjem Linq upita
  - Promijeniti željena svojstva
  - Pohraniti promjene u kontekstu

```
using (var context = new FirmaContext()) {
        Artikl artikl = context.Artikl.Find(12345678);
           //moglo je i context.Find<Artikl>(sifraArtikla);
        artikl.CijArtikla = 11m;
        context.SaveChanges();
}
```

CC BY NC SA

Brisanje zapisa

- **Primjer: 🗁 DataAccess \ EF_Firma \ Program.cs - ObrisiArtikl**
  - Dohvatiti entitet
  - Izbaciti ga iz konkretnog *DbSeta* ili označiti ga za brisanje pomoću *context.Entry*
  - Pohraniti promjene u kontekstu

```
using (var context = new FirmaContext())
{
  Artikl artikl = context.Artikl.Find(12345678);
  context.Artikl.Remove(artikl);
  //ili context.Entry(artikl).State = EntityStated.Deleted;
  context.SaveChanges();
}
```

CC BY NC SA

Upiti nad EF modelom

- **Where, OrderBy, OrderByDescending, ThenBy, First, Skip, Take, Select, …**
  - Davatelj usluge pretvara Linq upit u SQL upit
    - Nije uvijek moguće sve pretvoriti u SQL upit
- **Upit se izvršava u trenutku dohvata prvog podataka ili eksplicitnim pozivom postupka *Load (LoadAsync)***
  - Moguće ulančavanje upita (rezultat upita najčešće IQueryable<T>)
  - Podaci iz vezane tablice se učitavaju pri svakom dohvatu ili eksplicitno korištenjem postupka *Include* (kreira *join* upit u sql-u)
- 🗁 **DataAccess \ EF_Firma \ Program.cs – DohvatiNajskuplje**
  - Primjer upita za dohvat prvih n najskupljih artikala

```
var upit = context.Artikl.Include(a => a.Stavka)
                  .AsNoTracking()
                  .OrderByDescending(a => a.CijArtikla)
                  .Take(broj);
foreach (Artikl artikl in upit) { … }
```

CC BY NC SA

SQL nastao upitom kroz EF

- **Za prethodni primjer na SQL serveru će se izvršiti sljedeći upit**
  - Trebalo li sve podatke?
  - Možda samo trebamo ispisati koliko artikl ima stavki?

```
exec sp_executesql N'SELECT [s].[IdStavke],
[s].[IdDokumenta], [s].[JedCijArtikla], [s].[KolArtikla],
[s].[PostoRabat], [s].[SifArtikla]
FROM [Stavka] AS [s]
INNER JOIN (
    SELECT DISTINCT TOP(@__p_0) [a].[CijArtikla],
[a].[SifArtikla]
    FROM [Artikl] AS [a]
    ORDER BY [a].[CijArtikla] DESC, [a].[SifArtikla]
) AS [a0] ON [s].[SifArtikla] = [a0].[SifArtikla]
ORDER BY [a0].[CijArtikla] DESC,
[a0].[SifArtikla]',N'@__p_0 int',@__p_0=10
```

CC BY NC SA

Anonimni razredi kao rezultati upita

- **Rezultat upita ne mora biti neki od postojećih entiteta, već podskup ili agregacija više njih**
- **Rezultat je anonimni razred sa svojstvima navedenim u upitu**
  - Može se dati novo ime za pojedino svojstvo
- **🗁 DataAccess \ EF_Firma \ Program.cs – DohvatiNajskuplje_v2**

```
var upit = context.Artikl
                  .OrderByDescending(a => a.CijArtikla)
                  .Select(a => new
                          {
                             a.NazArtikla, a.CijArtikla,
                          BrojProdanih = a.Stavka.Count
                          })
                  .Take(n);

foreach (var artikl in upit)
  Console.WriteLine("...", artikl.NazArtikla,
                 artikl.CijArtikla, artikl.BrojProdanih);
```

CC BY NC SA

SQL nastao modificiranim upitom

- **SQL za prethodni EF upit je jednostavniji**

```
exec sp_executesql N'SELECT TOP(@__p_0) [a].[NazArtikla],
[a].[CijArtikla], (
    SELECT COUNT(*)
    FROM [Stavka] AS [s0]
    WHERE [a].[SifArtikla] = [s0].[SifArtikla]
)
FROM [Artikl] AS [a]
ORDER BY [a].[CijArtikla] DESC',N'@__p_0 int',@__p_0=10
```

CC BY NC SA

Ostale mogućnosti EF-a

- Nakon uspješnog upita EF automatski vrši dohvat primarnog ključa koji je definiran kao tip *identity*
- U trenutku pisanja ovih materijala EF Core ne podržava dodavanje procedura u model
- Moguće je samostalno napisati upit, ali rezultat je (trenutno) moguće samo pohraniti u neki od postojećih entiteta
- Poglede je moguće dodati u model, što će biti prikazano u poglavlju s web-aplikacijama

CC BY NC SA

Poboljšanje rada s postavkama za spajanje na BP iz EF (1)

- **Prethodno tvrdo kodirane postavke za spajanje na BP zamijenjen s odsječkom koji učitava postavke iz konfiguracijske datoteke**
- **Mane:**
  - Modelom se propisuje da konfiguracijska datoteka mora biti upravo appsettings.json
  - Mora se koristiti isključivo navedeni ključ

- **Alternativno može se definirati da konstruktor konteksta prima postavke**
- **Mana: Prilikom svakog poziva mora se poslati navedeni argument i napisati kod za dohvat vrijednosti iz konfiguracijske datoteke**

CC BY NC SA

Poboljšanje rada s postavkama za spajanje na BP iz EF (2)

- (Naoko zasad ne tako dobra) ideja:
- Definirati sučelje s metodom koja vraća postavke te u konstruktoru primati primjerak sučelja
  - 🗁 DataAccess \ EF_Firma_DI \ Model \ FirmaContext.cs

```
 private IConnectionStringTool tool;

 public FirmaContext(IConnectionStringTool tool) {
      this.connectionStringTool = tool;
 }

protected override void OnConfiguring(
             DbContextOptionsBuilder optionsBuilder)
{
  optionsBuilder.UseSqlServer(tool.GetConnectionString());
}
```

Poboljšanje rada s postavkama za spajanje na BP iz EF (3)

- **Implementacija sučelja vrši dohvat podataka iz konfiguracijske datoteke**
  - 🗁 DataAccess \ EF_Firma_DI \ ConnectionStringTool.cs

```
public class ConnectionStringTool : IConnectionStringTool {
    private string connectionString;
    public ConnectionStringTool() {
       var config = new ConfigurationBuilder()
                 .SetBasePath(Directory.GetCurrentDirectory())
                 .AddJsonFile("appsettings.json")
                 .Build();
      connectionString = config["ConnectionStrings:Firma"];
      connectionString = connectionString.Replace("sifra",
                              config["FirmaSqlPassword"]);
    }
    public string GetConnectionString() { //metoda sučelja
      return connectionString;
    }
```

CC BY NC SA

Poboljšanje rada s postavkama za spajanje na BP iz EF (4)

- **Ovakvo rješenje zahtjeva da se prilikom svakog instanciranja konteksta pošalje instanca vlastitog razreda**
- **Naoko izgleda kao korak nazad u odnosu na varijantu u kojoj se šalje samo *connection string*, ali…**
- **… .NET Core ima ugrađenu podršku za *Dependency Injection***
- **Umjesto direktnog stvaranja objekta konteksta s *new* koristit će se primjerak sučelja *IServiceProvider***
  - osigurava da se prilikom stvaranja nekog objekta automatski stvore i odgovarajući objekti potrebni u njegovom konstruktoru

.NET Core i Dependency Injection

- **Primjer 🗁 DataAccess \ EF_Firma_DI \ Program.cs**
- **Inicijaliziraju se postavke za DI te se postavljaju sljedeće postavke**
  - Svaki put kada se korištenjem *Service Providera* stvara objekt koji u konstruktoru kao parametar treba primjerak sučelja *IConnectionStringTool* neka mu se pošalje objekt tipa *ConnectionStringTool*
    - *AddSingleton* označava da će se za cijeli program objekt tipa *ConnectionStringTool* stvoriti samo jednom
  - Ako je potrebno stvoriti objekt tipa *FirmaContext*, neka se svaki put stvori novi (*AddTransient*) objekt tipa *FirmaContext*

```
private static IServiceProvider serviceProvider;
public static void Main(string[] args) {
 //Inicijaliziraj postavke za DI
 serviceProvider = new ServiceCollection()
                        .AddSingleton<IConnectionStringTool,
                                      ConnectionStringTool>()
                        .AddTransient<FirmaContext,
                                      FirmaContext>()
                        .BuildServiceProvider();
```

.NET Core, DI na primjeru EF konteksta

- **Primjer 🗁 DataAccess \ EF_Firma_DI \ Program.cs**

```
context = serviceProvider.GetService<FirmaContext>()
```

- ***Service Provider* treba vratiti objekt koji je pridružen razredu/sučelju *FirmaContext*, a to je upravo *FirmaContext*.**
- **Postavkama je određeno da to mora biti novi objekt**
- **Konstruktor razreda FirmaContext ima argument tipa *IConnectionStringTool***
- ***ServiceProvider* za IConnectionStringTool ima pridružen razred ConnectionStringTool i to s opcijom *singleton*.**
  - Ako objekt još nije stvoren, stvara se
  - Postojeći objekt tipa IConnectionStringTool se predaje konstruktoru razreda FirmaContext
- **Navedeno rješenja nam omogućava da jednostavno zamijenimo način pohrane i dohvata postavki za spajanje na BP, a da pritom naš kontekst ovisi samo o jednoj metodi sučelja**
  - Intenzivno se koristi unutar ASP.NET Core-a

Reference

- **Overview of ADO.NET**
  - https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview
- **Entity Framework Core**
  - https://docs.microsoft.com/en-us/ef/core
- **„Tajne” vrijednosti (user/app secrets)**
  - https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets
- **.NET Core i Dependency Injection**
  - https://andrewlock.net/using-dependency-injection-in-a-net-core-console-application/

CC BY NC SA

Web-aplikacije
ASP.NET Core MVC

2019/20.05

BY NC SA

Web-aplikacija

- **Što je web aplikacija?**
  - Programska aplikacija kojoj se pristupa preko internetskog preglednika ili drugog programa koji implementira HTTP (Hyper Text Transfer Protocol)
- **Da li je skup web stranica ujedno i web aplikacija?**
  - Aplikacija koristi programsku logiku da bi prikazala sadržaj korisniku
  - Kod web aplikacija obično se izvršava neki programski kod na serveru
  - Primjer: statički cjenik nije aplikacija; dinamički cjenik je aplikacija
- **Primjer web aplikacije: Edgar http://edgar.fer.hr**
  - Identifikacija i autorizacija korisnika
  - Pisanje provjere, pregled rezultata, ...
  - Baza podataka + programska logika za generiranje web stranica
- **Elementi web stranice**
  - HTML (HyperText Markup Language): osnovni jezik za definiranje web stranica
  - JavaScript: jezik za klijentske skripte koje izvodi preglednik
  - Kôd u nekom drugom jeziku koji se izvodi na poslužitelju

Karakteristike web-aplikacija

- Izvršava se i na poslužitelju i na klijentu
  - Klijentski dio mora biti napisan u nekom od jezika koji Web preglednik podržava (HTML, JavaScript).
- Ograničen pristup resursima na strani klijenta (npr. JavaScript koji se izvršava unutar preglednika ne može čitati s korisnikovog diska)

- **Prednosti**
  - Korisnik može biti bilo tko s pristupom Internetu
    - nema instalacijske procedure na strani klijenta
    - koriste se na bilo kojem OS, koji ima internetski preglednik
  - Jednostavno održavanje i nadogradnja na novu verziju
- **Nedostatci**
  - Složenija izrada u odnosu na samostojne klijentske aplikacije
    - Potreba za posebno dizajniranim sučeljem (Web design)
    - Mogući problemi pri prikazu u različitim preglednicima
    - Potrebna prilagodba regionalnim posebnostima korisnika
  - Sigurnosni problemi (neovlašten pristup aplikaciji i poslužitelju, zatrpavanje prometom)

BY NC SA

Kratka povijest razvoja web-aplikacija

…ili zašto danas neke stvari (ne) radimo na određeni način…

BY NC SA

Interaktivnost na klijentskoj strani

- **„Duga” povijest, ali šira upotreba tek u nekoliko zadnjih godina**

- **JavaScript 1995. godine**
  - Nakadno se pojavljuju (s više ili manje uspjeha) Flash, Java Applets i Silverlight
- **jQuery – prva verzija 2006. godine**
  - Verzija 1.5: 2011.godine
  - Verzija 2.0: 2013. godine
  - Verzija 3.1: 7. mjesec 2016.
  - Verzija 3.4.1: 4. mjesec 2019.

- **HTML 5 2014. godine**

- **CSS predložen 1994., objavljen prvi put 1996.**
  - Aktualna verzija CSS 3

BY NC SA

CGI skripte

- **Začeci između 1993. i 1995. – protokol Common Gateway Interface**
  - CGI skripte, obično u nekom od skriptnih jezika (npr. Perl)
  - Skripta na standardni izlaz ispisuje HTML dokument koji web server preusmjerava korisniku
  - Parametri se prenose *query stringom*
    - parovi oblika ključ=vrijednost iza znaka upitnik u adresi zahtjeva
  - U skripti dostupan unutar varijable okruženja QUERY_STRING

```
#!/bin/sh
echo "Content-type: text/html"
echo
echo "<html> <head> <title> CGI script </title> </head>
<body>"
echo "Sadržaj query stringa: $QUERY_STRING <br>"
echo "</body>"
```

  - U POST varijanti parametri sadržani u tijelu HTTP zahtjeva
- **Svaki poziv CGI skripte je novi proces**

CC BY NC SA

PHP i ASP

- **1995. i 1996. PHP i ASP (Active Server Pages)**
  - Kasnije i Java Servlets
  - Web server ima dodatak koji omogućava izvršavanje više zahtjeva unutar istog procesa
  - Odsječci koji počinju s <?php odnosno <% izvršavaju se na serveru
  - Veće mogućnosti izvršavanja na serveru i mogućnost direktnog pisanja HTML-a (bez *echo* za svaki tekst)
- **Stil pisanje aplikacije i dalje nalik CGI skriptama**
  - Izmiješan HTML sadržaj sa serverskom logikom
  - Teško za održavanje i čitanje koda.

```
<% Do While Not RSSes.EOF And BrRedova>0 %>
<TR><TD VALIGN=TOP><%= absPos %>.</TD>
<TD VALIGN=TOP><a
href="bib_details.asp?idRef=<%=RSSes("IdReference")%>"><%=RSSe
s("AutoriRef")%></A>,
<%=RSSes("Godina")%><BR><%=RSSes("Naslov")&RSSes("Naslov2")%>
<% If existData(RSSes("Volumen")) Then %>
, Vol. <%= RSSes("Volumen") %>
<% End If %>
```

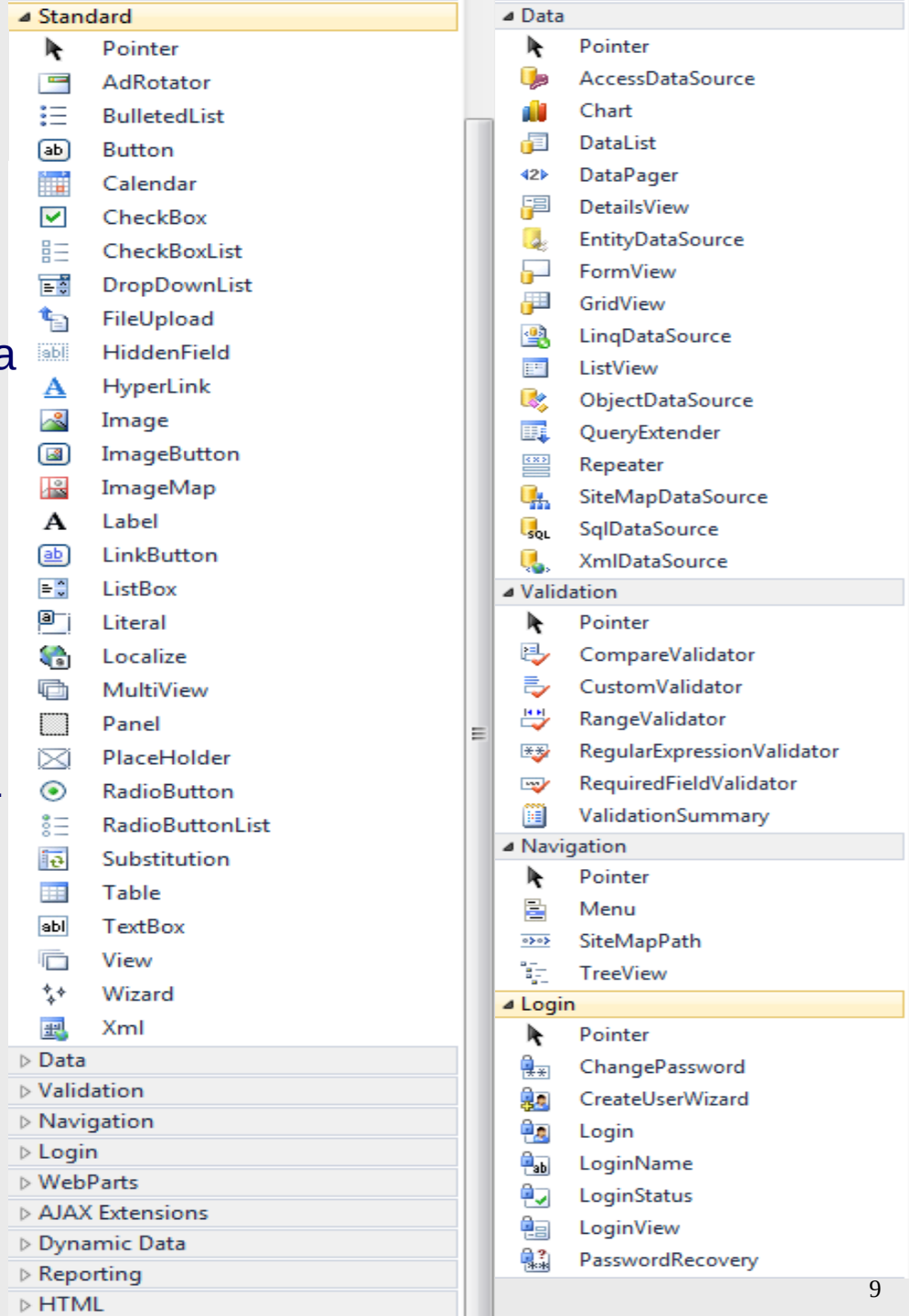
Pokušaji odvajanja prezentacijskih elemenata

- **Problem kako odvojiti prezentacijski dio od dohvata i pripreme podataka**
- **Izbjeći miješanje koda koji se izvršava na serveru i HTML-a koji se prikazuje korisniku.**
- **Kod PHP-a se počinju koristiti predlošci (npr. Smarty)**
  - u posebnim datotekama kreiraju se predlošci s posebno označenim varijablama i oznaka
  - u PHP skripti se vrši dohvat podataka koji se predaje predlošku.
  - Iz današnje MVC perspektive moglo bi se reći da je PHP skripta pripremila model koji se onda prezentirao pogledom pisanom u smarty-u.
- **Microsoft umjesto ASP-a 2002.g. stvara ASP.NET Web Forms**
  - Uspješan u svoje vrijeme, jer je nudio jednostavan način izrade web aplikacija, omogućen i onima koji su imali oskudna znanja HTML-a

BY NC SA

ASP.NET Web Forms

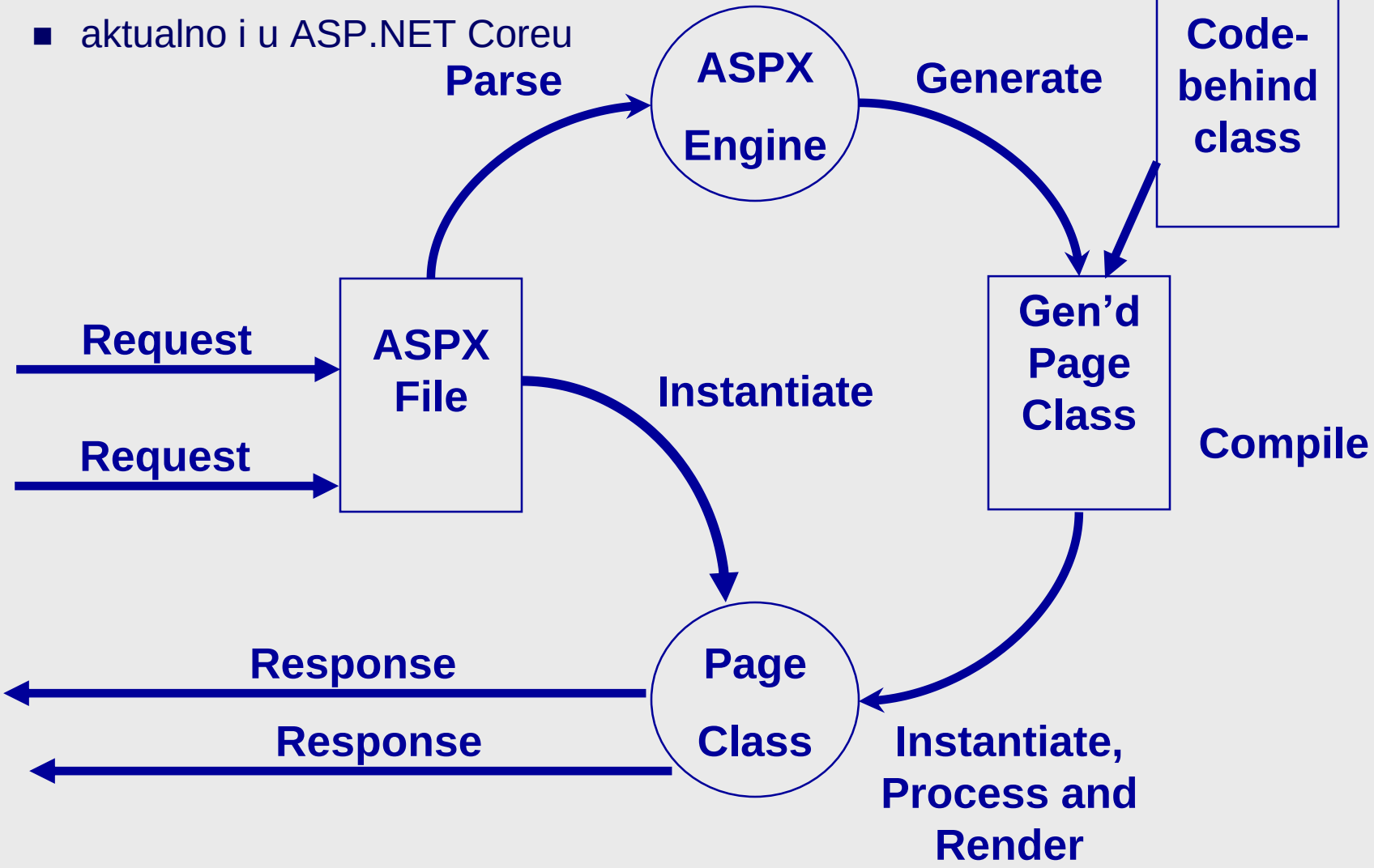
- **cca 60 unaprijed definiranih serverskih kontrola**
  - simulira se izrada web aplikacija nalik windows formama s ograničenjima web tehnologije
- **Realizacija**
  - Stranica.aspx + Stranica.aspx.cs + Stranica.aspx.designer.cs
  - .NET Framework vrši transformaciju kontrola u HTML
  - kontrole imaju mogućnosti HTML kontrola uz dodatnu funkcionalnost (npr. validacija)
  - Da bi se očuvao sadržaj stranice, podaci se prenose zajedno sa stranicom (rekonstrukcija sadržaja pomoću skrivenog polja __VIEWSTATE)



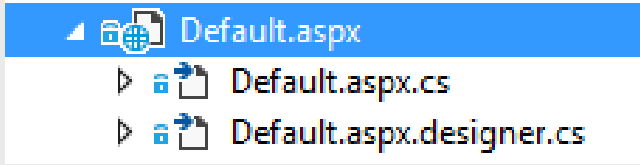
BY NC SA

Prevođenje i prikaz stranica

- poslužiteljska logika se piše u nekom od .NET jezika
- prevođenje → brže izvođenje nego kod skriptnih jezika (ASP, PHP)
  - aktualno i u ASP.NET Coreu



WebForms - Odvajanje dizajna od koda



- **Datoteka s kontrolama - Default.aspx**
  - ASPX stranica uobičajeno sadrži HTML, serverske kontrole, JavaScript i ostale prezentacijske elemente
  - Source i/ili Design pogled na istu stranicu

Default.aspx.cs | Default.aspx*

Client Objects & Events | (No Events)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitio
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Pozdrav (web)</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Button ID="Button1" runat="server"
        Text="Klikni me" OnClick="Button1_Click" />
        <hr /><asp:Label ID="Label1" runat="server"/>
    </form>
</body>
</html>
```

Design | Source | <html>

BY NC SA

Odvajanje dizajna od koda

- **Datoteka s kôdom (CodeFile, Code behind) – Default.aspx.cs i Default.aspx.designer.cs**
  - parcijalni razred
  - sadrži kôd koji se izvršava na serveru

```
public partial class Default : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e){
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
      Label1.Text = "Pozdrav, trenutno vrijeme je " +
                                DateTime.Now.ToString();
    }
}
```

- **Klikom na gumb izvodi se postupak Page_Load (ako postoji), a zatim slijede obrade događaja**
  - Rekonstrukcija sadržaja kontrola pri svakom zahtjevu

CC BY NC SA

Ključne karateristike Web Formi

- **Za svaku kontrolu automatski se generira odgovarajuća HTML kontrola**
- **Automatski se rekonstruira stanje/sadržaj kontrola između dvaju zahtjeva na server (koristi se skriveno polje __ViewState)**
- **Glavna stranica (Master)**
  - Predstavlja zajednički izgled (okvir, dizajn) za više stranica
  - Uobičajeno sadrži zajedničke elemente (izbornike, zaglavlje, podnožje, …)
  - Web aplikacija može imati više master stranica
  - Može sadržavati što i obična stranica uz jedan ili više okvira
- **Povezivanje podataka**
  - Kontrola se veže na neki ObjectDataSource (ili neki drugi DataSource) kojem su definirani CRUD postupci + straničenja, sortiranja i slično
  - Automatsko kreiranje tablica s podacima + kontrole za Edit/Update/Cancel i prelaske među njima
    - Korisnik piše samo postupke za manipulaciju podacima, ne i postupke za prihvat podataka s prezentacijskog sučelja
- **Izrada stranice u dizajnu ili direktnim pisanje koda**

Nedostatci ASP.NET web formi

- **ViewState proizvodi veliku količinu podataka pri svakom zahtjevu**
- **Kompliciran životni ciklus stranice**
- **Ograničena kontrola nad proizvedenim HTML-om**
- **Prevelike *.cs datoteke uz često miješanje prezentacijske i aplikacijske logike**
- **Slaba mogućnost testiranja**

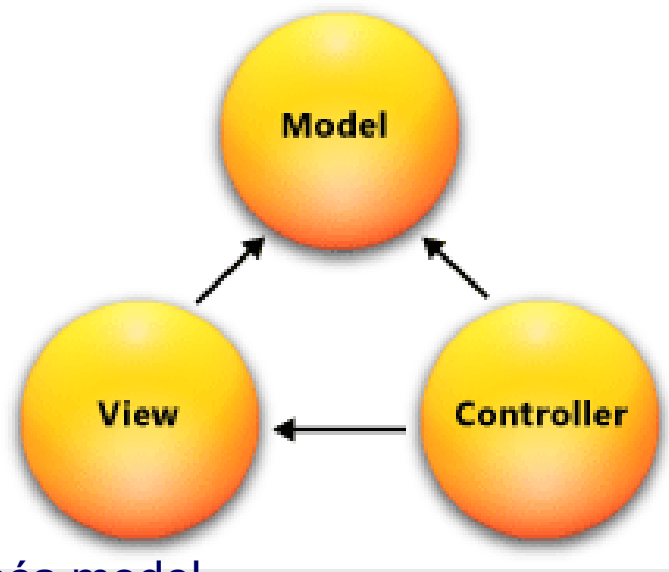
- **ASP.NET MVC kao rješenje navedenih problema**
  - Objedinjena iskustva MVC implementacija u drugim jezicima
  - MVC kao koncept nastao u kasnim sedamdesetim godinama prošlog stoljeća (Smalltalk projekt unutar Xeroxa)

CC BY NC SA

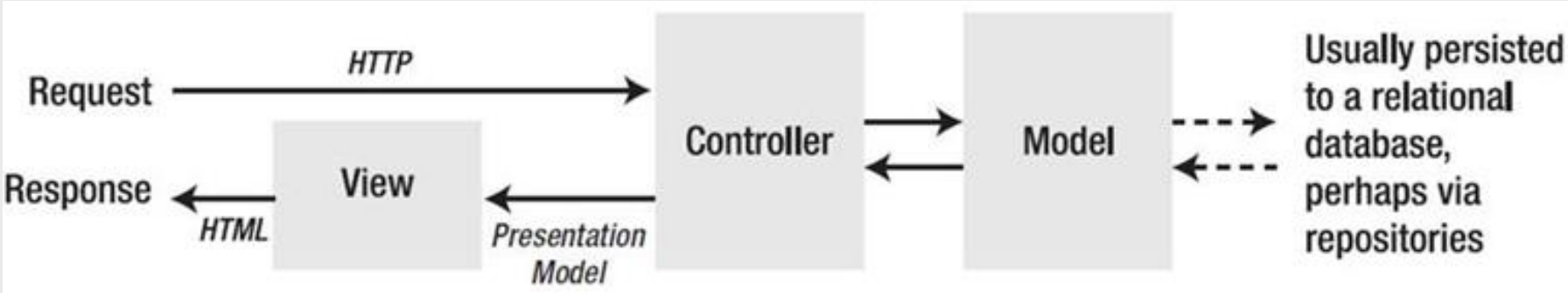
MVC

BY NC SA

ASP.NET MVC



- **Model - Pogled - Upravljač**
- **Detaljnija razrada <u>prezentacijskog sloja</u>**
  - Pogled definira izgled korisničkog sučelja
  - Obično vezan za objekt iz modela
- **<u>Upravljač predstavlja prezentacijsku logiku</u>**
  - Prima ulaz iz pogleda, obrađuje ga, puni i dohvaća model poziva niže slojeve i određuje redoslijed prikaza pogleda



- **U jednostavnim aplikacijama model objedinjava i poslovnu logiku i sloj pristupa podacima**
- **U složenijim kao model koristi se "pravi" poslovni model, a model unutar projekta služi za definiranje pomoćnih modela za lakši prikaz ("prezentacijski model", a ne model u smislu poslovnog objekta)**

CC BY NC SA

Prednost MVC-a nad web formama

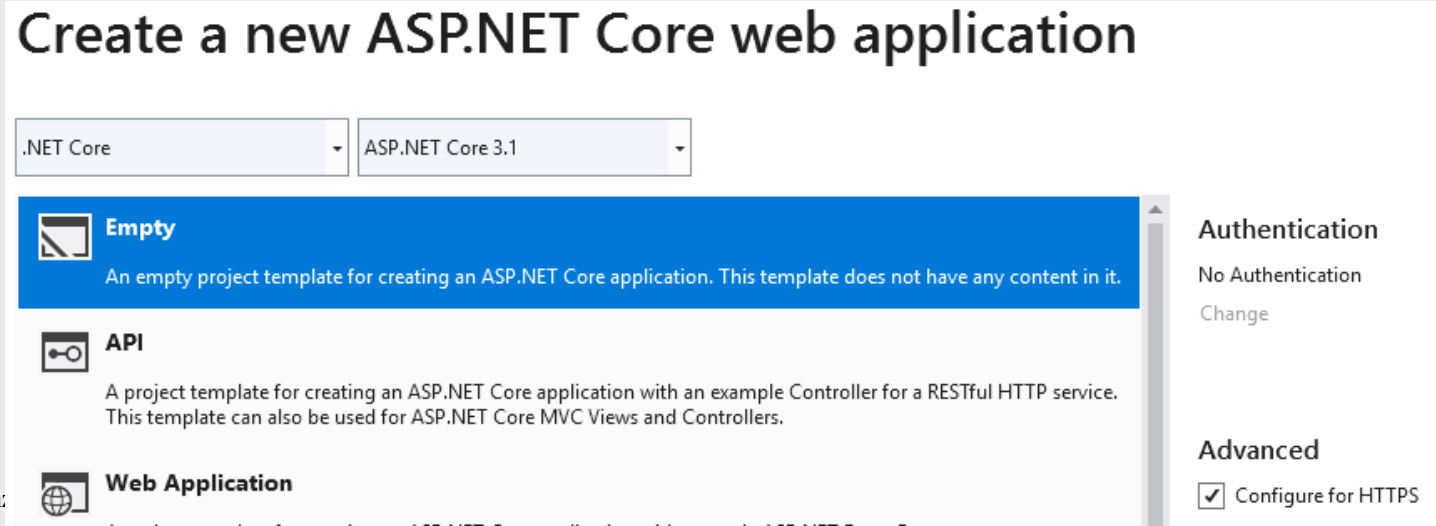
- **Smanjena složenost podjelom aplikacije u model, pogled i upravljač**
- **Ne koristi se *ViewState* ni serverske kontrole čime se omogućava potpuna kontrola ponašanja aplikacije.**
- **Zahtjevi centralizirani na jedan upravljač**
  - Front Controller pattern
  - bogata podrška za interpretiranje/usmjeravanje zahtjeva
- **Podrška za test-driven development (TDD).**
- **Dobar okvir i za veće razvojne timove i za dizajnere**

ASP.NET MVC Core

- **Podržan na različitim platformama (Windows, Linux, MacOs, Docker)**
- **Ključne promjene u odnosu na „stari” ASP.NET MVC**
  - Tag-helperi za formiranje poveznica i kontrola
    - proširuju postojeće HTML kontrole dodatnim atributima koji se koriste prilikom generiranja stranica, npr. za formiranje poveznica, popunjavanje polja za unos podatka i slično (više naknadno)
  - Veće mogućnosti konfiguracijskih datoteka
  - Intenzivno korištenjem tehnike *Dependency Injection*
    - posljedično lakše testiranje i veća modularnost

Stvaranje nove web-aplikacije

- **Iz naredbenog retka**
  - `dotnet new mvc --auth None -n Naziv`
    - obrisati višak
  - `dotnet new web -n Naziv`
    - dodati potrebne pakete i konfigurirati za MVC
- **Koristeći razvojno okruženje**
  - Create a new project → ASP.NET Core Web Application
    - Empty (ili Web Application) uz opciju No Authentication
- **U primjeru koji slijedi bit će stvorena prazna web aplikacija, pa će postupno biti dodavani potrebni dijelovi**



Sadržaj „prazne” web-aplikacije

- **Samo Program.cs i Startup.cs.**
  - Web-aplikacija se izvršava na privremenom web-serveru (localhost:port)
  - Ispisuje *HelloWorld* - definirano u postupku *Configure*
  - Primjer: 🗁 BiloKojaPraznaAplikacija \ Startup.cs

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env) {
    if (env.IsDevelopment())
    {
       app.UseDeveloperExceptionPage();
    }
    app.UseRouting();

    app.UseEndpoints(endpoints => {
          endpoints.MapGet("/", async context => {
            await context.Response.WriteAsync("Hello World!");
          });
    });
 }
```

Postavke pokretanja programa

- **Program.cs je „glavni program” web-aplikacije**
  - ASP.NET Core koristi *Kestrel* kao svoj web server
    - Može se spojiti s klasičnim web serverom (IIS, Apache, Nginx i slično)
  - Postavljaju se postavke aplikacije ili određuje razred u kojem se to obavlja
    - uobičajeno razred Startup
  - Primjer: 🗁Web \ Firma.Mvc \ Program.cs

```
public class Program {
  public static void Main(string[] args) {
    CreateHostBuilder(args).Build().Run();
  }

  public static IHostBuilder CreateHostBuilder(
                                          string[] args) =>
     Host.CreateDefaultBuilder(args)
         .ConfigureWebHostDefaults(webBuilder => {
            webBuilder.UseStartup<Startup>();
         });
}
```

Uključivanje podrške za MVC

- **Ukloniti odsječak za ispis *Hello World* u svakom zahtjevu**
- **U postupku za definiranje korištenih servisa dodati podršku za MVC i aktivirati MVC u postupku Configure**
  - Više o rutama i usmjeravanjima naknadno
- **Primjer: 🗁Web \ Firma.Mvc \ Startup.cs**

```
public void ConfigureServices(IServiceCollection services) {
      services.AddControllersWithViews();
…
public void Configure(...) {
       ...
       app.UseEndpoints(endpoints => {
           endpoints.MapGet("/", async context => {
             await context.Response.WriteAsync("Hello World!");
           });
       });

      app.UseEndpoints(endpoints =>
             endpoints.MapDefaultControllerRoute());
```

Način rada MVC aplikacije

- **Iz adrese zahtjeva i postavki usmjeravanja bit će odabran upravljač (razred izveden iz razreda *Controller*) na kojem će se izvesti određena akcija (postupak u odabranom razredu)**
- **Pozvani upravljač (najčešće) popunjava određeni model i vraća korisniku pogled u kojem su vrijednosti zamijenjene konkretnim vrijednostima iz modela**
- **Kako prepoznati koju akciju (i na kojem upravljaču) izvršiti?**
  - Inicijalno usmjeravanje definira rutu oblika
    `{controller=Home}/{action=Index}/{id?}`
    tj. pretpostavlja da je adresa oblika *controller/action/id* pri čemu su pretpostavljene vrijednosti:
    - upravljač: *Home*
    - akcija: *Index*
    - parametar id je opcionalan
  - Napomena: U primjeru s mjestima i dokumentima bit će prikazan način kako definirati drugačije umjeravanje

CC BY NC SA

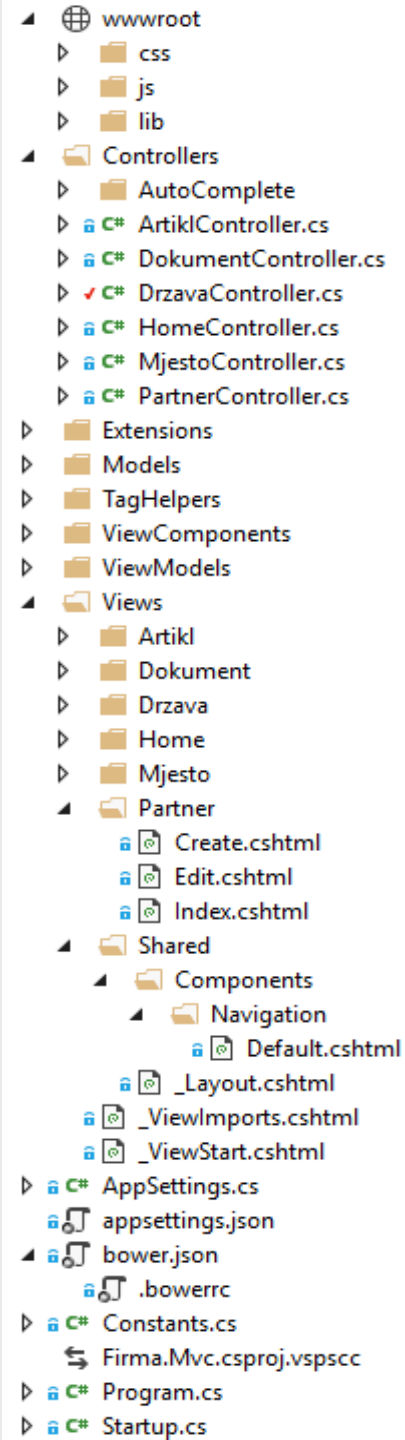
Organizacija mapa unutar projekta

- **Uobičajena konvencija**
  - Upravljači unutar mape *Controllers* u razredima sa sufiksom Controller
    - Primjer: 🗁Web \ Firma.Mvc \ Controllers \ HomeController.cs
  - Pogledi unutar mape *Views* podijeljeni u podmape po nazivima upravljača
    - Naziv datoteke obično odgovara nazivu akcije (postupka u upravljaču)
    - Primjer: 🗁Web \ Firma.Mvc \ Views \ Home \ Index.cshtml
- **Podjela po područjima (engl. Area)**
  - Svako područje u svojoj mapi ispod mape Areas prati uobičajenu konvenciju
    - Npr. Areas \ Podrucje1 \ Controllers \ DataLoadController.cs
  - Naziv područja dio adrese zahtjeva (npr. Podrucje1 / DataLoad / Akcija)
- **Podjela po funkcionalnosti - „Feature Folders”**
  - Svaka funkcionalnost ima svoju mapu koja sadrži i poglede i upravljače na istom nivou
  - Praktično kod velikih aplikacija

CC BY NC SA

Konačna struktura ogledne aplikacije

- **Uobičajene mape i datoteke**
  - *wwwroot*: statički sadržaj (css, skripte, klijentske biblioteke)
  - *Controllers*: upravljači unutar aplikacije
  - *Views*: pogledi podijeljeni u podmape za svaki upravljač i Shared za zajedničke poglede (npr. glavna stranica)
  - *Models*: razredi koji predstavljaju domenske modele (u slučaju manje aplikacije)
  - ViewModels: razredi koji služe za prijenos podataka pogledu i prihvat podataka iz pogleda
    - prezentacijski modeli
  - *TagHelpers*: vlastiti razredi s atributima kojim se proširuju uobičajene HTML oznake
  - *ViewComponents*: komponente (dijelovi aplikacije) koje se koriste na više mjesta, dizajniraju se zasebno te se uključuju u pojedinim pogledima
  - Konfiguracijske i projektne datoteke
  - Ostali razredi po potrebi



CC BY NC SA

Početna stranica (1)

- **Programski kod akcije Index na upravljaču Home kao rezultat vraća pogled ne navodeći ime pogleda**
  - Tip rezultata je *IActionResult*
    - Može se specificirati i konkretniji
  - Podrazumijeva se ime pogleda jednako nazivu postupka
    - Pogled se očekuje u datoteci Views \ Home \ Index.cshtml
      - ili istog imena negdje u mapi Shared
  - O sintaksi pogleda na kasnijim slajdovima

Index.cshtml

```
1   @{
2     ViewData["Title"] = "Početna stranica";
3   }
4
5   <p>
6     Primjerom se ilustrira nekoliko karakterist
7   </p>
8   <p>
9     Primjer s državama prikazuje iz podatke koj
10    prikazuje se po n podataka po stranici pri
11    zapisan u konfiguracijskoj datoteci appsett
12  </p>
13  <p>
14    Također se vodi računa o tome da ako krenem
15    na istu stranicu na kojoj smo bili prije od
16  </p>
17  <p>
18    Omogućeno je dodavanje novih država, ažurir
19    ažuriranje obavlja na posebnoj stranici
20  </p>
21  <p>
22    Primjer s partnerima je primjer hijerarhije
```

- **Primjer: 🗁Web \ Firma.Mvc \ Controllers \ HomeController.cs**

```
public class HomeController : Controller {
  public IActionResult Index() {
            return View();
  }
}
```

Uobičajeni rezultati akcije upravljača (izvedeni iz IActionResult)

Tip	Opis rezultata/povratne vrijednosti	Postupak u upravljaču
ViewResult	Prikazuje pogled	View
PartialViewResult	Prikazuje parcijalni pogled	PartialView
RedirectToRouteResult	Privremeno ili trajno preusmjerava zahtjev (HTTP kod 301 ili 302), stvarajući URL na osnovu postavki usmjeravanja	RedirectToAction RedirectToActionPermanent RedirectToRoute RedirectToRoutePermanent
RedirectResult	Privremeno ili trajno preusmjerava rezultat na određeni URL	Redirect RedirectPermanent
ContentResult	Vraća tekstualni sadržaj	Content
FileResult	Vraća binarni sadržaj	File
JsonResult	Vraća objekt serijaliziran u JSON format	Json
JavaScriptResult	Vraća JavaScript odsječak	JavaScript
HttpUnauthorizedResult	Vraća HTTP kod 401	-
HttpNotFoundResult	Vraća HTTP kod 404	HttpNotFound
HttpStatusCodeResult	Vraća određeni HTTP kod	-
EmptyResult	Bez povratne vrijednosti	-

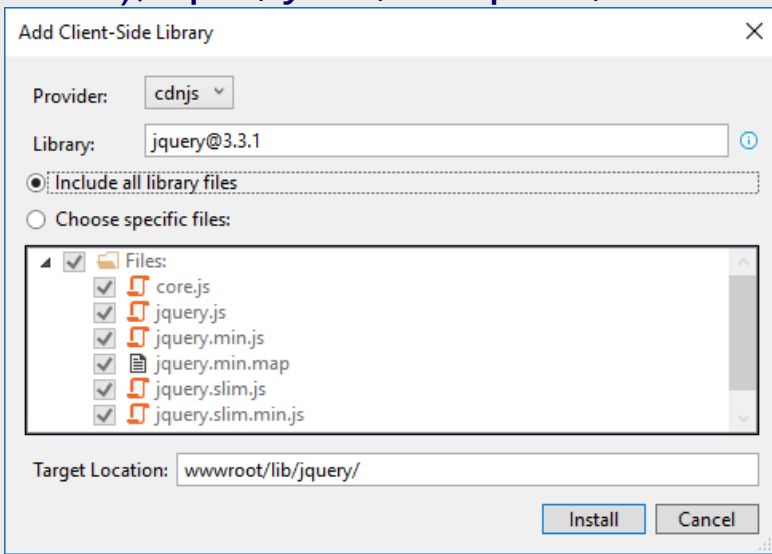
Sintaksa pogleda

- **Ako drugačije nije navedeno koristi se pogled čije ime odgovara pozvanoj akciji, a nalazi se u mapi Views\Pozvani upravljač**
  - Ekstenzija cshtml
  - Pogled predstavlja mješavinu html i mvc koda
  - MVC kod počinje oznakom @ iza kojeg slijedi naredba ili blok naredbi unutar vitičastih zagrada i html oznake
    - koristi se jednostavni kod (npr. petlja, grananje i sl) vezan uz prikaz
    - Komentari oblika @* *@
    - + dodatni atributi za html kontrole (tzv. *tag-helperi*)
  - Početni redak pogleda (opcionalno) sadrži podatak koji se model koristi
    - *@model naziv razreda* koji se koristi za model
    - Konkretne vrijednosti predanog modela dobije se s *@Model*
  - Tekst izvan html oznake prefiksira se s @: ili se stavlja oznaka <text>
  - Prostori imena uključuju se s @using
    - Često korišteni mogu se dodati u datoteku Shared_ViewImports.cshtml
- **Pogled može biti parcijalni**
  - nema html zaglavlje niti koristi glavnu stranicu

BY NC SA

Paketi klijentskih biblioteka

- **Bootstrap, jQuery, …**
- **Način uključivanja**
  - Samostalno skinuti distribuciju i staviti na odgovarajuće mjesto - uobičajeno wwwroot\lib
  - Uključiti direktno preko CDN-a
    - npr. https://code.jquery.com/jquery-3.3.1.js
    - Najjednostavnije i preporuča se za zadaće!
      - nedostatak: nemogućnost izvanmrežnog rada
  - Koristiti neki od alata za (ras)pakiranje klijentskih biblioteka
    - bower (nekoć ugrađen u VS, razvoj napušten), npm, yarn, webpack, LibMan, …
- **LibMan – ugrađen u Visual Studio**
  - Add > Client-Side Library
  - Naknadno: Manage Client-Side Libraries
  - Stvara datoteku libman.json
  - Izvori: cdnjs, unpkg ili neka lokalna mapa



CC BY NC SA

LibMan i paketi klijentskih biblioteka (1)

- **Napomena: Ako se koriste upravljači paketima u datoteku .gitignore dodati da se za mapu wwwroot\lib ne evidentiraju promjene**
  - LibMan automatski obnavlja sadržaj kod svakog korisnika te nakon promjene datoteke libman.json
    - Odredište ovisi o postavkama
  - Primjer: 🗁Web \ .gitignore

```
...
#bower, libman, ...  wwwroot/lib
**/wwwroot/lib
```

CC BY NC SA

LibMan i paketi klijentskih biblioteka (2)

```
{  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [{
      "library": "jquery@3.4.1",
      "destination": "wwwroot/lib/jquery/"
    }, {
      "library": "jquery-ui@1.12.1", "destination": "wwwroot/lib/jquery-ui/"
    }, {
      "provider": "unpkg",
      "library": "jquery-ui@1.12.1",
      "destination": "wwwroot/lib/jquery-ui/",
      "files": ["ui/i18n/datepicker-hr.js"]
    }{
      "library": "jquery-validation-unobtrusive@3.2.11",
      "destination": "wwwroot/lib/jquery-validation-unobtrusive/"
    }, {
      "library": "twitter-bootstrap@4.4.1",
      "destination": "wwwroot/lib/bootstrap/"
    }, {
      "library": "jquery-validate@1.19.1",
      "destination": "wwwroot/lib/jquery-validate/"
    }]}
```

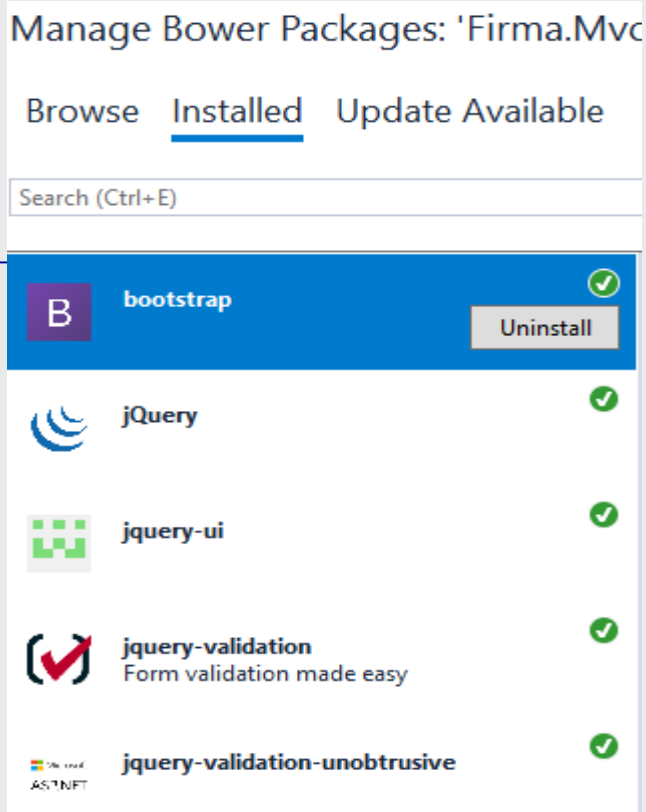
- **Primjeri koji slijede koriste sljedeće klijentske biblioteke**
  - jQuery biblioteke za validaciju na klijentskoj strani, prikaz kalendara umjesto direktnog unosa datuma te za dinamičko uklanjanje i dodavanje redaka prilikom ažuriranja
  - bootstrap za stil i ikone

- Primjer: 🗁Web \ Firma.Mvc \ libman.json

CC BY-NC-SA

Primjer konfiguracijske datoteke za neke druge upravljače paketima

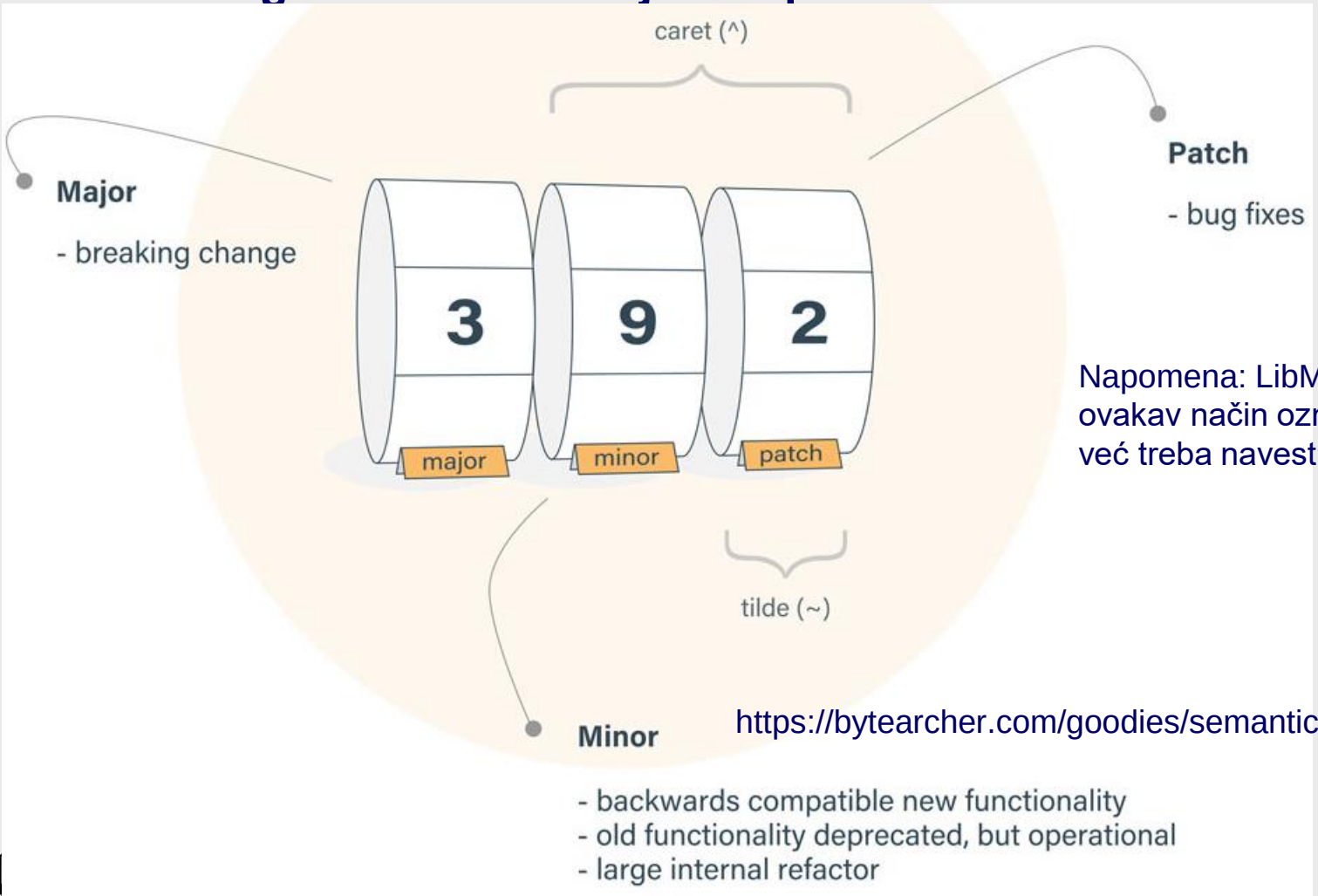
- U slučaju da su se koristili Bower, yarn ili npm tada bi konfiguracijske datoteke bile bower.json odnosno package.json i sadržaj bi bio sljedeći:




```
{
  "private": true,
  "dependencies": {
    "jQuery": "^3.3.1",
    "jquery-ui": "^1.12.1",
    "jquery-validation": "^1.17.0",
    "bootstrap": "^4.0.0",
    "jquery-validation-unobtrusive": "^3.2.9"
  }
}
```

Značenja znakova ^ i ~ u konfiguracijskoj datoteci

- ^ omogućava instaliranje novije verzije od navedene, sve dok je *major version* isti
- ~ omogućava instaliranje zakrpa navedene *minor verzije*



Napomena: LibMan ne podržava ovakav način označavanja paketa, već treba navesti točnu verziju

https://bytearcher.com/goodies/semantic-versioning-cheatsheet/

Uključivanje sadržaja mape wwwroot

- **Sadržaj u mapi wwwroot je namijenjen za statički sadržaj koji se koristi iz aplikacije i u konačnici će biti kopiran na produkcijski server**
- **Potrebno uključiti mapu sa statičkim sadržajem u aplikaciju**
  - Ne mora biti nužno wwwroot
- **Primjer: 🗁Web \ Firma.Mvc \ Startup.cs**

```
public void Configure(...) {
   ...
   app.UseStaticFiles();
```

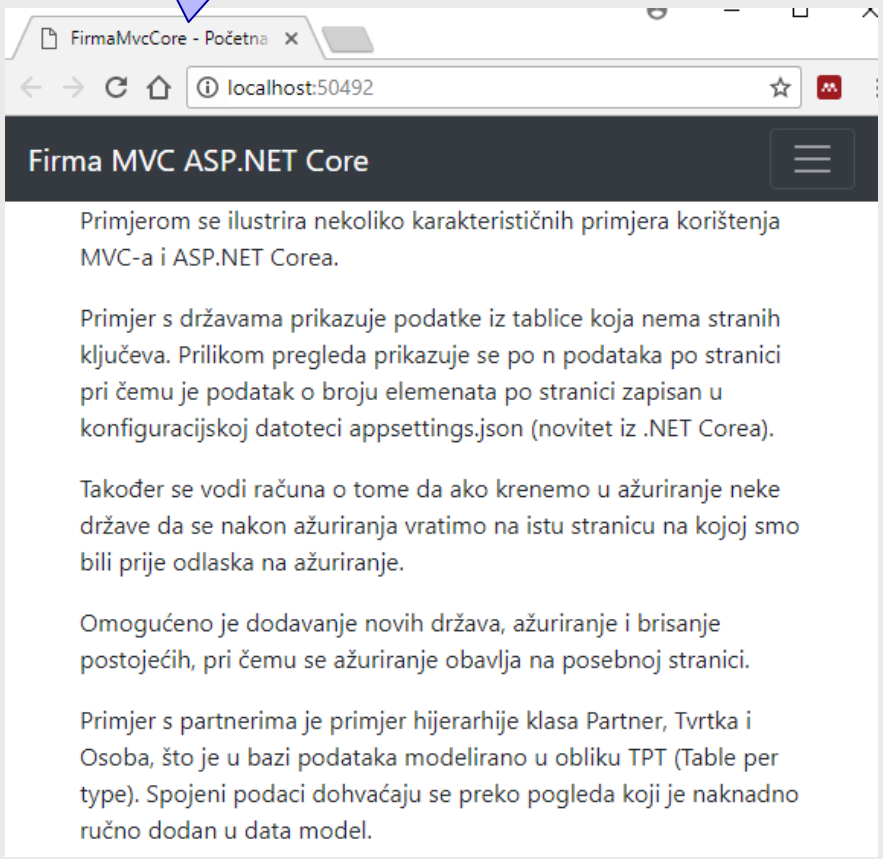
BY NC SA

Početna stranica (2)

Index.cshtml

```
1   @{
2     ViewData["Title"] = "Početna stranica";
3   }
4
5   <p>
6     Primjerom se ilustrira nekoliko karakterist
7   </p>
8   <p>
9     Primjer s državama prikazuje iz podatke koj
10    prikazuje se po n podataka po stranici pri
11    zapisan u konfiguracijskoj datoteci appsett
12  </p>
13  <p>
14    Također se vodi računa o tome da ako krenem
15    na istu stranicu na kojoj smo bili prije od
16  </p>
17  <p>
18    Omogućeno je dodavanje novih država, ažurir
19    ažuriranje obavlja na posebnoj stranici
20  </p>
21  <p>
22    Primjer s partnerima je primjer hijerarhije
```

- **Početna stranica aplikacije sadrži naslov, ali i dijelove koji nisu navedeni u Index.cshtml**
  - Koristi se tzv. glavna stranica koja predviđa okvir u koji pogledi upisuju svoj sadržaj



- Za oblikovanje koristi se Bootstrap i vlastita stilska biblioteka smještena u wwwroot \ css \ site.css

CC BY NC SA

Glavna stranica

- **Pretpostavljena glavna stranica za svaki pogled definirana u datoteci Views \ _ViewStart.cshtml**
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ _ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```

    - Očekuje se postojanje Views \ Shared \ _Layout.cshtml
  - Svaki pogled po potrebi može definirati svoju glavnu stranicu ili je uopće ne koristiti promjenom vrijednosti svojstva *Layout*

```
@{
    Layout = null;
}
```

  - Ako se unutar pojedinog pogleda ne postavi vrijednost za *Layout* koristi se pretpostavljena glavna stranica
- **Glavna stranica uključuje stil i javascript datoteke, definira navigaciju, prostor za javascript pojedinog pogleda…**
  - Konkretni sadržaj za neki zahtjev dobit će se pomoću *RenderBody*

Primjer glavne stranice

- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Shared \ _Layout.cshtml**

```
<!DOCTYPE html>
<html lang="hr_HR" xml:lang="hr_HR"…>
<head>
  <title>FirmaMvcCore - @ViewData["Title"]</title>
  <link rel="stylesheet"
       href="~/lib/bootstrap/css/bootstrap.css" />
  <link rel="stylesheet" href="~/css/site.css" />
  @RenderSection("styles", required: false)
</head><body>
       ... <vc:navigation /> ...
       ...
       @RenderBody()
       ...
  <script src="~/lib/jquery/jquery.js"></script>
  ...
  <script src="~/js/site.js" asp-append-version="true"></script>
   ...
  @RenderSection("scripts", required: false)
</body></html>
```

BY NC SA

Vlastite komponente (1)

- **Za sadržaje koji se pojavljuju na više mjesta, primjerice izbornik**
- **Komponente se izvode iz apstraktnog razreda ViewComponent, a rezultat se priprema u postupku Invoke**
  - Rezultat tipa IViewComponentResult
  - Odgovarajući „pogled” po sintaksi identičan ostalima
- **Primjer: 🗁 …\ ViewComponents \ NavigationViewComponent.cs**
  - Poveznica na trenutni upravljač će biti drugačije označena, pa je u postupku očitana aktualna vrijednost ako postoji
    - `referenca?.član` vraća član objekta na kojeg referenca pokazuje ili null u slučaju da je referenca null
      - referenca.clan bi mogao izvazvati NullReferenceException
    - O ViewBagu-u naknadno

```
public class NavigationViewComponent : ViewComponent {
    public IViewComponentResult Invoke()
    {
      ViewBag.Controller = RouteData?.Values["controller"];
      return View();
    }
}
```

Vlastite komponente (2)

- **Primjer: 🗁 Views\Shared\Components\Navigation\Default.cshtml**

```
<a class="..."
   asp-action="Index" asp-controller="Home">
    Početna stranica
</a>
<a class="nav-link
   @(ViewBag.Controller == "Drzava" ? "active": "")"
   asp-action="Index" asp-controller="Drzava">
    Države
</a>
...
```

- **Poveznice se formiraju tzv. tag-helperima.**
  - Na standardnu HTML oznaku dodaju se atributi asp-action, asp-controller, asp-nazivparametra i slično na osnovu kojih nastane konkretna poveznica
- **Za vlastite tag-helpere i komponente u _*ViewImports.cshtml* dodati**
  - @addTagHelper *, naziv_asemblija npr. @addTagHelper *, Firma.Mvc

CC BY NC SA

Konfiguracijske datoteke

- **Inicijalna konfiguracija uključuje datoteku appSettings.json i podvarijante ovisno o varijabli okruženja te datoteke s „tajnim ključevima”**
- **Dohvat vrijednost vrši se kroz objekt tipa Iconfiguration**
  - Umetnut u konstruktor razreda Startup
  - Primjer: 🗁Web \ Firma.Mvc \ Startup.cs

```
public class Startup
{
   public IConfiguration Configuration { get; }
   public Startup(IConfiguration configuration)
   {
     Configuration = configuration;
   }
```

CC BY NC SA

Dodavanje dodatne konfiguracijske datoteke

- **Po potrebi se može uključiti i neka dodatna datoteka, npr. Json ili slično**

```
public class Program {
   public static IHostBuilder CreateHostBuilder(string[] args)
=>
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((hostingContext, config) =>
        {
           config.AddJsonFile("menus.json",
                                 optional: true,
                                 reloadOnChange: true);
         }...
```

Preslikavanje konfiguracijske datoteke u vlastiti razred (1)

- **Definira se vlastiti razred koji svojom strukturom prati JSON sadržaj ili neki njegov dio iz konfiguracijske datoteke**
- **Primjer: 🗁Web \ Firma.Mvc \ AppSettings.cs**

```
public class AppSettings  {
    public int PageSize { get; set; } = 10;
    public int PageOffset { get; set; } = 10;
}
```

- **Primjer: 🗁Web \ Firma.Mvc \ appsettings.json**

```
{
  "AppSettings": {
    "PageSize": 10,
    "PageOffset": 5,
  }
}
```

CC BY NC SA

Preslikavanje konfiguracijske datoteke u vlastiti razred (2)

- **U početnim postavkama uspostavlja se preslikavanje između dijela konfiguracijske datoteke i vlastitog razreda**
  - Primjer: 🗁Web \ Firma.Mvc \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
  ...
  //application settings
  var appSection = Configuration.GetSection("AppSettings");
  services.Configure<AppSettings>(appSection);
```

Podsjetnik
Stvaranje EF modela na osnovu postojeće BP

1. **Instalirati dotnet-ef na računalu**

```
dotnet tool install --global dotnet-ef
```

2. **U mapi ciljanog projekta izvršiti sljedeće naredbe**

```
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

- ili dodati koristeći opciju Manage NuGet Packages

3. **U naredbenom retku izvršiti sljedeće dvije naredbe**

```
dotnet restore


dotnet-ef dbcontext scaffold
"Server=rppp.fer.hr,3000;Database=Firma;User
Id=rppp;Password=*"
Microsoft.EntityFrameworkCore.SqlServer -o Model -t
Artikl -t Dokument -t Drzava -t Mjesto -t Osoba -t
Partner -t Stavka -t Tvrtka
```

Postavke za spajanje na bazu podataka

- **Automatski generirani razred za EF kontekst sadrži tvrdo kodirane postavke za spajanje na bazu podataka u postupku *OnConfiguring***
  - Navedeni postupak treba obrisati i zamijeniti konstruktorom koji primljeni parametar tipa DbContextOptions proslijedi baznom razredu
  - Konkretni objekt bit će umetnut tehnikom *Dependency Injection*
- **Primjer: 🗁Web \ Firma.Mvc \ Models \ Firma.Context.cs**

```
protected override void OnConfiguring
            (DbContextOptions<FirmaContext> options) { ... }

public FirmaContext(DbContextOptions<FirmaContext> options) :
base(options) { }
```

- **Primjer: 🗁Web \ Firma.Mvc \ Startup.cs**

```
public void ConfigureServices(IServiceCollection services) {
    services.AddDbContext<Models.FirmaContext>(
                options =>   options.UseSqlServer(
                    Configuration.GetConnectionString("Firma")));
```

Akcija dohvata svih država (1)

- **Primjer: 🗁Firma.MVC \ Controllers \ DrzavaController.cs**
  - Upravljač iz primjera vrši dohvat podataka koristeći Entity Framework Core
  - Kontekst primljen u konstruktoru
  - ASP.NET Core instancira pojedini upravljač te na osnovi postavki iz Startup.ConfigureServices, koristeći tehniku *Dependeny Injection*, stvara potrebne argumente
    - Argumenti spremljeni kao varijable dostupne u akciji koja se poziva nakon konstruktora

```
public class DrzavaController : Controller
  {
    private readonly FirmaContext ctx;
    private readonly AppSettings appData;

    public DrzavaController(FirmaContext ctx,
                            IOptionsSnapshot<AppSettings> options)
    {
      this.ctx = ctx;
      appData = options.Value;
    }
```

Akcija dohvata svih država (2)

- **Primjer: 🗁Firma.MVC \ Controllers \ DrzavaController.cs**
  - Upravljač dohvati podatke, napuni model (lista država) i izazove prikaz pogleda s imenom *IndexSimple*.

```
public class DrzavaController : Controller {
  ...
  public IActionResult Index() {
      var drzave = ctx.Drzava
                      .AsNoTracking()
                      .OrderBy(d => d.NazDrzave)
                      .ToList();
      return View("IndexSimple", drzave);
  }
```

CC BY NC SA

Pregled svih država

- **Primjer: 🗁Firma.Mvc \ Views \ Drzava \ IndexSimple.cshtml**
  - Tip modela je IEnumerable<Drzava> (može i List<Drzava>, ali nije potrebno)
  - Za svako mjesto definiran redak tablice s podacima o mjestu
    - Izgled tablice i pojedinog retka definiran stilovima iz Bootstrapa
  - Vrijednost modela može se dohvatiti preko svojstva Model

```
@using Firma.Mvc.Models;
@model IEnumerable<Drzava>
...
<table class="table ... table-striped">
...
@foreach (var drzava in Model)
{
  <tr>
        <td class="text-center">@drzava.OznDrzave</td>
        <td class="text-left">@drzava.NazDrzave</td>
        <td class="text-center">@drzava.Iso3drzave</td>
        <td class="text-center">@drzava.SifDrzave</td>
    ...
```

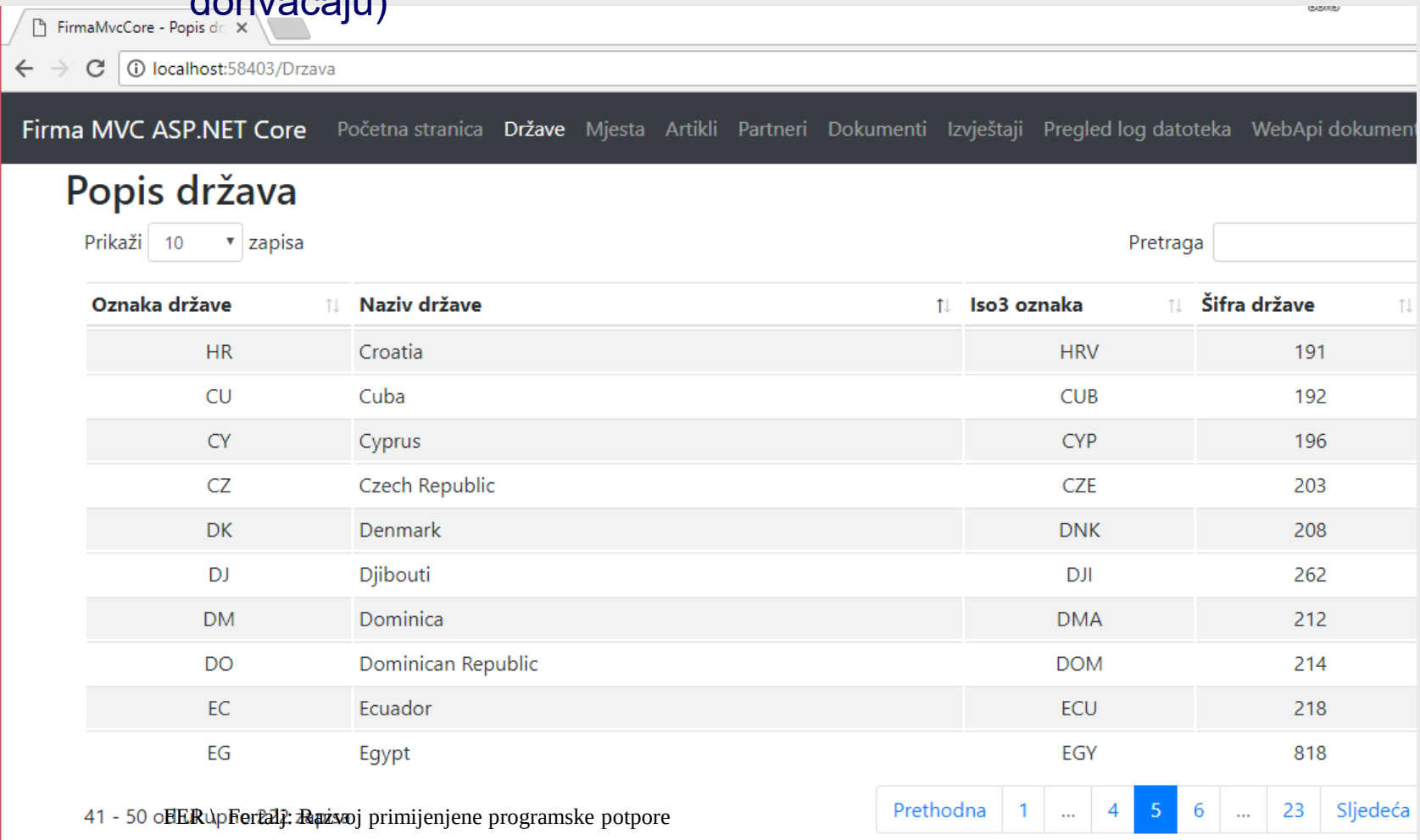
Straničenje na klijentskoj strani (1)

- **Javascript biblioteka https://datatables.net**
  - Dostupna i kao plugin za Bower (i ostale paketne alatne)
  - Aktivira se kodom u jQueryu, nakon što se dokument učita
  - Primjer: 🗁Firma.Mvc \ Views \ Drzava \ IndexSimple.cshtml

```
...
<table class="table ... table-striped" id="tabledrzave">
... </table>
@section styles{
  <link rel="stylesheet"
href="https://.../css/jquery.dataTables.min.css" />
}
@section scripts{
  <script src=".../js/jquery.dataTables.min.js"></script>
<script>
  $(document).ready(function(){
    $('#tabledrzave').DataTable();
  });
</script>
```

Straničenje na klijentskoj strani (2)

- **Brzo, jednostavno i vizualno privlačno rješenje…**
  - (prijevod se može postaviti prilikom inicijalizacije - vidi *IndexSimple.cshtml*)
  - …ali nije prikladno za velike količine podataka (svi podaci se uvijek dohvaćaju)



Straničenje na serverskoj strani

- **Napisati serverske postupke za *DataTables* ili vlastita izvedba?**
  - U ovom primjeru vlastita izvedba, kao motivacija za neke druge principe
- **Primjer: 🗁Firma.Mvc \ ViewModels \ PagingInfo.cs**
  - Razred za informacije o trenutnoj stranici, broju stranica i aktivnom sortiranju

```
public class PagingInfo
{
   public int TotalItems { get; set; }
   public int ItemsPerPage { get; set; }
   public int CurrentPage { get; set; }
   public bool Ascending { get; set; }
   public int TotalPages {
      get {
         return (int)Math.Ceiling(
                (decimal)TotalItems / ItemsPerPage);
      }
   }
   public int Sort { get; set; }
}
```

BY NC SA

URL i model za prikaz svih država

- **Adresa: /Drzava/Index**
  - Upravljač: Drzava(Controller), Akcija: Index
    - Zbog postavki usmjeravanja može i bez navođenja akcije Index
  - Opcionalno se predaje broj stranice i redoslijed sortiranja
    - Npr. /Drzava?page=27&sort=4&ascending=false
- **Primjer: 🗁Firma.Mvc \ ViewModels \ DrzaveViewModel.cs**
  - Model za prikaz država sadrži popis država i podatke o trenutnoj stranici, ukupnom broju stranica i redoslijedu sortiranja (razred *PagingInfo*)

```
namespace Firma.Mvc.ViewModels
{
  public class DrzaveViewModel
  {
    public IEnumerable<Drzava> Drzave { get; set; }
    public PagingInfo PagingInfo { get; set; }
  }
}
```

Parametri sortiranja i straničenja

- **Primjer: 🗁Firma.Mvc \ Views \ Drzava \ Index.cshtml**
  - Omogućeno sortiranje i straničenje
    - Zaglavlje tablice s podacima sadrži poveznice na trenutnu akciju (*Index)* na trenutnom upravljaču s parametrima za broj stranice i način sortiranja
    - Bit će dio adrese stranice
    - Postavlja se atributima asp-route-nazivparametra

```
@model Firma.Mvc.ViewModels.DrzaveViewModel
...
<table class="table table-striped">
  <thead>
    <tr>
      <th>
        <a asp-route-sort="2"
           asp-route-page="@Model.PagingInfo.CurrentPage"
           asp-route-ascending="@(Model.PagingInfo.Sort == 2 ?
!Model.PagingInfo.Ascending : true)">
          Naziv države
        </a>
      </th>...
```

Akcija dohvata podskupa država (1)

- **Primjer: 🗁Firma.MVC \ Controllers \ DrzavaController.cs**
  - Upravljač provjerava postoji li uopće bilo koja država u BP te ako ne postoji preusmjerava rezultat na akciju *Create* uz odgovarajuću poruku
    - O svojstvu *TempData* naknadno

```
public class DrzavaController : Controller {
  ...
  public IActionResult Index(int page = 1, int sort = 1,
                             bool ascending = true) {
      int pagesize = appData.PageSize;

      var query = ctx.Drzava
                  .AsNoTracking();

      int count = query.Count();
      if (count == 0) {
        TempData[Constants.Message] =
                                "Ne postoji niti jedna država.";
        TempData[Constants.ErrorOccurred] = false;
        return RedirectToAction(nameof(Create));
      }
```

Akcija dohvata podskupa država (2)

- **Primjer: 🗁Firma.MVC \ Controllers \ DrzavaController.cs**
  - Formira se podatak o stranicama i sortiranju, a zatim se provjerava je li unutar valjanog raspona te ako nije, preusmjerava se na zadnju stranicu
    - Drugi parametar pri preusmjeravanju je anonimni objekt koji se sastoji od postavki usmjeravanja

```
public class DrzavaController : Controller {
  ...
  public IActionResult Index(int page = 1, int sort = 1,
                             bool ascending = true) {
    ...
    var pagingInfo = new PagingInfo {
      CurrentPage = page, Sort = sort,
      Ascending = ascending, ItemsPerPage = pagesize,
      TotalItems = count
    };
    if (page > pagingInfo.TotalPages) {
      return RedirectToAction(nameof(Index),
        new { page = pagingInfo.TotalPages, sort, ascending});
}
```

Akcija dohvata podskupa država (3)

- **Primjer: 🗁Firma.MVC \ Controllers \ DrzavaController.cs**
  - Upit se formira tako da koristi odabrani sort

```
public class DrzavaController : Controller {
 ...
  public IActionResult Index(int page = 1, int sort = 1,
                             bool ascending = true) {
      var query = ctx.Drzava.AsNoTracking();
      ...
      System.Linq.Expressions.Expression<Func<Drzava, object>>
orderSelector = null;
      switch (sort) {
        case 1: orderSelector = d => d.OznDrzave; break;
        case 2: orderSelector = d => d.NazDrzave; break;
        case 3: orderSelector = d => d.Iso3drzave; break;
        case 4: orderSelector = d => d.SifDrzave;  break;
      }
      if (orderSelector != null)
        query = ascending ?
                query.OrderBy(orderSelector) :
                query.OrderByDescending(orderSelector);
```

Akcija dohvata podskupa država (4)

- **Primjer: 🗁Firma.MVC \ Controllers \ DrzavaController.cs**
  - Nakon pripreme upita, rezultat izvršavanja se pohrani u listu koja postane dio modela

```
public class DrzavaController : Controller {
 ...
  public IActionResult Index(int page = 1, int sort = 1,
                             bool ascending = true) {
       int pagesize = appData.PageSize;
       ...
      var drzave = query
                  .Skip((page - 1) * pagesize)
                  .Take(pagesize)
                  .ToList();

      var model = new DrzaveViewModel {
        Drzave = drzave,
        PagingInfo = pagingInfo
      };

      return View(model);
```

Dodavanje novog podatka

BY NC SA

Postupci za dodavanje novog podatka

- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Drzava \ Index.cshtml**
  - Poveznica na akciju Create na istoimenom upravljaču

```
<a asp-action="Create">Unos nove države</a>
```

- **Dva postupka naziva Create (ali zajednički pogled Create.cshtml)**
  - inicijalno otvaranje forme (GET zahtjev) – trivijalni kod, samo prikaz pogleda
  - prihvat popunjenih podataka (POST zahtjev)
  - određivanje akcije na osnovi atributa *HttpGet* i *HttpPost*
  - Primjer 🗁Web \ Firma.Mvc \ Controllers \ DrzavaController.cs

```
public class DrzavaController : Controller {
  [HttpGet]
  public IActionResult Create(){
       return View();
  }

  [HttpPost]
  public IActionResult Create(Drzava drzava) {
     …
  }
```

Kontrole za unos novog podatka

- **Za svako svojstvo priprema se HTML *input* kontrola za unos**
  - Tip kontrole (text, checkbox, number, datetime, hidden, password, …) automatski se određuje prema tipu svojstva i dodatnim atributima
- **Pogled sadrži poveznicu na popis (odustajanje od unosa) i gumb za slanje popunjenih podataka (*submit form*) na akciju Create**
- **Primjer: 🗁 Firma.Mvc \ Views \ Drzava \ Create.cshtml**

```
@model Drzava //dodati @using Firma.Mvc.Models u _ViewImports.cshtml
...
<form asp-action="Create" method="post">
   ...
    <input asp-for="OznDrzave" .../>
  ...
    <input asp-for="NazDrzave" class="form-control" />
   ...
  <button class="btn btn-primary" type="submit">Dodaj</button>
  <a asp-action="Index" class="btn btn-secondary">Odustani</a>
```

- **Prilikom generiranja stranice stvaraju se atributi id i name**

```
<input type="text" id="NazDrzave" name="NazDrzave" … >
```

CC BY NC SA

Tekst pored kontrola za unos novog podatka

- **Za svako svojstvo ispisuje se prikladno ime svojstva**
  - Primjer: 🗁 Firma.Mvc \ Views \ Drzava \ Create.cshtml

```
@model Drzava
...
<form asp-action="Create" method="post">
   ...
  <label asp-for="NazDrzave"></label>
  <input asp-for="NazDrzave" class="form-control" />
...
```

- **Tekst definiran atributom Display u samom modelu**
  - Dodano naknadno nakon generiranja modela
  - Moguće postaviti i druge podatke, npr. Watermark (Prompt)
  - Primjer: 🗁 Firma.Mvc \ Models \ Drzava.cs

```
public class Drzava{
  ...
  [Display(Name="Oznaka države", Prompt="Unesite naziv")]
  public string OznDrzave { get; set; }
```

Prihvat podataka

- **Postupak može primiti cijeli model i/ili bilo koji broj imenovanih parametara kojima se pokušavaju pridijeliti vrijednosti primljene prilikom zahtjeva**
- **Povezivanje se vrši na osnovu svojstva *name* pojedine html kontrole**

- **Redoslijed povezivanja (prvi koji bude pronađen):**
  1. Request.Form
  2. RouteData.Values
  3. Request.QueryString

  - Dodatnim atributima moguće je eksplicitno odrediti izvor i zanemariti ovaj redoslijed.
    - Detaljnije na https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding

CC BY NC SA

Provjera ispravnosti izvora zahtjeva

- **Atributom *ValidateAntiForgeryToken* provjera se je li zahtjev stigao sa stranice za unos novog podatka ili netko pokušava direktno izvršiti unos s nekog drugog mjesta**
  - skriptom ili kroz neki drugi alat (npr. Chrome Postman, Fidller, …)
  - pojam u literaturi poznat pod nazivom *Cross Site Request Forgery*
  - token generiran prilikom prikaza stranice za unos i zapisan u skriveni element forme pod nazivom __RequestVerificationToken
  - istovremeno kreiran *cookie* s identičnim sadržajem

- **Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs**

```
public class DrzavaController : Controller {
  [HttpPost]
  [ValidateAntiForgeryToken]
  public IActionResult Create(Drzava drzava) {
      ...
  }
```

CC BY NC SA

Postupak za dodavanje novog podatka

- **Podatak se doda u kontekst i pozove snimanje promjena**
  - U slučaju uspjeha, rezultat je preusmjeravanje na popis država
  - U slučaju pogreške, prikazuje se isti pogled, ali s dotad upisanim podacima (sadržani su u modelu)
  - Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs

```
public IActionResult Create(Drzava drzava) {
      ...
      try {
          ctx.Add(drzava);
          ctx.SaveChanges();
          TempData[Constants.Message] =
                   $"Država {drzava.NazDrzave} dodana.";
          TempData[Constants.ErrorOccurred] = false;
          return RedirectToAction(nameof(Index));
      }
      catch (Exception exc) {
          ModelState.AddModelError(string.Empty,
                                   exc.CompleteExceptionMessage());
          return View(drzava);
      }
}
```

Pogreške pri pohrani promjena u EF modelu

- **Pogreške na bazi podataka (npr. narušavanja integriteta određenog primarnim ili stranim ključem ili nekim jedinstvenim indeksom i slično) zamotane u neke druge iznimke.**
  - Potrebno dohvatiti unutarnju iznimku, pa tako rekurzivno dalje
  - Pomoćni postupak napisan u obliku ekstenzije
  - Primjer: 🗁Firma.Mvc \ Extensions \ ExceptionExtensions.cs

```
public static class ExceptionExtensions {
    public static string CompleteExceptionMessage(
                                        this Exception exc)
    {
      StringBuilder sb = new StringBuilder();
      while (exc != null)
      {
        sb.AppendLine(exc.Message);
        exc = exc.InnerException;
      }
      return sb.ToString();
    }
  }
```

BY NC SA

Prijenos podataka između zahtjeva

- **Pogled je vezan za model, a dodatni podaci se mogu prenijeti koristeći *ViewBag* ili *ViewState* (primjeri naknadno)**
- **Nakon uspješnog dodavanja nove države dolazi do preusmjeravanja na akciju pregleda svih država**
  - *ViewBag* ili *ViewState* nisu pogodni, jer se ne iscrtava pojedini pogled nego dolazi do preusmjeravanja
- **Poruke između zahtjeva stavljaju se u *TempData***
  - Podaci iz *TempData* se brišu nakon konzumiranja
    - Interno se koristi sjednica (session), tj. podaci pohranjeni unutar sjednice
  - *TempData* staviti u pogled ili u glavnu stranicu
  - Primjer: 🗁Firma.Mvc \ Views \ Shared \ _Layout.cshtml

```
@if (TempData[Constants.Message] != null) {
       bool error = false;
       var obj = TempData[Constants.ErrorOccurred];
       if (obj != null)
             error = (bool)obj;
      <div class="alert @(error ? "alert-danger" : "alert-
success")">@TempData[Constants.Message]</div>
```

Validacija podataka

BY NC SA

Validacijski atributi

- **Neispravni ili nepotpuni podaci o državi će uzrokovati pogrešku prilikom pohrane u bazi podataka**
  - Poželjno zaustaviti neispravne podatke što je moguće prije
- **Jednostavna validacijska pravila moguće je implementirati korištenjem atributa izvedenih iz *ValidationAttribute***
- **Required, MaxLength, Range, RegularExpression …**
  - Sadrže i odgovarajući javascript kôd koji prikazuje pogreške odmah prilikom unosa podatka i onemogućava slanje podataka na server
- **Primjer: 🗁Firma.Mvc \ Models \ Drzava.cs**

```
public partial class Drzava {
  [Required(ErrorMessage="Oznaka države je obvezno polje")]
  public string OznDrzave { get; set; }
  [Required(ErrorMessage = "Naziv države je obvezno polje")]
  public string NazDrzave { get; set; }
  [MaxLength(3, ErrorMessage="ISO3 oznaka može sadržavati
maksimalno 3 slova")]
  public string Iso3drzave { get; set; }
  ...
```

Provjera ispravnosti modela prije snimanja

- **Validacija na klijentskoj strani se može lako zaobići**
- **Prije pohrane, validaciju napraviti i na serveru**
- **Provjera se vrši koristeći svojstvo ModelState.IsValid**
  - Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Drzava drzava)
{
  if (ModelState.IsValid)
  {
    ... Dodaj državu ...
  }
  else
  {
    return View(drzava);
  }
}
```

  - *Digresija: U slučaju vraćanja primljenog modela za rekonstrukciju vrijednosti pomoću tag-helpera koristi se ModelState, a ne Model*

CC BY NC SA

Prikaz validacijskih pogrešaka

- **Validacijska pogreška pojedinog svojstva prikazuje se u html kontroli (obično span) s dodatnim atributom asp-validation-for**
- **Sumarne validacijske pogreške se prikazuju dodavanjem atributa asp-validation-summary="[All/ModelOnly/None]" nekoj kontroli (obično div)**
  - All – pogreške modela, ali i pojedinih svojstava
  - ModelOnly – pogreške na razini cijelog modela, ali ne i za pojedina svojstva
- **Primjer: 🗁Firma.Mvc \ Views \ Drzava \ Create.cshtml**

```
@model Drzava
...
<form asp-action="Create" method="post">
  <div asp-validation-summary="All"></div>
  <div class="form-group">
    <label asp-for="OznDrzave"></label>
    <div><span asp-validation-for="OznDrzave" class="text-
danger" /></div>
    <input asp-for="OznDrzave" class="form-control" />
  ...
    <span asp-validation-for="NazDrzave" ...
```

BY NC SA

Stil prikaza validacijskih pogrešaka

- **MVC u slučaju validacijske pogreške postavlja odgovarajuću css klasu na html element vezan uz svojstvo s pogreškom**
- **Nazivi css stilova za validacijske pogreške:**
  - *.input-validation-error* – stil kontrole za unos podatka
  - *.field-validation-error* – stil teksta za poruku o pogrešci pojedinog svojstva
  - *.validation-summary-errors*– stil teksta sa sumarnim pogreškama

  - U primjeru postavljen stil tako da neispravna polja budu obrubljena crvenom bojom, a tekst sumarnih pogrešaka prikazan nijansom crvene i podebljano
  - Primjer: 🗁Firma.Mvc \ Views \ Drzava \ Create.cshtml

```
.validation-summary-errors {
      font-weight: bold;
      color:#a94442;
}
.input-validation-error{
    border-color:red;
}
```

Validacija na klijentskoj strani

- **Dio pogrešaka moguće odmah otkriti na klijentskoj strani čime se poboljšava korisnički dojam pri radu s aplikacijom**
  - ne treba slati podatke na server i čekati odziv da se prikaže pogreška
- **Koristi se Microsoftova klijentska biblioteka *jQuery Unobtrusive Validation* kao dodatak na *jQuery Validate***
- **Potrebno uključiti za pojedinu stranicu ili na glavnoj stranici**
  - Umjesto navođenja cijele putanje, može se koristiti *tag-helper* asp-src-include i zamjenski znakovi
  - Primjer: 🗁Firma.Mvc \ Views \ Shared \ _Layout.cshtml

```
...
  <script asp-src-include="~/lib/jquery-
validate/**/jquery.validate.min.js"></script>

  <script asp-src-include="~/lib/jquery-validation-
unobtrusive/**/*.min.js"></script>
...
</body>
</html>
```

Brisanje podatka

CC BY NC SA

Forma za brisanje podatka

- **Za brisanje treba koristiti POST postupak**
  - GET postupak se preporuča za postupke koji ne mijenjaju stanja objekta ili aplikacije
- **Potrebno stvoriti po jednu POST formu i jedan *submit* gumb za svaku državu**
  - HTML oznaka form proširena tag-helperima za postavljanje parametara zahtjeva
  - U glavnoj stranici uključena skripta site.js koja svakoj kontroli sa css stilom *delete* dodaje kod za potvrdu brisanja
  - Identifikator države kao skriveno polje
- **Primjer: 🗁Firma.Mvc \ Views \ Drzava \ Index.cshtml**

```
@foreach (var drzava in Model.Drzave) {
  ...
  <form asp-action="Delete" method="post"
        asp-route-page="@Model.PagingInfo.CurrentPage"
        asp-route-sort="@Model.PagingInfo.Sort"
        asp-route-ascending="@Model.PagingInfo.Ascending">
    <input type="hidden" name="OznDrzave"
                                value="@drzava.OznDrzave" />
    <button type="submit" title="Obriši" class="delete ...">
        ...</button>
  </form>
```

Akcija brisanja

- **Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs**
  - Brisanje nema vlastiti pogled
  - Nakon odrađivanja posla, preusmjerava rezultat na neku akciju
  - *TempData* sadrži tekst pogreške ili poruku o uspjehu

```
[HttpPost]
public IActionResult Delete(string OznDrzave, ...) {
   var drzava = ctx.Drzava.AsNoTracking()
                  .Where(d => d.OznDrzave == OznDrzave)
                  .SingleOrDefault();
   ...
   try {
     ctx.Remove(drzava);
     ctx.SaveChanges();
     ...
   }
   catch (Exception exc) {
     TempData[Constants.Message] = ...
   }
   return RedirectToAction(nameof(Index), ...
```

BY NC SA

Potvrda brisanja podatka

- **Pri svakom kliku na kontrolu koja ima definiran css stil *delete* traži se potvrda korisnika**
- **Implementira se koristeći *on* iz jQuerya s događajem click**
  - za razliku od $(".delete").click(…) ovo se odnosi i na elemente koji će se pojaviti u budućnosti dinamičkim učitavanjem
- **Primjer: 🗁 Firma.Mvc \ wwwroot \ js \ site.js**

```
$(function () { //nakon što je stranica učitana
  $(document).on('click', '.delete', function (event) {
    if (!confirm("Obrisati zapis?")) {
      event.preventDefault();
    }
  });
});
```

- **Primjer: 🗁 Firma.Mvc \ Views \ Shared \ _Layout.cshtml**

```
  ...
  <script src="~/js/site.js" asp-append-version="true"></script>
  ...
 </body></html>
```

Ažuriranje jednostavnog podatka

BY NC SA

Poveznica za ažuriranje države

- **Primjer: 🗁Firma.Mvc \ Views \ Drzava \ Index.cshtml**
  - Ažuriranje vodi na akciju *Edit* u upravljaču *Drzava*
  - Postupak *Edit* u *DrzavaController* očekuje *id* kao identifikator države
    - *id* se postavlja na konkretni Ozn*Drzave* prilikom stvaranja poveznice
    - dodatni parametri su informacije o trenutnoj stranici i načinu sortiranja da se ne izgubi povratkom na popis država

```
@model DrzaveViewModel
...
@foreach (var drzava in Model.Drzave) {
    ...
    <a asp-action="Edit"
       asp-route-id="@drzava.OznDrzave"
       asp-route-page="@Model.PagingInfo.CurrentPage"
       asp-route-sort="@Model.PagingInfo.Sort"
       asp-route-ascending="@Model.PagingInfo.Ascending"
       class="btn btn-sm" title="Ažuriraj">
       <i class="fas fa-edit"></i>
    </a>...
}
```

CC BY NC SA

Akcije ažuriranja podataka

- **Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs**
  - Dva postupka Edit i Update koji se odazivaju na istu akciju
  - *Edit* predstavlja inicijalno otvaranje forme (GET zahtjev),
  - *Update* naknadno slanja popunjenih podataka (POST zahtjev)
    - U ovom primjeru oba postupka imaju iste argumente, pa moraju imati različite nazive
    - Postupci primaju i podatke o trenutnoj stranici i načinu sortiranja tako da se mogu vratiti na pravu stranicu nakon ažuriranja

```
public class DrzavaController : Controller {
  [HttpGet]
  public IActionResult Edit(String id, int page = 1,
                            int sort = 1, bool ascending = true){
      ...
  }
  [HttpPost, ActionName("Edit")]
  public async Task<IActionResult> Update(String id,
      int page = 1, int sort = 1, bool ascending = true) {
    …
  }
```

BY NC SA

Prijenos dodatnih vrijednosti pogledu

- **Osim samog modela ponekad je potrebno prenijeti i neke druge vrijednosti**
  - poruke o pogrešci
  - izbor vrijednosti za padajuću listu država
  - informacije o stranici i načinu sortiranja
  - željeni naslov stranice (koristi se na glavnoj stranici)
- **Mogu se koristiti svojstva upravljača ViewData ili ViewBag**
  - rade nad istim podacima
- **ViewData (tipa ViewDataDictionary)**
  - sadrži parove ključ (string) , vrijednost (objekt)
- **ViewBag (tipa dynamic)**
  - Može sadržavati bilo koje svojstvo (nema sintaksne provjere prilikom kompilacije)

Prikaz stranice za ažuriranje podataka

- **Nalik pogledu za unos podatka**
  - Postojeći podaci se automatski stavljaju kao *value* kontrole `<input asp-for = "nazivsvojstva" ...`
- **Identifikator podatka (obično primarni ključ) se ne mijenja i za njega se ne generira kontrola, već se koristi skriveno polje ili (u ovom primjeru) je dio rute (adrese zahtjeva)**
  - Primjer: 🗁Firma.Mvc \ Views \ Drzava \ Edit.cshtml

```
<form asp-route-page="@ViewBag.Page"
      asp-route-sort="@ViewBag.Sort"
      asp-route-ascending="@ViewBag.Ascending„
       asp-route-id="Model.OznDrzave"
      method="post">
  <div asp-validation-summary="All"></div>
  <div class="form-group">
    <label asp-for="NazDrzave"></label>
    <div><span asp-validation-for="NazDrzave"
class="text-danger"></span></div>
    <input asp-for="NazDrzave" class="form-control" />
    ...
```

Priprema stranice za ažuriranje

- **U slučaju neispravnog identifikatora vratiti *NotFound***
  - korisniku se prikazuje pogreška 404
- **Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs**

```
public class DrzavaController : Controller {
  [HttpGet]
   public IActionResult Edit(String id,
          int page = 1, int sort = 1, bool ascending = true) {
      var drzava = ctx.Drzava.AsNoTracking()
                     .Where(d => d.OznDrzave == id)
                     .SingleOrDefault();
      if (drzava == null) {
        return NotFound("Ne postoji država s oznakom: " + id);
      }
      else {
        ViewBag.Page = page; ViewBag.Sort = sort;
        ViewBag.Ascending = ascending;
        return View(drzava);
      }
    }
```

BY NC SA

Prihvat vrijednosti za ažuriranje

- **Nekoliko tehnika ažuriranja podatka**
  - prihvat kompletnog podatka pa kopčanje u kontekst i snimanje promjena
  - odabir svojstava koje će se povezati
  - dohvat podatka iz baze podataka u kontekst i ažuriranje svojstva
  - Detaljnije na https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/crud#update-the-edit-page
- **U primjeru koji slijedi koristit će se varijanta dohvata podatka iz baze podataka i ažuriranje samo određenih svojstava**
  - Koristi se asinkroni postupak *TryUpdateModelAsync*
  - Preopterećeni postupak
  - Koristit će se ona varijanta u kojoj se navode svojstva koja se žele izmijeniti temeljem podataka pristiglih s forme
    - Preciznije, umjesto navođenja naziva svojstava navodit će se lambda izrazi kojim se selektira neko svojstvo

Asinkroni postupci (ukratko)

- **Razred Task koristi se za opisivanje postupka koji će se izvršiti u nekoj drugoj dretvi**
  - Ako postupak vraća neki rezultat tipa T tada se postupak definira kao Task<T>
- **Čekanje dovršetka asinkronog zadatka i dohvat vrijednosti po završetku vrši se naredbom await**
  - Kôd iza await nastavlja se izvršavati tek nakon dovršetka zadatka
  - Pozivatelj čeka na dovršetak zadatka, ali se istovremeno omogućava grafičkoj dretvi ili web serveru da reagira na druge događaje što nije slučaj s klasičnim postupkom Wait
    - U ovom primjeru duže čekanje bi onemogućilo web serveru da posluži neki drugi zahtjev
- **Postupak u kojem se koristi *await* mora se označiti s *async***
  - *Napomena: Povratna vrijednost iz postupka oblika async Task<TResult> je tipa TResult.*

CC BY NC SA

Prihvat vrijednosti za ažuriranje

- **Primjer: 🗁Firma.Mvc \ Controllers \ DrzavaController.cs**

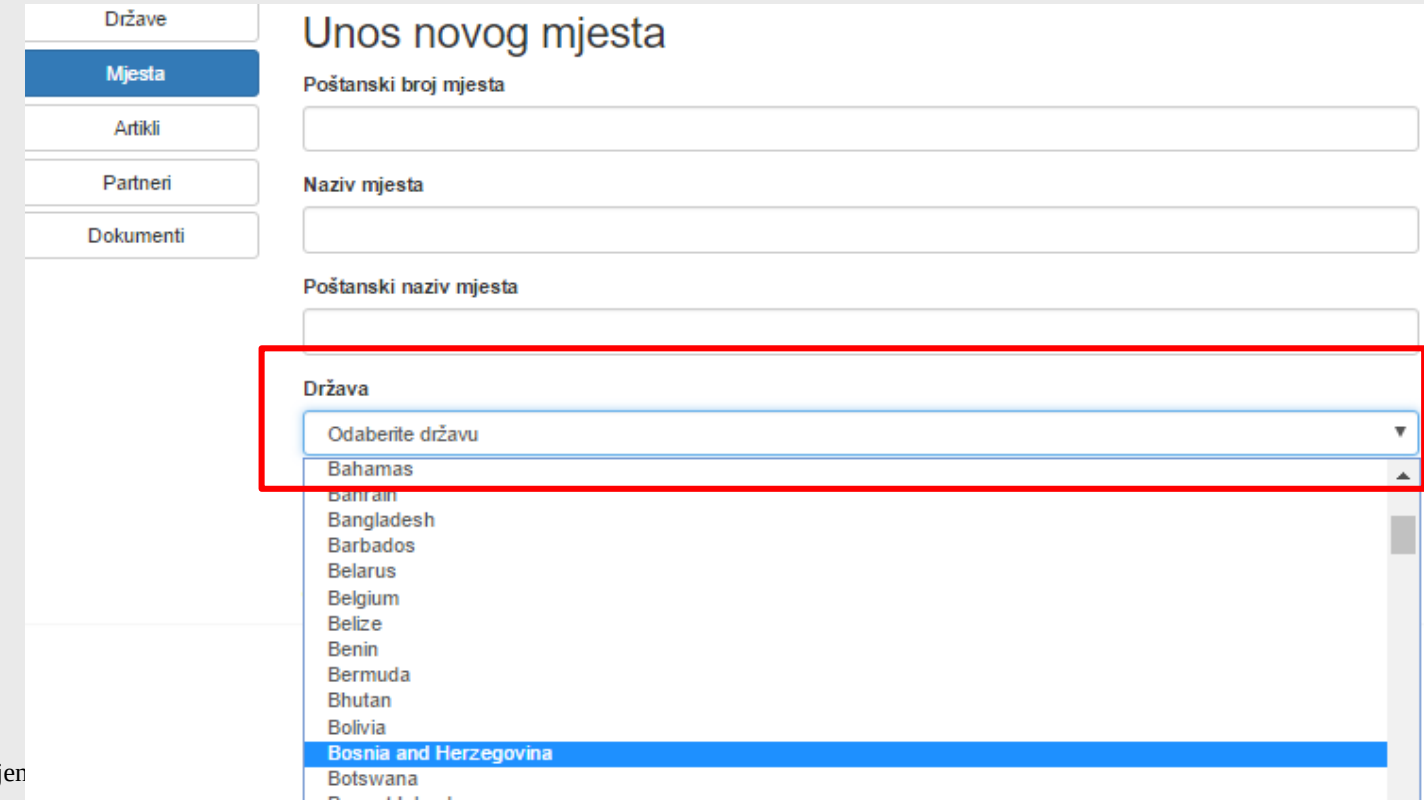
```
public class DrzavaController : Controller {
  [HttpPost, ActionName("Edit")]
  public async Task<IActionResult> Update(String id,
       int page = 1, int sort = 1, bool ascending = true) {
     …
     Drzava drzava = await ctx.Drzava
                           .Where(d => d.OznDrzave == id)
                           .FirstOrDefaultAsync();
     if (drzava == null) {
        return NotFound("Neispravna oznaka države: " + id);
     }

     if (await TryUpdateModelAsync<Drzava>(drzava, "",
       d => d.NazDrzave, d => d.SifDrzave, d => d.Iso3drzave)){
          ...
        await ctx.SaveChangesAsync();
       ...
```

Odabir vrijednosti iz ograničenog skupa vrijednosti

CC BY NC SA

Primjer unosa objekta s ograničenim skupom vrijednosti za neko svojstvo

- **Mjesto ima strani ključ na tablicu Država**
- **Umjesto unosa šifre države omogućiti korisniku da odabere državu iz popisa država.**
- **Popis država nije dio modela, već se prenosi koristeći *ViewBag* ili *ViewData***
  - nije dio modela, jer bi se inače prenosio i natrag u upravljač što nema smisla.



CC BY NC SA

Priprema podataka za padajuću listu

- **Na osnovu liste država stvara se objekt tipa *SelectList***
  - Navodi se izvor podataka, svojstvo koje predstavlja vrijednost odabranog elementa i svojstvo koje se prikazuje kao tekst u padajućoj listi
    - Umjesto tvrdo kodiranog naziva svojstva koristi se operator *nameof*
  - Stvoreni objekt se pogledu prenosi koristeći ViewBag (ili ViewData)
    - Može se upotrijebiti proizvoljno ime za novu vrijednost u ViewBagu
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ MjestoController.cs

```
[HttpGet]
public IActionResult Create() {
      PrepareDropDownLists();
      return View();
}
private void PrepareDropDownLists() {
   var drzave = ctx.Drzava
                .OrderBy(d => d.NazDrzave)
                .Select(d => new { d.NazDrzave, d.OznDrzave })
                .ToList();
   ViewBag.Drzave = new SelectList(drzave,
          nameof(Drzava.OznDrzave), nameof(Drzava.NazDrzave));
```

BY NC SA

Prikaz padajuće liste u pogledu

- **Padajuća lista se koristi unutar HTML oznake *select*, pri čemu se izvor navodi atributom *asp-items***
  - Potrebno i navesti *asp-for* za određivanje svojstva kojem će se odabir pridružiti
- **Moguće umetnuti i dodatne elemente u padajuću listu (osim onih koji su već pripremljeni).**
  - Dodaju se na početak
  - Npr. poruka da se odabere neki element iz liste koja je inicijalno odabrana
    - Nakon što se jednom promijeniti vrijednost (zbog atributa *disabled*) više se neće moći ponovo odabrati element s porukom
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Create.cshtml**

```
<select class="form-control" asp-for="OznDrzave"
                             asp-items="ViewBag.Drzave">
     <option disabled selected value="">
            Odaberite državu
     </option>
</select>
```

BY NC SA

Ponovna priprema podataka za padajuću listu

- **U slučaju neispravnih ili nepotpunih podataka potrebno je ponovno prikazati pogled za unos, ali i pripremiti podatke za padajuću listu**
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
 public IActionResult Create(Mjesto mjesto) {
     if (ModelState.IsValid) {
       try {
         ...
       }
       catch (Exception exc) {
         ...
         PrepareDropDownLists();
         return View(mjesto);
       }
     }
     else {
       PrepareDropDownLists();
       return View(mjesto);
     }
 }
```

Web-aplikacije (ASP.NET Core MVC)

Kreiranje vlastitog *tag-helpera*

2019/20.06-A

CC BY NC SA

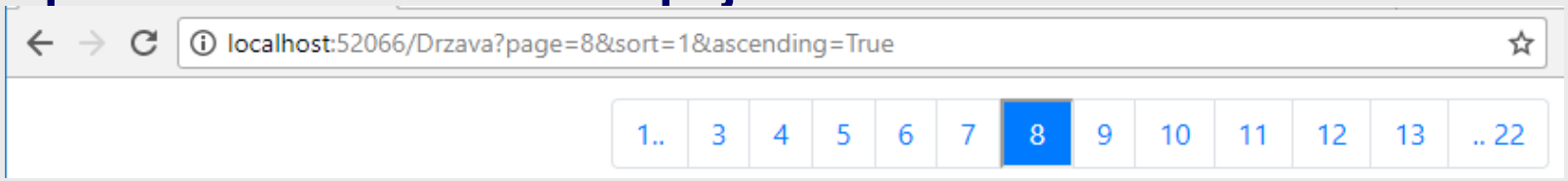
Napomena vezana za primjere koji slijede

- **Primjeri u ovim predavanjima predstavljaju složene koncepte koji uključuju korištenje CSS stilova i *jQueryja***
- **Primjeri su opsežni i ne mogu u potpunosti stati na slajdove te se preporuča isprobati primjer i detaljno proučiti programski kôd pri čemu slajdovi mogu poslužiti kao vodilja kojim redom proučavati primjere**

BY NC SA

Kreiranje poveznica za straničenje

- **Potrebno izraditi vlastitu komponentu/kontrolu koja će omogućiti prikaz poveznica za odlazak na pojedinu stranicu**



- **Željena funkcionalnost**
  - Trenutnu stranicu prikazati drugačijim stilom
  - Prikazati poveznice na prethodnih n i na sljedećih n stranica
    - n je parametar u konfiguracijskoj datoteci
  - U slučaju da je prva, odnosno zadnja stranica udaljena za više od n od trenutne stranice prikazati i poveznicu na prvu, odnosno zadnju stranicu
  - Omogućiti unos broja željene stranice
    - npr. klikom na trenutnu stranicu i upisom broja
    - Izvršiti provjeru unesenog broja prije preusmjeravanja

  - Komponenta mora raditi s bilo kojim entitetom iz primjera, a ne samo s državama i treba voditi evidenciju o trenutnom načinu sortiranja
    - Koristi se informacija iz razreda *PagingInfo*

BY NC SA

Podsjetnik na (neke) ugrađene *tag-helpere*

- **Želi li se stvoriti poveznica za akciju Show u upravljaču Contact gdje je vrijednost parametra *name* jednaka Pero Perić ugrađeni tag helperi će se primijeniti na standardnu HTML-ovu oznaku *a***

```
<a asp-controller="Contact"
   asp-action="Show"
   asp-route-name="Pero Perić"
     ...>Pero Perić</a>
```

  - Atributi asp-controller, asp-action, asp-route-naziv i slično se interpretiraju prilikom iscrtavanja pojedinog pogleda

- **Komponenta za straničenje bit će izvedena pomoću vlastitog *tag-helpera* s vlastitim atributima čija se vrijednosti dohvaćaju u kodu *tag-helpera* i uzrokuju generiranje potrebnih poveznica ispod nekog div elementa u HTML-u.**

```
<pager vlastitiatribut1="vrijednost"
     vlastitiatribut2="vrijednost" ... ></pager>
```

CC BY NC SA

Izrada vlastitog *tag-helpera*

- **Stvara se izvođenjem iz razreda *TagHelper***
- **Atributom *HtmlTargetElement* navodi se na koju HTML oznaku se može primijeniti (ako ne na onu koja odgovara nazivu tag-helpera) te koji skup atributa je obvezan**
  - Može se navesti skup obveznih atributa (odvajaju se zarezom)
- **Argumenti konstruktora stvaraju se koristeći *DependencyInjection***
  - Primjer: 🗁 Web \ Firma.Mvc \ TagHelpers \ PagerTagHelper.cs

```
[HtmlTargetElement(Attributes = "page-info")]
 public class PagerTagHelper : TagHelper {

    private readonly IUrlHelperFactory urlHelperFactory;
    private readonly AppSettings appData;

    public PageLinkTagHelper(
       IUrlHelperFactory helperFactory,
       IOptionsSnapshot<AppSettings> options)
    {
      urlHelperFactory = helperFactory;
      appData = options.Value;
    }
```

BY NC SA

Uključivanje vlastitog *tag-helpera*

- **Može se uključiti u pojedinom pogledu ili za sve poglede navođenjem u *_ViewImports.cshtml***

- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ _ViewImports.cshtml**

```
@using Firma.Mvc
@using Firma.Mvc.Models
@using Firma.Mvc.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper Firma.Mvc.TagHelpers.*, Firma.Mvc
```

Svojstva željenog *tag-helpera* (1)

- ***Tag-helper* će se primjenjivati na HTML oznaku div i imat će sljedeće vlastite atribute**
  - page-info: informacija o trenutnoj stranici i načinu sortiranja
  - page-action: akcija na koju poveznica vodi
  - page-title: tekst za tooltip za unos željene stranice
- **Korišteni stilovi od Bootstrapa**
  - Za polje za prikaz trenutne stranice i unos željene korišten vlastiti css stil pagebox
- **Primjer korištenja: 🗁 … Firma.Mvc \ Views \ Drzava \ Index.cshtml**

```
<pager page-info="@Model.PagingInfo"
     page-action="Index"
     page-title="Unesite željenu stranicu"
     class="float-right">
</pager>
```

Svojstva željenog *tag-helpera* (2)

- **Atributi *tag-helpera* navode se kao svojstva pri čemu se umjesto naziva s crticama koristi naziv u notaciji PascalCase**
  - Za svojstva koja nisu predviđena za korištenje u pogledu iznad svojstva se stavlja atribut *HtmlAttributeNotBound* (npr. kontekst konkretne upotrebe)
- **Primjer: 🗁 Web \ Firma.Mvc \ TagHelpers \ PagerTagHelper.cs**

```
[HtmlTargetElement(Attributes = "page-info")]
 public class PagerTagHelper : TagHelper {
    ...
    [ViewContext]
    [HtmlAttributeNotBound]
    public ViewContext ViewContext { get; set; }

    public PagingInfo PageInfo { get; set; }
    public string PageAction { get; set; }
    public string PageTitle { get; set; }
    ...
```

BY NC SA

Rezultat *tag-helpera*

- **Rezultat se priprema u nadjačanom postupku Process**
  - Kreira HTML na osnovi konteksta i vrijednosti svojstva. Umjesto direktnog stvaranja HTML-a koristi se razred *TagBuilder* za određenu HTML oznaku.
  - Sadržaj koji se ispiše pomoću *output.Content.AppendHtml* upisuje se u HTML oznaku na kojoj je atribut primijenjen
    - U primjeru oznaku pager promijeni u div
- **Primjer: 🗁 Web \ Firma.Mvc \ TagHelpers \ PagerTagHelper.cs**
  - Vlastiti postupak *BuildTagForPage* opisan na sljedećem slajdu

```
public override void Process(TagHelperContext context,
                             TagHelperOutput output) {
    int offset = appData.PageOffset;
    TagBuilder paginationList = new TagBuilder("ul");
    paginationList.AddCssClass("pagination");
    navTag.InnerHtml.AppendHtml(paginationList);
       ... Poveznice za trenutnu stranicu i +,- offset
    ...
    output.TagName = "nav";
    output.Content.AppendHtml(paginationList);
```

Kreiranje poveznica *tag-helperom*

- **Za kreiranje adrese koristi se objekt tipa *IUrlHelper* i anonimni razred s vrijednostima parametara**
  - Može se dobiti na osnovi trenutnog konteksta i objekta tipa *IUrlHelperFactory*
  - Primjer: 🗁 Web \ Firma.Mvc \ TagHelpers \ PagerTagHelper.cs

```
private TagBuilder BuildListItemForPage(int i, string text) {
      IUrlHelper urlHelper =
             urlHelperFactory.GetUrlHelper(ViewContext);
      TagBuilder a = new TagBuilder("a");
      a.InnerHtml.Append(text);
      a.Attributes["href"] = urlHelper.Action(PageAction,
             new { page = i, sort = PageInfo.Sort,
              ascending = PageInfo.Ascending });
      tag.AddCssClass("page-link");

      TagBuilder li = new TagBuilder("li");
      li.AddCssClass("page-item");
      li.InnerHtml.AppendHtml(a);
      return li;
}
```

Kreiranje poveznice za trenutnu stranicu (1)

- **Po uzoru na poveznicu za prvu stranicu kreiraju se i poveznice za stranice od** `max{1, trenutna - n} do min{ukupno, trenutna + n}`
  - Za trenutnu stranicu ne kreira se oznaka *a*, već tekstualni okvir
  - Postavljaju se dodatni atributi oblika data-naziv (vidi sljedeći slajd)
  - Stil *pagebox* služi da ga se *jQueryjem* može pronaći u dokumentu, a ujedno je stil definiran u stilskoj datoteci kako bi se odredila širina tog okvira
- **Primjer: 🗁 Web \ Firma.Mvc \ TagHelpers \ PagerTagHelper.cs**

```
  IUrlHelper urlHelper = urlHelperFactory
                         .GetUrlHelper(ViewContext);
  TagBuilder input = new TagBuilder("input");
  input.Attributes["type"] = "text";
  input.Attributes["value"] = broj trenutne stranice
    ... Postavljanje dodatnih atributa ...
  input.AddCssClass("page-link");
  input.AddCssClass("pagebox"); //za identifikaciju i širinu
  TagBuilder li = new TagBuilder("li");
  li.AddCssClass("page-item active");
  li.InnerHtml.AppendHtml(input);
  return li;
}
```

Kreiranje poveznice za trenutnu stranicu (2)

- **Za provjeru ispravnosti korisnikovog unosa potrebno je evidentirati dozvoljeni raspon stranica**
  - Evidentira se u atributima oblika data-naziv te se za dohvat vrijednosti koristi postupak *.data(naziv)* iz jQuerya
- **Korisnik će unijeti željeni broj stranice i on se treba ugraditi u odredišnu adresu. Mjesto gdje bi trebao stajati broj stranice prepoznaje se po nekom specijalnom nizu znakova**
  - U ovom primjeru -1, ali može biti i i nešto drugo
- **Primjer: 🗁 Web \ Firma.Mvc \ TagHelpers \ PagerTagHelper.cs**

```
TagBuilder input = new TagBuilder("input");
...
input.Attributes["data-current"] = text;
input.Attributes["data-min"] = "1";
input.Attributes["data-max"] = PageInfo.TotalPages.ToString();
input.Attributes["data-url"] = urlHelper.Action(PageAction,
   new { page = -1, sort = PageInfo.Sort,...});
...
```

CC BY NC SA

Označavanje sadržaja kontrole za unos stranice

- **Klikom na kontrolu koja prikazuje trenutnu stranicu, a ujedno služi i za unos željene stranice njen sadržaj se automatski označi**
  - Implementira se obradom događaja klika gumba
  - Na izvoru događaja poziva se postupak select()
  - Kontrolu prepoznamo po postojanju stila pagebox
- **Povezivanje događaja i obrada događaja odvija se nakon što je cijeli dokument učitan**
  - Konstrukcija $(function() { … }); u jQueryju
- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ gotopage.js**

```
$(function () { //nakon što je dokument učitan
  $('.pagebox').click(function () {
      //obrada klika na sve kontrole koje imaju stil pagebox
      $(this).select(); //selektiraj sadržaj
  });
```

Akcije na tipke Enter i ESC u pregledniku

- **Nakon svakog otpuštenog znaka na tipkovnici (događaj *keyup*) provjerava se radi li se o tipkama *enter* (kôd 13) ili *escape* (kôd 27)**
  - Aktiviraju prijelaz na novu stranicu (prethodno zamijenivši -1 s valjanim upisanim brojem), odnosno vraćaju originalnu vrijednost
  - Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ gotopage.js

```
$('.pagebox').keyup(function (event) {
   var keycode = event.which;
   var pageBox = $(this);
   if (keycode == 13) {
      if (validRange(pageBox.val(),
            pageBox.data("min"), pageBox.data("max"))) {
        var link = pageBox.data('url');
        link = link.replace('-1', pageBox.val());
        window.location = link;
      }
    }
    else if (keycode == 27) {
      pageBox.val(pageBox.data('current'));
    }
});
```

BY NC SA

Provjera ispravnosti unosa broja stranice

- **Provjera na klijentskoj strani korištenjem vlastite skriptne datoteke**
  - Koristi se regularni izraz za provjeru radi li se o broju, a zatim se provjera je li željeni broj unutar raspona
- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ gotopage.js**

```
function validRange(str, min, max) {
  var intRegex = /^\d+$/;
  if (intRegex.test(str)) {//da li je upisan broj?
    var num = parseInt(str);
    if (num >= min && num <= max)
      return true;
    else
      return false;
  }
  else {
    return false;
  }
}
```

BY NC SA

Web-aplikacije (ASP.NET Core MVC)

Parcijalni pogledi i dinamičko ažuriranje

2019/20.06b

CC BY NC SA

Napomena vezana za primjere koji slijede

- **Primjeri u ovim predavanjima predstavljaju složene koncepte koji uključuju korištenje CSS stilova i *jQueryja***
- **Primjeri su opsežni i ne mogu u potpunosti stati na slajdove te se preporuča isprobati primjer i detaljno proučiti programski kôd pri čemu slajdovi mogu poslužiti kao vodilja kojim redom proučavati primjere**

BY NC SA

Model za pregled svih mjesta

- **Umjesto entiteta Mjesto za pojedinačno mjesto koristi se novi prezentacijski pogled**
  - Umjesto oznake države sadrži naziv države
- **Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ MjestaViewModel.cs**

```
 public class MjestaViewModel {
    public IEnumerable<MjestoViewModel> Mjesta { get; set; }
    public PagingInfo PagingInfo { get; set; }
  }
```

- **Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ MjestoViewModel.cs**

```
public class MjestoViewModel {
    public int IdMjesta { get; set; }
    public int PostBrojMjesta { get; set; }
    public string NazivMjesta { get; set; }
    public string PostNazivMjesta { get; set; }
    public string NazivDrzave { get; set; }
  }
```

CC BY NC SA

Pregled svih mjesta

- **Kao i u primjeru s državama zaglavlje tablice sadrži poveznice za sortiranje po određenom stupcu u prikazu**
  - Izvedeno petljom i poljima koja sadrže nazive stupaca i parametre sortiranja
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Index.cshtml

```
<table class="..."> <thead>
    <tr>
      @{
        string[] nazivi = { "Poštanski broj", "Naziv mjesta",
"Poštanski naziv mjesta", "Država" };
        for (int i = 1; i <= nazivi.Length; i++) {
          <th>
            <a asp-route-sort="@i"
               asp-route-page="@Model.PagingInfo.CurrentPage"
               asp-route-ascending="@(Model.PagingInfo.Sort ==
i ? !Model.PagingInfo.Ascending? : true)">
              @nazivi[i - 1]
            </a>
          </th>
        }
```

Dodatne postavke usmjeravanja

- **Primjer: 🗁Web \ Firma.Mvc \ Startup.cs**

```
public void Configure(...) {
  ...
  app.UseMvc(routes => {
        endpoints.MapControllerRoute("Mjesta i artikli",
"{controller:regex(^(Mjesto|Artikl)$)}/Page{page}/Sort{sort:int
}/ASC-{ascending:bool}",
              new { action = "Index" }
              );
   endpoints.MapControllerRoute( name: "default",
         pattern: "{controller=Home}/{action=Index}/{id?}");
   });
```

- http://.../Mjesto/Page10
  - Upravljač Mjesto izvršava pretpostavljeni postupak (Index) , a za vrijednost parametra page uzima vrijednost 10
- http://.../Drzava?page=7
  - Upravljač Drzava izvršava Index, a page je 7

BY NC SA

Parcijalni pogled za prikaz mjesta (1)

- **Prikaz pojedinog retka izveden parcijalnim pogledom**
  - Parcijalni pogled je pogled koji ne koristi glavnu stranicu te se umeće u neke druge poglede
  - Može imati svoj model (u prikazanom primjeru je to pojedinačno mjesto)
- **U pogledu *Index* poziva se parcijalni pogled *Row* te se njegov sadržaj (tj. rezultat izvršavanja) umetne u konačni rezultat**
  - Poziv se obavlja korištenjem *tag-helpera* `partial`
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Index.cshtml**

```
<table class="table table-striped">
  ...
  <tbody>
    @foreach (var mjesto in Model.Mjesta)
    {
      <partial name="Row" model="mjesto" />
    }
  </tbody>
</table>
```

CC BY NC SA

Parcijalni pogled za prikaz mjesta (2)

- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Row.cshtml**
  - Model je tipa MjestoViewModel (sadrži naziv umjesto oznake države)

```
@model MjestoViewModel
<tr>
  <td class="text-left">@Model.PostBrojMjesta</td>
  <td class="text-left">@Model.NazivMjesta</td>
  <td class="text-left">@Model.PostNazivMjesta</td>
  <td class="text-left">@Model.NazivDrzave</td>
  <td>
    ...
  </td>
</tr>
```

CC BY NC SA

„Poveznica” za brisanje podatka bez osvježavanja cijele stranice

- **Za razliku od brisanje države, u primjeru s mjestima ne dolazi do osvježavanje cijele stranice**
  - Umjesto slanja zahtjeva za novom stranicom, koristit će se jQuery Ajax poziv
    - posljedično ne treba postojati skriveno polje s identifikatorom mjesta
  - Zbog *AntiforgeryTokena* potrebno dodati HTML oznaku *form* u pogled iako se neće koristiti
  - Podatak u mjestu kojeg treba obrisati postavljen kao atribut na gumbu *Obriši*
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Row.cshtml

```
<tr>
  ...
  <td>
@*Ne koristi se submit, ali treba zbog antiforgery tokena*@
    <form asp-action="Delete" method="post">
      <button data-idmjesta="@Model.IdMjesta"
            class="btn btn-sm btn-danger deleteajax"
            title="Obriši">
              <i class="fas fa-trash-alt"></i>
      </button>
    </form>
```

BY NC SA

Brisanje retka bez osvježavana stranice (2)

- **Brisanje obavlja vlastiti programski kod pisan u jQueryju**
  - Nakon učitavanja stranice svakoj kontroli koja ima postavljen stil *deleteajax* pridružuje se kod kojim se klikom na tu kontrolu izvršava brisanje mjesta
  - Kôd zapisan u funkciji *SetDeleteAjax*, jer se koristi na više mjesta
  - Funkcija je parametrizirana akcijom (tj. adresom koja odgovara akciji) koju treba izvršiti i nazivom parametra po kojem će se redak u tablici prepoznati
    - U ovom primjeru to je idmjesta te je akcija Delete
    - Umjesto pisanje url-a, koristi se postupak *Url.Action* koji vrati poveznicu na akciju na nekom (ili istom upravljaču)
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Index.cshtml**

```
@section scripts{
  <script type="text/javascript">
        $(function () {
...
            SetDeleteAjax(".deleteajax",
'@Url.Action("Delete")', 'idmjesta');
        });
  </script>
}
```

CC BY NC SA

Brisanje retka bez osvježavana stranice (3)

- **Klikom na kontrolu uklanja se prethodna statusna poruka i dohvaćaju podaci o podatku koji treba obrisati**
  - *selector* predstavlja informaciju kako će *jQuery* doći do elementa
  - *paramname* označava u kojem data atributu je identifikator podatka
- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ site.js**

```
function SetDeleteAjax(selector, url, paramname) {
  $(document).on('click', selector, function (event) {
    event.preventDefault(); //bitno ako je submit button
    var paramval = $(this).data(paramname);
    var tr = $(this).parents("tr");
    ...
      if (confirm('Obrisati zapis?')) {
        var token =
$('input[name="__RequestVerificationToken"]').first().val();
        clearOldMessage();
        ...
```

CC BY NC SA

Brisanje retka bez osvježavana stranice (4)

- **Skripta poziva odgovarajući url s identifikatorom podatka i *anti-forgery* tokenom**
  - Ispisuje se poruka o uspješnosti brisanja
    - successful i message su vlastite varijable u rezultatu (vidi sljedeći slajd)
  - U slučaju uspješnog brisanja uklanja se redak sa zapisom
  - U slučaju pogreške prilikom zahtjeva prikazuje se dijalog s opisom pogreška
  - Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ site.js

```
function SetDeleteAjax(selector, url, paramname) {
  ...
  $.post(url, { id: paramval,
                __RequestVerificationToken: token },
        function (data) {
          if (data.successful)
            $(tr).remove();
          $("#tempmessage").addClass(data.successful ?
                                  "alert-success" : "alert-danger");
          $("#tempmessage").html(data.message);
        }).fail(function (jqXHR) {
          alert(jqXHR.status + " : " + jqXHR.responseText);
        });
```

Akcija brisanja mjesta

- **Postupak vraća anonimni razred pretvoren u JSON**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ MjestoController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
 public IActionResult Delete(int id) {
      var mjesto = ctx.Mjesto.AsNoTracking()
                      .Where(m => m.IdMjesta == id)
                      .SingleOrDefault();
      if (mjesto != null) {
        try {
          string naziv = mjesto.NazMjesta;
          ctx.Remove(mjesto); ctx.SaveChanges();
          var result = new {
            message = $"Mjesto {naziv} sa šifrom {id}
obrisano.", successful = true };
          return Json(result);
        }
        catch (Exception exc) {
          var result = new {
            message = "Pogreška prilikom brisanja mjesta: " ...
```

BY NC SA

Ideja ažuriranja unutar retka

- **Klasično ažuriranje: poveznica u svakom retku vodi na posebnu stranicu za ažuriranje podatka**
- **Ažuriranje unutar retka**
  - Klikom na poveznicu ili neku drugu ikonu unutar retka trenutni redak se korištenjem JavaScripta zamijeni s nekim drugim retkom koji sadrži kontrole za ažuriranje
  - Ovisno o (ne)uspješnosti snimanja podataka kontrole za unos „nestaju”, a prikazuje se podatak s podacima ažuriranog retka

Poštanski broj	Naziv mjesta	Poštanski naziv mjesta	Država
53234	Breštane	Udbina	Croatia
33507	Breštanovci	Crnac	Croatia
43252	Breza	Prgomelje	Croatia
51213	Breza	Jurdani	Croatia

BY NC SA

Ikona za aktivaciju ažuriranja

- **Potrebno stvoriti HTML kontrolu (bilo koje vrste) koja u svojim atributima ima identifikator podatka kojeg treba ažurirati**
  - proizvoljni atribut oblika data-naziv s vrijednošću koja odgovara identifikatoru
  - bit će potrebno prepoznati sve takve ikone, pa im se uobičajeno dodijeli neka stilska oznaka (*editajax* u ovom primjeru)
    - ne mora nužno biti opisana u css datoteci
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Row.cshtml**

```
@model MjestoViewModel
<tr>
  ...
  <td>
    <a data-idmjesta="@Model.IdMjesta"
       class="btn btn-sm editajax"
       title="Ažuriraj">
      <i class="fas fa-edit"></i>
    </a>
  </td>
  ...
</tr>
```

CC BY NC SA

Sadržaj retka za ažuriranje

- **Parcijalni pogled sadrži redak s kontrolama za unos i odabir podataka te redak u kojem će se ispisati validacijske pogreške**
  - Atribut data-save koristi se iz vlastite skripte za pronalazak traženih kontrola
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Edit.cshtml

```
@model Mjesto
...
    <input asp-for="PostBrMjesta" data-save="" />
  ...
    <input asp-for="NazMjesta" data-save="" />
...
    <select asp-for="OznDrzave" asp-items="ViewBag.Drzave"
       data-save="">
      <option disabled selected value=""> Odaberite državu
      </option>
    </select>
   <button type="submit" class="... save" title="Spremi...
   <button class="... cancel" title="Odustani
...
    <div asp-validation-summary="All"></div>
    <input type="hidden" asp-for="IdMjesta" data-save="" />
```

BY NC SA

Akcija na upravljaču koja priprema redak za ažuriranje

- **Klikom na ikonu za ažuriranje podatka u nekom retku sadržaj retka se dinamički zamijeni novim retkom sa servera**
  - Upravljač vraća parcijalni pogled s kontrolama za promjenu podataka
  - Kao povratna vrijednost koristi se PartialView, a ne View!
- **Primjer: 🗁 Web \ Firma.Mvc \ Controller \ MjestoController.cs**

```
[HttpGet]
 public IActionResult Edit(int id) {
      var mjesto = ctx.Mjesto.AsNoTracking()
                          .Where(m => m.IdMjesta == id)
                          .SingleOrDefault();
      if (mjesto != null) {
        PrepareDropDownLists();
        return PartialView(mjesto);
      }
      else {
        return NotFound($"Neispravan id mjesta: {id}");
      }
    }
```

BY NC SA

Javascript kod za izmjenu redaka (1)

- **Nakon učitavanja stranice svakoj tako označenoj kontroli pridružuje se programski kod koji omogućava da se sadržaj tog retka zamijeni novim sadržajem na serveru**
  - Originalni redak se ne uklanja iz strukture dokumenta, već se samo skriva s naredbom *toggle*
- **Za sadržaj koji bude dostavljen sa servera potrebno je definirati ponašanje kontrola koje imaju ulogu akcija *Save* i *Cancel***
- **Kontrola kojoj je pridružena akcija *Cancel*, uklanja redak za ažuriranje te otkriva prethodno skriveni originalni redak**

Javascript kod za izmjenu redaka (2)

- **Potrebno pronaći sve kontrole koje imaju stilsku oznaku editajax**

```
<a data-idmjesta="@Model.IdMjesta" class="... editajax" …
```

- **Svakoj kontroli pridružiti kôd koji će omogućiti zamjenu retka**
  - Kôd zapisan u funkciji *SetEditAjax*, jer se koristi na više mjesta
  - Funkcija je parametrizirana akcijom (tj. adresom koja odgovara akciji) koju treba izvršiti i nazivom parametra po kojem će se redak u tablici prepoznati
    - U ovom primjeru to je idmjesta te je akcija Edit
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Mjesto \ Index.cshtml**

```
@section scripts{
  <script type="text/javascript">
        $(function () {
              SetEditAjax(".editajax",
                          '@Url.Action(„Edit")',
                          'idmjesta');
              ...
        });
  </script>
}
```

Javascript kod za izmjenu redaka (3)

- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ site.js**

```
function SetEditAjax(selector, url, paramname) {
  $(document).on('click', selector, function (event) {
    event.preventDefault(); //ako je submit button
    var paramval = $(this).data(paramname);
    var tr = $(this).parents("tr");
    var aktivan = $(tr).data('aktivan');
    if (aktivan != true) { //da spriječimo dva brza klika...
      $(tr).data('aktivan', true);
      $.get(url, { id: paramval }, function (data) {
        tr.toggle(); //sakrij trenutni redak
        //redak sa servera dodaj iza skrivenog retka
        var inserted = $(data).insertAfter(tr);
        SetCancelAndSaveBehaviour(tr, inserted, url);
      })
      .fail(function (jqXHR) {
        alert(jqXHR.status + " : " + jqXHR.responseText);
      });
    }
  });
}
```

Odustajanje od ažuriranja

- **Unutar umetnutog retka traži se HTML kontrola koji ima stil *cancel***
- **Klik na kontrolu uklanja umetnuti redak i postavlja vidljivim prethodno skriveni redak**
  - Atributom data-aktivan sprječava se dvostruki klik prije nego server stigne vratiti sadržaj
- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ site.js**

```
function SetCancelAndSaveBehaviour(hiddenRow, insertedData, url)
{
  //nađi cancel button
  insertedData.find(".cancel").click(function () {
    insertedData.remove(); //ukloni umetnuti redak
    hiddenRow.toggle(); //vrati vidljivost originalnom retku
    $(hiddenRow).data('aktivan', false);
  });
  ...
```

Definiranje ponašanja za kontrolu za snimanje

- Unutar umetnutog retka traži se HTML kontrola koji ima stil *save*
- Kontroli se pridružuje kôd koji će prikupiti unesene podatke, a zatim poslati na server upravljaču
- Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ site.js

```
function SetCancelAndSaveBehaviour(hiddenRow, insertedData, url)
{
  ...
  //nađi save button
  insertedData.find(".save").click(function () {
     event.preventDefault(); //ako je bio submit button
     ... dohvati podatke i spremi ih u json ...
     $.ajax({
         type: "POST",
         url: url,
         ...
     });
  });
```

BY NC SA

Obrada klika Save

- **Klikom na kontrolu vezanu za akciju Save izvršava se slanje podataka na server**
- **Budući da se istovremeno ažurira više svojstava istog retka, potrebno je prikupiti vrijednosti svih kontrola iz jednog retka**
- **Kontrole za unos prepoznaju se po atributu data-save**
  - Dodatno je potrebno obraditi pažnju radi li se o kontroli za slanje datoteke
- **Vrijednosti se pospremaju u JavaScript objekt FormData**
- **Potrebno je pronaći i antiforgerytoken**

Prikupljanje ažuriranih podataka

- **Kontrole za unos prepoznaju se po postojanju atributa data-save**
  - dodatno je potrebno obraditi pažnju o vrsti kontrole
  - vrijednosti se pospremaju u JavaScript objekt FormData
  - Primjer: 🗁 Firma.Mvc \ wwwroot \ site.js - *SetCancelAndSaveBehaviour*

```
var formData = new FormData();
insertedData.find("[data-save]").each(function (index, element){
   if ($(element).is(':checkbox'))
        formData.append($(element).attr('name'),
                        $(element).is(':checked') == true);
   else if ($(element).is("input[type=file]")) {
        var files = $(element).get(0).files;
        if (files.length > 0)
          formData.append($(element).attr('name'), files[0]);
   }
   else {
        var val = $.trim($(element).val());
        if (val != '')
             formData.append($(element).attr('name'),
                             $(element).val());
}});
```

Slanje ažuriranih podataka

- **Potrebno je pronaći i vrijednost anti-forgery tokena**
- **Prikupljeni podaci šalju se na server POST metodom**
  - Primjer: 🗁 Firma.Mvc \ wwwroot \ site.js – *SetCancelAndSaveBehaviour*

```
insertedData.find(".save").click(function () {
  var formData = new FormData();
  ...
  var token = $('input[name="__RequestVerificationToken"]')
              .first()
              .val();
  formData.append("__RequestVerificationToken", token);

  $.ajax({
      type: "POST",
      url: url,
      contentType: false,
      processData: false,
      data: formData,
   ...
```

Moguće situacije prilikom pohrane promjena podataka jednog retka

- **Prilikom zahtjeva za snimanje podataka mogu se dogoditi tri mogućnosti**

- **Pogreška prilikom zahtjeva**
  - Prikazuje se dijalog s opisom pogreške
- **Zahtjev je bio „uspješan”, ali podatak nije pohranjen, primjerice ako je došlo do validacijske pogreške**
  - Uklanja se „stari” redak za ažuriranje i mijenjao s novim kojeg je generirao server
    - Sadrži prethodno upisane podatke i validacijske pogreške
- **Podatak je uspješno snimljen te je server odgovorio sa statusnom porukom 302 koja ima značenje preusmjeravanja na drugu stranicu**
  - Nova stranica je parcijalni pogled s novim, ažurnim vrijednostima
  - Skriveni redak s prethodnim vrijednostima mijenja se novim sadržajem
  - Redak za ažuriranje se uklanja iz dokumenta

BY NC SA

Preusmjeravanje nakon uspješnog snimanja

- **Upravljač će u slučaju uspješnog snimanja podatka vratiti statusnu poruku 302 (preusmjeravanje)**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ MjestoController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(Mjesto mjesto) {
   ...
   try {
     ctx.Update(mjesto);
     ctx.SaveChanges();
     return StatusCode(302, Url.Action(nameof(Row),
                                 new { id = mjesto.IdMjesta }));
   }
   catch (Exception exc) {
      ModelState.AddModelError(string.Empty,
                    exc.CompleteExceptionMessage());
      return PartialView(mjesto);
   }
...
```

Zamjena retka nakon uspješnog snimanja

- **U slučaju statusne poruke 302 skripta uklanja redak koji je služio za ažuriranje, a skriveni redak mijenja s novim, osvježenim retkom**
  - 302 se smatra kao *error*, a ne kao *success* u zahtjevu postupkom *.ajax*
  - funkcionalnost ažuriranja automatski primijenjena i za novonastali redak
  - Primjer: 🗁 Firma.Mvc \ wwwroot \ site.js - *SetCancelAndSaveBehaviour*

```
$.ajax({
   type: "POST", url: url,
  ...
   error: function (jqXHR) {
      if (jqXHR.status == 302) { //data saved and redirect
        insertedData.remove();
        $.get(jqXHR.responseText, {}, function (refreshedRow) {
            $(hiddenRow).replaceWith(refreshedRow);
        })
      }
      else {
          alert(jqXHR.status + " : " + jqXHR.responseText);
      }
    }
});
```

Akcija u slučaju validacijske pogreške

- **U slučaju validacijske pogreške upravljač i akcija *Edit* vraćaju parcijalni pogled koji ponovno sadrži kontrole za unos, ali ovaj put s validacijskim pogreškama i prethodno popunjenim sadržajem**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ MjestoController.cs

```
[HttpPost]
public IActionResult Edit(Mjesto mjesto) {
 ...
  if (ModelState.IsValid) {
     try {
       ...
     }
     catch (Exception exc) {
         ModelState.AddModelError(...);
         PrepareDropDownLists();
         return PartialView(mjesto);
     }
   }
   else {
        PrepareDropDownLists();
        return PartialView(mjesto);
}
```

Zamjena retka u slučaju validacijske pogreške

- **U slučaju validacijske pogreške prethodni redak s kontrolama za unos podataka se uklanja, a HTML sadržaj kojeg je upravljač generirao dodaje se iza skrivenog retka**
- **Potrebno ponovno postaviti ponašanje kontrolama koje služe za snimanje i odustajanje od ažuriranja, jer te kontrole nisu bile dio dokumenta u trenutku kad je prvo postavljanje izvršeno**
  - Za razliku od Edit gumba, ovdje se koristi postupak .click, a ne .on gumba, jer se oni postavljaju na razini retka i ne koristi se on, već click
  - Primjer: 🗁 Firma.Mvc \ wwwroot \ site.js - *SetCancelAndSaveBehaviour*

```
$.ajax({
   type: "POST", url: url,
  ...
   success: function (data, textStatus, jqXHR) {
        insertedData.remove();
        var inserted = $(data).insertAfter(hiddenRow);
        SetCancelAndSaveBehaviour(hiddenRow, inserted, url);
      }
   ...
});
```

BY NC SA

Praćenje traga rada aplikacije

2019/20.07a

Praćenje traga rada aplikacije

- **Motivacija**
  - Evidentirati pogreške tijekom rada (neovisno o dojavama korisnika)
  - Detektirati uska grla aplikacije
  - Prikupljanje ostalih informacija
    - npr. parametri pretrage, prijave korisnika
  - …

- **Gdje evidentirati?**
  - Baza podataka
  - Datoteka
  - E-mail poruke, SMS
  - Event Log na Windowsima
  - …

- **Kako evidentirati?**
  - Na uniforman (centraliziran) način neovisan o odredištu traga

BY NC SA

Razine važnosti zapisa (1)

- **Trace**
  - Informacija namijenjena programeru u cilju lakšeg rješavanja problema
    - npr. ispis svih parametara nekog postupka
  - Bilježi male korake izvođenja programa
  - Nije preporučljivo koristiti u produkciji
    - smije sadržavati osjetljive podatke
    - veća količina zapisa u odnosu na ostale razine
- **Debug**
  - Informativna poruka u cilju otklanjanja pogrešaka
    - ispis određene vrijednosti kratkoročne koristi
  - Slično kao Trace, ali rjeđe (i manje detaljno)
    - nije nužno namijenjena samo programeru
  - Najčešće automatski isključeno iz produkcijske verzije
- **Information**
  - Zapis trajnijeg karaktera koji služi za praćenje toka rada aplikacije
    - npr. informacija o posjetu određenoj stranici ili evidencija postavljenih kriterija pretrage

CC BY NC SA

Razine važnosti zapisa (2)

- **Warning**
  - Za pogreške koje ne utječu na daljnji rad aplikacije, ali predstavljaju potencijalno opasne situacije te zahtijevaju naknadnu pažnju
    - npr. konfiguracijska datoteka ne sadrži traženu vrijednost, pa se koristi predodređena
- **Error**
  - Služi za evidentiranje pogrešaka i iznimki koje se ne mogu obraditi
  - Predstavljaju kritičnu pogrešku za određeni postupak, ali ne i za cijelu aplikaciju
    - npr. pogreška prilikom dodavanja novog podatka u bazu
- **Critical (Fatal)**
  - Situacije koje uzrokuju prekid rada cijele aplikacije
    - npr. nedostatak prostora na disku, neispravne postavke za spajanje na bazu podataka, …

- **Trace < Debug < Information < Warning < Error < Critical**

BY NC SA

Praćenje traga u ASP.NET Coreu

- **Microsoft.Extensions.Logging**
  - Podrška za različite pakete i odredišta zapisivanja tragova
  - Nekoliko ugrađenih implementacija
    - Npr. konzolni ispis
  - Moguće dodati vlastitu implementaciju

- **Zajedničko sučelje *ILogger* kao apstrakcija nad različitim implementacijama**

- **Koristi se tehnika *Dependency Injection***
  - Upravljači koji bilježe trag ovise o sučelju ILogger
  - ASP.NET stvara konkretni objekt temeljem podataka u Startup.cs

CC BY NC SA

Sučelje ILogger

- **`IsEnabled(LogLevel logLevel)`**
  - provjerava evidentira li konkretna implementacija zapise navedene razine

- **`void Log<TState>(LogLevel logLevel, EventId eventId, TState state, Exception exception, Func<TState, Exception, string> formatter)`**
  - evidentiranje zapisa određene razine i vrste događaja
    - Zapis (state) ne mora nužno biti string, već može biti bilo kojeg tipa
    - Vrsta događaja opisana strukturom EventId: Id i naziv
    - Uz zapis (state) može biti vezana određena iznimka (exception)
    - *formatter:* funkcija koja kreira string na osnovi zapisa i iznimke

- **`IDisposable BeginScope<TState>(TState state)`**
  - Služi za grupiranje više zapisa u jedan zajednički zapis
    - samo ako konkretna implementacija podržava

CC BY NC SA

Pokrate za postupak Log

- **Statički razred Microsoft.Extensions.Logging.Logger s proširenjima za sučelje ILogger**
- **Nekoliko preopterećenih postupaka:**
  - LogTrace
  - LogDebug
  - LogInformation
  - LogError
  - LogCritical
- **Pojednostavljuje zapisivanje traga**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DrzavaController.cs

```
public IActionResult Create(Drzava drzava){
   logger.LogTrace(JsonConvert.SerializeObject(drzava));
   ...
   catch (Exception exc) {
       logger.LogError("Pogreška prilikom dodavanje nove
           države: {0}", exc.CompleteExceptionMessage());
```

CC BY NC SA

NLog

- **NLog kao jedan od alata za praćenje traga kompatibilan s Microsoft.Extensions.Logging**
  - http://nlog-project.org/
- **Omogućava više vrsta (različitih) spremišta i formata ovisno o razini zapisa**
- **Upute za dodavanje u ASP.NET Core projekt**
  - https://github.com/NLog/NLog/wiki/Getting-started-with-ASP.NET-Core-3
  - Umjesto apsolutne putanje iz primjera na svim mjestima u konfiguracijskoj datoteci za NLog koristiti relativnu putanju
    - Primjer: 🗁 Web \ Firma.Mvc \ nlog.config

```
 <target xsi:type="File" name="allfile"
       fileName="logs\nlog-all-${shortdate}.log"
       layout="${longdate}|${event-
properties:item=EventId_Id}|${logger}|${uppercase:${level}
}|${message} ${exception}" />
```

CC BY NC SA

Konfiguracijska datoteka za NLog

- **Moguće definirati više različitih odredišta, a zatim pravila koja određuju gdje će sve pojedini zapis biti evidentiran**
  - Primjer: 🗁 Web \ Firma.Mvc \ nlog.config

```
<targets>
    <target xsi:type="File" name="allfile"
       fileName="logs\nlog-all-${shortdate}.log" ... />
    <target xsi:type="File" name="ownFile-web"
fileName="logs\nlog-own-${shortdate}.log"  ... />
    <!-- write to the void aka just remove -->
    <target xsi:type="Null" name="blackhole" />
  </targets>
  <rules>
    <logger name="*" minlevel="Trace" writeTo="allfile" />
    <!--Skip Microsoft logs and so log only own logs-->
    <logger name="Microsoft.*" minlevel="Trace"
             writeTo="blackhole" final="true" />
    <logger name="*" minlevel="Trace"
             writeTo="ownFile-web" />
  </rules>
```

CC BY NC SA

Aktivacija Nloga (1)

- **Konfiguracijskoj datoteci postaviti vrijednost svojstva *Copy to Output Directory* na *Copy if newer ili Copy always***
  - Kopira datoteku u bin\Debug\netcoreapp2.0 ili bin\Release\netcoreapp2.0
- **U glavnom programu prije poziva BuildWebHost instancirati NLog objekt i pokušati dodati inicijalni zapis u log**
  - Primjer: 🗁 Web \ Firma.Mvc \ Programs.cs

```
public static void Main(string[] args) {
     // NLog: setup the logger first to catch all errors
     var logger = NLog.Web.NLogBuilder
                        .ConfigureNLog("nlog.config")
                        .GetCurrentClassLogger();
     try
     {
       logger.Debug("init main");
       CreateHostBuilder(args).Build().Run();
     }
```

Aktivacija Nloga (2)

- **U postupku kreiranja web-servera pozvati proširenje kojim se među postojeće *loggere* dodaje i NLog**
  - Alternativno mogu se obrisati postojeći loggere i dodati samo Nlog (postupak `.ConfigureLogging`)
- **Napomena: Razina postavljena u konfiguracijskog datoteci određuje hoće li neka poruka uopće biti proslijeđena NLogu (i ostalim uključenim alatima)**
- **Primjer: 🗁 Web \ Firma.Mvc \ Program.cs**

```
public class Program {
  public static IHostBuilder CreateHostBuilder(string[] args) =>
     Host.CreateDefaultBuilder(args)
         .ConfigureWebHostDefaults(webBuilder => {
              webBuilder.UseStartup<Startup>();
         })
         .UseNLog();
```

Aktivacija Nloga (3)

- U konfiguracijskoj datoteci postavljena minimalna potrebna razina da bi poruka uopće bila upućena nekom od uključenih loggera
- Primjer: 🗁 Web \ Firma.Mvc \ appsettings.json

```
{
  "AppSettings": ...
  },
  "Logging": {
    "LogLevel": {
      "Default": "Trace"
    }
  }
}
```

CC BY NC SA

Stranica za prikaz svih zapisa

- **U oglednoj aplikaciji evidentiraju se poruke vezane za dodavanje i brisanje države.**
  - Izrađena stranica koja za odabrani datum pronalazi datoteku s korisničkim poruka i ispisuje ih na ekranu

localhost:52066/ViewLog/Show?dan=2018-05-08

Zapisi u log datoteci za dan 08.05.2018.

Prikaži 10 zapisa

Pretraga

Vrijeme	Id	Upravljač	Level	Message	Url	Action
14:09:09	0	Firma.Mvc.Program	DEBUG	init main	?	
14:09:11	0	Firma.Mvc.Controllers.DrzavaController	WARN	Ne postoji država s oznakom: ADi	http://localhost/Drzava/Edit/ADi?asd=2	Edit
14:16:54	0	Firma.Mvc.Controllers.DrzavaController	ERROR	Pogreška prilikom brisanja države: An error occurred while updating the entries. See the inner exception for details.The DELETE statement conflicted with the REFERENCE constraint "fk_Drzava_Mjesto". The conflict occurred in database "Firma", table "dbo.Mjesto", column 'OznDrzave'.The statement has been terminated.	http://localhost/Drzava/Delete?page=1,sort=1,ascending=True	Delete
14:17:09	0	Firma.Mvc.Controllers.DrzavaController	TRACE	{"OznDrzave":"_TEST","NazDrzave":"Test država","Iso3drzave":"ISO","SifDrzave":12,"Mjesto":[]}	http://localhost/Drzava/Create?	Create
14:17:09	0	Firma.Mvc.Controllers.DrzavaController	ERROR	Pogreška prilikom dodavanje nove države: An error occurred while updating the entries. See the inner exception for details.String or binary data would be truncated.The statement has been terminated.	http://localhost/Drzava/Create?	Create
14:17:16	0	Firma.Mvc.Controllers.DrzavaController	TRACE	{"OznDrzave":"_TE","NazDrzave":"Test država","Iso3drzave":"ISO","SifDrzave":12,"Mjesto":[]}	http://localhost/Drzava/Create?	Create
14:17:17	1000	Firma.Mvc.Controllers.DrzavaController	INFO	Država Test država dodana.	http://localhost/Drzava/Create?	Create
14:18:06	0	Firma.Mvc.Program	DEBUG	init main	?	
14:18:08	0	Firma.Mvc.Controllers.DrzavaController	INFO	Država Test država uspješno obrisana	http://localhost/Drzava/Delete?page=1,sort=1,ascending=True	Delete

21 - 29 od ukupno 29 zapisa

Prethodna 1 2 3 Sljedeća

BY NC SA

Odabir datuma za prikaz zapisa

- **Odabir datuma korištenjem jQueryUI datepickera**
  - Odabrani datum u formatu koji omogućava automatsko povezivanje u datumski tip u akciji Show
    - GET varijanta kako bi datum bio dio adrese, a ne tijela zahtjeva
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ ViewLog \ Index.cshtml

```
<form asp-action="Show" method="get">
...
      <input name="dan" class="form-control datum" />
...   <button type="submit"
             class="btn btn-xs btn-primary save" ...
</form>
@section scripts{
  ...
  <script>
    $(".datum").datepicker({
       dateFormat: "yy-mm-dd"
    });
  </script>
}
```

Provjera postoji li datoteka za odabrani dan

- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ViewLogController.cs**

```
public async Task<IActionResult> Show(DateTime dan)
{
   ViewBag.Dan = dan;
   List<LogEntry> list = new List<LogEntry>();
   string format = dan.ToString("yyyy-MM-dd");
   string filename = Path.Combine(AppContext.BaseDirectory,
                                  $"logs/nlog-own-{format}.log");
   if (System.IO.File.Exists(filename))
   {
       ...
```

Čitanje datoteke po retcima

- **Veza na datoteku korištenjem razreda FileStream koji se zatim pretvara u StreamReader**
  - Omogućava čitanje redak po redak
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ViewLogController.cs

```
public async Task<IActionResult> Show(DateTime dan) {
   ...
 String previousEntry = string.Empty;
 using (FileStream fileStream = new FileStream(filename,
                        FileMode.Open, FileAccess.Read)) {

   using (StreamReader reader = new StreamReader(fileStream))
   {
      string line;
      while ((line = await reader.ReadLineAsync()) != null) {
            ...
```

- **Poruka nekog zapisa mogla se nalaziti u više redaka**
  - Novi zapis prepoznaje se tako da počinje s datumom te se u tom trenutku stari dodaje u listu (vidi ostatak programskog koda)

BY NC SA

Model za prikaz zapisa

- **Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ LogEntry.cs**

```
public class LogEntry {
   public DateTime Time { get; set; }
   public int Id { get; set; }
   public string Controller { get; set; }
   public string Level { get; set; }
   public string Message { get; set; }
   public string Url { get; set; }
   public string Action { get; set; }
   internal static LogEntry FromString(string text) {
     string[] arr = text.Split('|');
     LogEntry entry = new LogEntry();
     entry.Time = DateTime.ParseExact(
            arr[0], "yyyy-MM-dd HH:mm:ss.ffff", null);
      ...
     entry.Controller = arr[2];
     entry.Level = arr[3];
     entry.Message = arr[4];
     entry.Url = arr[5].Substring(5); //url:
     entry.Action = arr[6].Substring(8); //action:
     return entry;
```

Programska dokumentacija

2019/20.07b

Dokumentacija programskog koda

- **XML dokumentacijski komentari**
  - omogućavaju pisanje dokumentacije koja se odnosi na programski kod
  - označavaju se trostrukom kosom crtom : ///
  - dodaju se ispred programskog bloka na koji se odnose
  - sadrže posebne XML oznake koje opisuju dokumentirani kod
  - Primjer: 🗁 Web \ Firma.Mvc \ Extensions \ ExceptionExtensions.cs

```
/// <summary>
/// Razred sa proširenjima za iznimke
/// </summary>
public static class ExceptionExtensions {
  /// <summary>
  /// Vraća sadržaj poruka cijele hijerarhije neke iznimke.
  /// </summary>
  /// <param name="exc">Iznimka čija se hijerarhija poruka
  ///dohvaća </param>
  /// <returns>String s porukaama svih unutarnjih iznimki.
  /// Poruka svake iznimmke dodana je u novi redak</returns>
  public static string CompleteExceptionMessage(
       this Exception exc) { ...
```

BY NC SA

Xml datoteka s programskom dokumentacijom

- **/doc opcija prevoditelja prilikom prevođenja uključuje traženje XML komentara u kodu te kreiranje dokumentacijske XML datoteke**
  - Alternativno: kontekstni izbornik projekta : Project properties \ Build \ XML documentation file
    - po želji promijeniti lokaciju generirane datoteke
  - prevođenje proizvede *NazivAsemblija.XML*
  - dodavanjem reference na neki dll, uzima se i istoimena xml datoteka (ako postoji) – služi za prikaz dokumentacijskih komentara
- **Služi kao pomoć prilikom programiranja**
  - Nudi opise razreda i postupaka iz referenciranog projekta

BY NC SA

XML oznake

- **XML oznake za komentiranje programskog koda**
  - `<summary>` - sažetak opisa razreda ili člana
  - `<remarks>` - nadopunjuje opis o razredu ili članu
  - `<param>` - opis parametra metode
  - `<paramref>` - oznaka da se opis odnosi na parametar
  - `<permission>` - prava pristupa
  - `<returns>` - povratne vrijednosti metode
  - `<value>` - opis vrijednosti svojstva
  - `<exception>` - iznimke koje mogu biti bačene u opisanom kodu. Sadrži referencu na iznimku.
  - `<typeparam>` - opis generičkog razreda ili metode
  - `<typeparamref>` - tip parametra kod generičkih razreda ili metoda

BY NC SA

XML oznake (nastavak)

- `<c>` - tekst unutar oznake je kod
- `<code>` - tekst unutar oznake je višeretkovni kod
- `<example>` - primjer
- `<include>` - uključivanje vanjske datoteke s XML dokumentacijom
- `<list>` - lista, nabrajanje elemenata
- `<para>` - paragraf teksta
- `<see>` - omogućava stvaranje linka
- `<seealso>` - link u “See also” sekciji

- **Za detaljniji opis oznaka pogledati:**
  - https://docs.microsoft.com/en-us/dotnet/csharp/codedoc

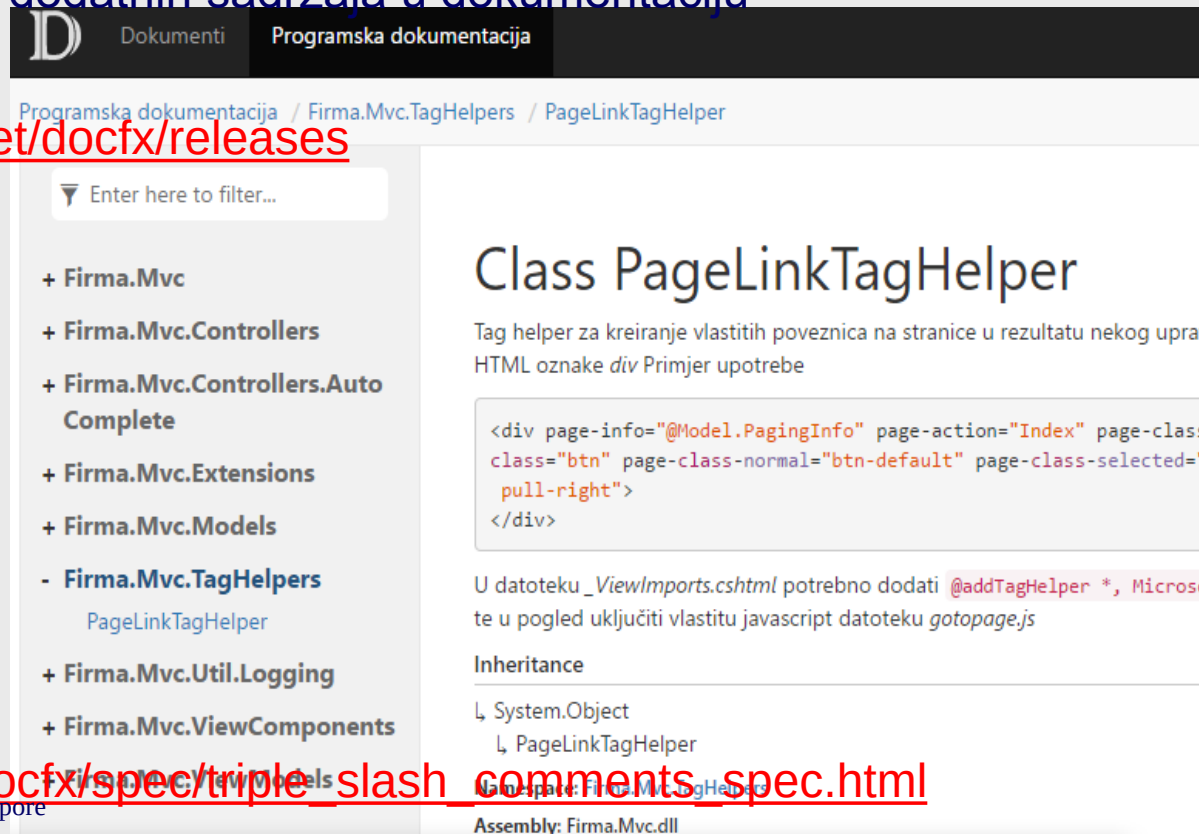
Primjer generirane XML datoteke

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>Firma.Mvc</name>
    </assembly>
    <members>
        <member name="T:Firma.Mvc.Extensions.ExceptionExtensions">
            <summary>
            Razred sa proširenjima za iznimke
            </summary>
        </member>
        <member name="M:Firma.Mvc.Extensions.ExceptionExtensions.CompleteExceptionMessage(Sy
            <summary>
            Vraća sadržaj poruka cijele hijerarhije neke iznimke. Za predanu iznimku provjer
            Ako da, poruka unutanje iznimke dodaje se u rezultat te se dalje provjerava post
            </summary>
            <param name="exc">Iznimka čija se kompletna hijerarhija poruka treba ispisati</p
            <returns>String formiran od poruka svih unutarnjih iznimki. Poruka svake iznimml
        </member>
        <member name="T:Firma.Mvc.TagHelpers.PageLinkTagHelper">
            <summary>
            Tag helper za kreiranje vlastitih poveznica na stranice u rezultatu nekog upravl
            Upotrebljava se kao atribut HTML oznake *div*
            <example>
            Primjer upotrebe
            <code>
            <div page-info="@Model.PagingInfo" page-action="Index" page-classes-enabled="tru
                page-class="btn" page-class-normal="btn-default"
                page-class-selected="btn-primary" class="btn-group pull-right">
            </div></code>
            U datoteku *_ViewImports.cshtml* potrebno dodati `@addTagHelper *, Microsoft.Asp
            te u pogled uključiti vlastitu javascript datoteku *gotopage.js*
            </example>
            </summary>
        </member>
```

- **Primjer: 🗁 Web \ Firma.Mvc \ bin \ Debug \ netcoreapp3.1 \ Firma.Mvc.xml**

Alat za generiranje programske dokumentacije

- **Generirana xml datoteka korisna unutar razvojnog sučelja, ali nije prikladna za čitanje**
- **Za izradu programske dokumentacije namijenjene za čitanje koristi se alat DocFx**
  - Generira HTML sadržaj na osnovi programske dokumentacije
  - Omogućava dodavanje dodatnih sadržaja u dokumentaciju
- **Instalacija DocFx-a**
  - https://github.com/dotnet/docfx/releases
  - Preporuka dodati u putanju
  - Alternativa: Unutar Visual Studia 2019 instalirati NuGet paket *docfx.Console*
- **Napomena: podržava podskup oznaka iz xml dokumenacije**
  - http://dotnet.github.io/docfx/spec/triple_slash_comments_spec.html



Neki primjeri korištenja DocFX-a iz prakse (1)

Documentation | Montelektro.MVVM | Montelektro.Util | CME.Data | CME.Dal | CME.StartupWorkflow | Search

Documentation / How to develop new form?

Enter here to filter...

- Introduction
- Configuration files
- navigation.json
- How to develop new form?
- + Table dependencies
- Startup workflow

```
    public PLCTypesRepository(IDatabaseWrapper dbWrapper)
    {
      this.db = dbWrapper;
      dependenyChecker = new DependencyRepository(dbWrapper);
    }
```

ActionResponse as a result

An easy way to catch an exception and return proper ActionResponse is to create anonymous function/action and call ActionResponse extension method

```
public ActionResponse InsertPlcType(PLCTypes type)
{
  Action func = () =>
  {
    string updateQuery = "INSERT INTO dbo.cme_PLCTypes (Name,Description) OUTPUT Inserted.Id V
alues (?,?)";  //ODBC does not support named parameters
    using (var command = db.Database.GetSqlStringCommand(updateQuery))
    {
      db.Database.AddInParameter(command, "@Name", DbType.String, type.Name);
      db.Database.AddInParameter(command, "@Description", DbType.String, type.Description);
      db.Database.ExecuteNonQueryCommand(command);
    }
  };
  return func.ActionResponse();
}
```

IN THIS ARTICLE

- Basic steps
- Create repository
- Data Transfer Obj
- Repository interfa
- > Repository impler
- > Repository impler
- ActionResponse a
- Data for combobo
- Saving changes
- Register repositor
- Create View Mode
- Choosing appropr

CC BY NC SA

Neki primjeri korištenja DocFX-a iz prakse (2)

Montelektro.MVVM / Montelektro.MVVM / CustomCommandInvoker

Enter here to filter...

- **Montelektro.MVVM**
  - ChangedValue
  - ChangedValue.ChangeState
  - CustomCommandInvoker
  - CustomObservableCollection<T>
  - EditableCollection<T>
  - EditableObject
  - IChangeLogIdentifiable
  - Messenger
  - ObservableObject
  - RelayCommand
- **Montelektro.MVVM.Extensions**

Class CustomCommandInvoker

Simple WPF TriggerAction to bind events into commands Does not define CommandParameters in order to pass additional parameter If ever needed code for adding command parameter could be implemented as in http://putridparrot.com/blog/a-simple-wpf-triggeraction-to-turn-events-into-commands/

Inheritance

↳ System.Object
  ↳ CustomCommandInvoker

Namespace: Montelektro.MVVM

Assembly: Montelektro.MVVM.dll

Syntax

```
public sealed class CustomCommandInvoker : TriggerAction<DependencyObject>
```

Examples

Example of use: setting Interaction Trigger (EventTrigger) that calls a Command when an Event occurs.

```
<TextBlock Text="{Binding Name}" Tag="#FF0000">
    <i:Interaction.Triggers>
        <i:EventTrigger EventName = "ContextMenuOpening">
               <mvvm:CustomCommandInvoker Command = "{Binding Path=DataContext.ContextMenuOpen
ingCommand, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" />
        </i:EventTrigger>
        <i:EventTrigger EventName = "ContextMenuClosing">
               <mvvm:CustomCommandInvoker Command = "{Binding Path=DataContext.ContextMenuClos
ingCommand, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" />
         </ i:EventTrigger>
    </i:Interaction.Triggers>
    ...
```

Fields

CommandProperty

Comand dependency property that allows (from WPF xaml file) binding an command that need to be invoked (e.g. when an event occurs)

Declaration

```
public static readonly DependencyProperty CommandProperty
```

Field Value

Type	Description
System.Windows.DependencyProperty	

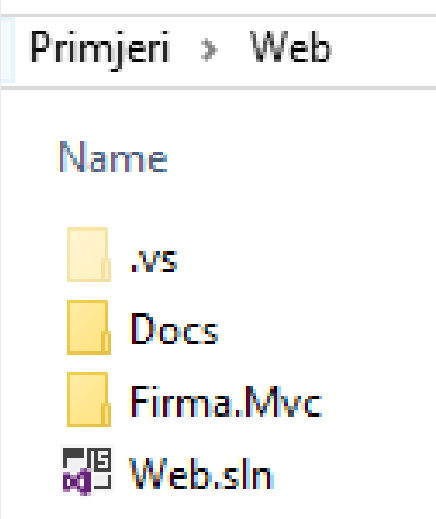
CC BY NC SA

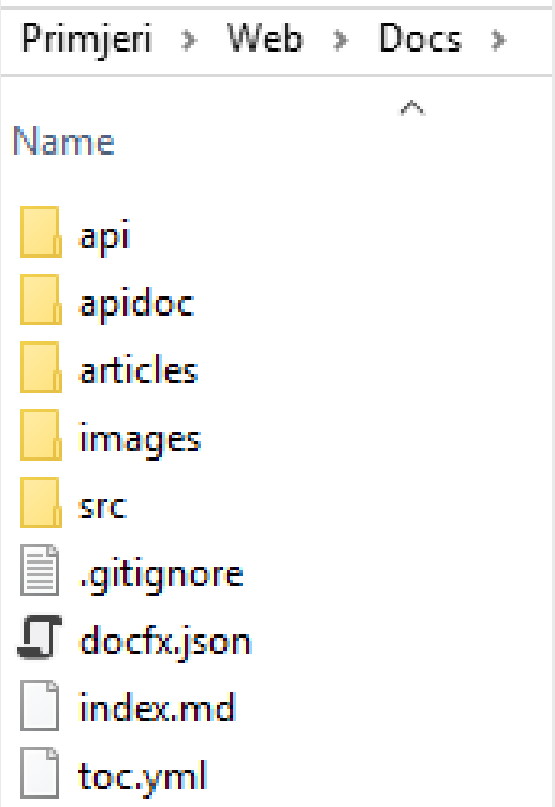
Generiranje programske dokumentacije

- **Upute koje slijede prilagođene su upute originalnih uputa https://dotnet.github.io/docfx/tutorial/docfx_getting_started.html**
  - Preduvjet: Instaliran DocFx, dodan u putanju i uspješno kompilirana web-aplikacija s napisanom dokumentacijom
- **U mapi u kojoj se nalazi sln datoteka (paralelno s mapom web-aplikacije) iz naredbenog retka pokrenuti**

```
docfx init -q -o Docs
```

  - stvara kostur projekta za programsku dokumentaciju
  - Alternativa: Visual Studio 2019:
    Projekt tipa ClassLibrary
    + NuGet paket
    docfx.Console

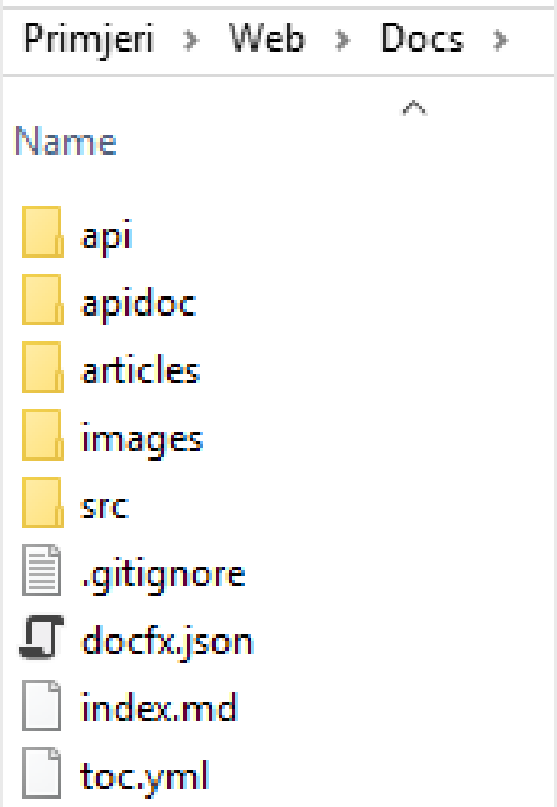




CC BY NC SA

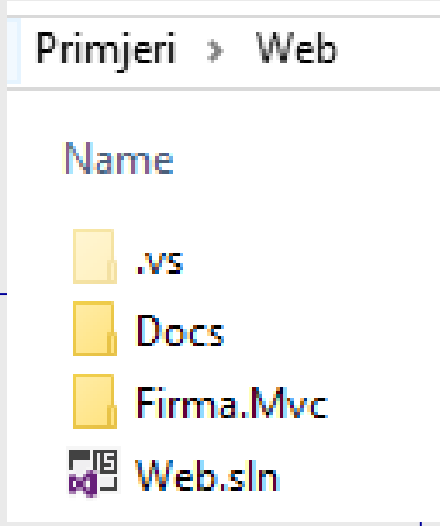
Uređivanje početne stranice dokumentacije

- **Docfx predstavlja strukturu projekta za izradu projektne dokumentacije**
  - Mapa api će naknadno sadržavati informacije o razredima iz web-aplikacije
  - Mapa articles predstavlja proizvoljne dokumente koji se mogu uključiti u programsku dokumentaciju
    - Izmijeniti sadržaj dodatne dokumentacije po uzoru na 🗁 Web \ Docs \ articles \ index.md i toc.yml
- **Toc.yml datoteke predstavljaju strukturu pojedinog dokumenta**
- **Sadržaj stranice piše se koristeći Markdown**
  - ekstenzija md
  - https://en.wikipedia.org/wiki/Markdown
- **Uključiti web-aplikaciju u dokumentaciju**
  - Vidi sljedeći slajd
- **Izmijeniti index.md i toc.yml po uzoru na 🗁 Web \ Docs \ index.md i toc.yml**



BY NC SA

Uključivanje web-aplikacije u izradu dokumentacije



- **Primjer 🗁 Web \ Docs \ docfx.json**

```
"metadata": [
{
 "src": [
   {
    ~~"files": [~~
~~      "src/**.csproj"~~
~~    ],~~
   }
 ],
 "dest": "api"
}
], ...
```

```
"metadata": [
    {
      "src": [
        {
          "files": ["*.csproj"],
          "src": "../Firma.Mvc",
        }
      ],
      "dest": "api"
    }
  ],
```

CC BY NC SA

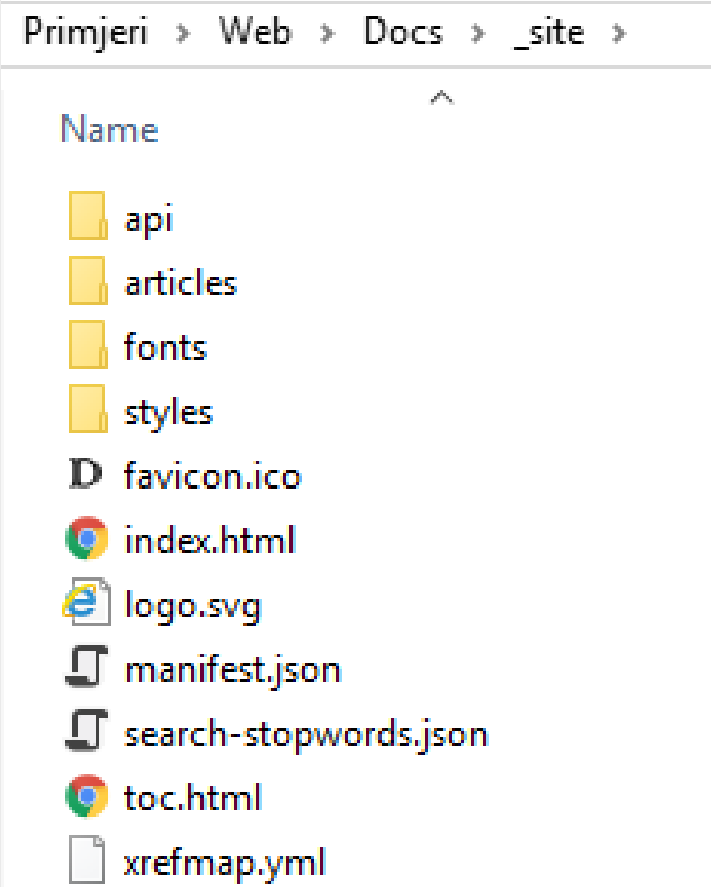
Generiranje dokumentacije

- **U mapi Docs pokrenuti `docfx`**
  - Generira podmapu _site s html stranicama generiranima na osnovi markdown datoteka
- **Za pregled dokumentacije pokrenuti**

```
docfx serve _site
```

  - Privremeni web server s dokumentacijom na adresi http://localhost:8080
- **…ili otvoriti index.html u pregledniku**
  - Ako docfx.json za template ima statictoc umjesto default

```
"template": [ "statictoc"],
```



CC BY NC SA

Izrada PDF-a s generiranom dokumentacijom (1)

- **Potrebno instalirati wkhtmltopdf**
  https://wkhtmltopdf.org/downloads.html
  - Nakon instalacije dodati C:\Program Files\wkhtmltopdf\bin u putanju
- **Kreirati podmapu pdf s definicijom poglavlja**
  - Primjer 🗁 Web \ Docs \ pdf \ toc.yml

```
- name: Dokumenti
  href: ../articles/toc.yml
- name: Programska dokumentacija
  href: ../api/toc.yml
```

- **U docfx.json dodati blok za generiranje pdfa.**
  - Detaljnije u 🗁 Web \ Docs \ docfx.json

  https://dotnet.github.io/docfx/tutorial/walkthrough/walkthrough_generate_pdf.html
- **Pokrenuti `docfx pdf`**
  - Nakon uspješne izgradnje pdf se nalazi u podmapi _site_pdf

CC BY NC SA

Web-aplikacije
ASP.NET Core MVC
Specijalizacija/Generalizacija

2019/20.08-A

CC BY NC SA

Problem prikaza specijalizacija nekog entiteta

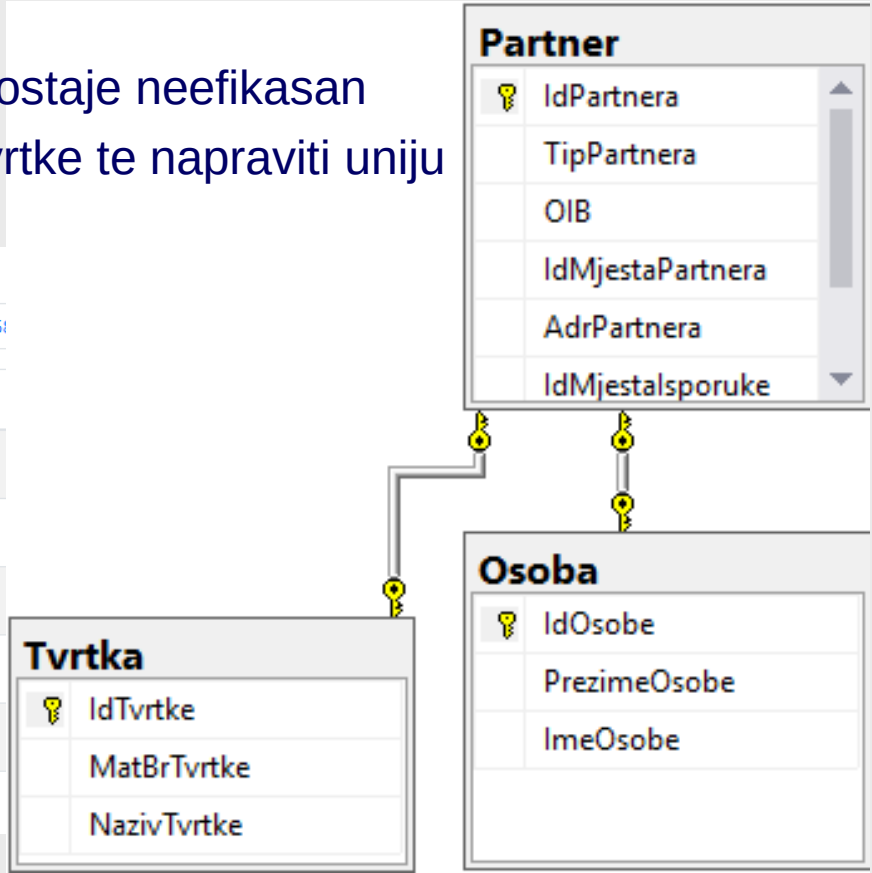
- **U oglednom modelu Osoba i Tvrtka su specijalizacije Partnera na principu TPT (Table per type) te ih EF Core preslikava u 3 tablice**
  - Alternativa TPH (Table per hierarchy): https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/inheritance?view=aspnetcore-3.1
- **Što ako u prikazu svih partnera želimo ispisati id partnera, vrstu partnera, OIB i naziv partnera?**
  - Upit korištenjem EF-a se komplicira i postaje neefikasan
    - potrebno dohvatiti sve osobe, pa tvrtke te napraviti uniju
  - Što ako treba sortirati podatke?

Popis partnera

Unos novog partnera

1.. 31 32 33 34 35 36 37 38 39 40 41 .. 58

Id partnera	Tip partnera	OIB	Naziv
20	Osoba	1410949396506	Mustapić, Hrvoje
750	Tvrtka	3319684	MZOPU
751	Tvrtka	3316041	NADA DIMIC
752	Tvrtka	3312400	NAMA
40	Osoba	1507971335042	Nekić , Ivan
753	Tvrtka	3308761	NEMO



BY NC SA

Pogled za dohvat podataka o partnerima

- **Rješenje prethodnog problema je napisati pogled u bazi podataka te ga uključiti u model**
  - Pogled ima sljedeću definiciju

```
CREATE VIEW [dbo].[vw_Partner]
AS
SELECT IdPartnera, TipPartnera, OIB,
       ISNULL(NazivTvrtke, NazivOsobe) AS Naziv
FROM
(
  SELECT IdPartnera, TipPartnera, OIB,
         PrezimeOsobe + ', ' + ImeOsobe AS NazivOsobe,
         NazivTvrtke
  FROM Partner
  LEFT OUTER JOIN Osoba ON Osoba.IdOsobe = Partner.IdPartnera
  LEFT OUTER JOIN Tvrtka ON Tvrtka.IdTvrtke = Partner.IdPartnera
) T
```

Uključivanje pogleda u EF-model (1)

- **Kreirati razred proizvoljnog imena koji odgovara rezultatu pogleda**
  - Primjer: 🗁 Firma.Mvc \ Models \ ViewPartner.cs
    - Dodatno napisano svojstvo koje opisno prikazuje tip partnera

```
public class ViewPartner {
        public int IdPartnera { get; set; }
        public string TipPartnera { get; set; }
        public string OIB { get; set; }
        public string Naziv { get; set; }
        public string TipPartneraText {
            get {
                if (TipPartnera == "O") {
                    return "Osoba";
                }
                else {
                    return "Tvrtka";
                }
            }
        }
    }
```

Uključivanje pogleda u EF-model (2)

- **U kontekst dodati *DbSet* razreda koji odgovara pogledu**
  - naziv svojstva obično odgovara nazivu pogleda, ali nije nužno
    - Može se navesti u `ToView("naziv pogleda")`
  - također, može se i navesti SQL upit koji vraća rezultat traženog tipa, npr. `ctx.vw_Partner.FromSqlRaw("SELECT * FROM vw_Partner");`
- **Primjer: 🗁 Firma.Mvc \ Models \ FirmaContext.cs**

```
public partial class FirmaContext : DbContext {
    ...
    public virtual DbSet<ViewPartner> vw_Partner { get; set; }
    ...
    modelBuilder.Entity<ViewPartner>(entity => {
        entity.HasNoKey();
        entity.ToView("vw_Partner")
      });
    ...
```

- **Ovako kreirani *DbSet* koristi se kao i drugi *DbSetovi*, ali samo za čitanje**

CC BY NC SA

Model za prikaz svih partnera

- (Kao i u prethodnim primjerima) model se sastoji od enumeracije razreda kojim se opisuje pojedinačni podatak i informacija o straničenju i sortiranju
- Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ PartneriViewModel.cs

```
namespace Firma.Mvc.ViewModels{
  public class PartneriViewModel
  {
     public IEnumerable<ViewPartner> Partneri { get; set; }
     public PagingInfo PagingInfo { get; set; }
  }
}
```

CC BY NC SA

Priprema modela za prikaz svih partnera

- **Podaci se pripremaju kreiranjem upita za pogled dodan u EF model**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ PartnerController.cs

```
public IActionResult Index(string filter, int page = 1,
                           int sort = 1, bool ascending = true) {
      int pagesize = appData.PageSize;
      var query = ctx.vw_Partner.AsQueryable();
      ...proširenje upita (sortiranje)
      var partneri = query
                     .Skip((page - 1) * pagesize)
                     .Take(pagesize)
                     .ToList();
      var model = new PartneriViewModel {
        Partneri = partneri,
        PagingInfo = pagingInfo
      };
      return View(model);
}
```

CC BY NC SA

Proširenje upita redoslijedom sortiranja

- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ PartnerController.cs**

```
public IActionResult Index(string filter, int page = 1,
                           int sort = 1, bool ascending = true) {
      ...
  System.Linq.Expressions.Expression<Func<ViewPartner,
object>> orderSelector = null;
  switch (sort) {
     case 1:
       orderSelector = p => p.IdPartnera; break;
     case 2:
       orderSelector = p => p.TipPartnera; break;
        ...
  }
  if (orderSelector != null) {
     query = ascending ?
             query.OrderBy(orderSelector) :
             query.OrderByDescending(orderSelector);
  }
       ...
```

Prikaz svih partnera

- **Primjer (po uzoru na prethodne) 🗁 Firma.Mvc\Views\Index.cshml**

```
@model PartneriViewModel
...
 @foreach (var partner in Model.Partneri) {
     <tr>
        <td class="text-left">@partner.IdPartnera</td>
        <td class="text-left">@partner.TipPartneraText</td>
        <td class="text-left">@partner.OIB</td>
        <td class="text-left">@partner.Naziv</td>
        <td>
             <a asp-action="Edit"
                asp-route-id="@partner.IdPartnera"
                asp-route-page="@Model.PagingInfo.CurrentPage"
                ... class="btn btn-warning btn-sm" title="Ažuriraj">
               <i class="fas fa-edit"></i></a>
        </td>
        <td>
          <form asp-action="Delete" method="post"
                   asp-route-page="@Model.PagingInfo.CurrentPage" ...>
          <input type="hidden" name="IdPartnera"
                               value="@partner.IdPartnera" />
          <button type="submit" title="Obriši">…
```

Stvaranje objekta kao jedne od specijalizacija

- **Za prijenos podataka između pogleda i upravljača za stvaranje novog partnera definiran je novi prezentacijski model koji sadrži sve atribute osobe, ali i tvrtke**
  - Alternativa: razviti dvije odvojene akcije i dva različita pogleda
  - Primjer: 🗁 Firma.Mvc \ ViewModels \ PartnerViewModel.cs

```
public class PartnerViewModel {
   public int IdPartnera { get; set; }
   RegularExpression("[OT]")]
   public string TipPartnera { get; set; }
   public string PrezimeOsobe { get; set; }
   public string ImeOsobe { get; set; }
   public string MatBrTvrtke { get; set; }
   public string NazivTvrtke { get; set; }
   [Required]
   [RegularExpression("[0-9]{11}")]
   public string Oib { get; set; }
   public string AdrPartnera { get; set; }
   public int? IdMjestaPartnera { get; set; }
   public string NazMjestaPartnera { get; set; }
...
```

CC BY-NC-SA

Priprema za unos novog partnera

- **Inicijalno postavljeno da se radi o osobi, ali moguće promijeniti prije samog unosa**
  - Primjer: 🗁 Firma.Mvc \ Controllers \ PartnerController.cs

```
[HttpGet]
public IActionResult Create()
{
   PartnerViewModel model = new PartnerViewModel
   {
     TipPartnera = "O"
   };
   return View(model);
}
```

BY NC SA

Odabir tipa partnera (1)

- **Odabir se vrši korištenjem *radiobuttona***
  - *radiobutton* ima naziv generiran na osnovi *tag-helpera asp-for* i naziva odgovarajućeg svojstva iz modela te pridruženu vrijednost
  - tekst se neovisno navodi ispred ili iza
  - Primjer: 🗁 Firma.Mvc \ Views \ Partner \ Create.cshtml

```
@model PartnerViewModel
...
<form asp-action="Create" method="post">
  <label class="radio-inline">
     <input type="radio" asp-for="TipPartnera" value="O">
  Osoba</label>
  <label class="radio-inline">
      <input type="radio" asp-for="TipPartnera" value="T">
   Tvrtka</label>
```

Odabir tipa partnera (2)

- **Dio kontrola treba prikazati samo ako se unosi nova osoba, odnosno ako se unosi nova tvrtka.**
  - Bit će izvršeno korištenjem *jQuerya*, ali je potrebno takve kontrole označiti odgovarajućim imenima ili stilovima
    - U primjeru se koristi stil koji nema svoje vizualne osobine već služi samo za pronalazak takvih kontrola
  - Primjer: 🗁 Firma.Mvc \ Views \ Partner \ Create.cshtml

```
@model PartnerViewModel
...
<form asp-action="Create" method="post">
  ...
  <div class="form-group samotvrtka">
        <label asp-for="MatBrTvrtke"></label>
         <input asp-for="MatBrTvrtke" class="form-control" />
  </div>
  <div class="form-group samoosoba">
        <label asp-for="ImeOsobe"></label>
        <input asp-for="ImeOsobe" class="form-control" />
  </div>
  ...
```

CC BY NC SA

Odabir tipa partnera (3)

- **Nakon učitavanja stranice te pri svakoj promjeni označenog radiobuttona skrivaju se odnosno prikazuje odgovarajuće kontrole**
  - Primjer: 🗁 Firma.Mvc \ Views \ Partner \ Create.cshtml

```
@section scripts{
    <script type="text/javascript">
        $(function () {
            $('input:radio').change(function () {
                OsobaIliTvrtka($(this).val());
            });
            OsobaIliTvrtka($('input:checked').val());
        });
        function OsobaIliTvrtka(tip) {
            if (tip == 'O') {
              $(".samotvrtka").hide(); $(".samoosoba").show();
            }
            else {
              $(".samoosoba").hide(); $(".samotvrtka").show();
            }
        }
    </script>
```

BY NC SA

Validacija prilikom unosa novog partnera (1)

- **Model *PartnerViewPartner* sadrži podatke i za osobu i za tvrtku**
  - Atribut *Required* na imenu osobe nema smisla ako je partner tvrtka
    - Štoviše morao bi se popuniti nekim besmislenim podatkom da bi se forma mogla poslati na server
- **Potrebno napraviti dodatnu provjeru vlastitim programskim kodom**
  - Moguće napisati vlastiti validacijski atribut ili implementirati sučelje *IValidatableObject*
  - Vraća enumeraciju s opisom pogrešaka (objekti tipa *ValidationResult*)
  - Validacija se odvija samo na serveru!
- **Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ PartnerViewModel.cs**

```
public class PartnerViewModel : IValidatableObject {
 public IEnumerable<ValidationResult>
            Validate(ValidationContext validationContext) {
...

ValidationResult("Potrebno je upisati ime osobe", new [] {
nameof(ImeOsobe) } );
...
```

CC BY NC SA

Validacija prilikom unosa novog partnera

- **Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ PartnerViewModel.cs**
- **yield return vraća jedan po jedan rezultat**
  - Može se koristiti ako je povratni tip IEnumerable ili IEnumerator

```
public IEnumerable<ValidationResult>
            Validate(ValidationContext validationContext) {
 if (TipPartnera == "O") {
    if (string.IsNullOrWhiteSpace(ImeOsobe))
        yield return new ValidationResult(
            "Potrebno je upisati ime osobe",
             new [] { nameof(ImeOsobe) } );
    if (string.IsNullOrWhiteSpace(PrezimeOsobe))
        yield return new ValidationResult(
            "Potrebno je upisati prezime osobe",
             new [] { nameof(PrezimeOsobe) } );
 }
 ...
```

Unos novog partnera (1)

- **Potrebno stvoriti novi objekt tipa Partner te kopirati svojstva iz modela (uključuje i inicijalizaciju svojstva Osoba ili Tvrtka)**
  - Vlastita metoda CopyValues prikazana na sljedećem slajdu
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ PartnerController.cs

```
public IActionResult Create(PartnerViewModel model) {
  if (ModelState.IsValid) {
       Partner p = new Partner();
       p.TipPartnera = model.TipPartnera;
       CopyValues(p, model); //kopiraj podatke iz modela u p
       try
       {
         ctx.Add(p);
         ctx.SaveChanges();
         ...
```

Unos novog partnera (2)

- **Kopiraju se potrebna svojstva te stvara nova instanca Osobe ili Tvrtke**
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
void CopyValues(Partner partner, PartnerViewModel model) {
      partner.AdrIsporuke = model.AdrIsporuke;
      partner.AdrPartnera = model.AdrPartnera;
      partner.IdMjestaIsporuke = model.IdMjestaIsporuke;
      partner.IdMjestaPartnera = model.IdMjestaPartnera;
      partner.Oib = model.Oib;
      if (partner.TipPartnera == "O") {
        partner.Osoba = new Osoba();
        partner.Osoba.ImeOsobe = model.ImeOsobe;
        partner.Osoba.PrezimeOsobe = model.PrezimeOsobe;
      }
      else {
        partner.Tvrtka = new Tvrtka();
        partner.Tvrtka.MatBrTvrtke = model.MatBrTvrtke;
        partner.Tvrtka.NazivTvrtke = model.NazivTvrtke;
      }
}
```

CC BY NC SA

Strani ključ i *identity* tip podatka

- **U postupku CopyValues stvoren novi objekt tipa Osoba ili Tvrtka**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ PartnerController.cs

```
public IActionResult Create(PartnerViewModel model) {
        ...
        Partner p = new Partner();
        ...
           //CopyValues: p.Osoba = new Osoba();
        ...
        ctx.Add(p);
        ctx.SaveChanges();
```

  - EF će automatski stvoriti odgovarajuće dvije *Insert* naredbe
  - Uz prvu Insert naredbu EF izvršava i upit za dohvat *identity* vrijednosti primarnog ključa koja se koristi kao primarni i strani ključ za tablicu Osoba odnosno Tvrtka.

CC BY NC SA

Nadopunjavanje umjesto padajuće liste

- **Izbor mjesta partnera**
  - Velik broj mogućih mjesta – nije prikladno za padajuću listu
- **Koristi se nadopunjavanje (engl. *autocomplete*), odnosno dinamička padajuća lista**
  - U pogledu se definira obično polje za unos kojem se pridružuje klijentski kod koji poziva određenu stranicu na serveru koja vraća tražene podatke osnovi trenutno upisanog teksta
    - Npr. za tekst `breg` stranica će vratiti sva mjesta koja u nazivu sadrže riječ `breg` (npr. Bregana, Lugarski Breg, Bregi, …)
    - Rezultat ovisi o postupku na serveru
  - Podaci koje stranica vraća bit će parovi oblika (identifikator, oznaka)
    - npr. `5489, "21000 Split"`
  - Upisani tekst predstavljat će naziv mjesta, a dohvaćeni identifikator mjesta će se pohraniti u skriveno polje: podaci su parovi oblika (identifikator, tekst)
- **Kako prepoznati kontrole kojima treba pridružiti dinamičke padajuće liste i gdje pohraniti identifikator?**
  - Te informacije bit će zapisane u *data* atribute oblika *data-naziv*

CC BY NC SA

Razred za pohranu rezultata servisa (1)

- **Podaci za dinamičku padajuću listu (nadopunjavanje) sastoje se od identifikatora i oznake**
  - Primjer: 🗁 Firma.Mvc \ Areas \ AutoComplete \ Models \ IdLabel.cs

```
public class IdLabel
{
    public string Label { get; set; }
    public int Id { get; set; }
    public IdLabel() { }
    public IdLabel(int id, string label) {
        Id = id;
        Label = label;
    }
}
```

- **Pretvorbom u JSON nastat će rezultat nalik sljedećem tekstu**

```
[{"label":"42000 Varaždin","id":6245},{"label":"42204 Varaždin
Breg","id":6246},{"label":"42223 Varaždinske Toplice","id":6247}]
```

pri čemu nije sigurno hoće li nazivi svojstava biti zapisani velikim ili malim slovom

CC BY NC SA

Razred za pohranu rezultata servisa (2)

- **Za izbjegavanje nedoumica i potencijalnih problema kod serijalizacije i deserijalizacije u JSON, eksplicitno se navode nazivi korištenjem atributa JsonPropertyName**
  - Primjer: 🗁 Firma.Mvc \ Areas \ AutoComplete \ Models \ IdLabel.cs

```
[DataContract]
public class IdLabel
{
    [JsonPropertyName("label")]
    public string Label { get; set; }
    [JsonPropertyName("id")]
    public int Id { get; set; }
    ...
```

BY NC SA

Upravljač za dohvat podataka za nadopunjavanje

- **Postupak ne vraća pogled, već enumeraciju parova (id, oznaka)**
  - ASP.NET Core automatski pretvara u JSON (ili xml ovisno o postavkama)
  - traži se podniz – ulazni argument se mora zvati *term*
    - u čemu tražimo podniz? uključujemo li i pretragu po poštanskog broja
  - projekcija iz skupa entiteta *Mjesto* u listu objekata tipa *Label*
  - Primjer: 🗁 Firma.Mvc\Areas\AutoComplete\Controllers\MjestoController.cs

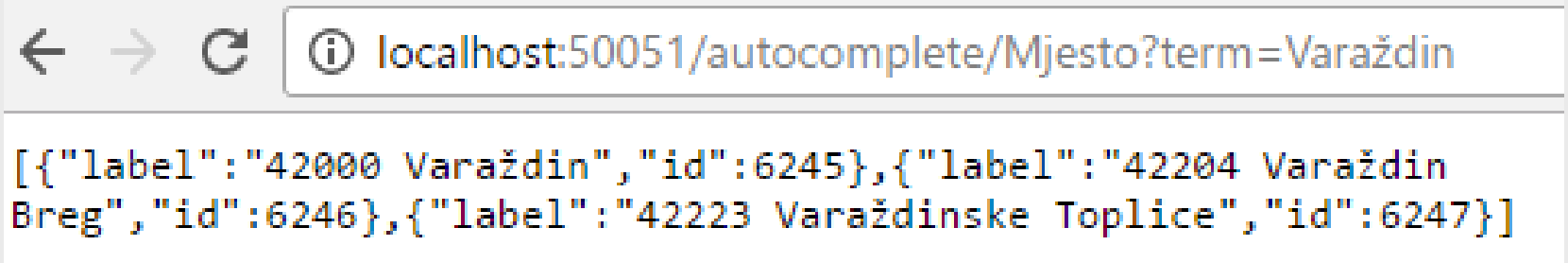
```
public IEnumerable<IdLabel> Get(string term) {
  var query = ctx.Mjesto
                 .Select(m => new IdLabel {
                   Id = m.IdMjesta,
                   Label = m.PostBrMjesta + " " + m.NazMjesta
                  })
                 .Where(l => l.Label.Contains(term));

  var list = query.OrderBy(l => l.Label)
                  .Take(appData.AutoCompleteCount)
                  .ToList();
  return list;
```

Usmjeravanje do upravljača (1)

- **Upravljač s prethodnog slajda nalazi se posebnom području (area)**
- **Usmjeravanje za područja se obično definira na sljedeći način**
  - Primjer: 🗁 Firma.Mvc \ Startup.cs - Configure

```
endpoints.MapControllerRoute(
                "AutoCompleteRoute",
                "autocomplete/{controller}/{action=Get}",
                new { area = "AutoComplete" }
              );
```

  - U demonstriranom primjeru sve akcije za nadopunjavanje će imati naziv Get
    - u kladu s imenovanjem u REST servisima (o REST-u neki drugi put)



CC BY NC SA

Usmjeravanje do upravljača (2)

- **Umjesto definiranja posebnog područja, putanja se mogla navesti i atributom *Route***
  - Atribute HttpGet označava postupak koji će se izvršiti pozivom oblika *autocomplete/Mjesto*

```
[Route("autocomplete/[controller]")]
public class MjestoController : Controller

[HttpGet]
public IEnumerable<IdLabel> Get(string term) {
       ...
```

BY NC SA

Priprema i označavanje kontrola za unos

- **U pogledu definirano obično polje za unos kojem će se naknadno pridružiti klijentski kod**
  - Upisani tekst predstavljat će naziv mjesta, a dohvaćeni identifikator mjesta će se pohraniti u skriveno polje: servis vraća parove oblika (id, tekst)
  - *data-autocomplete* sadrži relativnu adresu servisa
  - *s data-autocomplete-placeholder-name* označena vrijednost koja će se tražiti u atributu *data-autocomplete-placeholder* skrivenog unosa
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Partner \ Create.cshml**

```
<label asp-for="IdMjestaPartnera"></label>
<input class="form-control"
       data-autocomplete="mjesto"
       data-autocomplete-placeholder-name="mjestopartnera"
       value="@Model.NazMjestaPartnera" />
<input type="hidden" asp-for="IdMjestaPartnera"
       data-autocomplete-placeholder="mjestopartnera" />
...
@section scripts{
    <script src="~/lib/jquery-ui/jquery-ui.js"></script>
    <script src="~/js/autocomplete.js"></script>
```

BY NC SA

Aktiviranje nadopunjavanja (1)

- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ autocomplete.js**
  - Za svaki element koji ima definiran vlastiti atribut data-autocomplete:
    - dohvati relativnu adresu izvora podataka (iz data-autocomplete)
    - dohvati naziv elementa u koji se posprema dohvaćena vrijednost
    - brisanjem teksta i promjenom fokusa izbriši staru vrijednosti

```
$("[data-autocomplete]").each(function (index, element) {
  var url = $(element).data('autocomplete');
  var resultplaceholder =
            $(element).data('autocomplete-placeholder-name');
  if (resultplaceholder === undefined) resultplaceholder = url;
  $(element).change(function () {
      var dest = $("[data-autocomplete-placeholder='" +
resultplaceholder + "']");
      var text = $(element).val();
      if (text.length === 0 || text !== $(dest).data('selected-
label')) {
        $(dest).val('');
      }
    });
  ... aktiviraj autocomplete na elementu ...
```

BY NC SA

Aktiviranje nadopunjavanja (2)

- **Koristi *se jQuery autocomplete***
  - postavlja se adresa izvora podataka
  - minimalna potrebna duljina teksta za nadopunjavanje i slično
  - akcija koja će se izvršiti odabirom nekog elementa iz liste
  - u primjeru tekst će se kopirati u polje za unos, a identifikator u skriveno polje
- **Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ autocomplete.js**

```
... aktiviraj autocomplete na elementu ...
$(element).autocomplete({
            source: "/autocomplete/" + url,
            autoFocus: true,
            minLength: 1,
            select: function (event, ui) {
              $(element).val(ui.item.label);
              var dest = $("[data-autocomplete-
                   placeholder='" + resultplaceholder + "']");
              $(dest).val(ui.item.id);
              $(dest).data('selected-label', ui.item.label);
            }
        });
```

Pogreške prilikom dodavanja novog partnera

- **Model može biti neispravan ili se može dogoditi pogreška**
  - Prethodno povezani podaci su dio modela koji se vraćaju pogledu
  - Dodatno se vrši dohvat naziva mjesta na osnovu identifikatora mjesta
    - povezani u *model.IdMjestaIsporuke* i *model.IdMjestaPartnera*
    - potrebno u slučaju da je korisnik promijenio naziv mjesta, a nije odabrao neko mjesto iz padajuće liste
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
try {
          ctx.Add(p);
          ctx.SaveChanges();
...
}
catch (Exception exc) {
          DohvatiNaziveMjesta(model);
          ModelState.AddModelError(string.Empty,
                    exc.CompleteExceptionMessage());
          return View(model);
}
```

Dohvat podataka o partneru

- **Kao model za pogled koristi se isti model kao kod dodavanja**
  - Prvo se vrši dohvat zajedničkih podataka iz tablice Partner
  - Ostatak podataka puni se upitom na tablicu Osoba ili Tvrtka
    - Umjesto Find mogu se koristiti i varijante s Where
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
public IActionResult Edit(int id, ...) {
      var partner = ctx.Partner.Find(id);
       ...
      PartnerViewModel model = new PartnerViewModel {
          IdPartnera = partner.IdPartnera,
          IdMjestaIsporuke = partner.IdMjestaIsporuke,
          ...
          TipPartnera = partner.TipPartnera
      };
      if (model.TipPartnera == "O") {
         Osoba osoba = ctx.Osoba.Find(model.IdPartnera);
         model.ImeOsobe = osoba.ImeOsobe;
         model.PrezimeOsobe = osoba.PrezimeOsobe;
      }...
```

BY NC SA

Ažuriranje podataka o partneru (1)

- **Podaci povezani kroz model se provjeravaju na validacijske pogreške (slično kao kod *Create*)**
  - Ako je model ispravan, vrši se dohvat partnera iz BP te se (vlastitim postupkom) kopiraju vrijednosti iz primljenog modela u entitet iz EF
  - Primijetiti da se ne radi Include na Osoba ili Tvrtka
    - više na slajdovima koji slijede
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(PartnerViewModel model...) {
  var partner = ctx.Partner.Find(model.IdPartnera);
  ...
 if (ModelState.IsValid)
      {
        try
        {
          CopyValues(partner, model);
          ...
```

CC BY NC SA

Ažuriranje podataka o partneru (2)

- **Mijenjaju se sva svojstva osobe ili tvrtke**
  - prilikom dohvata partnera nije uključen i dohvat podataka o osobi ili tvrki
    - stoga je svojstvo Osoba (Tvrtka) jednako null te se instancira novi objekt
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
void CopyValues(Partner partner, PartnerViewModel model) {
      partner.AdrIsporuke = model.AdrIsporuke;
      partner.AdrPartnera = model.AdrPartnera;
      partner.IdMjestaIsporuke = model.IdMjestaIsporuke;
      partner.IdMjestaPartnera = model.IdMjestaPartnera;
      partner.Oib = model.Oib;
      if (partner.TipPartnera == "O") {
        partner.Osoba = new Osoba();
        partner.Osoba.ImeOsobe = model.ImeOsobe;
        partner.Osoba.PrezimeOsobe = model.PrezimeOsobe;
      }
      else {
        partner.Tvrtka = new Tvrtka();
        partner.Tvrtka.MatBrTvrtke = model.MatBrTvrtke;
        partner.Tvrtka.NazivTvrtke = model.NazivTvrtke; ...
```

Snimanje promjena

- **Budući da je povezani dio za osobu ili tvrtku nastao stvaranjem novog objekta, a ne ažuriranjem onog dohvaćenog iz BP, EF ga smatra novim objektom te bi za njega definirao insert upit**
  - Eksplicitno mijenjamo stanje tog objekta iz *Added* u *Modified i postavljamo vrijednost PK* => uzrokuje *update* upit, a ne *insert*
  - Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs

```
public IActionResult Edit(PartnerViewModel model, ... {
   ...
   var partner = ctx.Partner.Find(model.IdPartnera);
   ...
   CopyValues(partner, model);
   if (partner.Osoba != null) {
      partner.Osoba.IdOsobe = partner.IdPartnera;
      ctx.Entry(partner.Osoba).State = EntityState.Modified;
   }
   if (partner.Tvrtka != null) {
      partner.Tvrtka.IdTvrtke = partner.IdPartnera;
      ctx.Entry(partner.Tvrtka).State = EntityState.Modified;
   }
   ctx.SaveChanges();
```

Brisanje partnera

- **Definirano kaskadno brisanje u BP.**
  - Prilikom generiranja EF modela ta je činjenica uzeta u obzir
- **Dovoljno obrisati se entitet iz skupa Partner.**
  - Odgovarajući zapis iz tablice Osoba ili Tvrtka se automatski briše
- **Dohvat se može izvršiti s *Where* ili postupkom *Find* navođenjem vrijednosti primarnog ključa**
- **Primjer: 🗁 Firma.Mvc \ Controllers \ MjestoController.cs**

```
public IActionResult Delete(int IdPartnera, ...) {
   var partner = ctx.Partner.Find(IdPartnera);
   if (partner != null) {
        try {
          ctx.Remove(partner);
          ctx.SaveChanges();
...
```

  - Umjesto ctx.Remove moglo se napisati i

```
ctx.Entry(partner).State = EntityState.Deleted;
```

BY NC SA

Web-aplikacije
ASP.NET Core MVC
Slanje datoteke i prikaz slike

2019/20.08

BY NC SA

Prikaz svih artikala

- **Prikaz pojedinačnog artikla izveden u parcijalnom pogledu *Row.cshtml* koji se uključuje u *Index.cshtml* za svaki artikl**
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Artikl \ Index.cshtml

```
@foreach (var artikl in Model.Artikli) {
     <partial name="Row" model="artikl" />
}
```

  - Motivacija kao u primjeru s mjestima - dinamičko ažuriranje unutar retka
- **Slika artikla pohranjena u tablici Artikl → Što sadrži model za prikaz pojedinog artikla**
  - Preuzeti sliku zajedno s ostalim podacima o artiklu?

    → sporiji dohvat i korisnik više čeka na rezultat
  - U upitu dohvatiti samo osnovne podatke o artiklu, a sliku isporučiti na zahtjev?

- ***Dilema neovisna o načinu dohvata podataka:***
  - *Gdje spremati slike?*
  - *Serverski cache za slike?*

CC BY NC SA

Problem promjene slike artikla

- **Korisnikov preglednik posprema dohvaćene elemente u *cache***
- **Što ako promijenimo sliku artiklu?**
- **Nova slika = nova adresa!**
  - Svakoj slici pridijeliti jedinstveni broj?
  - Koristiti neku varijantu *checksuma*?
- **Dodano novo polje u tablicu Artikl**
  - Izračunato polje CHECKSUM(SlikaArtikla)

RPPP2\SQL2012.Firma - dbo.Artikl

Column Name	Data Type	Allow Nulls
SifArtikla	int	☐
NazArtikla	nvarchar(255)	☐
JedMjere	nvarchar(5)	☐
CijArtikla	money	☐
ZastUsluga	bit	☐
TekstArtikla	nvarchar(MAX)	☑
SlikaArtikla	varbinary(MAX)	☑
SlikaChecksum		☑

Column Properties

Computed Column Specification	(checksum([SlikaArtikla]))
(Formula)	(checksum([SlikaArtikla]))
Is Persisted	No

CC BY NC SA

Izmjena EF modela za Artikl

- **EF model osvježen s izračunatim svojstvom za sliku artikla**
  - Primjer: 🗁 Web \ Firma.Mvc \ Models\ Artikl.cs

```
public partial class Artikl
{
 ...
 public int? SlikaChecksum { get; set; }
```

  - Primjer: 🗁 Web \ Firma.Mvc \ Models\ FirmaContext.cs

```
protected override void OnModelCreating(
                                ModelBuilder modelBuilder) {
  modelBuilder.Entity<Artikl>(entity => {
        entity.HasKey(e => e.SifArtikla)
            .IsClustered(false);
         ...
        entity.Property(e => e.SlikaChecksum)
              .HasComputedColumnSql(
                    "(checksum([SlikaArtikla]))");
      });
```

Model za prikaz artikala

- **Primjer: 🗁 Web \ Firma.Mvc \ Models \ ArtikliViewModel**

```
public class ArtikliViewModel
  public IEnumerable<ArtiklViewModel> Artikli { get; set; }
  public PagingInfo PagingInfo { get; set; }
}
```

- **Model za pojedinačni artikl je razred *ArtiklViewModel***
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ ArtiklViewModel.cs

```
public class ArtiklViewModel {
    public int SifraArtikla { get; set; }
    public string NazivArtikla { get; set; }
    public string JedinicaMjere { get; set; }
    public decimal CijenaArtikla { get; set; }
    public bool Usluga { get; set; }
    public string TekstArtikla { get; set; }
    public bool ImaSliku { get; set; }
    public int? ImageHash { get; set; }
  }
```

Dohvat svih artikala

- **Koristi se posebni razred umjesto razreda Artikl iz EF modela kako se ne bi prilikom dohvata svih artikala istovremeno preuzimale i slike svih artikala**
  - Umjesto sadržaja slike, evidentira se postoji li slika
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ArtiklController.cshtml**

```
var artikli = ctx.Artikl
                 .Select(a => new ArtiklViewModel {
                          SifraArtikla = a.SifArtikla,
                          NazivArtikla = a.NazArtikla,
                          JedinicaMjere = a.JedMjere,
                          CijenaArtikla = a.CijArtikla,
                          Usluga = a.ZastUsluga,
                          TekstArtikla = a.TekstArtikla,
                          ImaSliku = a.SlikaArtikla != null,
                          ImageHash = a.SlikaChecksum})
                 .Skip(...
                 .Take(...
             ...
```

Poveznica za prikaz slike

- **Ako artikl koji se prikazuje u pojedinom retku ima sliku, stvara se HTML *img* kontrola**
- **Adresa slike je akcije *GetImage* na upravljaču *Artikl***
  - Adresi slike se dodaje parameter *hash* kako bi preglednik mogao prepoznati novu sliku artikla u odnosu na onu koju ima spremljenu u svojoj memoriji
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Artikl \ Row.cshtml**

```
@if (Model.ImaSliku) {
     <img src="@Url.Action("GetImage",
                           "Artikl",
                          new { id = @Model.SifraArtikla,
                                hash=Model.ImageHash })"
          class="malaslika" />
}
```

Akcija za dohvat slike

- **Polje bajtova koje predstavlja sliku artikla dohvati se EF upitom**
  - Rezultat postupka je *FileContentResult* koji nastane pozivom nasljeđenog postupka *File* iz upravljača.
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ArtiklController.cs

```
public FileContentResult GetImage(int id) {
      byte[] image = ctx.Artikl
                        .Where(a => a.SifArtikla == id)
                        .Select(a => a.SlikaArtikla)
                        .SingleOrDefault();
      if (image != null)
        return File(image, "image/jpeg");
      else
        return null;
}
```

  - Napomena: Povratna vrijednost je mogla biti i IActionResult, pa je umjesto `return null` moglo pisati `return NotFound()`

CC BY NC SA

Forma za slanje datoteke

- **Prilikom unosa novog artikla može se poslati datoteka sa slikom artikla**
- **Za unos se koristi HTML *input* kontrola tipa *file***
  - Naziv može biti proizvoljan, ali mora odgovarati argumentu u akciji upravljača
- **Forma mora imati atribut *enctype* postavljen na *multipart/form-data***
- **Primjer: 🗁 Web \ Firma.Mvc \ Views \ Artikl \ Create.cshtml**

```
<form asp-action="Create" method="post"
      enctype="multipart/form-data">
...
 <label asp-for="SlikaArtikla" ...></label>
     <div class="col-sm-5">
       <input type="file" name="slika" />
     </div>
...
 <button ... type="submit">Dodaj</button>
```

CC BY NC SA

Prihvat datoteke na upravljaču

- **Postupak prima objekt tipa Artikl (rekonstruiran na osnovi podataka iz forme) i objekt tipa *IFormFile***
  - Naziv argumenta jednak atributu *name* u kontroli za odabir slike
  - Sadržaj poslane podatke se može kopirati u *MemoryStream* i dobiti kao polje bajtova i pospremiti u entitet Artikl
    - U primjeru se originalna slika smanji prije pohrane u bazu podataka
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ArtiklController.cs**

```
[HttpPost][ValidateAntiForgeryToken]
public IActionResult Create(Artikl artikl, IFormFile slika){
  ...
  if (slika != null && slika.Length > 0) {
    using (MemoryStream stream = new MemoryStream()) {
              slika.CopyTo(stream);
              byte[] image = stream.ToArray();
              image = CreateThumbnail(image);
              artikl.SlikaArtikla = image;
    }
  }
  ctx.Add(artikl);ctx.SaveChanges();     ...
```

BY NC SA

Alati za smanjivanje slike

- **Ugrađena podrška u CoreCompact.System.Drawing**
  - različiti dodatne dll datoteke za Windows/Linux
- **Neki od paketa za rad sa slikama**
  - https://andrewlock.net/using-imagesharp-to-resize-images-in-asp-net-core-a-comparison-with-corecompat-system-drawing/
  - https://blogs.msdn.microsoft.com/dotnet/2017/01/19/net-core-image-processing/
- **LibVips kao jedan od najbržih alata**
  - https://libvips.github.io/libvips/
    - Od 4. mj. 2018 NetVips kao premosnica za .NET
      - https://kleisauke.github.io/net-vips/tutorial/getting_started.html
  - U primjeru koji slijedi demonstrirat će se varijanta pozivanja vanjskog programa
    - Pozvat će se naredba nalik sljedećoj

```
vipsthumbnail.exe -t velika.jpg -o mala.jpg --size=150x100
```

    - Potrebno instalirati libvips i u konfiguracijskoj datoteci navesti putanju do bin direktorija

Dodatne aplikacijske postavke

- **Konfiguracijska datoteka proširena putanjom do izvršne datoteke za izradu manje slike i visinom na koju treba smanjiti sliku**
  - Primjer: 🗁 Web \ Firma.Mvc \ AppSettings.cs

```
 public class AppSettings {
    ...
    public ImageSettingsData ImageSettings { get; set; }
    public class ImageSettingsData {
      public int ThumbnailHeight { get; set; } = 100;
      public string LibVipsThumbnail { get; set; }
    }
}
```

  - Primjer: 🗁 Web \ Firma.Mvc \ appsettings.json

```
{
  "AppSettings": {
    "PageSize": 10, ...
    "ImageSettings": {
      "ThumbnailHeight": 100,
      "LibVipsThumbnail": "C:\\libvips\\vips-dev-
8.6\\bin\\vipsthumbnail.exe"
    }
```

Pokretanje vanjskog procesa

- **Koristi se statički postupak *Start* iz razreda *Process***
  - Parametri pohranjeni u objektu tipa *ProcessStartInfo*
- **Primjer: 🗁 Web \ Firma.Mvc \ Util \ ImageTool \ LibVips.cs**

```
string arguments = $"-t {input.Path} -o {output.Path} --
size={maxwidth}x{maxheight}";
ProcessStartInfo pi = new ProcessStartInfo(
                            exeThumbnail, arguments);
pi.UseShellExecute = false;
pi.RedirectStandardOutput = true;
pi.RedirectStandardError = true;
using (var process = Process.Start(pi)) {
    process.WaitForExit();
    result.ExitCode = process.ExitCode;
    result.StandardOutput = process.StandardOutput
                                    .ReadToEnd();
    result.StandardError = process.StandardError
                                    .ReadToEnd();
}
result.Data = File.ReadAllBytes(output.Path);
```

BY NC SA

Privremena datoteka (1)

- **Libvips treba datoteku na disku, a slika se nalazi u memoriji**
- **Stvoriti privremenu datoteku i obrisati nakon korištenja**
  - *System.IO.Path.GetTempFileName()* stvara privremenu datoteku
    - Problem: Može se stvoriti maksimalno 65535 privremenih datoteka bez brisanja prethodnih
  - Osigurati brisanje nakon korištenja → IDisposable
  - Bit će implementirao u vlastitom razredu TempFile
  - Primjer: 🗁 Web \ Firma.Mvc \ Util \ ImageTool \ LibVips.cs

```
using (TempFile input = new TempFile()) {
   File.WriteAllBytes(input.Path, data);
   using (TempFile output = new TempFile(input.Path+".jpg"))
   {
       ... Pozovi libvips ...
       result.Data = File.ReadAllBytes(output.Path);
    }
}
```

BY NC SA

Privremena datoteka (2)

- **Implementacija osigurava da se datoteka obriše čak i ako korisnik ne upotrijebi sučelje IDisposable**
  - Primjer potrebe za finalizatorom!
  - Primjer: 🗁 Web \ Firma.Mvc \ Util \ TempFile.cs

```
class TempFile : IDisposable {
  string path;
  public TempFile():this(System.IO.Path.GetTempFileName()) { }
  public TempFile(string path) { this.path = path; }
  public string Path {
    get {
        if (path == null)
            throw new ObjectDisposedException(GetType().Name);
        return path;
     }
  }
  ~TempFile()
  {
      Dispose(false); //Dispose(bool) je vlastiti postupak
  }
```

Privremena datoteka (3)

- **Potrebno provjeriti zašto se poziva Dispose(bool)**
  - Dispose iz IDisposable ili finalizator zbog sakupljača smeća
    - GC.SuppressFinalize – uputa CLR-u da ne pozove finalizator objekta
  - Primjer: 🗁 Web \ Firma.Mvc \ Util \ TempFile.cs

```
class TempFile : IDisposable {
   public void Dispose() {
      Dispose(true);
    }
    private void Dispose(bool disposing) {
      if (disposing) {
        GC.SuppressFinalize(this);
      }
      if (path != null) {
        try {
          File.Delete(path);
        }
        catch { } // best effort
        path = null;
      }
    }
```

Ažuriranje artikla

- **Ažuriranje osim podataka o artiklu prima novu sliku (ako postoji) i informaciju treba li možda obrisati sliku**
  - Ne može se koristiti varijanta ctx.Update(artikl), jer slika nije prenošena u pogled → dohvatiti artikl iz b.p. i ažurirati samo ono što treba
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ArtiklController.cs

```
[HttpPost] [ValidateAntiForgeryToken]
public IActionResult Edit(Artikl artikl,
                          IFormFile slika, bool obrisiSliku) {
  Artikl dbArtikl = ctx.Artikl.Find(artikl.SifArtikla);
  ...
  dbArtikl.JedMjere = artikl.JedMjere;
  dbArtikl.NazArtikla = artikl.NazArtikla;
  dbArtikl.ZastUsluga = artikl.ZastUsluga;
  dbArtikl.TekstArtikla = artikl.TekstArtikla;
  dbArtikl.CijArtikla = artikl.CijArtikla;
  if (slika != null && slika.Length > 0) {
            ... izvuci byte[] image iz streama
              dbArtikl.SlikaArtikla = image;
  } else if (obrisiSliku) dbArtikl.SlikaArtikla = null;
  ctx.SaveChanges();
```

CC BY NC SA

Udaljena validacija na klijentskoj strani

- **Prije pokušaja unosa artikla provjerava se postoji li već artikl s navedenom šifrom**
  - sprječava se postavljanje upita za koji će sigurno biti neuspješan
- **Kako bi se korisniku ta informacija pružila i prije slanja forme koristi se tzv. udaljena validacija**
  - Automatski se generira se javascript kod koji poziva postupak na serveru koji vraća true/false
  - Primjer: 🗁 Web \ Firma.Mvc \ Models \ Artikl.cs

```
public partial class Artikl {
  [Required]
  [Remote(action:nameof(ArtiklController.ProvjeriSifruArtikla),
          controller: "Artikl",
      ErrorMessage = "Artikl s navedenom šifrom već postoji")]
   public int SifArtikla { get; set; }
```

  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ArtiklController.cs

```
public bool ProvjeriSifruArtikla(int SifArtikla) {
  bool postoji = ctx.Artikl.Any(a => a.SifArtikla==SifArtikla);
  return !postoji;
```

CC BY NC SA

Napomena uz udaljenu validaciju

- **Za razliku od ostalih atributa poput [Required], [Range] i slično, udaljena validacija se izvodi samo na klijentskoj strani**
  - U slučaju da javascript kod nije ispravno izveden, model će se na serveru smatrati valjanim (tj. postupci za udaljenu validaciju neće biti pozvani)
- **Moguće je napisati i vlastite validacijske atribute**
  - Moguće dodati i vlastiti javascript kod za validaciju na klijentu
  - Više na: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#custom-attributes

Web-aplikacije
ASP.NET Core MVC

2019/20.09-A

Primjer zaglavlje-stavke

CC BY NC SA

Napomena uz master-detail primjer

- **Primjer za master-detail predstavlja složene koncepte koji uključuju korištenje CSS stilova i *jQueryja***
- **Primjer je opsežan te pojedini dijelovi koda ne mogu u potpunosti stati na slajdove te se preporuča isprobati primjer i detaljno proučiti programski kôd pri čemu slajdovi mogu poslužiti kao nit vodilja kojim redom proučavati primjer**

- **Tablični prikaz dokumenata u pogledu s ikonama za ažuriranje i brisanje izveden kao i u ostalim primjerima (npr. prikaz država)**
  - Model sadrži podatke o dokumentima te informacije o straničenju i sortiranju te sadržajem po kojem treba filtrirati podatke
- **Ažuriranje dokumenta i stavki izvedeno dinamički slično kao kod mjesta i artikala**

BY NC SA

Pogled za prikaz svih dokumenata

- **Kao i u primjeru s partnerima za dohvat svih dokumenata koristi se pogled iz baze podataka**
  - Značajno pojednostavljuje kôd za dohvat podataka u upravljaču

```
CREATE VIEW [dbo].[vw_Dokumenti] AS
SELECT  dbo.Dokument.IdDokumenta, dbo.Dokument.VrDokumenta,
        dbo.Dokument.BrDokumenta, dbo.Dokument.DatDokumenta,
       dbo.Dokument.IdPartnera, dbo.Dokument.IdPrethDokumenta,
       dbo.Dokument.PostoPorez, dbo.Dokument.IznosDokumenta,
       CASE dbo.Partner.TipPartnera WHEN 'O' THEN
          dbo.Osoba.PrezimeOsobe + ', ' + dbo.Osoba.ImeOsobe
       ELSE dbo.Tvrtka.NazivTvrtke END
       + ' (' + dbo.Partner.OIB + ')' AS NazPartnera
FROM  dbo.Dokument INNER JOIN dbo.Partner ON
dbo.Dokument.IdPartnera = dbo.Partner.IdPartnera
LEFT OUTER JOIN dbo.Osoba
         ON dbo.Partner.IdPartnera = dbo.Osoba.IdOsobe
LEFT OUTER JOIN dbo.Tvrtka
         ON dbo.Partner.IdPartnera = dbo.Tvrtka.IdTvrtke
```

(Podsjetnik) Dodavanje pogleda u EF model (1)

- **Potrebno definirati razred koji svojoj strukturom odgovara rezultatu pogleda**
  - Svojstva koja imaju set dio, a ne odgovaraju nekom stupcu iz rezultata označavaju se atributom *NotMapped*
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ ViewDokumentInfo.cs

```
public class ViewDokumentInfo {
        public int IdDokumenta { get; set; }
        public decimal PostoPorez { get; set; }
        public int? IdPrethDokumenta { get; set; }
        public DateTime DatDokumenta { get; set; }
        public int IdPartnera { get; set; }
        public string NazPartnera { get; set; }
        public decimal IznosDokumenta { get; set; }
        public string VrDokumenta { get; set; }
        public int BrDokumenta { get; set; }

        [NotMapped]
        public int Position { get; set; } //Position in result
    }
```

CC BY NC SA

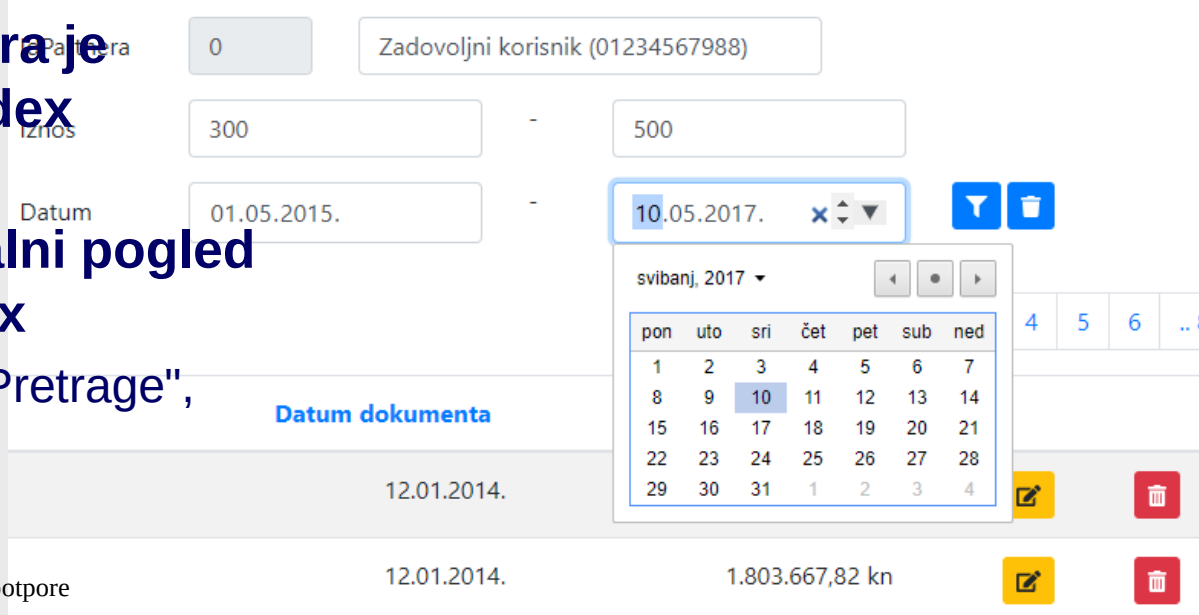
(Podsjetnik) Dodavanje pogleda u EF model (2)

- **U definiciju konteksta dodati novi *DbSet* za pogled**
  - Primijetiti da naziv svojstva u ovom primjeru ne odgovara nazivu pogleda
- **Primjer: 🗁 Web \ Firma.Mvc \ Models \ FirmaContext.cs**

```
public partial class FirmaContext : DbContext {
  ...
  public virtual DbSet<ViewDokumentInfo> ViewDokumentInfo {
       get; set;
  }
  ...
  protected override void OnModelCreating(
                                ModelBuilder modelBuilder) {
      modelBuilder.Entity<ViewDokumentInfo>(entity =>
      {
        entity.HasNoKey();
        entity.ToView("vw_Dokumenti");
      });
```

Filtriranje podataka

- **Popis dokumenata moguće filtrirati po partneru, iznosu ili datumu.**
- **Za odabir partnera koristi se nadopunjavanje, a za odabir datuma prikazuje se kalendar**
  - Posljedica atributa *[DataType(DataType.Date)]* iznad svojstva za datume
- **Gumb na formi za unos kriterija šalje popunjene podatke na akciju *Filter* koja spaja kriterije u jedan string koji se šalje kao parametar akciji Index**
  - Primjerice za odabir sa slike i aktiviranje filtera vrijednost parametra filter bit će `filter=0-01.01.2015-10.05.2017-300,00-500,00`
- **Gumb za uklanjanje filtra je poveznica na akciju Index bez parametara**
- **Kriterij pretrage parcijalni pogled uključen u pogled Index**
  - @Html.Partial("KriterijPretrage", Model.Filter)



CC BY NC SA

Parametar ili sjednica?

- **Prednosti korištenja paramet(a)ra za informaciju o filtriranju**
  - Moguće imati nekoliko aktivnih filtara unutar više kartica istog preglednika
  - Moguće pohraniti poveznicu za buduće posjete
    - Informacija se ne gubi istekom sjednice
  - Može se koristiti ako server koristi *load balancing*

- **Mane:**
  - Potreba za korištenjem više parametara, odnosno pronalaskom efikasnog načina za spremanje više kriterija unutar istog stringa
  - Potrebno prenositi parametre u različite akcije
    - vidi primjer s državama i parametrima page, sort ascending
  - Treba obratiti pažnju na znakove koji mogu utjecati na rekonstrukciju vrijednosti pojedinog filtra
  - Prevelik broj kriterija i dugi tekstovi kriterija mogli bi uzrokovati adresu izvan raspona dopuštene duljine
    - Alternativa: kriterij pohraniti negdje (sjednica, BP) i pridružiti mu neku vrijednost koja bi se koristila unutar adrese

Razred za informacije o filtru (1)

- **Za prihvat podataka o filteru koristi se razred *DokumentFilter***
  - Osim svojstava za prihvat unesenih vrijednosti sadrži i nekoliko pomoćnih metoda kojima se olakšava rad s filtriranjem
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ DokumentFilter.cs

```
public class DokumentFilter {
      public int? IdPartnera { get; set; }
      public string NazPartnera { get; set; }
      public DateTime? DatumOd { get; set; }
      public DateTime? DatumDo { get; set; }
      public decimal? IznosOd { get; set; }
      public decimal? IznosDo { get; set; }
      public bool IsEmpty() {
            bool active = IdPartnera.HasValue
                        || DatumOd.HasValue
                        || DatumDo.HasValue
                        || IznosOd.HasValue
                        || IznosDo.HasValue;
            return !active;
        }
        ...
```

CC BY-NC-SA

Razred za informacije o filtru (2)

- **Uneseni podaci mogu se spojiti u string te rekonstruirati iz stringa**
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ DokumentFilter.cs

```
public class DokumentFilter {
   ...
   public override string ToString() {
      return string.Format("{0}-{1}-{2}-{3}-{4}",
         IdPartnera, DatumOd?.ToString("dd.MM.yyyy"),
         DatumDo?.ToString("dd.MM.yyyy"), IznosOd, IznosDo);
   }
   public static DokumentFilter FromString(string s) {
      var filter = new DokumentFilter();
      var arr = s.Split(new char[] { '-' },
                        StringSplitOptions.None);
      filter.IdPartnera = string.IsNullOrWhiteSpace(arr[0]) ?
                        new int?() : int.Parse(arr[0]);
      filter.DatumOd = string.IsNullOrWhiteSpace(arr[1]) ?
                  new DateTime?() : DateTime.ParseExact(arr[1],
                 "dd.MM.yyyy", CultureInfo.InvariantCulture);
                 ...
            return filter;
```

BY NC SA

Razred za informacije o filtru (3)

- **Upit za dohvat svih dokumenata filtrira se po odabranim kriterijima**
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ DokumentFilter.cs

```
public class DokumentFilter {
  ...
  public IQueryable<ViewDokumentInfo> Apply(
                    IQueryable<ViewDokumentInfo> query) {

    if (IdPartnera.HasValue)
        query = query
                .Where(d => d.IdPartnera == IdPartnera.Value);

    if (DatumOd.HasValue)
        query = query
                .Where(d => d.DatDokumenta >= DatumOd.Value);
    ...
```

Dohvat svih partnera za dinamičke padajuće liste

- **Izvedeno korištenjem pogleda korištenog za pregled svih partnera**
  - Primjer: 🗁 Firma.Mvc \ Areas \ AutoComplete \ Controllers \ PartnerController.cs

```
[HttpGet]
 public IEnumerable<IdLabel> Get(string term) {
    var query = ctx.vw_Partner
                   .Select(p => new IdLabel
                    {
                        Id = p.IdPartnera,
                        Label = p.Naziv + " (" + p.OIB + ")"
                    })
                    .Where(l => l.Label.Contains(term));

    var list = query.OrderBy(l => l.Label)
                    .ThenBy(l => l.Id)
                    .ToList();
    return list;
}
```

Digresija: Dohvat svih partnera upitom bez pogleda

- **Bez pogleda, kôd za upit korištenjem EF-a bi bio puno složeniji**
  - Primjer: 🗁 Firma.Mvc \ Areas \ AutoComplete \ Controllers \ PartnerController.cs

```
var queryOsobe = ctx.Osoba.Select(o => new IdLabel {
                            Id = o.IdOsobe,
                            Label = o.PrezimeOsobe + ", " +
        o.ImeOsobe + " (" + o.IdOsobeNavigation.Oib + ")"
                          })
                          .Where(l => l.Label.Contains(term));

     var queryPartneri = ctx.Tvrtka.Select(t => new IdLabel {
                                 Id = t.IdTvrtke,
                                 Label = t.NazivTvrtke + ", " +
                   " (" + t.IdTvrtkeNavigation.Oib + ")"
                                 })
                          .Where(l => l.Label.Contains(term));
     var list = queryOsobe.Union(queryPartneri)
                           .OrderBy(l => l.Label)
                           .ThenBy(l => l.Id)
                           .ToList();
```

CC BY NC SA

Aktiviranje nadopunjavanja za partnere

- **Izvedeno slično kao kod padajuće liste za mjesta prilikom dodavanja novog partnera**
  - Vlastiti skripta aktivira nadopunjavanje za sve kontrole koje imaju definiran atribut data-autocomplete
  - Unutar vlastite *data* oznake informacija o relativnoj adresi izvora podataka
  - Razlika je u tome što se id odabranog partnera vidi na formi
    - Nije skriveno polje, ali se ne može mijenjati (atribut *readonly*)
    - Primijetiti da se veže za svojstvo NazPartnera iz razreda DokumentFilter
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Dokument \ KriterijPretrage.cshtml

```
<input asp-for="IdPartnera"
       readonly="readonly"
       class="form-control"
       data-autocomplete-placeholder="partner" />

<input data-autocomplete="partner"
       class="form-control"
       asp-for="NazPartnera" />
```

BY NC SA

Identifikator i naziv partnera u filtru

- **Naziv odabranog partnera dio razreda DokumentFilter, ali nije dio teksta koji se prenosi u adresi unutar parametra *filter***
  - Potrebno ga je obnoviti upitom temeljem *IdPartnera*
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs**

```
public IActionResult Index(string filter, ...
  ...
  DokumentFilter df = new DokumentFilter();
  if (!string.IsNullOrWhiteSpace(filter)) {
      df = DokumentFilter.FromString(filter);
      if (!df.IsEmpty()) {
           if (df.IdPartnera.HasValue) {
                 df.NazPartnera = ctx.vw_Partner
                    .Where(p => p.IdPartnera == df.IdPartnera)
                    .Select(vp => vp.Naziv)
                    .FirstOrDefault();
            }
            query = df.Apply(query);
       }
  }
```

CC BY NC SA

Forma za ažuriranje dokumenta

- **Forma za ažuriranje dokumenta sastoji se od**
  - zaglavlja za ažuriranje podataka koji pripadaju tablici Dokument (*master*)
    - Odabir partnera i prethodnog dokumenta izveden nadopunjavanjem
  - detalja sa stavkama dokumenta (*detail*)
    - Moguće promijeniti vrijednost stavke (količina i rabat), obrisati je ili dodati novu (poseban redak nakon svih stavki)
- **Zajednička akcija za spremanje podataka (zaglavlje + stavke)**

Dokument br: 5555

Vrsta dokumenta: R-1 | Porez (u %): 22 | Datum: 05.11.2016.

Broj: 11250 | Partner: 509 | CETINA (5914624)

Prethodni dokument: | Iznos: 2.414,38 kn

Artikl	Količina	Rabat [0-1]	Jedinična cijena	Iznos
AC Adapter ASIIN Mitac - 442687900004	3,00000	0,00	106,00 kn	318,00 kn
ArtiklProba F	3,00000	0,00	35,00 kn	105,00 kn
AC/DC adapter za 3Pod KINGSINGTON, 70W (33335EU)	2,00000	0,00	778,00 kn	1.556,00 kn

CC BY NC SA

Model za rad s dokumentima (1)

- **Prezentacijski model s validacijskim atributima**
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ DokumentViewModel.cs

```
public class DokumentViewModel {
  public int IdDokumenta { get; set; }
  ...
  [DisplayFormat(DataFormatString = "{0:dd.MM.yyyy.}",
                                 ApplyFormatInEditMode = false)]
  [DataType(DataType.Date)]
  [Display(Name = "Datum")]
  [Required(ErrorMessage = "Potrebno je ... dokumenta")]
  public DateTime DatDokumenta { get; set; }

  [Display(Name = "Porez (u %)")]
  [Required(ErrorMessage = "Potrebno je postotak poreza")]
  [Range(0, 100, ErrorMessage = "Porez mora biti 0-100")]
   public int StopaPoreza { get; set; }
   public decimal PostoPorez {
     get { return StopaPoreza / 100m; }
     set { StopaPoreza = (int) (100m * value); }
  ...
```

BY NC SA

Model za rad s dokumentima (2)

- **Osim atributa koji će u konačnici završiti u tablici Dokument, model sadrži i kolekciju stavki**
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ DokumentViewModel.cs

```
public class DokumentViewModel {
  ...
  public IEnumerable<StavkaViewModel> Stavke { get; set; }
  public DokumentViewModel() {
            this.Stavke = new List<StavkaViewModel>();
  }
}
```

  - Stavke prikazane vlastitim prezentacijskim modelom

cc BY NC SA

Model za rad sa stavkama

- **Novi razred kao model za rad sa stavkama da se izbjegnu problemi s vezom prema tablici Dokument**
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ StavkaViewModel.cs

```
public class StavkaViewModel
    {
        public int IdStavke { get; set; }
        public int SifArtikla { get; set; }
        public string NazArtikla { get; set; }
        public decimal KolArtikla { get; set; }
        public decimal JedCijArtikla { get; set; }
        public decimal PostoRabat { get; set; }

        public decimal IznosArtikla {
            get
            {
                return KolArtikla * JedCijArtikla
                            * (1 - PostoRabat);
            }
        }
    }
```

Priprema dokumenta za ažuriranje (1)

- **Za početak ažuriranja dokumenta i njegovih stavki potrebno iz BP dohvatiti osnovne podatke o dokumentu i popuniti model iz prethodnih slajdova**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
[HttpGet]
public IActionResult Edit(int id ...) {
   var dokument = ctx.Dokument.AsNoTracking()
                     .Where(d => d.IdDokumenta == id)
                     .Select(d => new DokumentViewModel {
                        BrDokumenta = d.BrDokumenta,
                        DatDokumenta = d.DatDokumenta,
                        IdDokumenta = d.IdDokumenta,
                        IdPartnera = d.IdPartnera,
                        IdPrethDokumenta = d.IdPrethDokumenta,
                        IznosDokumenta = d.IznosDokumenta,
                        PostoPorez = d.PostoPorez,
                        VrDokumenta = d.VrDokumenta
                      })
                      .FirstOrDefault();}
```

CC BY NC SA

Priprema dokumenta za ažuriranje (2)

- **Odabir partnera i prethodnog dokumenta vrši se nadopunjavanjem, pa ne treba pripremati podatke za padajuće liste**
- **Ipak, … potrebno dohvatiti nazive partnera i dokumenta kako bi se prikazali korisniku**
  - Za partnera provjeriti tip i dohvatiti ime i prezime, odnosno naziv tvrtke
  - Slično za dokument
  - Potrebno koristiti isti format koji se koristi u upravljaču koji služi kao izvor za nadopune
- **Primjer: 🗁 Web \ Firma.Mvc \ Areas \ AutoComplete \ Controllers \ DokumentController.cs**
  - Programski kod prevelik za slajd, ali relativno jednostavan…

CC BY NC SA

Priprema dokumenta za ažuriranje (3)

- **Nakon osnovnih podataka vrši se dohvat stavki dokumenta**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
[HttpGet]
public IActionResult Edit(int id ...) {
   ...
   var stavke = ctx.Stavka.Where(s => s.IdDokumenta ==
                                      dokument.IdDokumenta)
                  .OrderBy(s => s.IdStavke)
                  .Select(s => new StavkaViewModel {
                            IdStavke = s.IdStavke,
                            JedCijArtikla = s.JedCijArtikla,
                            KolArtikla = s.KolArtikla,
         NazArtikla = s.SifArtiklaNavigation.NazArtikla,
                            PostoRabat = s.PostoRabat,
                            SifArtikla = s.SifArtikla
                   })
                   .ToList();
   dokument.Stavke = stavke;
   return View(dokument);
```

Pogled za ažuriranje dokumenta

- **Dio sa zaglavljem nalik ostalim formama za ažuriranje pojedinačnog podatka**
  - Input kontrole + kontrole za nadopunjavanje
  - Gumb za povratak na popis dokumenata, osvježavanje trenutnog dokumenta i spremanje promjena
- **Na dnu forme se nalazi poziv parcijalnog pogleda koji će prikazati sve stavke dokumenta**
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Dokument \ Edit.cshtml

```
@model DokumentViewModel
...
<form id="form" method="post" asp-action="Edit" ... >
    <input type="hidden" asp-for="@Model.IdDokumenta"/>
    ...
    <input asp-for="StopaPoreza" class="form-control" />
    ...
   <button id="save" type="submit" title="Spremi">...</button>
    ...
   <partial name="Stavke" model="Model.Stavke" />
```

Povezivanje kolekcije podataka

- **U slučaju da postoji više kontrola s istim atributom *name*, unesene vrijednosti se na upravljaču prihvaćaju u istoimenom argumentu koji je polje nekog tipa vrijednosti.**
- **Što ako treba povezati više složenih podataka (npr. više stavki)?**
  - Za atribut *name* se koristi oblik NazivKolekcije[indeks].NazivSvojstva
    - npr. name="Stavke[0].IdStavke", name="Stavke[1].IdStavke", name="Stavke[2].IdStavke" …
  - Indeksi moraju biti bez prekida počevši od 0
- **Što ako nije moguće osigurati neprekinuti niz indeksa?**
  - Mogu se koristiti bilo koje oznake (čak ni ne moraju biti brojevi), ali dodatno treba stvoriti (skrivene) kontrole oblika Stavka.Index čija je vrijednost jednaka korištenoj oznaci

```
<input type="hidden" name="Stavke.Index" value="knjiga" />
<input type="text" name="Stavke[knjiga].IdStavke" ... />

<input type="hidden" name="Stavke.Index" value="5" />
<input type="text" name="Stavke[5].IdStavke" ... />
```

Parcijalni pogled za ažuriranje stavki

- ***DokumentViewModel* sadrži svojstvo *Stavke***
  - Povezivanje kolekcije stavki i njenih svojstva ostvareno varijantom sa Stavke.Index
    - Kao indeks koristi se šifra artikla
  - Primjer: 🗁 Web \ Firma.Mvc \ Views \ Dokument \ Stavke.cshtml

```
@model IEnumerable<StavkaViewModel>
...
@foreach(var stavka in Model) {
       <input type="hidden" name="Stavke.Index"
                                value="@stavka.SifArtikla"/>

        <input type="hidden"
              name="Stavke[@stavka.SifArtikla].IdStavke"
              value="@stavka.IdStavke" />
...
       <input name="Stavke[@stavka.SifArtikla].KolArtikla"
              value="@stavka.KolArtikla"/>
        ...
```

Brisanje stavke

- **Gumb za brisanje stavke (prepoznaje se po stilu *deleterow*) uklanja stavku iz strukture HTML dokumenta.**
  - Primjer: 🗁 Web \ Firma.Mvc \ wwwroot \ js \ dokument.js

```
$(document).on('click', '.deleterow', function () {
    event.preventDefault();
    var tr = $(this).parents("tr");
    tr.remove();

    //očisti staru poruku da je dokument uspješno snimljen
    $("#tempmessage").siblings().remove();
    $("#tempmessage").removeClass("alert-success");
    $("#tempmessage").removeClass("alert-danger");
    $("#tempmessage").html('');
});
```

- **Posljedično bit će izbrisana iz dokumenta nakon klika na gumb za snimanje**

CC BY NC SA

Izmjena dokumenta i njegovih stavki (1)

- **Prilikom dohvata dokumenta potrebno je dohvatiti i sve njegove stavke**
- **Kao prvi korak vrši se kopiranje podataka dokumenta iz povezanog modela u objekt koji je dohvaćen iz BP**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(DokumentViewModel model, ...
  var dokument = ctx.Dokument
                  .Include(d => d.Stavka)
                  .Where(d =>
                         d.IdDokumenta == model.IdDokumenta)
                  .FirstOrDefault();

... Kopiraj podatke o dokumentu iz modela u objekt dokument
... dokument.PostoPorez = model.StopaPoreza / 100m;
    dokument.VrDokumenta = model.VrDokumenta;
```

CC BY NC SA

Izmjena dokumenta i njegovih stavki (2)

- **U listu cijelih brojeva spremaju se identifikatori povezanih stavki**
  - Predstavlja stavke koje nisu uklonjene iz HTML-a prije snimanja
  - Iz konteksta se izbacuju sve stavke dokumenta koje se ne nalaze u toj listi
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(DokumentViewModel model, ...
  ...
   List<int> idStavki = model.Stavke
                             .Where(s => s.IdStavke > 0)
                             .Select(s => s.IdStavke)
                             .ToList();
   //izbaci sve koje su nisu više u modelu
   ctx.RemoveRange(dokument.Stavka
                 .Where(s => !idStavki.Contains(s.IdStavke)));
```

Izmjena dokumenta i njegovih stavki (3)

- **Stavke koje se nalaze u povezanom modelu su ili nove stavke ili neke od postojećih**
  - Postojeće treba dohvatiti iz objekta dohvaćenog iz baze podataka
  - Nove treba stvoriti i dodati u dokument
  - U oba slučaja u novi objekt kopirati svojstva iz povezanog modela
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs**

```
foreach (var stavka in model.Stavke) {
  Stavka novaStavka;
  if (stavka.IdStavke > 0) {
       novaStavka = dokument.Stavka
              .First(s => s.IdStavke == stavka.IdStavke);
  }
  else {
       novaStavka = new Stavka();
       dokument.Stavka.Add(novaStavka);
  }
  novaStavka.SifArtikla = stavka.SifArtikla;
  novaStavka.KolArtikla = stavka.KolArtikla;
  …kopiranje ostalih vrijednosti
```

BY NC SA

Izmjena dokumenta i njegovih stavki (4)

- **Potrebno izračunati iznos dokumenta i spremiti promjene**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(DokumentViewModel model, ...
   ...
   dokument.IznosDokumenta = (1 + dokument.PostoPorez)
                             * dokument.Stavka
                                        .Sum(s =>
    s.KolArtikla * (1 - s.PostoRabat) * s.JedCijArtikla);

  ctx.SaveChanges();
```

Odabir artikla za novu stavku

- **Poseban redak na dnu postojećih stavki u kojem se nadopunjavanjem bira artikl**
  - Povratni podaci sa servera sadrže šifru artikla, naziv artikla i cijenu artikla
  - Cijena artikla se kopira u odgovarajuće polje

- **Pogledati programski kôd u sljedećim datotekama:**
  - 🗁 Web \ Firma.Mvc \ Areas \ AutoComplete \ Models \ Artikl.cs
  - 🗁 Web \ Firma.Mvc \ Areas \ AutoComplete \ Controllers \ ArtiklController.cs
  - 🗁 Web \ Firma.Mvc \ wwwroot \ js \ autocomplete.js

CC BY NC SA

Dodavanje nove stavke

- **Skrivena tablica s identifikatorom *template* koja služi za dodavanje novog retka na osnovu vrijednosti iz posebnog retka na dnu stavki**
  - Provjera ispravnosti vrijednosti, provjera postoji li duplikat artikla i slično
  - Dodavanje novog retka
  - Ponašanje tipke ENTER

- **Redak s predloškom se uklanja neposredno prije snimanja forme kako ne bi sudjelovao u povezivanju**

- **Pogledati programski kôd u sljedećim datotekama:**
  - 🗁 Web \ Firma.Mvc \ Views \ Dokument \ Stavke.cshtml
  - 🗁 Web \ Firma.Mvc \ wwwroot \ js \ dokumenti.js

CC BY NC SA

Dodatna funkcionalnost (1)

- **Na stranici za ažuriranje dokumenta poveznica za povratak na listu svih dokumenata**
  - Pamti se prethodna stranica i način sorta
- **Dodatno, poveznica na prethodni i sljedeći dokument**
  - Svaki dokument ima svoju poziciju unutar baze podataka u ovisnosti o trenutnom sortu i filtru
  - Inicijalno pridijeljeno prilikom dohvata dokumenata
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
public IActionResult Index(string filter, int page = 1...

  ... Dohvati dokumente u ovisnosti o filtru i sortu ...

  for(int i=0; i<dokumenti.Count; i++)
        dokumenti[i].Position = (page - 1) * pagesize + i;
```

BY NC SA

Traženje sljedbenika i prethodnika

- **Korištenjem proširenja Skip**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs

```
private void SetPreviousAndNext(int position, string filter,
                                int sort, bool ascending) {

var query = ctx.ViewDokumentInfo.AsQueryable();
  ... proširi upit sa filtrom i sortiranjem ...
  if (position > 0)
     ViewBag.Previous = query.Skip(position - 1)
                             .Select(d => d.IdDokumenta)
                             .First();

  if (position < query.Count() - 1) {
      ViewBag.Next = query.Skip(position + 1)
                          .Select(d => d.IdDokumenta)
                          .First();
```

CC BY NC SA

Kreiranje novog dokumenta

- **Kao početni model za unos novog dokumenta stvara se novi objekt kojem se postavlja trenutni datum i sljedeći broj dokumenta**
  - Samo kao primjer početne inicijalizacije, jer u slučaju istovremenog dodavanja novih dokumenata može doći do ponavljanja istog broja
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ DokumentController.cs**

```
[HttpGet]
public IActionResult Create() {
    int maxbr = ctx.Dokument.Max(d => d.BrDokumenta) + 1;
    var dokument = new DokumentViewModel {
                DatDokumenta = DateTime.Now,
                BrDokumenta = maxbr
    };
    return View(dokument);
}
```

- **Ostatak izveden slično kao kod ažuriranja (razlika u tome što su sve stavke nove).**

CC BY NC SA

Web-aplikacije
ASP.NET Core MVC

2019/20.09-B
Izrada izvješća

CC BY NC SA

Alat za izradu izvješća

- **Za izradu izvješća u PDF formatu koristit će se alat PdfReport.Core**
  - https://github.com/VahidN/PdfReport.Core

PdfRpt.Core by Vahid Nasiri v1.2.7

PdfReport.Core is a code first reporting engine, which is built on top of the iTextSharp.LGPLv2.Core and EPPlus.Core libraries.

- **Primjeri sadrže 3 vrsta izvješća:**
  - Jednostavni tablični izvještaj
    - popis svih država
  - Tablični izvještaj sa slikama i sumom vrijednosti nekog stupca
    - prikaz 10 najskupljih artikala sa slikom
  - Master-detail primjer koji demonstrira grupiranje elemenata
    - n „najboljih” kupnji (tj. dokumenata s najvećom vrijednosti zajedno s popisom stavki)

Inicijalne postavke za izvješća (1)

- **Razred za rad s izvješćima je *PdfReport* koji sadrži postupke koji vraćaju referencu na vlastiti objekt, pa se postupci mogu vezati**
  - Inicijalno se postavljaju metapodaci (autori, naslov, …) i orijentacija
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
private PdfReport CreateReport(string naslov) {
     var pdf = new PdfReport();
     pdf.DocumentPreferences(doc =>
     {
       doc.Orientation(PageOrientation.Portrait);
       doc.PageSize(PdfPageSize.A4);
       doc.DocumentMetadata(new DocumentMetadata {
         Author = "FER-ZPR", Application = "Firma.MVC Core",
         Title = naslov
       });
       doc.Compression(new CompressionSettings {
         EnableCompression = true,
         EnableFullCompression = true
       });
     })
```

BY NC SA

Inicijalne postavke za izvješća (2)

- **Postavlja se vrsta predloška, automatska širina stupaca, može se ograničiti broj redaka po stranici, broj grupa po stranici i slično.**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
private PdfReport CreateReport(string naslov) {
     var pdf = new PdfReport();
     ...
     .MainTableTemplate(template => {
            template.BasicTemplate(
                   BasicTemplate.ProfessionalTemplate);
     })
     .MainTablePreferences(table => {
            table.ColumnsWidthsType(
                   TableColumnWidthType.Relative);
            //table.NumberOfDataRowsPerPage(20);
            table.GroupsPreferences(new GroupsPreferences {
                   GroupType = GroupType.HideGroupingColumns,
                   RepeatHeaderRowPerGroup = true,
                   ShowOneGroupPerPage = true
       });
       table.SpacingAfter(4f);
```

BY NC SA

Zaglavlje i podnožje jednostavnih izvješća

- **Jednostavna izvješća će u zaglavlju imati naslov izvješća, a u podnožju trenutni datum**
  - Potrebno postaviti smjer teksta na LeftToRight
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public async Task<IActionResult> Drzave() {
  ...
  PdfReport report = CreateReport(naslov);
  report.PagesFooter(footer => {
                      footer.DefaultFooter(DateTime.Now
                           .ToString("dd.MM.yyyy."));
                     })
      .PagesHeader(header => {
                    header.CacheHeader(cache: true);
                    header.DefaultHeader(defaultHeader => {
                          defaultHeader.RunDirection(
                               PdfRunDirection.LeftToRight);
                          defaultHeader.Message(naslov);
                    });
      });
```

Izvor podataka za izvješće

- **U svim primjerima kao izvor podataka služit će lista čiji su elementi objekti nekog prezentacijskog modela ili anonimnog razreda**
  - Podaci se u izvještaj dodaju redom kojim se nalaze u listi
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public async Task<IActionResult> Drzave() {
  var drzave = await ctx.Drzava
                        .AsNoTracking()
                        .OrderBy(d => d.NazDrzave)
                        .ToListAsync();
```

```
public async Task<IActionResult> Artikli() {
  var artikli = await ctx.Artikl
                       .AsNoTracking()
                       .Where(a => a.SlikaArtikla != null)
                       .OrderByDescending(a => a.CijArtikla)
                       .Select(a => new {
                              a.SifArtikla, a.NazArtikla,
                              a.CijArtikla, a.SlikaArtikla
                        })
                       .Take(10).ToListAsync();
```

BY NC SA

Postavljanje izvora podataka

- **Izvor se postavlja pozivom postupka *MainTableDataSource* i pisanjem odgovarajućeg delegata tipa *Action<MainTableDataSourceBuilder>***
  - Nad podatkom tipa *MainTableDataSourceBuilder* poziva se postupak *StronglyTypedList* te se kao argument predaje pripremljena lista
    - Alternative: postupci SqlDataReader, CustomDataSource, …
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs**

```
public async Task<IActionResult> Drzave() {
  var drzave = await ctx.Drzava
                        .AsNoTracking()
                        .OrderBy(d => d.NazDrzave)
                        .ToListAsync();

  report.MainTableDataSource(dataSource =>
      dataSource.StronglyTypedList(drzave));
```

Definiranje stupaca

- **Željeni stupci se navode u postupku *MainTableColumns***
  - Stupcima se postavlja poredak (ako se izostavi koristi se redoslijed navođenja) relativna širina, vidljivost, zaglavlje te naziv svojstva
    - Moguće imati redak s rednim brojem
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs (Drzave)

```
report.MainTableColumns(columns => {
    columns.AddColumn(column => {
      column.IsRowNumber(true);
      column.CellsHorizontalAlignment(
             HorizontalAlignment.Right);
      column.IsVisible(true); column.Order(0);column.Width(1);
      column.HeaderCell("#",
             horizontalAlignment: HorizontalAlignment.Right);
    });
    columns.AddColumn(column => {
          column.PropertyName(nameof(Drzava.OznDrzave));
         column.Order(1); column.Width(2);
          column.HeaderCell("Oznaka države");
        });
```

BY NC SA

Preuzimanje izvješća

- **Binarni sadržaj izrađenog izvješća vraća se korisniku korištenjem postupka File (naslijeđen iz razreda *Controller*)**
- **Dva načina:**
  - Sugerirati pregledniku da isporučeni sadržaj prikaže unutar preglednika
    - content-disposition:inline
  - Preglednik nudi odabir mape za snimanje datoteke
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs**

```
public async Task<IActionResult> Drzave() {
 ...
 byte[] pdf = report.GenerateAsByteArray();
 if (pdf != null) {
        Response.Headers.Add("content-disposition",
                    "inline; filename=drzave.pdf");
        return File(pdf, "application/pdf");
        //return File(pdf, "application/pdf", "drzave.pdf");
        // samo gornji redak otvara Save as dialog
 }
 else return NotFound();
```

Stupci s posebnim predloškom

- **Za svaki artikl prikazana slika u izvješću**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public async Task<IActionResult> Artikli()
  ...
  report.MainTableColumns(columns => {
      columns.AddColumn(column => {
          column.PropertyName(nameof(Artikl.SlikaArtikla));
          ...
          column.ColumnItemsTemplate(t =>
                    t.ByteArrayImage(string.Empty,
                                     fitImages: true));
      });
  ...
```

Stupci sa sumom vrijednosti u podnožju

- **Postavlja se konačna suma svih artikala i tekst ispred sume**
  - moguće postaviti i međusume za svaku stranicu *PageSummarySettings*
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public async Task<IActionResult> Artikli()
  ...
  report.MainTableSummarySettings(summarySettings => {
        summarySettings.OverallSummarySettings("Ukupno");
  });
  report.MainTableColumns(columns => {
      columns.AddColumn(column => {
          column.PropertyName<Artikl>(x => x.CijArtikla);
          column.ColumnItemsTemplate(template => {
            template.TextBlock();
            template.DisplayFormatFormula(obj =>
                    string.Format("{0:C2}", obj));
          });
          column.AggregateFunction(aggregateFunction => {
            aggregateFunction.NumericAggregateFunction(
                            AggregateFunction.Sum);
            aggregateFunction.DisplayFormatFormula(...
```

Primjer izvještaja oblika zaglavlje stavke

- Za svaki dokument ispisani osnovni podaci nakon čega slijede sve stavke
  - Nakon ispisa stavki ispisana suma bez PDV-a
- Stupac *Ukupno* je izračunati stupac

localhost:50051/Report/Dokumenti

10 najvećih kupnji

Id dokumenta:	2237	**Datum dokumenta:**	11.02.2014
Partner:	KBC	**Iznos dokumenta:**	2.178.811,73 kn

#	Naziv artikla	Količina	Jedinična cijena	Rabat	Ukupno
1	CD/MP3 radio kazetofon, PHILIP MORRIS PH-AZ1038, 1 x kazeta	1,00	289,00 kn	5,00%	274,55 kn
2	DVD/CD/MP3/DivX player, linijski, BIONEER DV-300-K + 7 crtića, crni	10,00	549,00 kn	9,00%	4.995,90 kn
3	GPS uređaj GURMAN GPS 72	1,00	1.291,00 kn	7,00%	1.200,63 kn
4	Knjiga "Access 2003 za Windows"	2,00	157,00 kn	9,00%	285,74 kn
5	Knjiga "Skok u Linux"	2,00	199,00 kn	2,00%	390,04 kn
6	MBO EVGA, s. 775, 650i Ultra, NVIDIA nForce 650i Ultra, BUS 1333 MHz, serial ATA II, RAID, 7.1 zvuk, 1Gbps, SSSR 2, ATX 2	1,00	684,00 kn	1,00%	677,16 kn
7	Mikrovalna pećnica LGG MS-1924V, 19 litara	1,00	419,00 kn	1,00%	414,81 kn
8	Mobilni uređaj MOTOCOLA V3, GPRS, EMS, Bluetooth, ekran u boji	1,00	999,00 kn	2,00%	979,02 kn
9	Mobilni uređaj NUKIA N73, GPRS, Bluetooth, MMS, FM Radio, kamera, ekran u boji, crveni	2,00	3.499,00 kn	1,00%	6.928,02 kn
10	MP3/WMA player, DESTRUCTIVE Zen Vision M 60 GB, crni	3,00	2.299,00 kn	6,00%	6.483,18 kn
11	MP3/WMA/FM player, CS, MC309F, 1 GB	5,00	403,00 kn	1,00%	1.994,85 kn
12	Notebook ACER Aspire 5610 NWLMi, Kintel Solo Core T1350(1.83ghz), TFT 15.4" WXGA, 1GB (512MB + 512MB POKLON), DVDRW, HDD 60 GB, bežična mreža, Linux (LX.AU60C.004)	4,00	4.799,00 kn	7,00%	17.852,28 kn
13	Notebook PH Compaq 6715b (GB833EA), KPD Sempron 3800+ (2.2GHz), 15.4" WXGA (1280x800), RAM 1GB SSSR2, HDD 120GB SATA, DVD+-RW DL, ATi Radeon X1250, modem, LAN, WLAN, BT, Windows Vista Basic	4,00	5.192,00 kn	8,00%	19.106,56 kn
14	PC računalo HGspotVR PREMIUM II + Windows VISTA Home Premium, KPD Athlon 64 X2 4400+, 1 GB SSSR2, HDD 250 GB SATA-II, GeForce 8500 GT 256 MB, DVD±RW, TV Tuner+zvučnici 2.1 + 19" LCD BENQ monitor + poklon Chat Pack i BITDEFENDER Internet Security 2008	2,00	5.199,00 kn	10,00%	9.358,20 kn
15	RAM, 1 GB, SSSR 2, PC-5300, 667 MHz,	10,00	178,00 kn	2,00%	1.744,40 kn
16	Torba za notebook Marasst MFL30	4,00	141,00 kn	8,00%	518,88 kn
17	Torbica za Strukkle triPOD trio nano 3-1, kožna, siva	3,00	114,00 kn	4,00%	328,32 kn
18	TV LCD 82 cm, SHARP LC32WD1E, 16:9, stereo, HD Ready, 2 x HDMI, DVB -T	1,00	7.399,00 kn	9,00%	6.733,09 kn
19	TV LCD 94 cm, PHILIP MORRIS 37PFL9732, FULL HD, DVB-T	100,00	16.999,00 kn	1,00%	1.682.901,00 kn
				Ukupno:	763.166,63 kn

CC BY NC SA

Oblik podataka za izvješća s grupiranjem

- **Potrebno koristiti denormalizirane podatke**
  - Primjer: 🗁 Web \ Firma.Mvc \ ViewModels \ StavkaDenorm.cs

```
public class StavkaDenorm {
    public string OIB { get; set; }
    public string NazPartnera { get; set; }
    public int IdDokumenta { get; set; }
    public DateTime DatDokumenta { get; set; }
    public decimal IznosDokumenta { get; set; }
    public int IdStavke { get; set; }
    public int SifArtikla { get; set; }
    public decimal KolArtikla { get; set; }
    public decimal JedCijArtikla { get; set; }
    public decimal PostoRabat { get; set; }
    public string NazArtikla { get; set; }
    [NotMapped] public string UrlDokumenta { get; set; }
  }
```

  - Svojstvo *UrlDokumenta* predstavljat će adresu dokumenta
    - Formirat će se nakon dohvata na osnovu usmjeravanja i *IdDokumenta*
    - Ne postoji u bazi podataka pa se mora označiti s *NotMapped*

CC BY NC SA

Funkcija za dohvat n najboljih kupnji

- **Ispis n dokumenata s najvećim iznosom**
  - Složen upit te je izveden korištenjem funkcije na BP
  - Kao i u primjeru pogleda u kontekst potrebno dodati DbSet<StavkaDenorm> i navesti podatak koji jednoznačno određuje podatak ili označiti da nema ključ

```
CREATE FUNCTION [dbo].[fn_NajveceKupnje](@N int)
RETURNS TABLE AS
RETURN(
  SELECT P.OIB, P.Naziv AS NazPartnera,
       D.IdDokumenta, D.DatDokumenta, D.IznosDokumenta,
       S.IdStavke, S.SifArtikla, S.KolArtikla,
       S.JedCijArtikla, S.PostoRabat, A.NazArtikla
   FROM
       (SELECT TOP (@N) * FROM Dokument
              ORDER BY IznosDokumenta DESC) D
       INNER JOIN vw_Partner P ON P.IdPartnera = D.IdPartnera
       LEFT OUTER JOIN Stavka S
              ON S.IdDokumenta = D.IdDokumenta
       LEFT OUTER JOIN Artikl A ON A.SifArtikla = S.SifArtikla
)
```

Poziv funkcije s parametrima

- **Za dohvat iz DbSeta koji se odnosi na funkciju, potrebno napisati SQL upit (u ovom slučaju parametrizirani)**
- **<u>Podaci moraju biti poredani po stupcima po kojima se želi izvršiti grupiranje</u>**
  - Poredak nakon toga po želji
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs**

```
public async Task<IActionResult> Dokumenti() {
  int n = 10;
  var param = new SqlParameter("@N", n);
  string naslov = $"{n} najvećih kupnji";
  var stavke = await ctx.StavkaDenorm
                          .FromSqlRaw(
              "SELECT * FROM fn_NajveceKupnje(@N)", param)
                          .OrderBy(s => s.IdDokumenta)
                          .ThenBy(s => s.NazArtikla)
                          .ToListAsync();
```

BY NC SA

Postavljanje adrese za ažuriranje dokumenta

- **Unutar PDF-a bit će dodana poveznica za ažuriranje dokumenta**
- **Potrebno pripremiti svojstvo s adresom**
  - Akcija Edit na upravljaču Dokument s parametrom id
  - Sprema se u svojstvo UrlDokumenta za svaki element dohvaćene liste
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs**

```
public async Task<IActionResult> Dokumenti() {
  ...
  var stavke = await ctx.StavkaDenorm
                        ...
                        .ToListAsync();

   stavke.ForEach(s =>
       s.UrlDokumenta = Url.Action("Edit",
                                   "Dokument",
                                 new { id = s.IdDokumenta }
                        ));
```

Definiranje stupaca po kojima se vrši grupiranje

- **Prilikom definiranja stupaca u prikazu, prvo se navode stupci po kojima se vrši grupiranje**
  - Neće biti prikazani u tablici, već iznad nje
- **Potrebno napisati kriterij pripadaju li dvije vrijednosti tog „stupca" istoj grupi**
- **Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs**

```
public async Task<IActionResult> Artikli()
  ...
  report.MainTableColumns(columns => {
      columns.AddColumn(column => {
            column.PropertyName("IdDokumenta");
            column.Group(
                  (val1, val2) =>
                  {
                    return (int)val1 == (int)val2;
                  });
      });
```

CC BY NC SA

Stupac s automatskim izračunom vrijednosti

- **Vrijednost stupca s ukupnom cijenom neke stavke računa se na osnovu ostalih vrijednosti iz trenutnog retka**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public async Task<IActionResult> Artikli()
  ...
  report.MainTableColumns(columns => {
      columns.AddColumn(column => {
            column.CalculatedField(
               list => {
                  if (list == null) return string.Empty;
                  decimal kolArtikla =
                     (decimal)list.GetValueOf("KolArtikla");
                  decimal postoRabat =
                     (decimal)list.GetValueOf("PostoRabat");
                  decimal jedCijArtikla =
                   (decimal)list.GetValueOf("JedCijArtikla");
                  var iznos = jedCijArtikla * kolArtikla
                                 * (1 - postoRabat);
                   return iznos;
               });
```

Zaglavlje grupe

- **Kao predložak za zaglavlje pojedine grupe potrebno napisati vlastiti predložak kojim se definira kako izgleda zaglavlje izvještaja i zaglavlje pojedine grupe**
- **Predložak se postavlja postupkom *CustomHeader* u *PagesHeader***
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public async Task<IActionResult> Artikli()
  ...
  report.PagesHeader(header => {
        header.CacheHeader(cache: true);
        header.CustomHeader(new MasterDetailsHeaders(naslov)
        {
          PdfRptFont = header.PdfFont
        });
      });
```

Predložak za zaglavlje izvještaja

- **Zaglavlje izvještaja sastoji se od jednog retka s naslovom**
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public class MasterDetailsHeaders : IPageHeader {
      private string naslov;
      public MasterDetailsHeaders(string naslov) {
        this.naslov = naslov;
      }
      public IPdfFont PdfRptFont { set; get; }

      public PdfGrid RenderingReportHeader(...
      {
        var table = new PdfGrid(numColumns: 1)
                                { WidthPercentage = 100 };
        table.AddSimpleRow(
           (cellData, cellProperties) => {
             cellData.Value = naslov;
             ...
        });
        return table.AddBorderToTable();
      }
```

Predložak za zaglavlje grupe

- **Vrijednosti grupe mogu se dobiti iz ulaznog argumenta tipa *IList<CellData>***
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public class MasterDetailsHeaders : IPageHeader {
  public PdfGrid RenderingGroupHeader(...
                    IList<CellData> newGroupInfo ...){
      var idDokumenta = newGroupInfo
                        .GetSafeStringValueOf("IdDokumenta");
        ...
       var iznosDokumenta = (decimal) newGroupInfo
                          .GetValueOf("IznosDokumenta");
```

BY NC SA

Predložak za zaglavlje grupe

- **Zaglavlje grupe definirano kao tablica s 4 stupca s podacima grupe**
  - Svaka ćelija može sadržavati fiksni tekst ili biti vezana za neku vrijednost
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ ReportController.cs

```
public class MasterDetailsHeaders : IPageHeader {
  public PdfGrid RenderingGroupHeader( ...
      ...
      var table = new PdfGrid(relativeWidths: new[]
            { 2f, 5f, 2f, 3f }) { WidthPercentage = 100 };

       table.AddSimpleRow(
            ...
          (cellData, cellProperties) => {
            cellData.Value = datDokumenta;
            cellProperties.PdfFont = PdfRptFont;
            cellProperties.HorizontalAlignment =
                  HorizontalAlignment.Left;
            cellProperties.DisplayFormatFormula =
             obj => ((DateTime)obj).ToString("dd.MM.yyyy");
          });
```

BY NC SA

Refleksija

2019/20.10-A

Refleksija

- **Proces kojim program može pregledavati i modificirati vlastitu strukturu tijekom izvođenja**
- **Refleksija se upotrebljava za:**
  - dohvat metapodataka (sadržanih u atributima)
  - otkrivanje tipa podatka
    - pristup informacijama o učitanim asemblijima i tipovima definiranim unutar njih (pregled interakcija i instanciranje tipova)
  - dinamičko povezivanje na svojstva i postupke
    - pozivanje svojstava ili postupaka dinamički instanciranog objekta na temelju otkrivenog tipa (*dynamic invocation*), npr. *bind* fonta ili boje
  - stvaranje i korištenje novih tipova za vrijeme izvršavanja programa
- **Primjeri upotrebe**
  - razvoj aplikacija za reverzno inženjerstvo
  - razvoj preglednika razreda
  - razvoj editora svojstava razreda (kao *Properties* prozor)
  - dinamičko uključivanje dodataka (*pluginova*)

cc BY NC SA

Važniji razredi i prostori imena za refleksiju

- **Prostor imena *System.Reflection***
  - sadrži razrede i sučelja za dohvat informacija o tipovima i članovima programskog koda pri izvođenju, kao i dinamičko kreiranje tipova
    - `Assembly` – pruža informacije o asembliju i omogućava dohvat podataka o tipovima definiram unutar njega
    - `MemberInfo` (apstraktni razred)
      - Omogućuje pristup metapodacima o članovima razreda te dinamičko pozivanje postupaka
    - `ConstructorInfo, PropertyInfo, MethodInfo, …`
      - Pristup metapodacima konstruktora, svojstava, odnosno postupaka
- **Razred System.Type**
  - pruža informacije o nekom tipu podatka (puni naziv, je li apstraktan, javan, iz kojeg tipa je izveden, informacije o tipovima koji su korišteni za parametrizaciju, …)
  - služi za dohvat informacija o članovima razreda (atributi, svojstva, postupci, …)

BY NC SA

Stvaranje objekta razreda *System.Type (1)*

- **Ako je razred prisutan (referenciran) u trenutku kompilacije:**
  - Pozivom postupka `GetType` na nekom postojećem objektu:

```
MojRazred r = new MojRazred();
…
Type t = r.GetType();
```

  - Korištenjem operatora `typeof`

```
Type t = typeof(MojRazred);
```

  - Korištenjem statičkog postupka `GetType` iz razreda `Type` uz navođenje punog imena naziva razreda

```
Type t = Type.GetType("MojProjekt.MojRazred");
```

BY NC SA

Stvaranje objekta razreda *System.Type (2)*

- **Ako razred nije prisutan (referenciran) u trenutku kompilacije:**
  - Korištenjem statičkog postupka `GetType` iz razreda `Type` uz navođenje punog imena naziva razreda pripadajućeg asemblija (odvojen zarezom i razmakom)
    - `Type t = Type.GetType ("MojProjekt.MojRazred, MojProjekt");`
    - Ako je razred generički iza naziva imena dodaje se znak ` i broj tipova s kojima je razred parametriziran

      `Type.GetType("System.Collection.Generic.Dictionary`2");`
    - Postupak `Type.GetType` je preopterećen te po želji omogućuje ignoriranje velikih i malih slova

Dinamičko učitavanje asemblija

- **Asemblij (dll ili exe) može se dinamički učitati unutar programa**
  - Potrebno uključiti NuGet paket System.Runtime.Loader
- **Učitavanje vrši primjerak tipa AssemblyLoadContext**
  - postupak *LoadFromAssemblyPath*
- **Potrebno navesti punu putanju do asemblija**
  - Apsolutna putanja može se dobiti iz relativne pomoću Path.GetFullPath
- **Iz učitanog asemblija mogu se dobiti informacije o svim tipovima podataka ili o nekom konkretnom**
- **Primjer: 🗁 Refleksija \ Refleksija \ Program.cs**

```
string assemblyLocation =" ... LotoImplementacija.dll";
AssemblyLoadContext loadctx = AssemblyLoadContext.Default;
Assembly asm = loadctx.LoadFromAssemblyPath(
                    Path.GetFullPath(assemblyLocation));

Type type = asm.GetType("LotoImplementacija.Loto");
...
```

BY NC SA

Naknadno povezivanje (engl. *late binding*)

- **Nakon što su dostupne informacije o nekom tipu (objekt tipa Type) moguće stvoriti novi objekt tog tipa i pozivati njegove postupke**
- **Postupak *CreateInstance* unutar razreda *Activator* stvara novi objekt određenog tipa**
  - Primjer: 🗁 Refleksija \ Refleksija \ Program.cs

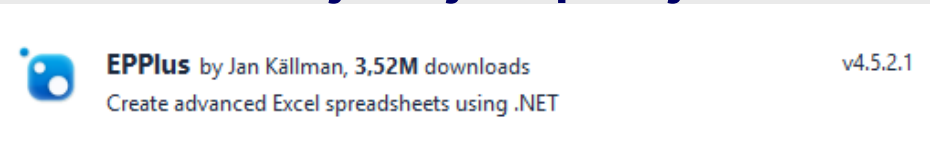
```
Type type = asm.GetType("LotoImplementacija.Loto");
object obj = Activator.CreateInstance(type, 7, 39);

MethodInfo info = type.GetMethod("IzvuciBrojeve");
object result = info.Invoke(obj, new object[] {false});

string ispis = string.Join(", ", (List<int>)result);
Console.WriteLine(ispis);
```

- **Bolja varijanta: definirati sučelje koje dinamički stvoreni tip implementira (ILoto u primjeru)**

CC BY NC SA

Izvoz podataka u Excel

- **Prije još jedne demonstracije upotrebe refleksije slijedi primjer izvoza podataka u Excel**
  - Potrebno instalirati paket EPPlus



- **Primjer: 🗁 Web \ Firma.Mvc \ ReportController.cs**

```
using (ExcelPackage excel = new ExcelPackage()) {
   var worksheet = excel.Workbook.Worksheets.Add("Države");
   worksheet.Cells[1, 1].Value = "Oznaka države";
   worksheet.Cells[1, 2].Value = "Naziv države";
   ...
   for (int i = 0; i < drzave.Count; i++) {
      worksheet.Cells[i + 2, 1].Value = drzave[i].OznDrzave;
       ...
   content = excel.GetAsByteArray();
}
return File(content, ExcelContentType, "drzave.xlsx");
```

CC BY NC SA

Korištenje refleksije za izvoz u Excel (1)

- **Ideja: napraviti generičko rješenje za izvoz neke kolekcije u Excel**
  - Pomoću refleksije dobiti informacije o svim svojstvima nekog tipa, a zatim dinamički uzimati vrijednost pojedinog svojstva
  - Rješenje napisano kao proširenje sučelja IEnumerable<T>
    - Proširenje nazvano CreateExcel (vidi sljedeći slajd)
- **Primjer korištenja: 🗁 Web \ Firma.Mvc \ DokumentController.cs**

```
public async Task<IActionResult> DokumentiExcel() {
    var dokumenti = await ctx.ViewDokumentInfo
                    ...
                    .ToListAsync();
    byte[] content;
    using (ExcelPackage excel =
                    dokumenti.CreateExcel("Dokumenti")) {
        content = excel.GetAsByteArray();
    }
    return File(content, ExcelContentType, "dokumenti.xlsx");
}
```

BY NC SA

Korištenje refleksije za izvoz u Excel (2)

- **Nazivi stupaca jednaki nazivima (jednostavnih) svojstava**
  - Primjer 🗁 Web \ Firma.Mvc \ Extensions \ ExcelCreator.cs

```
public static ExcelPackage CreateExcel<T>(
       this IEnumerable<T> data, string worksheetName) {
  ExcelPackage excel = new ExcelPackage();
  var worksheet = excel.Workbook
                        .Worksheets.Add(worksheetName);
  int row = 1;int col = 1;
  PropertyInfo[] props = typeof(T)
                         .GetProperties(BindingFlags.Instance
                                        | BindingFlags.Public);
  foreach (var prop in props) {
    if (prop.PropertyType is IEnumerable)
          continue; //preskoči kolekcije
    string name = prop.Name;
    worksheet.Cells[row, col++].Value = name;
  }
```

CC BY NC SA

Korištenje refleksije za izvoz u Excel (3)

- **Vrijednost nekog svojstva nekog objekta dobije se iz objekta tipa *PropertyInfo***
  - Primjer 🗁 Web \ Firma.Mvc \ Extensions \ ExcelCreator.cs

```
public static ExcelPackage CreateExcel<T>(
       this IEnumerable<T> data, string worksheetName) {
  ...
  foreach (T t in data) {
     ++row; col = 1;
     foreach (PropertyInfo prop in props) {
        if (prop.PropertyType is IEnumerable)
            continue; //preskoči kolekcije

        object value = prop.GetValue(t);
        worksheet.Cells[row, col].Value = value;
        ++col;
     }
  }
```

Atributi

- **Atributi su nadodane oznake tipovima, poljima, postupcima i svojstvima**
  - Definirani uglatim zagradama [ ] prije deklaracije entiteta koji opisuju
  - Dostupni programski tijekom izvođenja programa
- **Primjeri već definiranih atributa**
  - `Required` – validacijski atribut
  - `Display` – koristi se kod MVC tag-helpera *label-for*
  - `Obsolete` – označava da je neki postupak zastario (upozorenja pri kompilaciji)
  - …
- **Razred *MemberInfo* – (iz kojeg se izvodi PropertyInfo i slični)**
  - `GetCustomAttribute` – vraća atribut danog tipa *Type* primijenjenog na asemblij, član razreda, …
  - `GetCustomAttributes` – vraća polje atributa
  - `IsDefined` – određuje je li ijedan atribut zadanog tipa *Type* definiran

CC BY NC SA

Definiranje naziva stupca atributom

- **Umjesto naziva svojstva za naziv stupca uzima se vrijednosti atributa Display**
  - Slično radi i MVC ako se koristi *label-for*
  - Primjer 🗁 Web \ Firma.Mvc \ Extensions \ ExcelCreator.cs

```
public static ExcelPackage CreateExcel<T>(
       this IEnumerable<T> data, string worksheetName) {
  ...
  PropertyInfo[] props = typeof(T).GetProperties(...
  foreach (var prop in props) {
    string name = prop.Name;
    if (prop.IsDefined(typeof(DisplayAttribute))) {
      name = prop.GetCustomAttribute<DisplayAttribute>().Name;
    }
    worksheet.Cells[row, col++].Value = name;
```

CC BY NC SA

Vlastiti atributi

- **Vlastiti atributi nasljeđuju *System.Attribute***
- **Naziv uobičajeno završava s *Attribute*, ali nije nužno**
  - Ako završava s *Attribute*, taj se sufiks prilikom korištenja može se izostaviti

```
public class PrviAttribute : System.Attribute{
       public string Naziv { get; set; }
       public int Broj { get; set; }
}
public class DrugiAttribute : System.Attribute{
       public string X { get; set; }
}
[Prvi(Broj=5, Naziv="Test")]
[DrugiAttribute(X = "Proba")]
public class MojRazred{...}
```

BY NC SA

Primjer vlastitog atributa

- **Primjer 🗁 Web \ Firma.Mvc \ Util \ ExcelFormatAttribute.cs**
  - Atribut kojim će se definirati format ćelije u Excelu
  - Može se primijeniti samo na svojstva
    - određeno atributom AttributeUsage

```
[AttributeUsage(AttributeTargets.Property)]
public class ExcelFormatAttribute : Attribute {
    public string ExcelFormat { get; set; } = string.Empty;

    public ExcelFormatAttribute(string format)
    {
      ExcelFormat = format;
    }
 }
```

BY NC SA

Primjer korištenja vlastitog atributa (1)

- **Primjer 🗁 Web \ Firma.Mvc \ Models \ ViewDokumentInfo.cs**

```
public class DokumentViewModel {
  ...
  [ExcelFormat("0.00%")]
  public decimal PostoPorez { get; set; }
  [DisplayAttribute(Name = "Datum dokumenta")]
  [ExcelFormat("dd.mm.yyyy")]
  ...
  [ExcelFormat("#,###,##0.00")]
  public decimal IznosDokumenta { get; set; }
```

  - Napomena: EPPlus uvijek koristi engleske regionalne postavke za decimalni zarez odnosno točku

Primjer korištenja vlastitog atributa (2)

- **Primjer 🗁 Web \ Firma.Mvc \ Extensions \ ExcelCreator.cs**

```
PropertyInfo[] props = typeof(T).GetProperties(...
...
foreach (T t in data) {
  ++row; col = 1;
  foreach (var prop in props)
      ...
      object value = prop.GetValue(t);
      worksheet.Cells[row, col].Value = value;
      if (prop.IsDefined(typeof(ExcelFormatAttribute))) {
            string format = prop
                 .GetCustomAttribute<ExcelFormatAttribute>()
                 .ExcelFormat;
            if (!string.IsNullOrWhiteSpace(format)) {
              worksheet.Cells[row, col].Style
                    .Numberformat.Format = format;
```

Web servisi.

2019/20.10-B

Servis

- **Jedna ili više funkcionalnih komponenti (ili cijeli sustav) s kojim se komunicira putem javno objavljenih i precizno definiranih sučelja**
  - mrežno dostupan podatkovni i/ili računalni resurs
  - osigurava mehanizam za pozivanje udaljenih postupaka
  - prima jedan ili više zahtjeva i vraća jedan ili više odgovora
- **Princip crne kutije**
- **Heterogeni klijenti**
- **Koristi otvorene web standarde:**
  - HTTP, XML, JSON, SOAP, OData …

SOA

- **SOA = Servisno orijentirana arhitektura**
  - engl. Service Oriented Architecture
  - omogućava izradu distribuiranih aplikacija između različitih platformi koje komuniciraju razmjenom podataka među servisima
- **Skup precizno definiranih, međusobno neovisnih servisa povezanih u logički jedinstvenu aplikaciju**
  - Objektno orijentirana aplikacija povezuje objekte
  - Servisno orijentirana aplikacija povezuje servise
- **Distribuirani sustav u kojem sudjeluje više autonomnih servisa međusobno šaljući poruke preko granica**
  - Granice mogu biti određene procesom, mrežom, …

BY NC SA

Problemi i preporuke prilikom izradu servisa

- **Sigurnost komunikacije**
- **Konzistentnost stanja**
  - Neuspjeh prilikom izvršavanja servisa ne smije ostaviti sustav u stanju pogreške
- **Problem višenitnosti**
- **Pouzdanost i robusnost servisa**
  - Klijent treba znati je li servis primio poruku.
  - Pogreške u servisu treba obraditi
- **Interoperabilnost**
  - Tko sve može pozvati servis?
- **Skalabilnost**
- **Brzina obrade postupka**

BY NC SA

Osnovna pravila servisno orijentirane arhitekture

- **Jasno određene granice**
  - Jasno iskazana funkcionalnost i struktura podataka
  - Implementacija = crna kutija
- **Neovisnost servisa**
  - Servis ne ovisi o klijentu, nekom drugom servisu, lokaciji i vrsti instalacije
  - Verzije se razvijaju neovisno o klijentu. Objavljene verzije se ne mijenjaju.
- **Ugovor, a ne implementacija**
  - Korisnik servisa i implementator servisa dijele samo listu javnih postupaka i definiciju struktura podataka
    - Dijeljeni podaci trebaju biti tipovno neutralni
    - Tipovi specifični za pojedini jezik moraju se moći pretvoriti u neutralni oblik i obrnuto
  - Implementacijski postupci ostaju tajna
- **Semantika, a ne samo sintaksa**
  - Logička kategorizacija servisa, smisleno imenovanje postupaka

Standardi za web servise - SOAP

- **SOAP – Simple Object Access Protocol**
  - Donedavno *de facto* standard za izradu web servisa i razmjenu informacija u distribuiranim, heterogenim okruženjima ≈ HTTP POST + XML
  - Orijentiran na akcije (engl. action driven)
  - Postupak web-servisa koji treba pozvati zapisan u sadržaju zahtjeva
- **WSDL - Web Services Description Language**
  - XML shema za opis SOAP web-servisa
  - definira format postupaka koje pruža Web servis
- **UDDI - Universal Description, Discovery, and Integration**
  - propisuje način dokumentiranja servisa Discovery (URI i WSDL opisi) koji bi se objavljivali u registracijskim bazama (npr. http://uddi.xml.org)
  - ideja UDDI napuštena
- **WS-***
  - Skup standarda za sigurnost, podatke i opise SOAP web-servisa

BY NC SA

Standardi za web servise - REST

- **REST = Representational State Transfer**
  - Alternativa SOAP protokolu
    - Nije protokol, već način izrade servisa (konceptualno ne mora biti baziran nad HTTP protokolom)
  - Orijentiran na resurse (eng. resourse driven)
- **REST web-servisi izlažu „resurse”.**
  - Adresa servisa jednoznačno određuje resurs
- **Akcije određene vrstom HTTP zahtjeva**
  - GET – dohvat podataka
  - POST - stvaranje novog podatka
  - PUT i PATCH – izmjene cijelog ili dijela podatka
  - DELETE – brisanje podatka
- **Standard za dokumentiranje OpenAPI (Swagger)**

cc BY NC SA

REST vs SOAP

- **Slanjem SOAP poruke na pristupnu točku provjerava se sadržaj i na osnovu sadržaja se određuje postupak koji se treba izvršiti**
- **Korištenjem REST-a postupak se određuje na osnovu url-a i http metode (GET, POST, DELETE, PUT)**
- **Prednosti REST-a u odnosu na SOAP**
  - Veći broj potencijalnih klijenata
    - Dovoljna je podrška za Http i XML ili JSON
    - Nije potrebno implementirati složene WS-* standarde
  - Lakše *cacheiranje*
  - Manje poruke
    - POX (Plain old XML) – XML poruke bez SOAP zaglavlja
    - JSON – (JavaScript Object Notation)
- **Nedostatak**
  - Nema wsdl-a, veća mogućnost nepravilne poruke
  - Praktično se rješava alatima za dokumentiranje servisa (Swagger i OpenAPI) i generatorima klijentskih razreda

CC BY NC SA

REST i vrste HTTP poziva

- **GET**
  - Služi za dohvat podataka
    - Dohvat svih podataka
    - Dohvat jednog podatka po primarnom ključu
    - (Opcionalno) dohvat na osnovu kriterija pretrage
  - Vraća rezultat ili HTTP status 404
- **POST**
  - Služi za kreiranje novog podatka
  - Za uspješno stvoreni podatak vraća HTTP status 201
- **PUT/PATCH**
  - Služi za ažuriranje cijelog podatka (PUT) ili dijela podatka (PATCH)
  - Nakon uspješnog ažuriranja vraća HTTP status 200 ili 204
- **DELETE**
  - Služi za brisanje podatka
  - Ako je podatak uspješno obrisan ili je već ranije bio obrisan vraća HTTP status 200 ili 204
- **U kompletu čini serverski Web API**

BY NC SA

Kostur Web API upravljača

```
[Route("api/[controller]")]
 public class ValuesController : BaseController {
   // GET: api/values
   [HttpGet]
   public IEnumerable<string> Get() {
        return new string[] { "value1", "value2" };
   }
   // GET api/values/5
   [HttpGet("{id}")]
   public string Get(int id) {
        return "value";
   }
   // POST api/values
   [HttpPost]
   public void Post([FromBody]string value) {      }
   // PUT api/values/5
   [HttpPut("{id}")]
   public void Put(int id, [FromBody]string value) {   }
   // DELETE api/values/5
   [HttpDelete("{id}")]
   public void Delete(int id) {    }
```

BY NC SA

Primjer Web API servisa – Web API za države

- **Definira se slično kao i ostali MVC upravljači, ali ima definirano vlastito usmjeravanje te nasljeđuje ControllerBase**
  - Ovisnosti o sučeljima i razredima u konstruktoru kao i za ostale upravljače
  - Primjer: 🗁 Web \ Firma.Mvc \ Areas \ WebApi \ Controllers \ DrzavaController.cs

```
[Route("api/[controller]")]
public class DrzavaController : ControllerBase  {
    private readonly FirmaContext ctx;
    private ILogger logger;

    public DrzavaController(FirmaContext ctx,
                            ILogger<MjestoController> logger) {
      this.logger = logger;
      this.ctx = ctx;
    }
```

BY NC SA

Web API – dohvat svih država (1)

- **Naziv postupka nebitan – postupak se određuje prema atributu HttpGet i usmjeravanju upravljača**
  - Konkretno u ovom primjeru /api/drzava
  - Poželjno koristiti zasebni model neovisno o sličnosti s modelom iz EF-a
    - Omogućava naknadno neovisnu izmjenu podatkovnog sloja
  - Primjer: 🗁 Web \ Firma.Mvc \ Areas \ WebApi \ Controllers \ DrzavaController.cs

```
[HttpGet]
public async Task<List<Firma.Api.Dto.Drzava>> Get() {
   var list = await ctx.Drzava
                        .Select(d => new Firma.Api.Dto.Drzava {
                             SifDrzave = d.SifDrzave,
                             OznDrzave = d.OznDrzave,
                             Iso3drzave = d.Iso3drzave,
                             NazDrzave = d.NazDrzave
                         })
                         .ToListAsync();
      return list;
}
```

BY NC SA

Web API – dohvat svih država (2)

- **Otvaranjem adrese http://.../api/drzava dobit će se JSON sadržaj s popisom svih država**
  - Popis država serijaliziran u JSON
    - Konfiguracijskom datotekom može se postaviti i drugi format (npr. Xml)

localhost:50492/api/drzava

{"OznDrzave":"IT","NazDrzave":"Italy","Iso3drzave":"ITA","SifDrzave":380},{"OznDrzave":"JM","NazDrzave":"J
{"OznDrzave":"JP","NazDrzave":"Japan","Iso3drzave":"JPN","SifDrzave":392},{"OznDrzave":"KE","NazDrzave":"K
{"OznDrzave":"KH","NazDrzave":"Cambodia","Iso3drzave":"KHM","SifDrzave":116},{"OznDrzave":"KI","NazDrzave"
{"OznDrzave":"KN","NazDrzave":"Saint Kitts and Nevis","Iso3drzave":"KNA","SifDrzave":659},{"OznDrzave":"KP
of","Iso3drzave":"KOR","SifDrzave":410},{"OznDrzave":"KW","NazDrzave":"Kuwait","Iso3drzave":"KWT","SifDrza
{"OznDrzave":"KZ","NazDrzave":"Kazakhstan","Iso3drzave":"KAZ","SifDrzave":398},{"OznDrzave":"LA","NazDrzav
{"OznDrzave":"LC","NazDrzave":"Saint Lucia","Iso3drzave":"LCA","SifDrzave":662},{"OznDrzave":"LI","NazDrza
{"OznDrzave":"LR","NazDrzave":"Liberia","Iso3drzave":"LBR","SifDrzave":430},{"OznDrzave":"LS","NazDrzave":
{"OznDrzave":"LU","NazDrzave":"Luxembourg","Iso3drzave":"LUX","SifDrzave":442},{"OznDrzave":"LV","NazDrzav

  - Ne odredi li se drugačije (serijalizacijskim atributima ili postavkama u Startup.cs), koristit će se camelCase
    - Primjer 🗁 Web \ Firma.Mvc \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
    services.AddControllersWithViews()
            .AddJsonOptions(options =>
options.JsonSerializerOptions.PropertyNamingPolicy = null);
```

CC BY NC SA

Web API – dohvat države s određenom oznakom

- **Putanja oblika /api/drzava/oznakadrzave (npr. /api/drzava/HR)**
  - Usmjeravanje imenovano posebnim nazivom kao bi se moglo naknadno referencirati (vidi primjer za stvaranje nove države)
  - Vraća jednu državu, ali povratna vrijednost tipa IActionResult
    - Ako država ne postoji, potrebno vratiti status 404 (Not found)
    - Od ASP.NET Corea 2.1 uveden ActionResult<T>
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ WebApi \ DrzavaController.cs

```
[HttpGet("{oznDrzave}", Name = "DohvatiDrzavu")]
public async Task<IActionResult> Get(string oznDrzave) {
      var drzava = await ctx.Drzava.AsNoTracking()
                        .Where(d => d.OznDrzave == oznDrzave)
                        .FirstOrDefaultAsync();
      if (drzava == null)
        return NotFound("Tražena država ne postoji");
      else  {
        var model = new Firma.Api.Dto.Drzava  {
          SifDrzave = drzava.SifDrzave, ...

        return Ok(model);//status 200 + podatak
```

Web API – dodavanje države

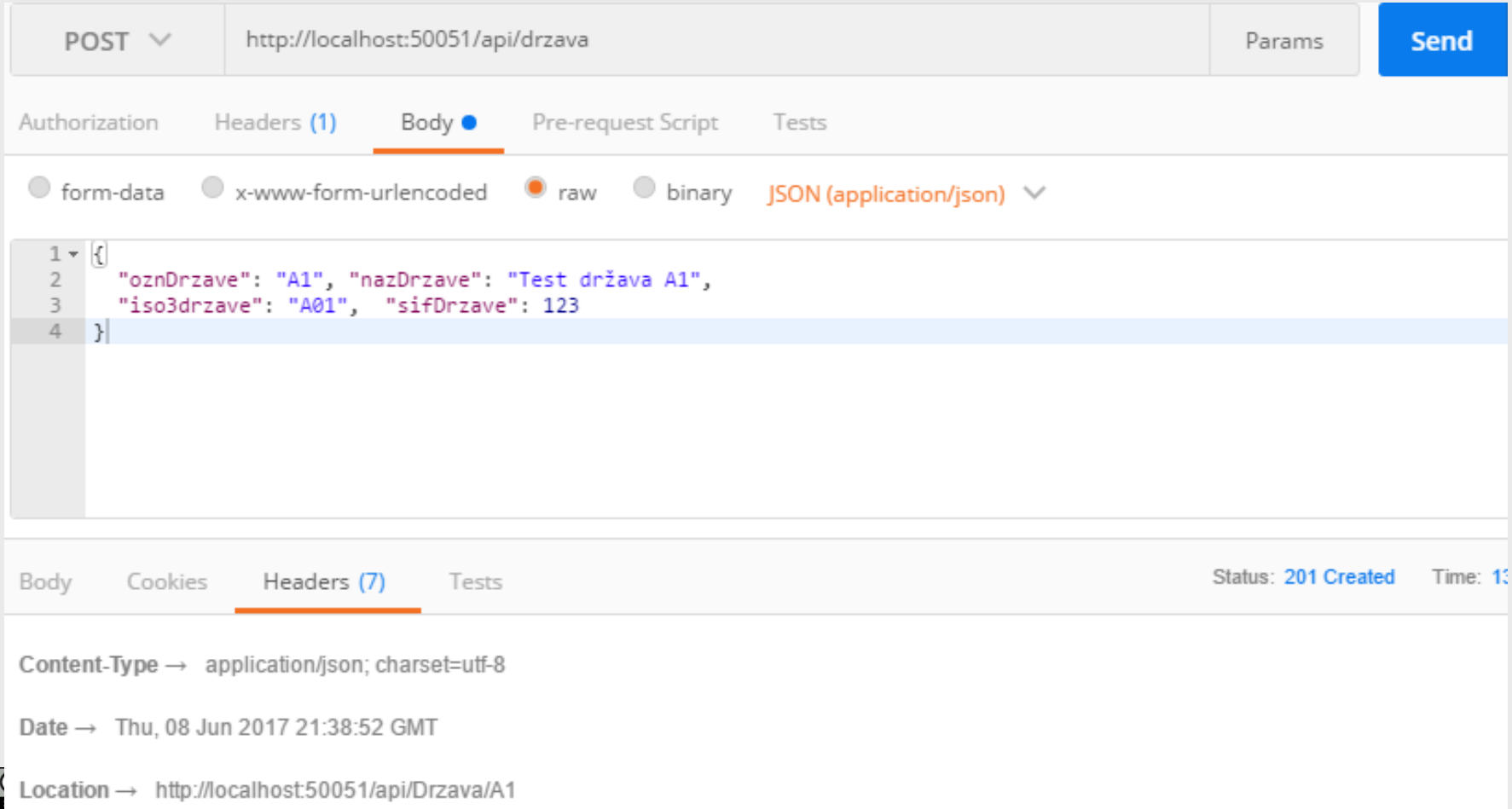
- **Postupak POST proizvoljnog imena u kojem se model rekonstruira iz tijela zahtjeva**
  - Ako je model ispravan posprema podatak u bazu podataka i vraća 201
    - Uz status vraća pohranjeni podataka i adresu podatka
      - Adresa je dio zaglavlja odgovora, podatak je u tijelu
  - Neispravni model uzrokuje statusnu poruku 400 (BadRequest)
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ WebApi \ DrzavaController.cs

```
[HttpPost]
public async Task<IActionResult> Create([FromBody]
                                Firma.Api.Dto.Drzava model)  {
      if (model != null && ModelState.IsValid) {
        Drzava drzava = new Drzava {
          SifDrzave = model.SifDrzave ...
        ctx.Add(drzava);
        await ctx.SaveChangesAsync();
        return CreatedAtRoute("DohvatiDrzavu",
              new { oznDrzave = drzava.OznDrzave }, model);
      }
      else return BadRequest(ModelState);
```

BY NC SA

Kako isprobati POST i ostale zahtjeve?

- **Postman – REST Client (https://www.getpostman.com/)**
  - Alternativa Fiddler ili slični alati, vlastita konzolna aplikacija
  - Postaviti Content-Type na application/json i poslati odgovarajući JSON



CC BY NC

Web API – ažuriranje države

- **Postupak PUT proizvoljnog imena**
  - Potrebno poslati cijeli model, a ne samo parcijalne promjene (PATCH)
  - Oznaka države iz zahtjeva i ona iz modela moraju biti jednake
  - Uspješna promjena podatka vraća statusnu poruku 204
    - Neispravni podaci: 400 ili 404 (ako država te oznake ne postoji)
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ WebApi \ DrzavaController.cs

```
[HttpPut("{oznDrzave}")]
public async Task<IActionResult> Update(string oznDrzave,
                                        [FromBody] DrzavaApiModel model) {
      if (model == null || model.OznDrzave != oznDrzave ||
!ModelState.IsValid)
              return BadRequest(ModelState);
      else
        var drzava = await ctx.Drzava.FindAsync(oznDrzave);
        if (drzava == null) return NotFound("... ne postoji");
        else {
          drzava.SifDrzave = model.SifDrzave; ...
          await ctx.SaveChangesAsync();
          return NoContent();
```

BY NC SA

Web API – brisanje države

- **Postupak DELETE proizvoljnog imena sa šifrom države u adresi**
  - Uspješno brisanje vraća statusnu poruku 204
    - Nepostojeća država 404
  - Primjer: 🗁 Web \ Firma.Mvc \ Controllers \ WebApi \ DrzavaController.cs

```
[HttpDelete("{oznDrzave}")]
public async Task<IActionResult> Delete(string oznDrzave) {
      var drzava = await ctx.Drzava.FindAsync(oznDrzave);
      if (drzava == null)
      {
        return NotFound("Tražena država ne postoji");
      }
      else
      {
        ctx.Remove(drzava);
        await ctx.SaveChangesAsync();
        return NoContent();
      };
    }
```

BY NC SA

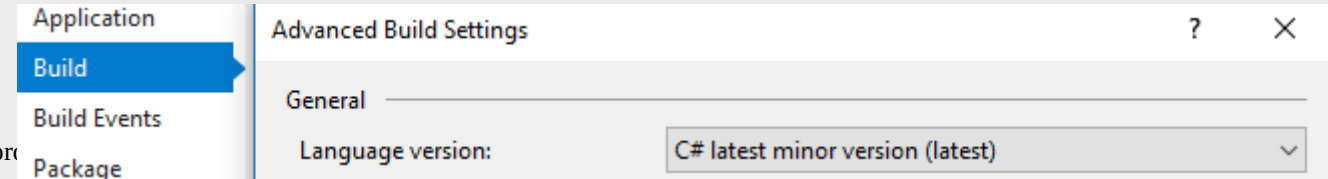
Primjer poziva REST servisa iz konzolne aplikacije (1)

- **Primjer: 🗁 Web\WebApiClients\ConsoleCallsWebApi\Program.cs**
  - Razred *HttpClient* za pozivanje REST servisa
  - Paket Microsoft.AspNet.WebApi.Client za proširenje ReadAsAsync<T>

.NET **Microsoft.AspNet.WebApi.Client** by Microsoft, **39,2M** downloads v5.2.6
This package adds support for formatting and content negotiation to System.Net.Http.

```
private static async Task DohvatiSveDrzave() {
  using (var client = new HttpClient()) {
     var response = await client.GetAsync(url);
     if (response.IsSuccessStatusCode) {
        var drzave = await response
                                .Content.ReadAsAsync<Drzava[]>();
        Console.WriteLine("Ispis prvih 5 drzava: ");
        for (int i = 0; i < drzave.Length && i < 5; i++) {
            Console.WriteLine(drzave[i].ToString());
        }...
```

  - HttpClient sadrži samo asinkrone postupke, pa je potrebno koristiti asinkronu varijantu glavnog programa (static async Task Main(string[] args)
    - C# 7.1 ili više



Primjer poziva REST servisa iz konzolne aplikacije (2)

- **Primjer: 🗁 Web\WebApiClients\ConsoleCallsWebApi\Program.cs**
  - Potrebno postaviti tip sadržaja na application/json
  - Poziva se postupak *PostAsJsonAsync*
    - Slično u primjerima koji slijede PutAsJsonAsync, DeleteAsync, …

```
private static async Task DodajDrzavu(string oznDrzave,
             string nazDrzave, string iso3, int sifra) {
  Drzava drzava = new Drzava {
        OznDrzave = oznDrzave, NazDrzave = nazDrzave, ...
  };
  using (var client = new HttpClient()) {
     var response = await client
                    .PostAsJsonAsync<Drzava>(url, drzava);
     if (response.IsSuccessStatusCode) {
          Console.WriteLine("Drzava dodana. Adresa: {0}",
                    response.Headers.Location);
        ...
```

CC BY NC SA

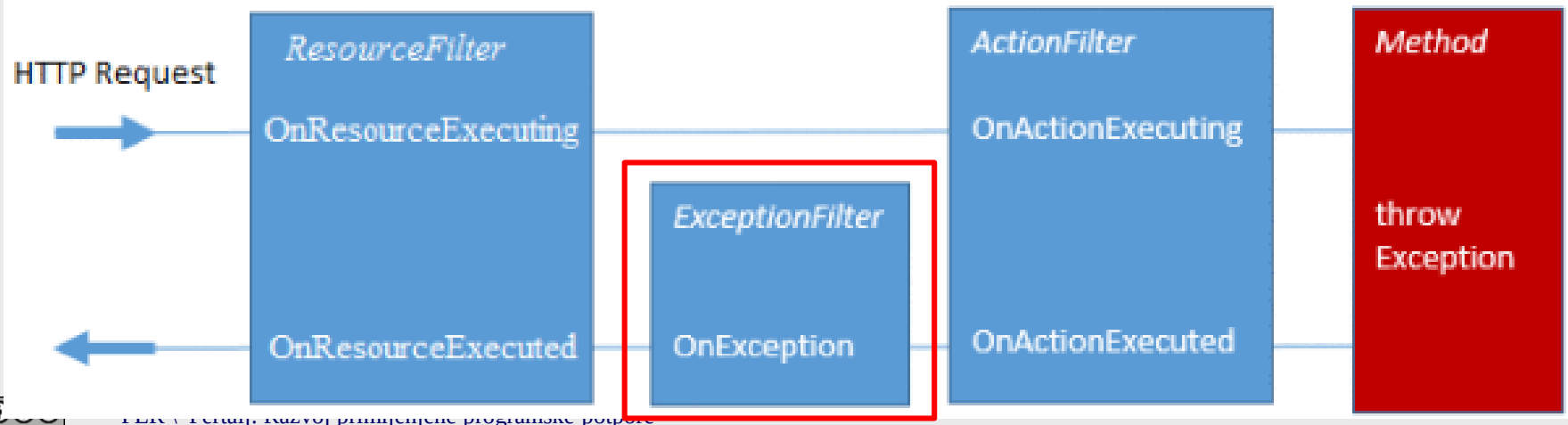
Što u slučaju pogreške unutar upravljača?

- **Pogreška prilikom spremanja podatka?**
- **Neuhvaćena iznimka?**
- **…**
- **Različiti pristupi**
  - „Zabijanje glave u pijesak”
    - pravimo se da se iznimka neće dogoditi
    - Ako se dogodi izaziva statusnu poruku broj 500 (Interval Server Error) koja korisniku ne znači previše, a može otkriti neželjene interne podatke
  - „Sve OK”
    - Čitavi programski kôd servisa omotan u try-catch block, a rezultat je razred koji sadrži omotane podatke koje je trebao vratiti i informaciju o uspješnosti postupka i eventualnoj pogrešci
    - Korisnik uvijek dobiva statusnu poruku 200
      - Primjer 🗁 Firma.Api.Dto \ ActionResponse.cs i
        🗁 Firma.Api.Dto \ ActionResponseExtensions.cs
        🗁 Firma.Mvc \ … \ ExcDemoApiController.cs
        🗁 ConsoleCallApiException \ Programs.cs
  - Nešto treće?

CC BY NC SA

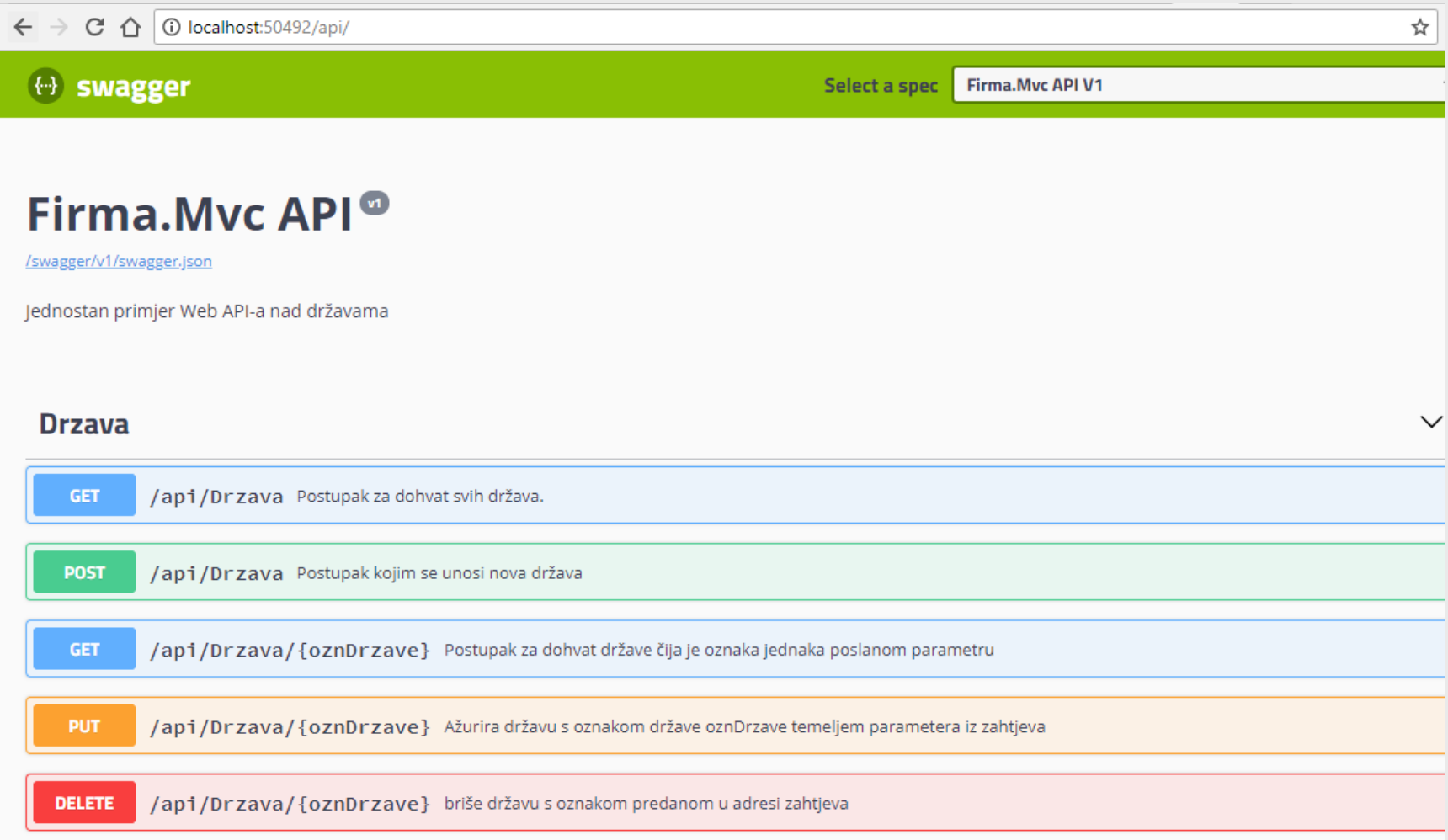
Obrada iznimke korištenjem atributa

- **Obrada iznimke unutar postupka**
  - Napisati vlastiti atribut implementiranjem sučelja IExceptionFilter ili izvođenjem iz razreda ExceptionFilterAttribute
  - Zamjenjuje naporno pisanje try-catch blokova u svakom postupku i centralizira ponašanje u slučaju nepredviđene iznimke
  - Koristi se kao tip unutar atributa ServiceFilter, npr. [ServiceFilter(typeof(FirmaExceptionFilter))]
    - Potrebno registrirati u razredu Startup services.AddScoped<FirmaExceptionFilter>();
- **Primjeri:** 🗁 Firma.Mvc \ Util \ FirmaExceptionFilter.cs
  🗁 Firma.Mvc \ … \ExcDemoApi.cs – postupak GetFilter



Dokumentacija za web servise

- Za dokumentiranje Web API servisa koristi se alat Swagger
- Osim informacija o tipovima podataka i rezultata omogućava i pozivanje servisa



Aktivacija Swaggera (1)

- **Dodati NuGet paket *Swashbuckle.AspNetCore***
- **U *Startup.cs* dodati aktivaciju *Swaggera***
  - *Swagger* automatski pronalazi sve upravljače koji imaju definiranu rutu i atribute Http[Get|Post|…]
  - U postavkama projekta uključiti kreiranje XML dokumentacije
    - Navesti putanje do svih xml datoteka koje treba uključiti
- **Primjer 🗁 Web \ Firma.Mvc \ Startup.cs**

```
public void ConfigureServices(...
  ...
  services.AddSwaggerGen(c => {
     c.SwaggerDoc("v1", new Info { Title = "Firma.Mvc API",
              ...
     });
     var xmlFile =
      $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
     var xmlPath = Path.Combine(
                   AppContext.BaseDirectory, xmlFile);
     c.IncludeXmlComments(xmlPath);
```

Aktivacija Swaggera (2)

- **Primjer 🗁 Web \ Firma.Mvc \ Startup.cs**

```
public void Configure(...
  ...
   app.UseSwagger();
   app.UseSwaggerUI(c =>
      {
        c.SwaggerEndpoint("/swagger/v1/swagger.json",
                          "Firma.Mvc API V1");
        c.RoutePrefix = "api";
      });
```

- **Nakon navedenih postavki automatski generirana dokumentacija za WebApi servise je dostupna na http://.../api/**
  - Npr. http://localhost:50492/api/

BY NC SA

Informacije o statusnim porukama

- **Swagger pretpostavlja da svi postupci vraćaju status 200**
  - U slučaju da nije tako, potrebno eksplicitno navesti moguće vrijednosti iznad postupka
  - Koristi se atribut *ProducesResponseType*
  - Primjer 🗁 Web \ Firma.Mvc \ Controller \ WebApi \ DrzavaController.cs

```
[HttpPut("{oznDrzave}")]
[ProducesResponseType((int)HttpStatusCode.NotFound)]
[ProducesResponseType((int)HttpStatusCode.NoContent)]
[ProducesResponseType((int)HttpStatusCode.BadRequest)]
public async Task<IActionResult> Update(
     string oznDrzave, [FromBody] DrzavaApiModel model)
```

CC BY NC SA

Informacije o povratnim porukama

- **Swagger provjerava tip povratne vrijednosti iz servisa**
  - *IAsyncResult* je općeniti rezultat, pa *Swagger* nema automatski tu informaciju - alternativa: koristiti ActionResult<povratni_tip>
  - Informacija se postavlja atributom *ProducesResponseType* ili izvedenim iz njega *SwaggerResponseAttribute*
    - Izvedeni razred sadrži mogućnosti dodavanja opisa
  - Primjer 🗁 Web \ Firma.Mvc \ Controller \ WebApi \ DrzavaController.cs

```
[[HttpGet("{oznDrzave}", Name = "DohvatiDrzavu")]
[ProducesResponseType(typeof(Firma.Api.Dto.Drzava),
                      (int)HttpStatusCode.OK)]
[ProducesResponseType((int)HttpStatusCode.NotFound)]
public async Task<IActionResult> Get(string oznDrzave)
...
[HttpGet]
[SwaggerResponseAttribute((int)HttpStatusCode.OK,
            typeof(IEnumerable<Firma.Api.Dto.Drzava>),
     Description = "Vraća enumeraciju država") ]
 public async Task<IEnumerable<DrzavaApiModel>> Get()
```

BY NC SA

Generiranje klijenta na osnovi Swagger dokumentacije (1)

- **Na temelju Swaggerove json datoteke i alata NSwag moguće generirati klijente razrede**
  - Generator dio paketa Nswag.CodeGeneration.CSharp

- **Primjer 🗁 Web \ GenerateWebApiClient \ Program.cs**

```
var doc = await SwaggerDocument.FromJsonAsync(json);
var settings = new
      SwaggerToCSharpClientGeneratorSettings {
            ClassName = "FirmaWebApiClient",
            CSharpGeneratorSettings = {
                  Namespace = "GeneratedWebApiClientDemo"
            }
      };
var generator = new SwaggerToCSharpClientGenerator(
                                    doc, settings);
var code = generator.GenerateFile();
```

Generiranje klijenta na osnovi Swagger dokumentacije (2)

- **Generirani klijent sadrži**
  - razrede koji služe kao ulazno/izlazni podaci postupaka web-servisa
  - postupke čiji **naziv odgovara nazivu rute** i s parametrima koji određuju pojedini resurs
  - Smanjuje mogućnost pogrešnog poziva u odnosu na prethodni primjer s razredom *HttpClient*.
- **Primjer 🗁 Web \ GeneratedWebApiClientDemo \ Program.cs**

```
using(HttpClient client = new HttpClient()) {
  var firma = new FirmaWebApiClient(url, client);
  await firma.DodajDrzavuAsync(model);
```

- **Primjer 🗁 Web \ Firma.Mvc \ … \ WebApi \ … DrzavaController.cs**

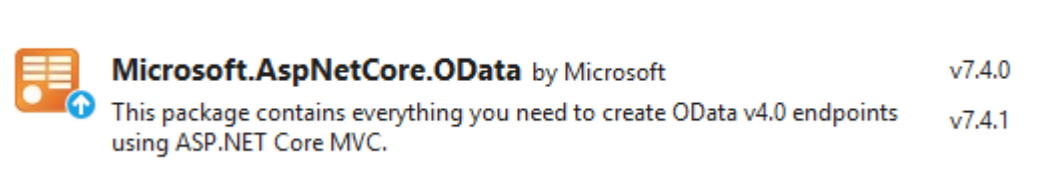
```
[HttpPost(Name = "DodajDrzavu")]
public async Task<IActionResult> Create([FromBody]
                        Firma.Api.Dto.Drzava model)
```

CC BY NC SA

Standardi za web servise - OData

- **OData – Open Data Protocol**
  - Standard za izgradnju *RESTful API-a*
- **Standardizira oblik adrese pojedinog REST servisa**
- **Definira sintaksu za oblikovanje/filtriranje rezultata**
  - Npr. servis *Dokument* na adresi s prefiksom http://localhost:50492/odata/ je oblikovan po OData standardu te omogućava filtriranje rezultata, sortiranje po nekom atributu i slično
  - Podaci se pripremaju na serveru/servisu
  - Primjer: *http://localhost:50492/odata/Dokument?$filter=contains(NazPartnera,'Stipanović') and DatDokumenta lt 2016-01-01 and IznosDokumenta gt 10000&$top=3&$orderby=IznosDokumenta desc*
- **Za detalje o standardu pogledati na http://www.odata.org/**

Izrada REST servisa prema standardu OData (1)



- **Uključiti paket Microsoft.AspNetCore.Odata**
- **Uključiti podršku za OData**
- **Dodati u postavke za Dependency Injection razred koji opisuje model podataka kojeg OData servis koristi (vidi sljedeći slajd)**
  - Primjer 🗁 Web \ Firma.Mvc \ Startup.cs

```
public void ConfigureServices(
      IServiceCollection services) {
      ...
      services.AddOData();
      ...
      services.AddTransient<FirmaODataModelBuilder>();
public void Configure(...
     app.UseEndpoints(endpoints =>       {
        endpoints.MapODataRoute("ODataRoutes", "odata",
odataModelBuilder.GetEdmModel());
```

Izrada REST servisa prema standardu OData (2)

- **Navesti razrede koje se koriste kao ulazno/izlazni podaci Odata servisa i omogućiti pojedine upite**
  - Primjer 🗁 Web \ Firma.Mvc \ Areas \ OData \ FirmaODataModelBuilder.cs

```
public class FirmaODataModelBuilder {
  public IEdmModel GetEdmModel(
            IServiceProvider serviceProvider) {
     var builder = new
          ODataConventionModelBuilder(serviceProvider);
     builder.EntitySet<Firma.Api.Dto.Mjesto>("mjesto"))
            .EntityType
            .Filter() // Allow for the $filter Command
            .Count() // Allow for the $count Command
            .OrderBy() // Allow $orderby
            .Page() // Allow  $top and $skip
            .Select();// Allow for the $select Command;
```

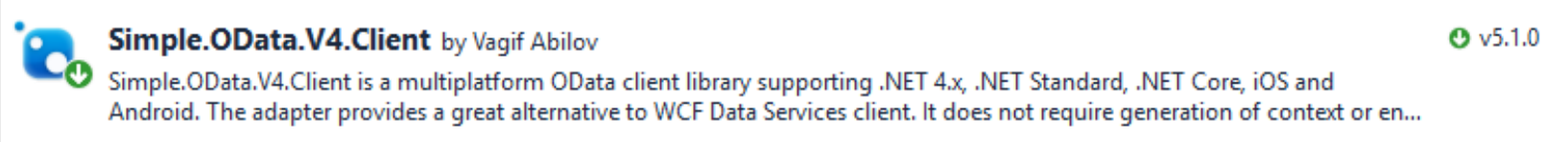
Izrada REST servisa prema standardu OData (3)

- **Upravljač OData web-servisa izveden iz razred ODataController**
  - Atribut EnableQuery za za dohvat više podataka (vraća IQueryable)
  - Patch za promjenu dijela podatka prima objekt tipa Delta<T>
  - Ostali postupci slični kao kod ostalih REST servisa
  - Primjer 🗁 Web\Firma.Mvc\Areas\OData\Controllers\MjestoController.cs

```
[Route("odata")]
public class MjestoController : ODataController {
    ...
  [EnableQuery]
  public IQueryable<Firma.Api.Dto.Mjesto> Get() { ...

  [HttpPost] // POST /odata/Mjesto
  public IActionResult Post([FromBody] Mjesto model) {

 [HttpPatch] // PATCH /odata/Mjesto(key)
 public IActionResult Patch(int key,
                  [FromBody] Delta<Mjesto> model)
```

Pozivanje OData servisa iz konzolne aplikacije

- **Korištenjem nekih od standardnih klijenata**
  - http://www.odata.org/getting-started/understand-odata-in-6-steps/
  - http://www.odata.org/libraries/
- **U primjeru korišten Simple.OData.Client**



- **Za detalje pogledati 🗁 Web\ FirmaODataClient \ Program.cs**

Tehnike rada s podacima u Windows aplikacijama

2017/18.11

Napomena uz primjere

- **Primjeri ilustrirani u sljedećim slajdovima predstavljaju neke osnovne koncepte koji se koriste u radu s Windows aplikacijama i pri povezivanju podataka na formama**
  - Primjeri su izrađeni korištenjem tehnologije WPF, ali slični koncepti vrijede i za WindowsForms te Universal Windows Platform (UWP)
  - Primjeri su opsežni i ne mogu u potpunosti stati na slajdove te se preporuča isprobati primjer i detaljno proučiti programski kôd pri čemu slajdovi mogu poslužiti kao vodilja kojim redom proučavati primjere

- **WPF nije predmet ovih predavanja te je implementirano onoliko koliko je potrebno da se demonstriraju primjeri bez ulaženja u detalje kako pojedina kontrola radi**

CC BY NC SA

Događaji

- **Svaka WPF forma sastoji se od dvije datoteke**
  - *.xaml datoteka sadrži XML kojim se opisuje prezentacijski dio forme
    - djelomično nalik HTML-u: unutar 4 vrste kontejnera (StackPanel, Dock, Grid, Canvas) smještaju se ostale jednostavne ili složene kontrole
  - *.xaml.cs : programski kôd koji se izvršava ovisno o pojavi nekog događaja
- **Događaj (engl. *event*) - mehanizam razreda za dojavu zbivanja nad objektom**
  - najčešće se odnosi na grafičko korisničko sučelje
    - npr. klik mišem na gumb, unos teksta u kućicu formulara, učitavanje/zatvaranje forme, ...
  - … ali se koristi i šire
- **Rukovatelj događajem (engl. *Event handler*)**
  - postupak koji obrađuje događaje
    - pretplatnik na događaj
  - uobičajenog naziva ControlName_EventName
  - uobičajeni argumenti su objekt događaja i podaci događaja

CC BY NC SA

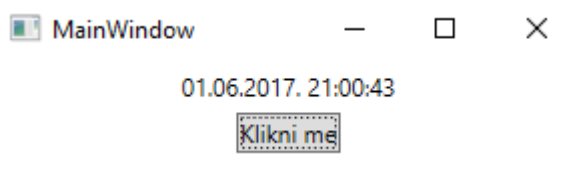
Primjer događaja

- **Primjer: 🗁Windows \ SimpleWpfApp \ MainWindow.xaml**
  - Klikom na gumb trenutno vrijeme se upisuje u labelu
  - Obrađuje se klik na gumb

```
<Window x:Class="SimpleWpfApp.MainWindow" ... Title="MainWindow">
  <StackPanel Orientation="Vertical">
    <Label x:Name="lbl1" Content="..."/>
    <Button x:Name="btn1" Content="Klikni me" Click="btn1_Click" />
  </StackPanel>
</Window>
```

- **Primjer: 🗁Windows \ SimpleWpfApp \ MainWindow.xaml.cs**

```
 private void btn1_Click(object sender, RoutedEventArgs e) {
      lbl1.Content = DateTime.Now.ToString("dd.MM.yyyy. HH:mm:ss");
 }
```



CC BY NC SA

Programsko pridruživanje obrade događaja

- **U prethodnom primjeru kompilacijom nastaje datoteka MainWindow.g.cs koja (između ostalog) sadrži:**

```
this.btn1.Click += new RoutedEventHandler(this.btn1_Click);
```

Događaj — Tip delegata — *Event handler*

- **Osim kroz dizajn pridruživanje postupka događaju može se napraviti i programski**
  - Primjer: 🗁Windows \ SimpleWpfApp \ MainWindow.xaml.cs

```
public partial class MainWindow : Window {
   public MainWindow() {
     InitializeComponent();
     btn1.Click += ShowMessage;
     btn1.Click += (s, e) => {
                     MessageBox.Show("Još jedna poruka");
                   };
   }
   private void ShowMessage(object sender, RoutedEventArgs e) {
     MessageBox.Show("Gumb kliknut...");
   }
```

BY NC SA

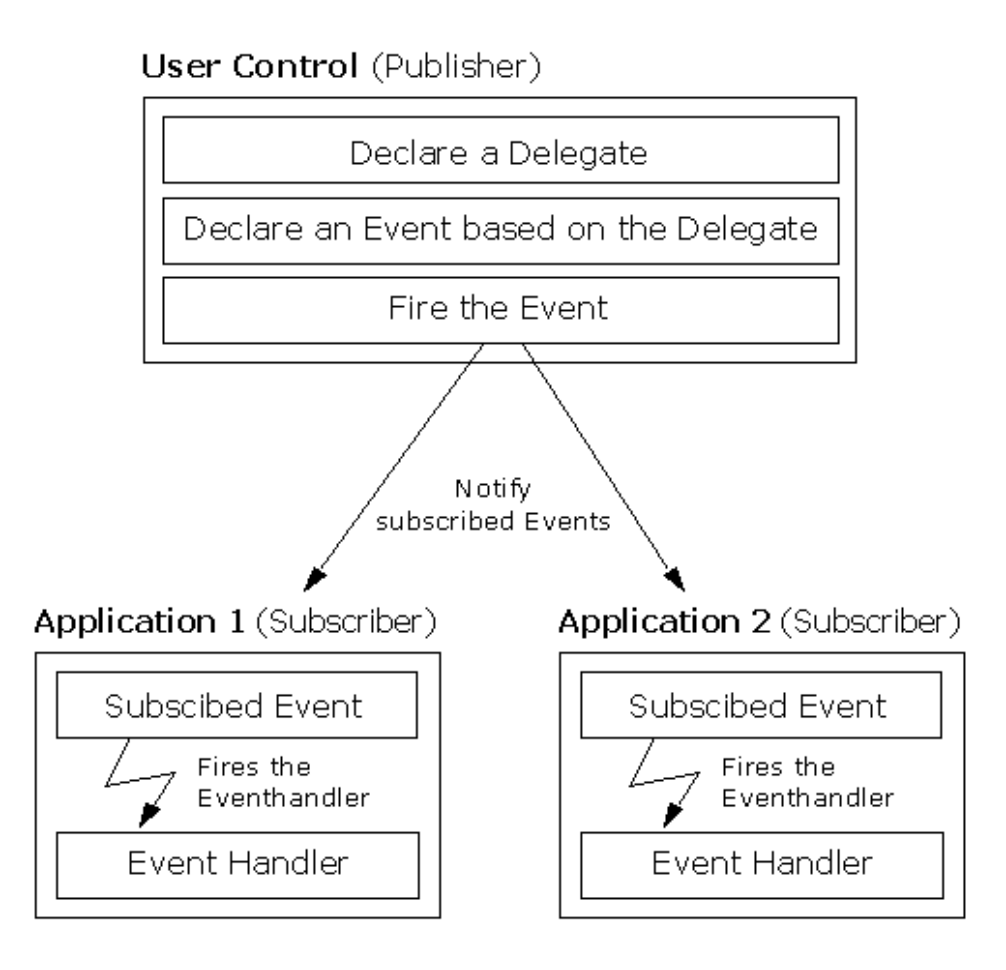
Događaji i delegati

- **Tip delegata**
  - opisuje kako izgleda postupak (za npr. obradu događaja)
  - objekt ovog razreda ne sadrži podatke, već reference na jedan ili više postupaka (slično pokazivaču na funkciju u C/C++)
  - postupci se registriraju na listi poziva delegata objekta
    - Pridruživanje jednog postupka delegatu =
    - Dodavanje postupka delegatu +=
    - Uklanjanje nekog postupka iz delegata −=
- **Događaj je delegat kojem su dozvoljene samo operacije += i -=**
- ***Event multicasting***
  - može postojati više postupaka za obradu istog događaja

cc BY NC SA

Vlastiti događaji

- **Događaj ne mora nužno biti vezan uz grafičko sučelje**
- **Model izdavač-pretplatnik (publisher-subscriber)**
  - Objekt “objavljuje” događaj, a aplikacija se na njega “pretplaćuje”



Definiranje vlastitog događaja

- **Primjer: 🗁Windows \ WeakReference \ EventSource.cs**
  - Vlastiti razred definira događaj *CustomEvent*
  - Postupci za obradu ovog događaja pohranjeni u delegatu tipa *EventHandler*

    public delegate void EventHandler(object sender, EventArgs e);

    → pretplatnik mora biti void postupak s parametrima *object* i *EventArgs*
  - „Izazivanje” događaja nalik pozivu postupka
    - prethodno potrebno provjeriti ima li pridruženih rukovatelja događajem

```
public class EventSource {
    public event EventHandler CustomEvent;
    public void RaiseEvent() {
      Console.WriteLine("Source is raising event...");
      if (CustomEvent != null) {
        CustomEvent(this, EventArgs.Empty);
      }
    }
  }
```

BY NC SA

Obrada vlastitog događaja

- **Primjer: 🗁Windows \ WeakReference \ CustomEventListener.cs**

```
public class CustomEventHandler  {
    public CustomEventHandler(EventSource source) {
       ...
      source.CustomEvent += Source_CustomEvent;
    }

    private void Source_CustomEvent(object sender, EventArgs e) {
      Console.WriteLine("CustomEventHandler received event.");
    }
```

BY NC SA

Događaji i zaostale reference

- **Pretplaćivanje na događaje često može biti uzrok curenja memorije (memory leak)**
  - Objekt živi dok na njega postoji barem jedna referenca
  - Izdavač događaja čuva referencu na pretplatnika
  - Što ako je životni vijek izdavača znatno dulji od pretplatnika?
  - Primjer: 🗁Windows \ WeakReference \ Program.cs (StrongReference1)

```
EventSource source = new EventSource();
CustomEventHandler listener = new CustomEventHandler(source);
source.RaiseEvent(); //listener obradio događaj

listener = null;
TriggerGC(); //GC ne može očistiti custom listenera

source.RaiseEvent(); //listener obrađuje i ovaj događaj iako logički
                     // više ne bi trebao postojati
```

CC BY NC SA

Izbjegavanje zaostalih referenci na pretplatnika

- **Rješenje prethodnog problema je otkazati pretplatu na događaj**
  - Primjer: 🗁Windows \ WeakReference \ Program.cs (StrongReference2)

```
EventSource source = new EventSource();
CustomEventHandler listener = new CustomEventHandler(source);
source.RaiseEvent(); //listener obradio događaj
listener.CancelListening();
listener = null;
TriggerGC(); //GC briše custom listenera
source.RaiseEvent(); //nema više pretplatnika
```

  - Primjer: 🗁Windows \ WeakReference \ CustomEventHandler.cs

```
public class CustomEventHandler {
    public void CancelListening() {
      source.CustomEvent -= Source_CustomEvent;
    }
```

CC BY NC SA

Slaba referenca (Weak reference)

- **Ugrađeni razred WeakEventManager**
  - Omogućava stvaranje reference prema nekom objektu, ali omogućava sakupljaču smeća da obriše referencirani objekt.
- **Iz perspektive izdavača ništa se ne mijenja**
- **Pretplatnik umjesto += i -= koristi AddHandler i RemoveHandler**
  - Primjer: 🗁Windows \ WeakReference \ WeakReferenceEventHandler

```
 public class WeakReferenceEventHandler {
    EventSource source;
    public WeakReferenceEventHandler(EventSource source) {
      this.source = source;
      WeakEventManager<EventSource, EventArgs>.AddHandler(source,
nameof(EventSource.CustomEvent), Source_CustomEvent);
    }

    public void CancelListening() {
      WeakEventManager<EventSource, EventArgs>.RemoveHandler(source,
nameof(EventSource.CustomEvent), Source_CustomEvent);
    }
```

BY NC SA

Višenitnost, paralelnost i asinkronost u C#-u

- **Nit ili dretva (eng. *thread*)**
  - Osnovne jedinice u kojima OS raspodjeljuje procesorsko vrijeme
  - Višenitnost (*multithreading*) => Više niti može se izvoditi unutar jednog procesa. Višenitni program simulira istovremeno izvođenje različitih niti
- **C# i višenitnost**
  - Asinkrono pozivanje postupaka
    - delegat ima mogućnost asinkronog poziva (*BeginInvoke*, *EndInvoke*)
  - Kontrola *BackgroundWorker*
  - Stavljanje postupka u postojeći red niti (razred *ThreadPool*)
  - Stvaranje nove niti (razred *Thread*)
  - TPL (eng. *Task Parallel Library*)
    - Stvaranje jednog ili više zadataka (razred *Task*)
    - Paralelno izvršavanje dijelova koda (razred *Parallel*)
    - Paralelni LINQ upiti (PLINQ)
  - C# 5.0 / .NET 4.5: korištenje asinkronih verzija postupaka (ako takvi postoje za određenu namjenu) + ključne riječi *async* i *await*

CC BY NC SA

Primjer potrebe za paralelnim izvršavanjem

- **Primjer: 🗁 Windows \ AsyncPiCalculator \ PiCalculator.cs**
  - Izračun broja π i istovremeni prikaza statusa.
  - Za provjeru prekida postupka koristi se zastavica (boolean varijabla *abort*)
    - Može se koristiti struktura *CancellationToken*
  - Podizanje vlastitog događaja nakon svakog koraka izračuna

```
public delegate void ShowProgressDelegate(string pi, int digitsSoFar);
```

```
public class PiCalculator {
  public event ShowProgressDelegate ProgressChanged;
  ...
  public string CalcPi(int digits) {
    abort = false;
    StringBuilder pi = new StringBuilder("3.", digits + 2);
    for (int i = 0; i < digits && !abort; i += 9) {
        ...
        //prikaz trenutnog stanja podizanjem vlastitog događaja
        ProgressChanged(pi.ToString(), i + digitCount);
    ...
    return pi.ToString();
```

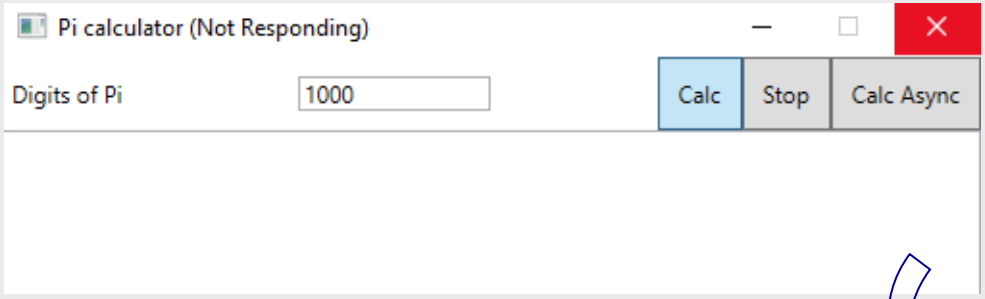
BY NC SA

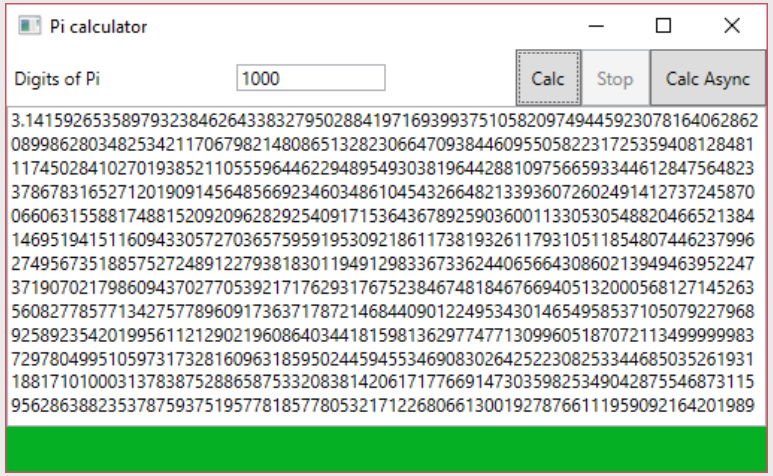
Problem jednonitnog izvršavanja

- **Primjer: 🗁 Windows \ AsyncPiCalculator \ MainWindow.xaml.cs**
  - Instancira se postupak za izračun broja pi i pretplaćuje se na događaj promjene statusa
  - Izračun započinje na istoj niti

```
void btnCalc_Click(object sender, EventArgs e) {
  ...
  int digitsToCalc = (int)digitsUpDown.Value;
  piCalculator = new PiCalculator();
  piCalculator.ProgressChanged += ShowProgress;
  string pi = piCalculator.CalcPi(digitsToCalc);
```

  - Problem: Forma ne reagira dok postupak izračuna ne završi





Stvaranje zadatka u pozadini

- **Koriste se razredi *Task* i/ili *Task<TResult>***
  - Za definiranje postupka u pozadini bez povratne vrijednosti (ili tipa *TResult*)
  - Konstruktor razreda *Task* očekuje
    - Delegat tipa *Action* ili *Action<object>* : *void* postupak bez argumenata ili s jednim argumentom tipa *object*
  - Konstruktor razreda *Task<TResult>* očekuje
    - Delegat tipa *Func<TResult>* ili *Func<object, TResult>* : postupak bez argumenata (ili s argumentom tipa *object)* koji vraća objekt tipa *TResult*
  - Ostali (opcionalni) parametri konstruktora uključuju argument koji se prosljeđuje postupku te razred za signalizaciju otkazivanja zadatka
  - Umjesto eksplicitno definiranih delegata uobičajeno je koristiti lambda izraze
- **Primjer: 🗁 Windows \ AsyncPiCalculator \ PiCalculator.cs**

```
public Task<string> CalcPiAsync(int digitsToCalc){
     Func<string> calcFunc = () => {
       string pi = CalcPi(digitsToCalc);
       return pi;
     };
     Task<string> task = Task.Run(calcFunc);
     return task;
  }
```

BY NC SA

Čekanje zadataka i asinkroni postupci

- ***await* + postupak koji vraća Task ili Task<TResult>**
  - Pozivatelj čeka na dovršetak zadatka, ali se istovremeno omogućava formi da reagira na druge događaje
  - Kôd iza await nastavlja se izvršavati tek nakon dovršetka zadatka
  - *await* + zadatak tipa Task<TResult> vraća rezultat tipa TResult
  - Postupak u kojem se koristi *await* mora se označiti s *async*
    - *Napomena: Povratna vrijednost iz postupka oblika async Task<TResult> je tipa TResult.*
  - Rukovatelji događaja mogu se označiti s *async*
- **Primjer: 🗁 Windows \ AsyncPiCalculator \ MainWindow.xaml.cs**
  - Izračun pokrenut kao zadatak (*Task*) (vidi prethodni slajd)

```
private async void btnCalcAsync_Click(...) {
  ...
  piCalculator = new PiCalculator();
  piCalculator.ProgressChanged += ShowProgress;
  string pi = await piCalculator.CalcPiAsync(digitsToCalc);
  ...
```

CC BY NC SA

Pristup kontrolama korisničkog sučelja iz drugih niti

- **Promjena vrijednosti ili stanja kontrole u grafičkom sučelju treba biti izvršena iz niti koja je stvorila tu kontrolu**

- **Koristi se postupak Dispatcher.Invoke koji sinkronizirano izvršava akciju na grafičkoj niti.**
  - Primjer: 🗁 Windows \ AsyncPiCalculator \ MainWindow.xaml.cs

```
private void ShowProgress(string pi, int digitsSoFar) {
      Dispatcher.Invoke(() => {
       //rad s GUI kontrolama je sada siguran...
       tbPi.Text = pi;
       progressBar.Value = digitsSoFar;
     });
    }
```

BY NC SA

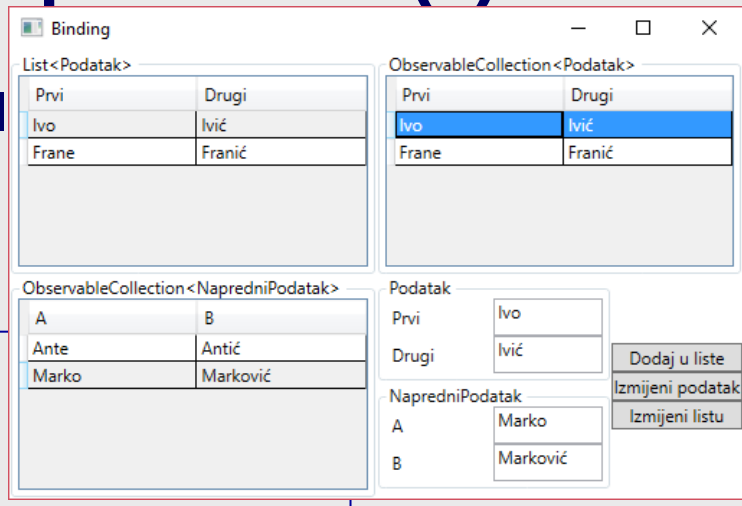
Povezivanje podataka

- ***Binding***
  - povezivanje svojstva (tekst, boja, vidljivost, …) komponente (labela, gumb, polje za unos …) sa svojstvom pridruženog objekta
  - povezivanje izvora podataka neke kontrole (lista, …) s nekom kolekcijom
- ***DataContext* – pridruženi objekt**
  - svaka grafička komponenta u WPF-u može imati svoj *DataContext*
  - uobičajeno se cijelom prozoru (kontroli) pridružuje objekt sa svojstvima koja se kasnije pridružuju svojstvima komponenti unutar prozora
    - obično se takav objekt naziva prezentacijski model (view model)
    - slično modelu u MVC web-aplikaciji
    - Primjer: 🗁 Windows \ Binding \ MainWindow.xaml
      - Nakon inicijalizacije WPF forme, poziva se konstruktor razreda ViewModel

```
<Window x:Class="Binding.MainWindow" ...
        xmlns:local="clr-namespace:Binding">
  <Window.DataContext>
    <local:ViewModel/>
  </Window.DataContext>
...
```

CC BY NC SA

Primjer forme s povezanim podacima (1)



- **Primjer: 🗁 …\Binding\MainWindow.xaml**
  - Tri komponente tipa *DataGrid* povezane s 3 različita izvora podataka

```
<DataGrid  ...
 ItemsSource="{Binding PodaciList}" >
   <DataGrid.Columns>
       <DataGridTextColumn Binding="{Binding Prvi}"
            Header="Prvi" Width="*"/>
       <DataGridTextColumn Binding="{Binding Drugi}"
            Header="Drugi" Width="*"/>
   </DataGrid.Columns>
</DataGrid>
```

  - Svakom DataGridu stupci povezani na odgovarajuće svojstvo

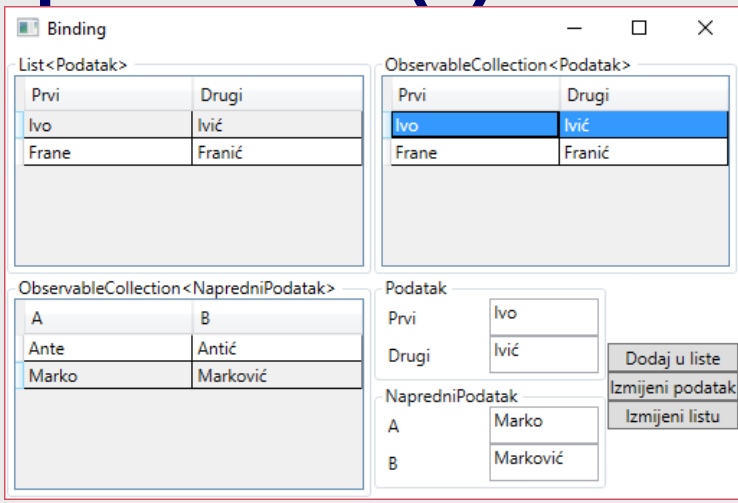
```
public class Podatak {
  public string Prvi { get; set; }
  public string Drugi { get; set; }
}
```

```
public class ViewModel{
   public List<Podatak> PodaciList
        { get; set; }
   ...
```

CC BY NC SA

Primjer forme s povezanim podacima (2)

- **Primjer: 🗁 …\Binding\MainWindow.xaml**
  - Polja za unos teksta povezana na trenutni element određenog DataGrida




```
<Label ... Content="Prvi"/>
<TextBox … Text="{Binding ElementName=Grid2, Path=SelectedItem.Prvi}"/>
<Label ... Content="Drugi"/>
<TextBox … Text="{Binding ElementName=Grid2, Path=SelectedItem.Drugi}"/>
```

```
public class Podatak {
  public string Prvi { get; set; }
  public string Drugi { get; set; }
}
```

Naredbe umjesto obrade događaja

- **Gumbima pridružena „naredba”**
  - Primjer: 🗁 Windows \ Binding\ MainWindow.xaml

```
<StackPanel DockPanel.Dock="Right">
   <Button Content="Dodaj u liste" Command="{Binding Dodaj}" />
   <Button Content="Izmijeni podatak" Command="{Binding Azuriraj}" />
   <Button Content="Izmijeni listu"
       Command="{Binding AzurirajListu}"  />
</StackPanel>
```

  - Opcionalno naredba može imati jedan parametar (ne koristi se u primjeru)
- **Naredba je svojstvo prezentacijskog modela koje implementira sučelje *ICommand***

```
public class ViewModel {
private ICommand dodaj;
    public ICommand Dodaj {
      get {
          ...
          return dodaj;
```

Implementacija sučelja ICommand

- **Uobičajena implementacija parametrizirana akcijom koju treba izvršiti i predikatom koji određuje može li se akcija izvršiti**
  - Primjer: 🗁 Windows \ Binding \ RelayCommand.cs

```
public class RelayCommand : ICommand {
    readonly Action<object> execute;
    readonly Predicate<object> canExecute;
    public RelayCommand(Action<object> execute):this(execute, null){}
    public RelayCommand(Action<object> execute,
                        Predicate<object> canExecute) {
      this.execute = execute; this.canExecute = canExecute;
    }
    public bool CanExecute(object parameter) {
      return canExecute == null ? true : canExecute(parameter);
    }
    public event EventHandler CanExecuteChanged {
      add { CommandManager.RequerySuggested += value; }
      remove { CommandManager.RequerySuggested -= value; }
    }
    public void Execute(object parameter) { execute(parameter); }
```

BY NC SA

Korištenje implementacije sučelja ICommand

- **Instancira se samo jedan primjerak *RelayCommand* po naredbi**
  - Primjer: 🗁 Windows \ Binding \ ViewModel.cs

```
private ICommand dodaj;
public ICommand Dodaj {
    get {
      if (dodaj == null)
         dodaj = new RelayCommand(DodajPodatak);
      return dodaj;
    }
  }
private void DodajPodatak(object obj) {
    Podatak p = new Podatak() {
      Prvi = "Mate",
      Drugi = "Matić"
    };
    PodaciList.Add(p);
    PodaciObservable.Add(p);
      ...
```

CC BY NC SA

Osvježavanje povezivanja

- **U prikazanom primjeru promjena povezane vrijednosti na formi automatski mijenja sadržaj na izvoru sadržaj komponenti povezanih na isti podataka**
  - Promjena podataka u kôdu ne mora ažurirati povezane komponente!
- **Primjer: 🗁 Windows \ Binding**
  - *List<Podatak>* - dodavanje novih elemenata u listu ili ažuriranje elementa ne propagira promjene na formu
  - *ObservableCollection<Podatak>* - dodavanje elemenata u listu ili promjena reference za neku poziciju u listi propagira promjene na formu (događaj *CollectionChanged*)
  - *ObservableCollection<NapredniPodatak>* - isto kao i *ObservableCollection<Podatak>* + propagira promjenu vrijednosti svojstva
    - *NapredniPodatak : INotifyPropertyChanged*
    - događaj *PropertyChanged* prilikom promjene svojstva objekta

BY NC SA

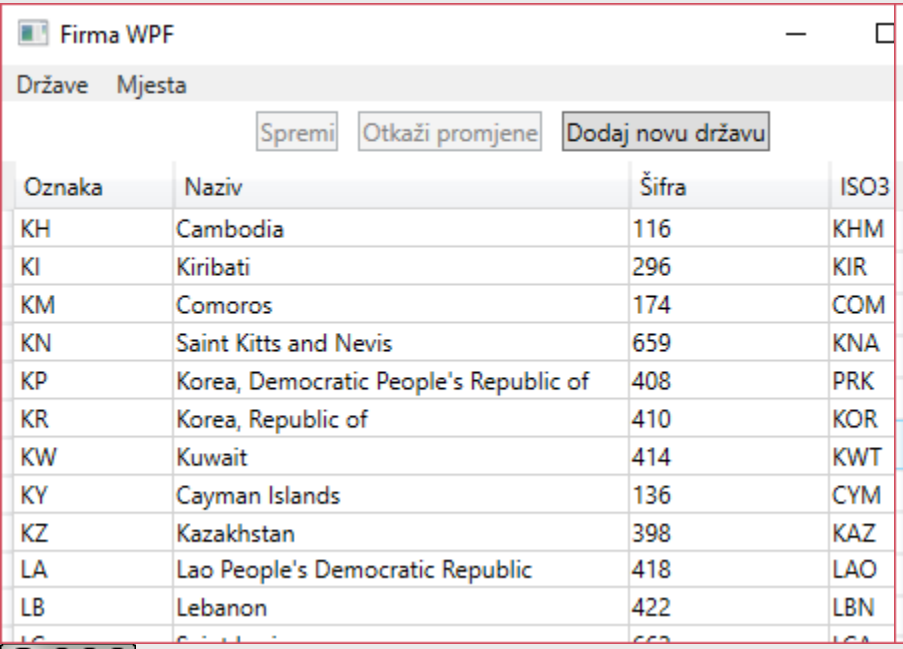
Smjer i osvježavanje povezivanja

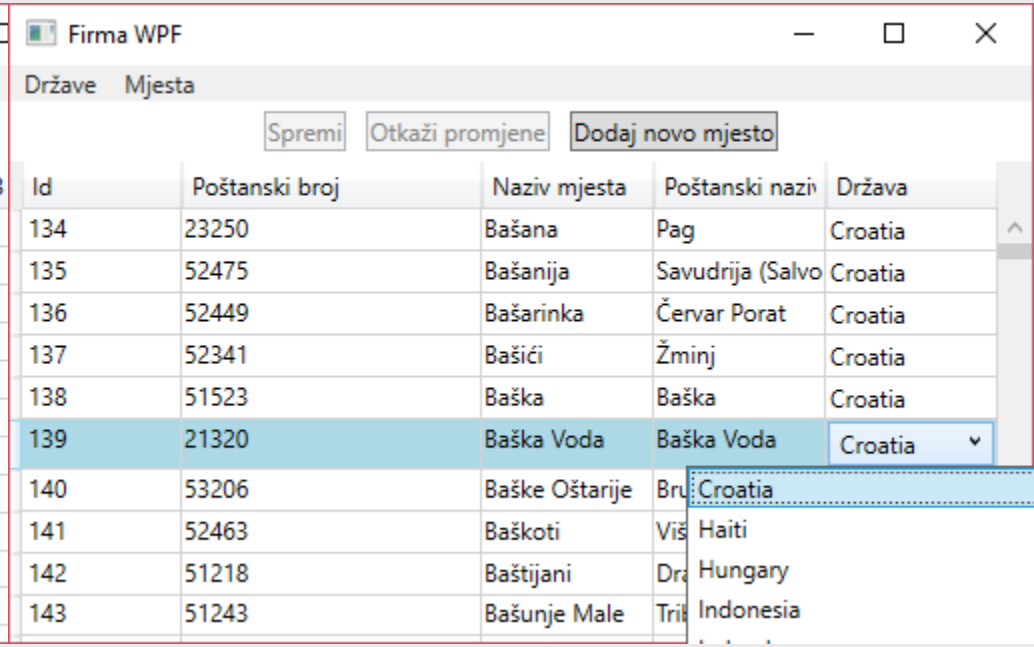
- ***TwoWay***
  - promjena podatka na izvoru ažurira stanje povezane komponente i obrnuto
- ***OneWay***
  - promjena vrijednosti podatka na izvoru propagira se prema povezanoj komponenti
  - Koristi se kad su komponente samo za čitanje/prikaz
- ***OneWayToSource***
  - OneWay ali u suprotnom smjeru.
- ***OneTime***
  - vrijednost za izvora se pridružuje komponenti jednokratno prilikom pokretanja aplikacije ili prilikom promjene konteksta
  - npr. za naziv neke labele, stupca, …
  - jednostavnije varijanta OneWay povezivanja s boljim performansama
- ***Default***
  - ponašanje ovisno o pojedinoj komponenti (npr. tekstualna polja, checkbox i slične komponente za unos podataka koriste TwoWay, ostale komponente OneWay)

cc BY NC SA

Primjer upotrebe prikazanih koncepata na primjeru popisa država

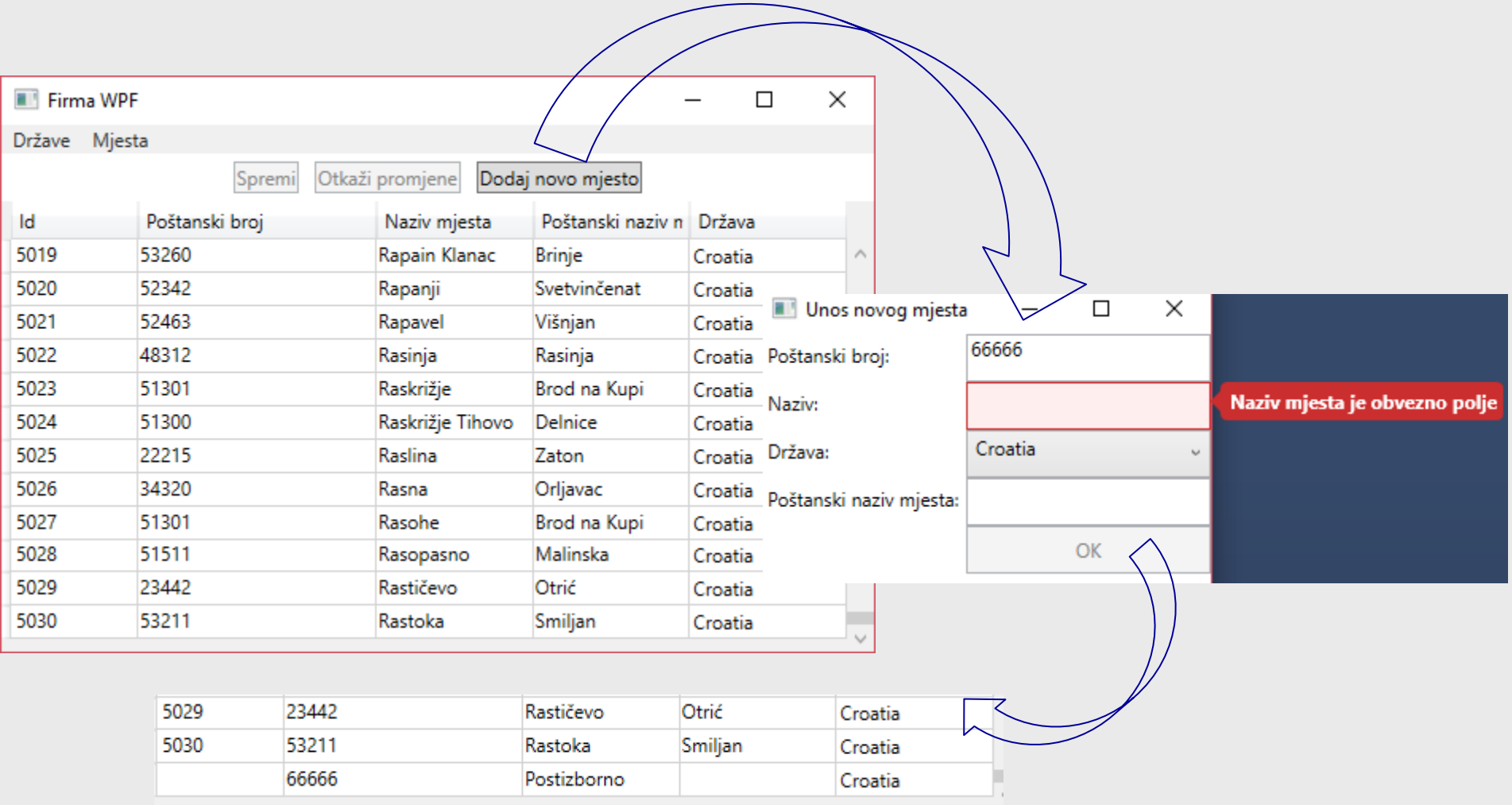
- **Primjer: 🗁 Windows \ Binding**
  - Forme prikazuje popis država/mjesta te omogućava ažuriranje i brisanje podataka
  - Gumbi za spremanje i otkazivanje promjena omogućavaju se samo ako promjena postoji i ako je objekt ispravan
  - Unos nove države/mjesta izveden u modalnom dijalogu





Prijenos parametara među formama (1)

- Forma poziva drugu formu i opcionalno prenosi podatke pozivajućoj
- Izmjene na drugoj formi koriste se za ažuriranje na pozivajućoj formi



CC BY NC SA

Prijenos parametara među formama (2)

- **Prijenos parametara iz pozvane forme u pozivajuću**
  - Primjer: 🗁 Windows \ Firma.Wpf \ ViewModels \ MjestaViewModel
  - Klik na gumb izvršava narednu koja stvara novu formu i pokazuje je kao modalni dijalog

```
private void OpenAddForm() {
     var form = new Views.NovoMjesto();
     bool? ok = form.ShowDialog();
     if (ok.HasValue && ok.Value) {
       Mjesto mjesto = form.Mjesto;
       Items.Add(mjesto);
     }
   }
```

  - Ako je rezultat otvaranja dijaloga istinit (bit će ako se klikne OK), može se očitati vrijednost svojstva Mjesto
    - Ako se forma zatvori drugačiji vrijednosti se ne dohvaćaju (rezultat je null ili false)

CC BY NC SA

Prijenos parametara među formama (3)

- **Pozvana forma definiran svojstvo s podatkom kojeg treba vratiti te obrađuje klik na gumb tako da za rezultat dijaloga vrati istinu**
  - Primjer: 🗁 Windows \ Firma.Wpf \ Views \ NovoMjesto.xaml.cs

```
public partial class NovoMjesto : Window {
  ...
  public Mjesto Mjesto {
      get {
        return DataContext as Mjesto;
      }
  }
  private void CloseForm(object sender, RoutedEventArgs e)
    {
      this.DialogResult = true;
      this.Close();
    }
```

  - Na sličan način preko svojstva se može izvršiti prijenos parametara u pozvanu formu

BY NC SA

Forma za unos nove države/mjesta

- **Primjer: 🗁 Windows \ Firma.Wpf \ Views \ NovaDrzava.xaml**
- **Ključne karakteristike**
  - Forma povezana (DataContext) na objekt tip Drzava
  - Labele i tekstualna polja za unos
  - Prikazuju se validacijske pogreške (ako ih ima)
  - Onemogućava se klik na gumb ok ako postoji validacijska pogreška

- **Primjer: 🗁 Windows \ Firma.Wpf \ Views \ NovoMjesto.xaml**
  - Sve što vrijedi za unos nove države + države prikazane u padajućoj listi

- **Za ispravno funkcioniranje formi potrebno implementirati sučelja INotifyPropertyChanged i validaciju**
  - Zajednička funkcionalnost izdvojena u razrede ObservableObject i EditableObject

BY NC SA

Vlastiti razred ObservableObject

- **Bazni razred za sve razrede koji se koriste za povezivanje**
  - Obavještava pretplatnike u slučaju promjene vrijednosti nekog svojstva
    - *CallerMemberName* dohvaća naziv postupka pozivatelja
  - Primjer: 🗁 Windows \ Firma.Wpf \ Core \ ObservableObject.cs

```
public abstract class ObservableObject : INotifyPropertyChanged{
    public event PropertyChangedEventHandler PropertyChanged;
    protected void RaisePropertyChanged(
            [CallerMemberName] string propertyName = null) {
       if (PropertyChanged != null)
          PropertyChanged(this,
             new PropertyChangedEventArgs(propertyName));
    }
    protected virtual void SetProperty<T>(ref T property,
       T value, [CallerMemberName] string propertyName = null) {
      if (!Equals(property, value)) {
        property = value;
        RaisePropertyChanged(propertyName);
      } ...
```

Vlastiti razred *EditableObject*

- **Bazni razred za sve razrede čije vrijednosti unosi korisnik**
  - Primjer: 🗁 Windows \ Firma.Wpf \ Core \ EditableObject.cs

```
public abstract class EditableObject :
       ObservableObject,
       INotifyDataErrorInfo,
       IEditableObject,
       IRevertibleChangeTracking  {

       ... Proučiti programski kod razreda ...
```

  - *IEditableObject*
    - Omogućava stvaranje kopije podataka prije početka ažuriranja i vraćanja u slučaju odustajanja
  - *INotifyDataErrorInfo*
    - Standardno sučelje za validaciju
      - GetErrors(propertyName), svojstvo HasErrors, događaja ErrorsChanged
  - *IRevertibleChangeTracking*
    - Sučelje kojim bi se objektu poslao zahtjev da odbaci promjene

CC BY NC SA

yield return

- **Koristi se unutar postupaka koji predstavljaju iteratore**
- **Vraća jedan podatak te nakon zahtjeva za sljedećim članom nastavlja dalje sa svojim izvršavanjem**
  - Prednost: nije potrebno unaprijed stvoriti iterabilnu kolekciju kao rezultat
    - Elementi se vraćaju jedan po jedan dok pozivatelj (unutar foreacha) ne odustane ili dok se kolekcija ne iscrpi
  - Primjer: 🗁 Windows \ Firma.Wpf \ Core \ EditableObject.cs

```
protected Dictionary<string, List<string>> errors = new
Dictionary<string, List<string>>();

IEnumerable<KeyValuePair<string, string>> GetAllErrors() {
   foreach (var entry in errors) {
      foreach (var error in entry.Value) {
        yield return
           new KeyValuePair<string, string>(entry.Key, error);
      }
   }
}
```

BY NC SA

Pohrana promjena

- **Sve obrisane podatke iz liste, obrisati iz baze podataka**
- **Sva ažuriranja propagirati u bazu podataka**
- **Sve nove podatke potrebno je pohraniti u bazu podataka**
  - Ako postoje automatski izračunata polja treba ih dohvatiti i kopirati u povezani objekt (npr. Id mjesta)
- **ObservableCollection podiže događaje prilikom promjene kolekcije, ali ne vodi evidenciju što se promijenilo**
  - → potrebno napisati vlastite implementacije

Id	Poštanski broj	Naziv mjesta	Poštanski nazi	Država
5021	52463	Rapavel	Višnjan	Croatia
5022	48312	Rasinja	Rasinja	Croatia
5023	51301	Raskrižje	Brod na Kupi	Croatia
5024	51300	Raskrižje Tihovc	Delnice	Croatia
5025	22215	Raslina	Zaton	Croatia
5026	34320	Rasna	Orljavac	Croatia
5027	51301	Rasohe	Brod na Kupi	Croatia
5028	51511	Rasopasno	Malinska	Croatia
5029	23442	Rastičevo	Otrić	Croatia
5030	53211	Rastoka	Smiljan	Croatia
	66666	Postizborno		Croatia

5030	53211	Rastoka	Smiljan	Croatia
35344	66666	Postizborno		Croatia

CC BY NC SA

CustomObservableCollection

- **Primjer: 🗁 … \ Firma.Wpf \ Core \ CustomObservableCollection.cs**

```
public class CustomObservableCollection<T> :
   ObservableCollection<T>, IRevertibleChangeTracking
   where T : INotifyPropertyChanged, IRevertibleChangeTracking {

 private HashSet<T> removedItems = new HashSet<T>();
 private HashSet<T> newItems = new HashSet<T>();
 private HashSet<T> changedItems = new HashSet<T>();
```

- Nadjačava postupak *OnCollectionChanged* kako bi evidentirao nove i obrisane elemente iz kolekcije
- Za svaki postojeći element stvara se pretplata na promjene kako bi se mogao dodati u skup promijenjenih elemenata
- Postupci za odbacivanje i potvrđivanje promjena

CC BY NC SA

EditableCollection

- **Primjer: 🗁 … \ Firma.Wpf \ Core \ EditableCollection.cs**

```
public class EditableCollection<T> : EditableObject
                                     where T:EditableObject {
    private CustomObservableCollection<T> items;
    public CustomObservableCollection<T> Items {
      get { return items; }
      set { SetProperty(ref items, value); }
    }
```

- Ugnježđuje CustomEditableCollection i predstavlja EditableObject koji se može povezati na forme
- Nadjačava postupke za prihvat i odbacivanje promjena te provjeru pogrešaka
  - Osim same promjene liste moguće je da se promijenio i pojedini element, odnosno da ima validacijsku pogrešku

CC BY NC SA

Autentifikacija i autorizacija korisnika

2017/18.12

Provjera autentičnosti

- **engl. *authentification***
  - provjera autentičnosti (izvornosti, valjanosti) korisnika, utvrđivanje (dokazivanje) identiteta korisnika
    - „autentičan = koji je izvoran, koji zaista potječe od onoga kome se pripisuje; istinit, nepatvoren, vjerodostojan”
      - Hrvatski jezični portal, http://hjp.znanje.hr/index.php?show=search_by_id&id=e11hXA%3D%3D
  - *„Autentifikacija* je utvrđivanje autentičnosti, tj. identiteta osobe koja pristupa podatcima”
    - Leksikografski zavod Miroslav Krleža http://www.enciklopedija.hr/Natuknica.aspx?ID=68380
- **Različiti mehanizmi**
  - Lozinka, pin, kartica, token, …
- **Cookie**
  - *int.* kratki, tekstualni skup informacija koje web stranice zapisuju na korisničko računalo prigodom posjećivanja; kolačić
    - Hrvatski jezični portal

BY NC SA

Autorizacija korisnika

- **engl. *authorization***
- **„Postupak provjere ima li korisnik, računalo ili program pravo pristupa nekomu računalnomu sustavu”**
  - Leksikografski zavod Miroslav Krleža
- **„Stjecanje prava korištenja sustava i podataka pohranjenih na njemu”**
  - Hrvatski jezični portal

BY NC SA

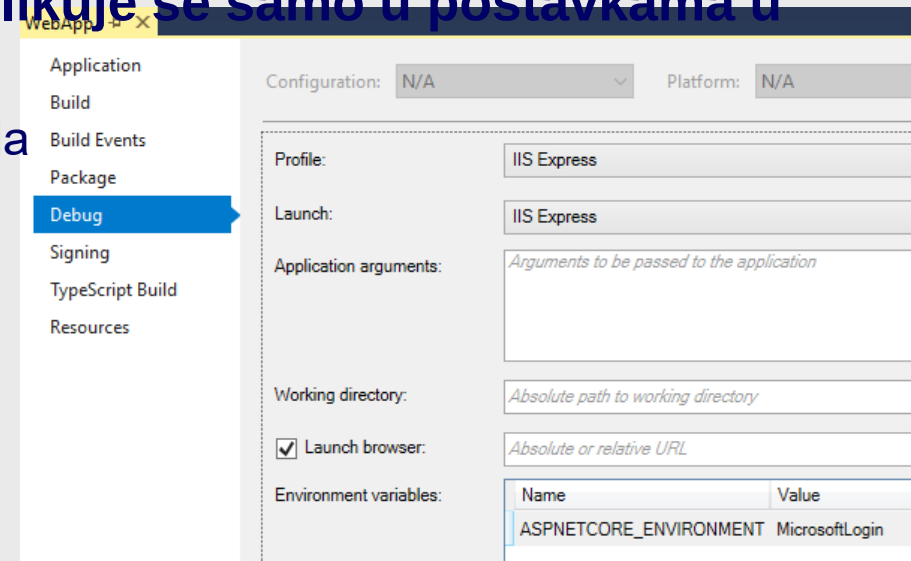
Ideja oglednog primjera (1)

- **U web-aplikaciju u projektu Security / WebApp ugrađena je prijava korisnika te postavke autorizacije pristupa pojedinim stranicama**
  - funkcionalnost aplikacije nalik prethodnim primjerima, ali dovršena toliko da se mogu demonstrirati mehanizmi autentifikacije i autorizacije
- **Početnu stranicu mogu vidjeti svi korisnici**
- **Stranicu s popisom artikala mogu vidjeti samo prijavljeni korisnici**
  - administratori vide poveznice za dodavanje, ažuriranje i brisanje artikla
- **Stranicu s popisom dokumenata vide samo prijavljeni korisnici**
  - svaki korisnik vidi samo svoje dokumente
  - administratori vide sve dokument
- **Izrađen trivijalni web servis s tri postupka različitih prava pristupa**
  - javno, za prijavljene korisnike, za administratore
- **„Obični" korisnici su svi partneri te se prijavljuju svojim OIB-om**
  - To je ujedno i lozinka za svrhu demonstracije primjera
- **Administratori zavedeni u tablici Korisnik**
  - Za potrebe demonstracije lozinka je čisti tekst, a u stvarnosti treba koristiti *hash + salt*.

BY NC SA

Ideja oglednog primjera (2)

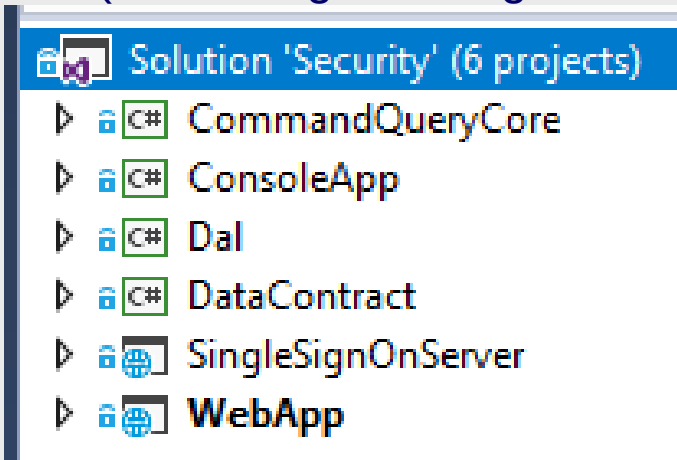
- **Demonstrirana tri mehanizma prijave korisnika**
  - jednostavna prijava korisničkim imenom i lozinkom
    - ne koriste se ASP.NET Core Identity
  - prijava korištenjem vanjskog pružatelja usluge provjere korisnika
    - prijava preko Microsoftovog računa (protokol OAuth)
    - za razliku od uobičajenih primjera ne povezuje se s lokalnim računom
  - prijave preko vlastitog servisa za jedinstvenu prijavu (engl. single sign on)
    - protokol OpenID Connect
    - implementacija pomoću programske biblioteke IdentityServer4
- **Programski kod web-aplikacije razlikuje se samo u postavkama u datoteci Startup.cs**
  - Ponašanje ovisno o varijabli okruženja ASPNETCORE_ENVIRONMENT
    - CookieDemo
    - MicrosoftLogin
    - OpenIdConnect



BY NC SA

Organizacija oglednog primjera

- **Rješenje podijeljeno u 6 projekata**
  - *WebApp* – ogledna web-aplikacija na kojoj se demonstrira autentifikacija i autorizacija korisnika
  - *SingleSignOnServer* – vlastiti server za jedinstvenu prijavu
  - *ConsoleApp* – konzolna aplikacija koja traži pristupni token od SSO servera kako bi pristupila zaštićenom web servisu u WebApp
  - 3 projekta tipa *ClassLibrary* za razdvajanje logike pristupa podacima od interakcijske logike web-aplikacije
    - pristup podacima se apstrahira i web-aplikacija ovisi o sučeljima kojim se definira interakcija sa slojem pristupa podacima
    - ne predstavlja klasično uslojavanje aplikacije, jer nema izdvojenog poslovnog sloja s posebnim objektnim modelom (osim onog nastalog ORM-om), ali omogućava lakše održavanje projekta i omogućava eventualnu zamjenu načina pristupa bazi podataka
      - Više na slajdovima koji slijede



BY NC SA

Motivacija za reorganizaciju koda

- **Primjer: 🗁 Web\Firma.Mvc\...\ArtiklController.cs**
- **U navedenom primjeru upravljač vrši prihvat ulaznih argumenata, dohvat podataka i pripremu modela**
  - Problem je u načinu dohvatu podataka
    - umjesto na „što”, upravljač u prethodnom primjeru fokusiran na „kako” (slaganje EF upita)
    - „prepametan” upravljač → debeli klijent
  - Što ako umjesto EF-a treba koristiti neku drugu tehniku pristupa podacima?
    - Hoće li entiteti i dalje biti isti?
      - …i hoće li ih biti ako se ne koristi ORM?
    - Što ako su podaci agregirani iz više izvora?
    - …

```
public IActionResult Index(int page = 1, int sort = 1,
{
  int pagesize = appData.PageSize;
  var query = ctx.Artikl.AsNoTracking();
  int count = query.Count();

  var pagingInfo = new PagingInfo
  {
    CurrentPage = page,
    Sort = sort,
    Ascending = ascending,
    ItemsPerPage = pagesize,
    TotalItems = count
  };
  if (page < 1)
  {
    page = 1;
  }
  else if (page > pagingInfo.TotalPages)
  {
    return RedirectToAction(nameof(Index), new { page
  }

  System.Linq.Expressions.Expression<Func<Artikl, obje
  switch (sort)
  {
    case 1:
      orderSelector = a => a.SlikaArtikla; //ima smisl
      break;
    case 2:
      orderSelector = a => a.SifArtikla;
      break;
    case 3:
      orderSelector = a => a.NazArtikla;
      break;
    case 4:
      orderSelector = a => a.JedMjere;
      break;
    case 5:
      orderSelector = a => a.CijArtikla;
      break;
    case 6:
      orderSelector = a => a.ZastUsluga;
      break;
  }
  if (orderSelector != null)
  {
    query = ascending ?
            query.OrderBy(orderSelector) :
            query.OrderByDescending(orderSelector);
  }

  var artikli = query
                .Select(a => new ArtiklViewModel
                {
                  SifraArtikla = a.SifArtikla,
                  NazivArtikla = a.NazArtikla,
                  JedinicaMjere = a.JedMjere,
```

CC BY NC SA

Repozitoriji umjesto konkretnog ORM-a? (1)

- **Zamjenom ORM alata može se pretpostaviti da će i dalje postojati isti koncepti i entiteti u neznatno izmijenjenom obliku**
  - Entiteti su posljedica modela baze podataka
- **Konkretni objekti tipa DbSet<T> iz EF-a mogu se zamijeniti sučeljima s postupcima za CRUD operacije - *repozitoriji***
  - Kontekst iz EF-a se tako mijenja sa sučeljem koje koordinira više repozitorija i predstavlja jedinstveni kontekst pristupa podacima - *Unit of Work*
  - Ako se i ne koristi ORM alat, mogu postojati repozitoriji i Unit of Work
    - u tom slučaju entiteti su zamijenjeni razredima kojima se opisuju izlazni podaci iz postupaka
- **Upravljači tada ovise o sučeljima repozitorija, ali konkretna implementacija se umetne tehnikom *Dependeny Injection***
  - U ovom slučaju rješava samo dio problema
    - skriva način perzistencije objektnog modela u relacijsku bazu i djelomično olakšava testiranje programa
    - moguće napisati testnu implementaciju repozitorija

CC BY NC SA

Repozitoriji umjesto konkretnog ORM-a? (2)

- **Nije promijenjen način rukovanja repozitorijima iz upravljača**
  - Prethodno prikazani upravljač bi se neznatno promijenio – i dalje bi slagao upit, ali ovaj put nad nekim nepoznatim repozitorijem
  - Koji je tip povratne vrijednosti kod operacija čitanja podataka?
    - IQueryable<T> bi omogućio daljnje upite (filtriranje, sortiranje, …)
      - Iz čega je nastao taj IQueryable? Što sve podržava?
    - IEnumerable<T> - nije li možda upit već evaluiran, pa su svi podaci morali biti dovučeni u memoriju?
- **Što ako podaci nisu u bazi podataka ili su agregirani iz više izvora?**
  - Kako dodatno oblikovati upit po želji?
- **Bilo bi dobro kad bi upravljač pozivao jedan postupak koji bi mu vratio upravo one podatke koje treba**
  - Krivi smjer rješenja: proširiti repozitorije s nekoliko metoda koje primaju različite parametre
    - traži po nekoj vrijednosti, po kombinaciji vrijednosti, vrati samo dio složene po nekom kriteriju, …
    - stvara prevelike repozitorije koje nije lako implementirati i otežava održavanje i testiranje

BY NC SA

Odvajanje upita od naredbi

- **Command-query separation**
  - odvojena sučelja za čitanje podataka (upiti, engl. queries) od onih koji mijenjaju podatke (naredbe, engl. commands)
    - ovisno o rješenju mogu koristiti različita spremišta
    - u implementaciji i naredbe i upiti mogu koristiti repozitoriji
  - U primjerima koji slijedi bit će ilustriran samo dio vezan za upite
- **Više upita u istom sučelju narušava SOLID principe**
  - *Single Responsibility, Open/Closed i Interface Segregation Principle*

→ **Svaki upit predstavljen jednim sučeljem**

BY NC SA

Opisi upita

- **„Upit” specificira tip rezultata tog upita za neke ulazne podatke**
  - Upit ne mora imati argumente i može se opisati generičkim sučeljem
  - Primjer: 🗁 CommandQueryCore \ IQuery.cs

```
public interface IQuery<TResult> {}
```

- **Primjeri opisa upita 🗁 DataContact \ Queries \ …**
  - Opis upita koji treba vratiti broj artikala

```
public class ArtiklCountQuery : interface IQuery<int> {}
```

  - Opis upita koji vraća sliku nekog artikla određenog šifrom artikla

```
public class ArtiklImageQuery : IQuery<byte[]> {
    public int SifArtikla { get; set; }
}
```

  - Opis upita koji vraća podskup artikala (na nekoj stranici nekim redoslijedom)

```
public class ArtiklQuery : IQuery<IEnumerable<ArtiklDto>> {
    public int Sort { get; set; }
    public bool Ascending { get; set; }
    public int Page { get; set; }  public int PageSize { get; set; }
  }
```

CC BY NC SA

Rukovatelj upitom

- **Upit (engl. *query*) opisan razredom IQuery<TResult> bit će izvršen u nekom rukovatelju upita (engl. *query handler*)**
- **Sučeljem se standardizira kako rukovatelji upita izgledaju**
  - Može se definirati „obična” i asinkrona varijanta
  - Primjer: 🗁 CommandQueryCore \ IQueryHandler[Async].cs

```
public interface IQueryHandler<TQuery, TResult>
                                where TQuery : IQuery<TResult>
{
    TResult Handle(TQuery query);
}
```

```
public interface IQueryAsyncHandler<TQuery, TResult>
                                where TQuery : IQuery<TResult>
{
    Task<TResult> HandleAsync(TQuery query);
}
```

  - Upit predstavlja opis upita (ulazne podatke i povratnu vrijednost), a rukovatelj izvršava tako opisani upit

CC BY NC SA

Primjeri rukovatelja upitom (1)

- **Primjer sučelja 🗁 DataContact \ QueriesHandlers \ …**
  - Opisuje rukovanje upitom za dohvat broja artikala

```
public interface IArtiklCountQueryHandler :
                        IQueryAsyncHandler<ArtiklCountQuery, int> { }
```

- **Primjer implementacije 🗁 Dal \ …**

```
public class ArtiklCountQueryHandler : IArtiklCountQueryHandler {
    private readonly FirmaContext ctx;

    public ArtiklCountQueryHandler(FirmaContext ctx)
    {
      this.ctx = ctx;
    }
    public async Task<int> HandleAsync(ArtiklCountQuery query)
    {
      var count = await ctx.Artikl.CountAsync();
      return count;
    }
  }
```

CC BY NC SA

Primjeri rukovatelja upitom (2)

- **Primjer sučelja 🗁 DataContact \ QueriesHandlers \ …**
  - Opisuje rukovanje upitom za dohvat podskupa artikala u određenom poretku

```
public interface IArtiklQueryHandler :
        IQueryAsyncHandler<ArtiklQuery, IEnumerable<ArtiklDto>> { }
```

- **Implementacija 🗁 Dal \ ArtiklQueryHandler.cs**

```
public class ArtiklQueryHandler : IArtiklQueryHandler {
    ...
  async Task<IEnumerable<ArtiklDto>> HandleAsync(ArtiklQuery query){
     var dbQuery = ctx.Artikl.AsNoTracking();
     Expression<Func<Artikl, object>> orderSelector = null;
     switch (query.Sort) { ... }
     if (orderSelector != null) {
        dbQuery = query.Ascending ?
                    dbQuery.OrderBy(orderSelector) :
                    dbQuery.OrderByDescending(orderSelector);
     }
     var artikli = await dbQuery
                          .Select(a => new ArtiklDto ...
```

BY NC SA

Korištenje rukovatelja upitom iz upravljača

- Upravljač ovisi o sučelju, a konkretnu implementaciju rukovatelja upitom prima u konstruktoru preko DI-a
- Primjer sučelja 🗁 WebApp \ Controllers \ ArtiklController.cs

```
public class ArtiklController…
    public ArtiklController(IArtiklCountQueryHandler countHandler,
IArtiklQueryHandler handler, ..., ILogger<ArtiklController> logger){
      this.countHandler = countHandler;
      this.handler = handler;
      ...

    public async Task<IActionResult> (...) {
        int count = await countHandler.HandleAsync(
                                  new ArtiklCountQuery());
         ...
       ArtiklQuery query = new ArtiklQuery {
        Ascending = ascending, Page = page,
        PageSize = pagesize, Sort = sort };
      var artikli = await handler.HandleAsync(query);
```

CC BY NC SA

Postavljanje ovisnosti

- **Povezivanje za DI postavljano u razredu Startup web-aplikacije**
  - Primjer 🗁 WebApp \ Startup.cs

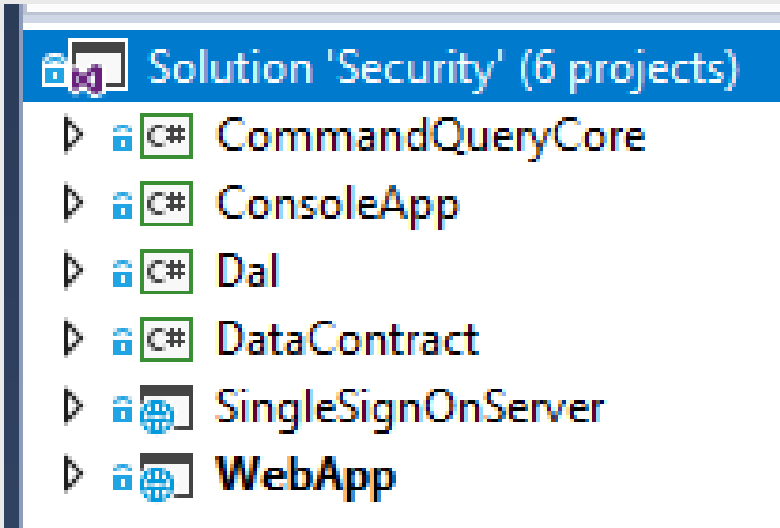
```
public class Startup {
  public void ConfigureServices(IServiceCollection services) {
      services.AddMvc();
      var appSection = Configuration.GetSection("AppSettings");
      services.Configure<AppSettings>(appSection);
      string connectionString = appSection["ConnectionString"];
      services.AddDbContext<FirmaContext>(
          options => options.UseSqlServer(connectionString));

     services.AddTransient<IArtiklQueryHandler,
                           ArtiklQueryHandler>();
     services.AddTransient<IArtiklCountQueryHandler,
                           ArtiklCountQueryHandler>();
    ...
```

  - Umjesto repetitivnog pisanja može se riješiti refleksijom tražeći i registrirajući sve klase koje implementiraju neki IQueryHandler<,> iz typeof(ArtiklQueryHandler).Assembly

CC BY NC SA

Međusobne ovisnosti projekata

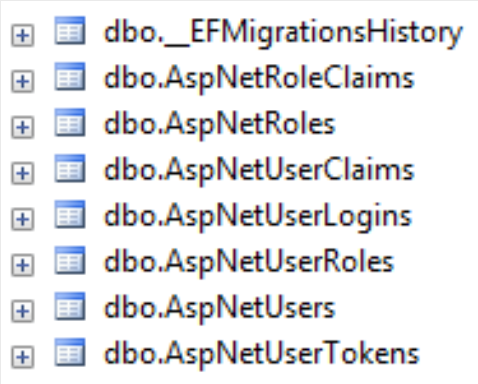
- **Smisao pojedinog projekta**
  - *CommandQueryCore* – generička sučelja za upit, naredbu i njihove rukovatelje
  - *DataContract* – opis svih upita koji se koriste u rješenju i pomoćni razreda koji služe za razmjenu podataka s pozivateljem
    - Data transfer objekti (Value object) – samo sadrže vrijednosti
  - *Dal* – implementacija postupaka iz DataContract
- **Nijedan element web-aplikacije (upravljači, pogledi, …) osim razreda Startup ne zna za projekt Dal i ovise samo o DataContract**
  - Startup.cs postavlja Dependency Injection pa stoga postoji referenca iz projekta WebApp na Dal



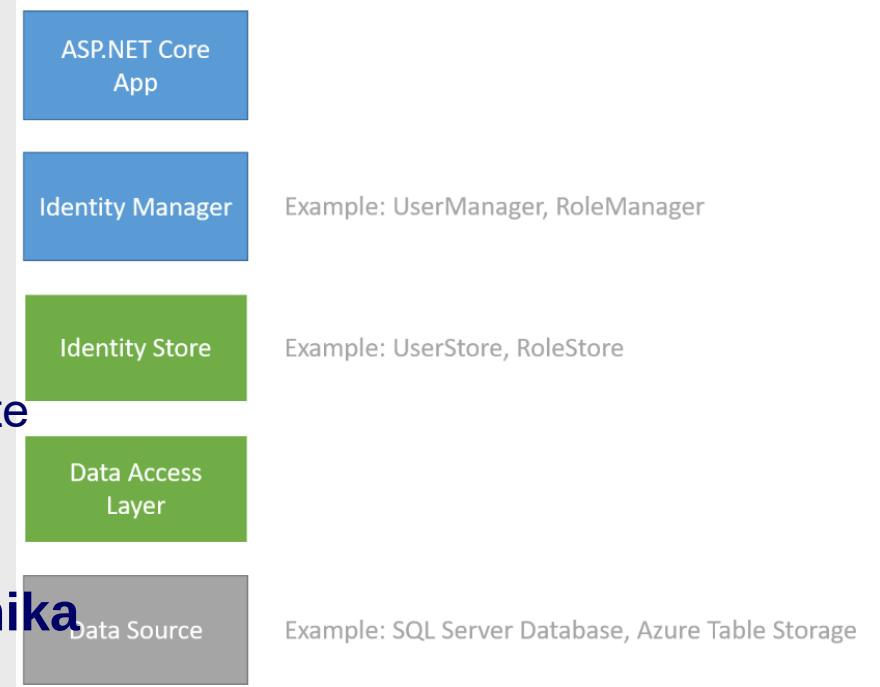
BY NC SA

ASP.NET Core i ugrađeni mehanizmi autentifikacije

- **ASP.NET Core Identity je programska biblioteka s ugrađenom funkcionalnošću prijave korisnika, registracije novih, opcijom oporavka zaboravljene lozinke i slično**
  - Gotovi pogledi i upravljači
  - Podrška za EF i neke druge pakete
  - Inicijalna implementacija uzrokuje stvaranje nekoliko specifičnih tablica naziva AspNet*



- **Moguće proširiti s dodatnim podacima ili napisati vlastitu implementaciju s proizvoljnim tablicama**
  - Kompleksno, zahtjeva implementaciju mnoštva postupaka propisanih sučeljem za manipulaciju korisnicima i rolama
    - zeleno i sivo označene komponente



- **Komplicirano rješenje u slučaju postojeće tablice s korisnicima ili jednostavne varijante prijave korisnika**

BY NC SA

Postavke autentifikacije i autorizacija zajedničke za sva tri primjera

- **U postavkama aplikacije uključiti autentifikacijske mehanizme**
  - Primjer: 🗁 WebApp \ Startup.cs

```
public class Startup {
  ...
   public void Configure(IApplicationBuilder app,
                                IHostingEnvironment env) {
      app.UseAuthentication();
      ...
```

  - Iznad akcija ili cijelih upravljača koje treba omogućiti samo prijavljenim korisnicima dodati atribut Authorize
    - Primjer: 🗁 WebApp \ Controllers \ ArtiklController.cs

```
[Authorize]
public class ArtiklController : Controller
```

CC BY NC SA

Jednostavna autentifikacija

ASPNETCORE_ENVIRONMENT=CookieDemo

Uključivanje jednostavne autentifikacije

- **Jednostavna autentifikacija se bazira na postojanju cookiea koji sadrži informacije o trenutnom korisniku**
  - Primjer: 🗁 WebApp \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
  services.AddAuthentication(
            CookieAuthenticationDefaults.AuthenticationScheme)
          .AddCookie(options => {
                      options.Cookie.Name = "RPPP-WebApp";});
```

  - U primjeru je promijenjen naziv cookiea, što nije bilo nužno

BY NC SA

Prijava korisnika (1)

- **Potrebno kreirati upravljač *Account* i akciju *Login* za prijavu korisnika**
  - Korisnik će automatski biti preusmjeren na tu stranicu pokušajem pristupa zaštićenom resursu
    - Stranica na koju se treba vratiti zapisana u parametru *returnUrl*
- **Prije potvrde prijave provjeriti model, a zatim ime i lozinku**
  - Primjer: 🗁 WebApp \ Controllers \ AccountController.cs

```
[HttpPost] [AllowAnonymous] [ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model,
                                       string returnUrl = null) {
      ViewData["ReturnUrl"] = returnUrl;
      if (ModelState.IsValid) {
        ValidateUserQuery query = new ValidateUserQuery {
          Username = model.Username, Password = model.Password
        };
        var user = await validateUser.HandleAsync(query);
        if (user != null) {
               ... Obavi prijavu korisnika ...
```

CC BY NC SA

Prijava korisnika (2)

- Nakon postavljanja željenih tvrdnji stvoriti objekt tipa ClaimIdentity i pokrenuti prijavu
- Primjer: 🗁 WebApp \ Controllers \ AccountController.cs

```
public async Task<IActionResult> Login(LoginViewModel model,
                                       string returnUrl = null)
...
  var claimsIdentity = new ClaimsIdentity(claims,
           CookieAuthenticationDefaults.AuthenticationScheme);
  var authProperties = new AuthenticationProperties {
            AllowRefresh = true, IsPersistent = false
  };
  await HttpContext.SignInAsync(
         CookieAuthenticationDefaults.AuthenticationScheme,
         new ClaimsPrincipal(claimsIdentity),
         authProperties);

  return RedirectToLocal(returnUrl);
}
```

BY NC SA

Claims

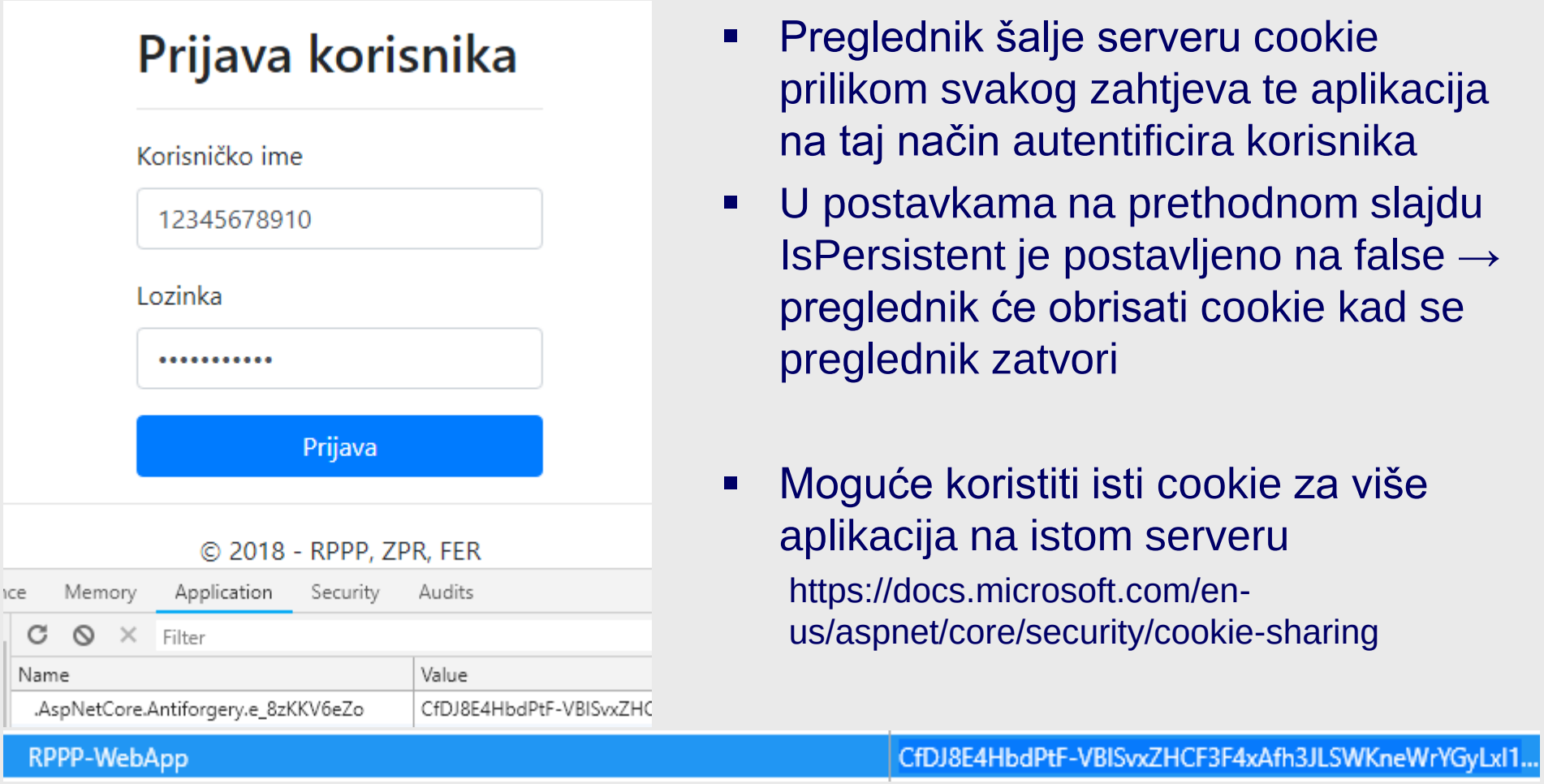
- **Korisnik opisan razredom *ClaimsIdentity* i načinom prijave**
- **Claim**
  - činjenica (tvrdnja) o nekom korisniku, vrijednost nekog korisničkog atributa
    - korisničko ime, pripadnost grupi, e-mail ili bilo što drugo što može koristiti potencijalnoj klijentskoj aplikaciji u kontekstu autorizacije
  - Tvrdnje opisane parovima naziv, vrijednosti
    - Mogu se definirati vlastite tvrdnje pored onih u razredu ClaimTypes
- **Primjer: 🗁 WebApp \ Controllers \ AccountController.cs**

```
var claims = new List<Claim> {
            new Claim(ClaimTypes.Name, user.Naziv)
          };
if (user.IsAdmin) {
   claims.Add(new Claim(ClaimTypes.Role, "Admin"));
}
if (user.IdPartnera.HasValue) {
   claims.Add(new Claim(CustomClaims.IdPartnera,
                        user.IdPartnera.ToString()));
}
```

BY NC SA

Autentifikacijski cookie

- **Uspješnom prijavom server će korisnikovom pregledniku isporučiti cookie s informacijom o korisniku**



- Preglednik šalje serveru cookie prilikom svakog zahtjeva te aplikacija na taj način autentificira korisnika
- U postavkama na prethodnom slajdu IsPersistent je postavljeno na false → preglednik će obrisati cookie kad se preglednik zatvori
- Moguće koristiti isti cookie za više aplikacija na istom serveru

  https://docs.microsoft.com/en-us/aspnet/core/security/cookie-sharing

Autorizacijska politika

- engl. *policy* = politika (ali u kontekstu upravljanja i planiranja)
- Pristup resursu se može ograničiti temeljem nekog pravila postavljanjem svojstva *Policy* u atributu *Authorize*

```
[Authorize(Policy = "Admin")]
```

  ili pozivom autorizacijskog servisa i provjere zadovoljava li trenutni korisnik trenutnu autorizacijsku politiku
  - Primjer: 🗁 WebApp \ Controllers \ DocumentController.cs

```
public async Task<IActionResult> Index(...
   var isAdminResult = await
           authorizationService.AuthorizeAsync(User, "Admin");
   if (!isAdminResult.Succeeded)
```

- Autorizacijske politike se postavljaju u datoteci Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
  services.AddAuthentication(...
  services.AddAuthorization(options => {
        options.AddPolicy("Admin",
      policy => policy.RequireClaim(ClaimTypes.Role, "Admin"));
```

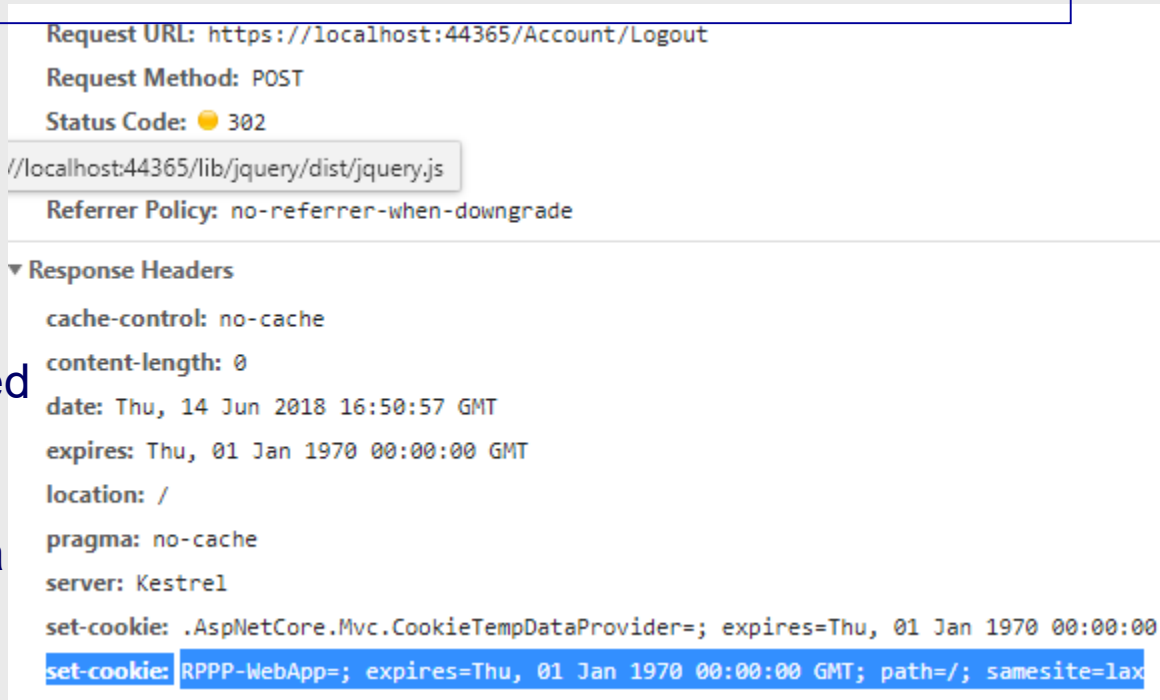
CC BY NC SA

Odjava korisnika

- **Primjer: 🗁 WebApp \ Controllers \ AccountController.cs**

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Logout() {
      await HttpContext.SignOutAsync();
      return RedirectToAction("Index", "Home");
}
```

- **Odjavom preglednik briše cookie**
- **Odjavu treba obaviti koristeći POST zahtjev**
  - Preglednik može unaprijed pokušati pristupiti nekim poveznicama, pa tako slučajno odjaviti korisnika



CC BY NC SA

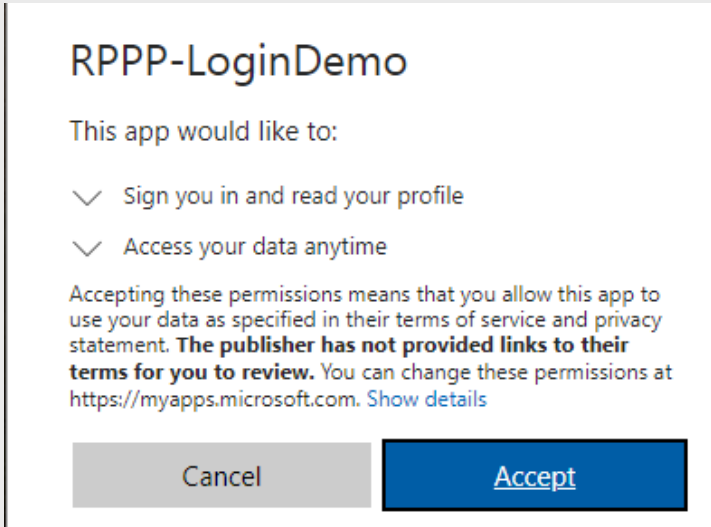
Prijava korištenjem vanjskih servisa

Prijava Microsoftovim računom

ASPNETCORE_ENVIRONMENT=MicrosoftLogin

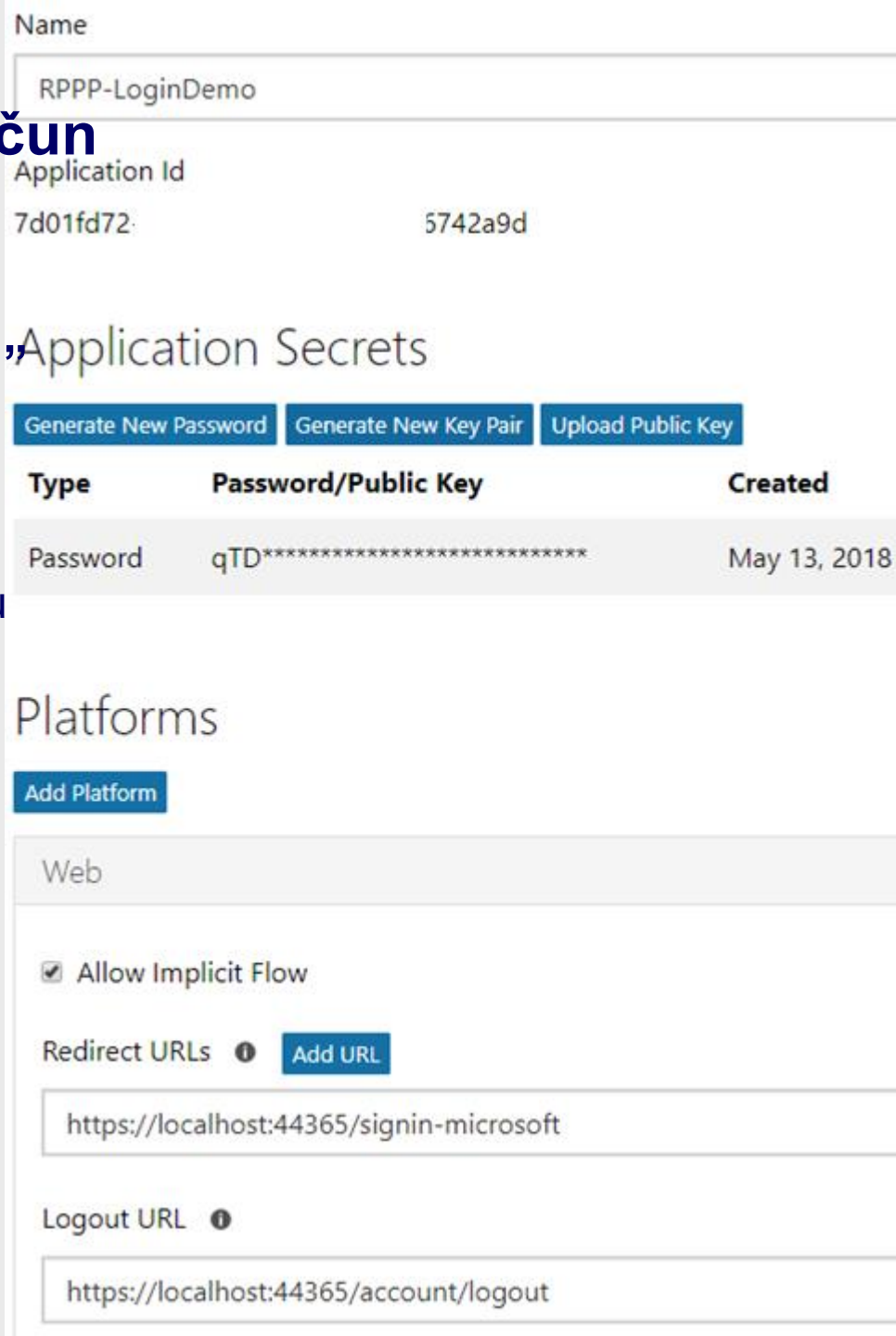
Tok prijave korisnika

- **Korisnik u pregledniku zahtijeva zaštićeni resurs (https://localhost:44365/Artikl)**
  - Aplikacija preusmjerava korisnika na Microsoftov servis za prijavu https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=... &response_type=code&redirect_uri=https%3A%2F%2Flocalhost%3A44365 %2Fsignin-microsoft…&state=…
- **Korisnik se prijavljuje na Microsoftovom servisu**
  - Ako je korisnik već prijavljen, samo odabire račun za prijavu
  - Microsoftov servis (po uspješnoj prijavi) preusmjerava korisnika natrag na aplikaciju https://localhost:44365/signin-microsoft?code=M9012261d-c5c9-85cb-2a6b... šaljući nekoliko cookiea
    - Ako se korisnik prvi put prijavljuje u ovoj aplikaciji pojavit će se upit slaže li se da aplikacija pruži svoje podatke
- **Ugrađena programska komponenta ASP.NET Corea interpretira podatke, postavlja autentifikacijski cookie i preusmjerava na prikaz artikala**



Registracije aplikacije za prijavu na Microsoftov račun

- **Aplikacija je slala svoj identifikator i lozinku**
- **Prethodno potrebno „registrirati" vlastitu aplikaciju na adresi** https://apps.dev.microsoft.com
  - za vrijeme razvoja id i lozinku pospremiti u tajnu datoteku
  - U postavkama navesti adresu koja zna primiti autorizacijski kod i cookie nakon uspješne prijave (Redirect URLs)
    - ugrađena funkcionalnost na adresi /signin-microsoft
  - Logout URL: Adresa na koju se šalje informacija da se korisnik negdje drugdje odjavio s Microsoftovog računa te bi ga trebalo odjaviti i iz aplikacije
    - ne radi trenutno



CC BY NC SA

Aktiviranje ugrađene funkcionalnosti za prijavu korištenjem Microsoftovog računa

- **Pri pokušaju pristupa zaštićenom resursu, korisnika se preusmjerava na Microsoftov servis (DefaultChallengeScheme)**
  - Dozvola korištenja servisa na osnovi identifikatora i lozinke vlastite aplikacije
  - Primjer: 🗁 WebApp \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
  services.AddAuthentication(options => {
        options.DefaultScheme =
               CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
               ...MicrosoftAccountDefaults.AuthenticationScheme;
      })
      .AddCookie(options => {
        options.Cookie.Name = "RPPP-ExternalCookie";
      })
      .AddMicrosoftAccount(microsoftOptions => {
        microsoftOptions.ClientId =
          Configuration["Authentication:Microsoft:ApplicationId"];
        microsoftOptions.ClientSecret =
          Configuration["Authentication:Microsoft:Password"];
      });
```

CC BY NC SA

Prijava korisnika

- **Ne koristi se vlastita stranica**
  - Korisnik će automatski biti preusmjeren na Microsoftov servis pokušajem pristupa zaštićenom resursu
  - Može se eksplicitno zatražiti prijava
    - Primjer: 🗁 WebApp \ Controllers \ AccountController.cs

```
[HttpGet]
[AllowAnonymous]
public IActionResult Login(string returnUrl = null) {
  returnUrl = Url.IsLocalUrl(returnUrl) ? returnUrl :
                Url.Action(nameof(HomeController.Index), "Home");
  var authenticationProperties = new AuthenticationProperties {
       RedirectUri = returnUrl
  };
  return Challenge(authenticationProperties, "Microsoft");
}
```

CC BY NC SA

Autorizacijska politika i zahtjevi

- **Mogu se koristiti tvrdnje koje su dio Microsoftovog računa.**
  - Za demonstraciju administratori će biti osobe čiji e-mail za Microsoftov račun završava s fer.hr
- **Koristi se mehanizam autorizacijskih zahtjeva**
  - Zahtjev je objekt razreda koji implementira sučelje *IAuthorizationRequirement*
  - Primjer: 🗁 WebApp \ Security \ EmailRequirement

```
public class EmailRequirement : IAuthorizationRequirement {
    public string Domena { get; private set; }
    public EmailRequirement(string domena) {
      Domena = domena;...
```

  - Primjer: 🗁 WebApp \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
  services.AddAuthentication(...
  services.AddAuthorization(options => {
        options.AddPolicy("Admin", policy =>
policy.Requirements.Add(new Security.EmailRequirement("fer.hr")));
```

CC BY NC SA

Rukovatelj zahtjevom

- **Rukovatelj zahtjevom provjera je li zahtjev zadovoljen**
  - Primjer: 🗁 WebApp \ Security \ EmailAuthorizationHandler

```
public class EmailAuthorizationHandler :
                    AuthorizationHandler<EmailRequirement> {
  protected override Task HandleRequirementAsync(
                        AuthorizationHandlerContext context,
                        EmailRequirement requirement) {
    if (!context.User.HasClaim(c => c.Type == ClaimTypes.Email))
        return Task.CompletedTask;
    string email = context.User.Claims
                          .Where(c => c.Type == ClaimTypes.Email)
                          .Select(c => c.Value).FirstOrDefault();
      if (!string.IsNullOrWhiteSpace(email) &&
                         email.EndsWith(requirement.Domena))
        context.Succeed(requirement);
```

  - Rukovatelj se registira kao i ostali razredi u lancu ovisnosti
  - Primjer: 🗁 WebApp \ Startup.cs

```
services.AddSingleton<IAuthorizationHandler,
                       Security.EmailAuthorizationHandler>();
```

CC BY NC SA

Odjava korisnika

- **Primjer: 🗁 WebApp \ Controllers \ AccountController.cs**

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Logout() {
      await HttpContext.SignOutAsync();
      return RedirectToAction("Index", "Home");
}
```

- **Odjavom iz aplikacije na ovaj način korisnik i dalje ostaje prijavljen na Microsoftovom računu i u ostalim aplikacijama!**
  - Uklonjen je cookie vezan za našu aplikaciju, ali ne i cookie Microsoftove jedinstvene prijave

CC BY NC SA

Jedinstvena odjava korisnika

- **Primjer: 🗁 WebApp \ Controllers \ AccountController.cs**

```
[HttpPost] ValidateAntiForgeryToken]
public async Task<IActionResult> LogoutMicrosoft() {
      await HttpContext.SignOutAsync();
      string url =
"https://login.microsoftonline.com/{tenant}/oauth2/v2.0/logout?post_logout_redirect_uri=" +
        WebUtility.UrlEncode(Request.Scheme + "://" +
Request.Host.Value); //npr.  "https://localhost:44365"
      return Redirect(url);
}
```

- **Gornji primjer izaziva odjavu s Microsoftovog računa i povratak na našu aplikaciju**

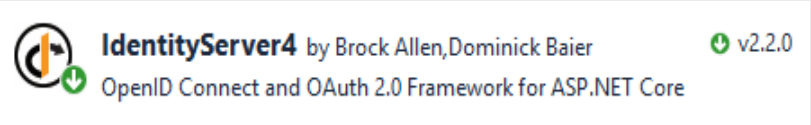
BY NC SA

Izrada vlastitog servera za jedinstvenu prijavu korisnika

ASPNETCORE_ENVIRONMENT=OpenIdConnect

Vlastiti SSO server

- **SSO = engl. *Single Sign On* (jedinstvena prijava)**
- **U prethodnom primjeru korišten je Microsoftov servis za jedinstvenu prijavu korisnika u više aplikacija**
- **Sličan koncept može se izvesti korištenjem vlastitog servera**
  - Implementacija paketom IdentityServer4



    - koristi se protokol OpenId Connect
    - sama izvedba prijave korisnika unutar servera nalik rješenju s jednostavnom prijavom (upiti i rukovatelji upitima)
  - kao i kod rješenja s Microsoftom, aplikaciju potrebno prijaviti
    - prikazano malo kasnije nakon prikaza rješenja
    - nije produkcijsko rješenje
      - razvojni potpisni certifikat, postavke (klijenti, resursi) tvrdo-kodirane
  - Vlastiti SSO u primjeru opisanom kroz slajdove bit će na adresi https://localhost:44314
    - bitno zbog međusobnog povezivanja – potrebno navesti adresu SSO servera

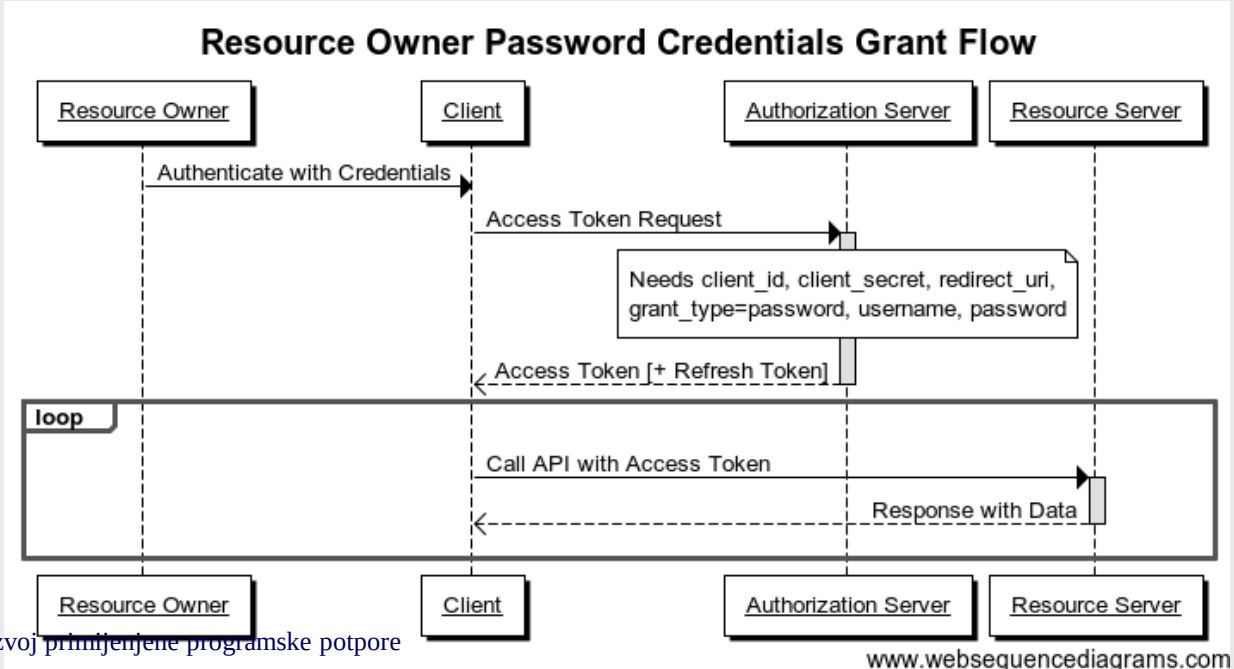
CC BY NC SA

Pojmovi u SSO scenariju

- **Vlasnik resursa (engl. Resource Owner)**
  - načelno krajnji korisnik
- **Server s podacima (engl. Resource Server)**
  - Server sa zaštićenim podacima
- **Klijent (engl. Client)**
  - Aplikacija koja zahtjeva pristup nekom zaštićenom podatku na serveru s podacima. Mora biti registrirana na autorizacijskom serveru.
- **Autorizacijski server (engl. Authorization Server)**
  - Server koji izdaje pristupni token. Ovim token klijent pristupa serveru s resursima
- **Token**
  - niz slučajno odabranih znakova generiranih na zahtjev nekog klijenta
  - Bearer authentication: „give access to the bearer of this token”

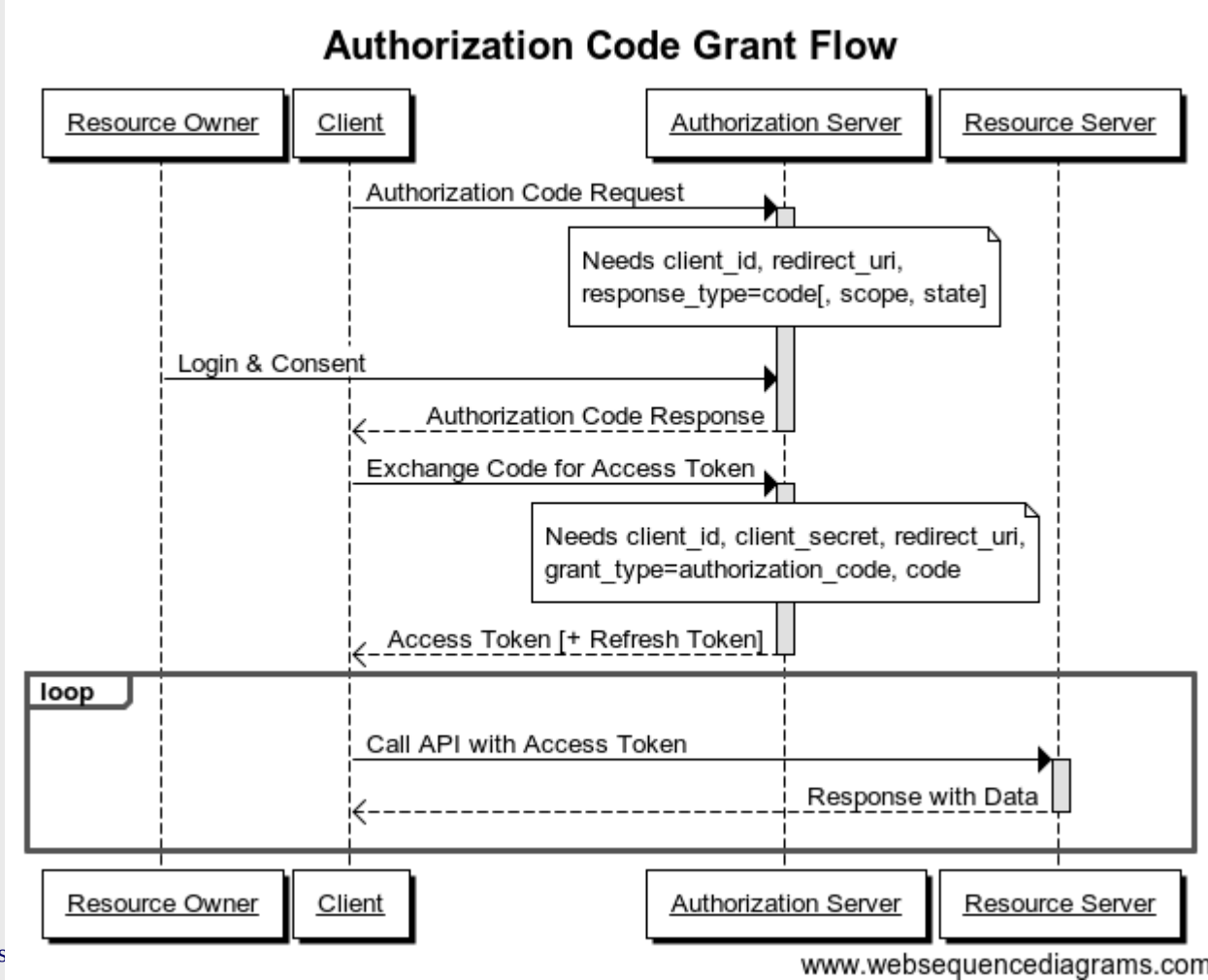
Tipovi interakcije klijenata i SSO servera (1)

- U primjerima će se koristiti *Resource Owner Password* i *Hybrid*
- Više o tipovima interakcija
  - http://docs.identityserver.io/en/release/topics/grant_types.html
  - http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/
  - https://auth0.com/docs/api-auth/which-oauth-flow-to-use
- Resource owner password
  - aplikacija traži od SSO servera token za korisnika šaljući korisnikovo korisničko ime i lozinku (bit će korišteno iz konzolne aplikacije)



Tipovi interakcije klijenata i SSO servera (2)

- **Hybrid (authorization code + token)**
  - klijentska aplikacija komunicira sa serverom svojom šifrom, a korisnika preusmjerava na autorizacijski server za prijavu te naknadno dobije njegov pristupni token



Uključivanje prijave korištenjem vlastitog SSO servera (1)

- **DefaultChallengeScheme treba postaviti na oidc**
  - Naziv autentifikacijske sheme proizvoljan. Primjer: 🗁 WebApp \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
  string authSchemeName = "cookie";
  services.AddAuthentication(options => {
        options.DefaultScheme = authSchemeName;
        options.DefaultChallengeScheme = "oidc";
  }).AddCookie(authSchemeName)
  .AddIdentityServerAuthentication( //zbog tokena iz konzolne aplik.
        JwtBearerDefaults.AuthenticationScheme, options => {
              options.Authority = "https://localhost:44314/";
              options.ApiName = "demo_api";
        })
  .AddOpenIdConnect("oidc", options => {
     options.SignInScheme = authSchemeName;
     options.Authority = "https://localhost:44314/";
     options.ClientId = "SecurityDemoWebApp";
     options.ClientSecret = "psst-tajna";
     options.ResponseType = OpenIdConnectResponseType.CodeIdToken;
```

CC BY NC SA

Uključivanje prijave korištenjem vlastitog SSO servera (2)

- **Navesti koji skup informacija (scope) se traži**
  - Na osnovi primljenih informacija stvaraju se tvrdnje za prijavljenog korisnika
  - Potrebno riješiti nesklad između naziva tvrdnji za role i naziv korisnika
- **Primjer: 🗁 WebApp \ Startup.cs**

```
public void ConfigureServices(IServiceCollection services) {
   ...
   options.Scope.Add("openid");
   options.Scope.Add("profile");
   options.Scope.Add("rppp.profile");
   options.Scope.Add("role");

   options.SaveTokens = true;
   options.GetClaimsFromUserInfoEndpoint = true;

   options.TokenValidationParameters = new TokenValidationParameters{
         NameClaimType = "name",
         RoleClaimType = "role",
      };
   });
```

BY NC SA

Prijava korisnika

- **Ne koristi se vlastita stranica**
  - Korisnik će automatski biti preusmjeren na SSO server pokušajem pristupa zaštićenom resursu
  - Može se eksplicitno zatražiti prijava
    - Slično kao u primjeru s Microsoftovim računom – razlika je jedino u shemi prijave (oidc)
    - Primjer: 🗁 WebApp \ Controllers \ AccountController.cs

```
[HttpGet]
[AllowAnonymous]
public IActionResult Login(string returnUrl = null) {
  returnUrl = Url.IsLocalUrl(returnUrl) ?
                        returnUrl : Url.Action("Index", "Home");
  var authenticationProperties = new AuthenticationProperties {
       RedirectUri = returnUrl
  };
  return Challenge(authenticationProperties, "oidc");
}
```

CC BY NC SA

Autorizacijska politika

- **Ista autorizacijska politika kao i kod jednostavne prijave**
  - Različiti naziv tvrdnje za grupu ClaimTypes i JwtClaimTypes koje koristi IdentityServer
- **Primjer: 🗁 WebApp \ Controllers \ Startup.cs**

```
public void ConfigureServices(IServiceCollection services) {
  services.AddAuthentication(...
  services.AddAuthorization(options => {
        options.AddPolicy("Admin",
      policy => policy.RequireClaim(ClaimTypes.Role, "Admin"));
        options.AddPolicy("AdminForApi",
    policy => policy.RequireClaim(JwtClaimTypes.Role, "Admin"));
  });
```

Odjava korisnika iz aplikacije

- **Primjer: 🗁 WebApp \ Controllers \ AccountController.cs**

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Logout() {
      await HttpContext.SignOutAsync();
      return RedirectToAction("Index", "Home");
}
```

- **Odjavom iz aplikacije korisnik i dalje ostaje prijavljen na SSO serveru**
  - Ponovnim klikom na *Prijava*, korisnik će bez pitanja biti prijavljen u aplikaciju.

CC BY NC SA

Jedinstvena odjava korisnika

- **Primjer: 🗁 WebApp \ Controllers \ AccountController.cs**

```
 [HttpPost] ValidateAntiForgeryToken]
public async Task<IActionResult> LogoutOpenIdConnect() {
      await HttpContext.SignOutAsync("cookie");
      await HttpContext.SignOutAsync("oidc");
}
```

- **Gornji primjer odjavljuje korisnika i iz trenutne aplikacije i sa SSO servera pri čemu je odjava inicirana iz neke klijentske aplikacije**

LogoutOpenIdConnect	POST	302	Other
endsession?post_logout_redirect_uri=https%3A%2F%2F...client-SKU=ID_NETSTAND...	GET	302	:44365/Account/LogoutOpenIdConnect
logout?logoutId=CfDJ8E4HbdPtF-VBISvxZHCF3F7IzMVXGm...AMVywmIIJGU933Z9D...	GET	200	:44314/connect/endsession?post_logout_redirec
signout-redirect.js	GET	200	:44314/account/logout?logoutId=CfDJ8E4HbdPt
callback?endSessionId=CfDJ8E4HbdPtF-VBISvxZHCF3F6K...LLwgNiuabezc4X7XTkBB...	GET	200	:44314/account/logout?logoutId=CfDJ8E4HbdPt
logout?sid=933601030f44c9709c405cc971dbe558&iss=https%3A%2F%2Flocalhost...	GET	(failed)	:44314/connect/endsession/callback?endSessior
signout-callback-oidc?state=CfDJ8E4HbdPtF-VBISvxZH...xFS627JN-Mdtnn-OKzQ3x...	GET	302	:44314/js/signout-redirect.js:4
localhost	GET	200	:44365/signout-callback-oidc?state=CfDJ8E4Hb

CC BY NC SA

Odjava korisnika sa SSO servera

- **Što ako se korisnik odjavi s SSO servera ili iz neke druge aplikacije?**
- **SSO vodi evidenciju o prijavljenim klijentima te ih obavještava o jedinstvenoj odjavi šaljući identifikator sjednice (sid)**
  - Npr. https://localhost:44365/account/logout?sid=ee5e81301a441aa27227309782&iss=https%3A%2F%2Flocalhost%3A44314
  - GET zahtjev
- **Svaka od aplikacija mora moći primiti informaciju o odjavi**
  - Primjer: 🗁 WebApp \ Controllers \ AccountController.cs

```
[HttpGet]
public async Task Logout(string sid) { //Front channel logout uri
  var sidClaim = User.Claims?.Where(
                          c => c.Type == JwtClaimTypes.SessionId)
                      .FirstOrDefault();
      if (sidClaim != null && sidClaim.Value == sid){
        await HttpContext.SignOutAsync();
      }
    }
}
```

Konfiguracija vlastitog SSO servera

- Za demonstaciju koriste se tvrdo kodirani klijenti i resursi
- Primjer: 🗁 SingleSignOn \ Controllers \ AccountController.cs

```
public void ConfigureServices(IServiceCollection services) {
      services.AddMvc();
      services.AddIdentityServer()
              .AddDeveloperSigningCredential()
              .AddInMemoryIdentityResources(
                      Resources.GetIdentityResources())
              .AddInMemoryApiResources(
                      Resources.GetApiResources())
              .AddInMemoryClients(Clients.GetClients())
              .AddResourceOwnerValidator<
                      CustomResourceOwnerPasswordValidator>();

      string connectionString = Configuration["ConnectionString"];
      services.AddDbContext<FirmaContext>(
          options => options.UseSqlServer(connectionString));
      services.AddTransient<IValidateUserQueryHandler,
                              ValidateUserQueryHandler>();
```

Definiranje klijenata (1)

- **Primjer: 🗁 SingleSignOn \ Data\ Clients.cs**
  - Demo web aplikacija

```
var securityDemoWebApp = new Client{
        ClientId = "SecurityDemoWebApp",
        ClientSecrets = { new Secret("psst-tajna".Sha256()) },
        ClientName = "RPPP Security Demo Web Application",
        AllowedGrantTypes = GrantTypes.Hybrid,
        RedirectUris = new List<string> {
                      "https://localhost:44365/signin-oidc" },
        PostLogoutRedirectUris = new List<string> {
                "https://localhost:44365/signout-callback-oidc" },
        FrontChannelLogoutUri =
                      "https://localhost:44365/account/logout",
        RequireConsent = false,
        AllowedScopes = {
                    IdentityServerConstants.StandardScopes.OpenId,
                    IdentityServerConstants.StandardScopes.Profile,
                    "rppp.profile", "role"
                },
        AlwaysIncludeUserClaimsInIdToken = true
```

Definiranje klijenata (2)

- **Primjer: 🗁 SingleSignOn \ Data\ Clients.cs**
  - Konzolna aplikacija

```
...
var interactiveClient = new Client {
       ClientId = "client",
       AllowedGrantTypes = GrantTypes.ResourceOwnerPassword,
      ClientSecrets = { new Secret("psst...".Sha256()) },
      AllowedScopes = {
                  "demo_api"
       },
       AlwaysIncludeUserClaimsInIdToken = true
     };

     return new List<Client> {
      securityDemoWebApp,
      interactiveClient
     };
```

CC BY NC SA

Definiranje resursa

- **Primjer: 🗁 SingleSignOn \ Data\ Resources.cs**
  - Definira koje informacije SSO dozvoljava/nudi aplikacijama

```
static IEnumerable<IdentityResource> GetIdentityResources() {
      var rpppProfile = new IdentityResource(
        name: "rppp.profile",
        displayName: "RPPP profile",
        claimTypes: new[] { CustomClaims.IdPartnera });

    return new List<IdentityResource>
    {
        new IdentityResources.OpenId(),
        new IdentityResources.Profile(),
        rpppProfile,
        new IdentityResource {
              Name = "role",
              UserClaims = new List<string> { "role" }
            }
    };
```

Definiranje resursa

- **Primjer: 🗁 SingleSignOn \ Data\ Resources.cs**
  - Definira koje informacije SSO dozvoljava/nudi za api pozive

```
static IEnumerable<ApiResource> GetApiResources() {
   return new List<ApiResource> {
          new ApiResource(
           "demo_api", "Demo API za demonstraciju bearer tokena",
               new[] { CustomClaims.IdPartnera, "role" })
      };
    }
```

CC BY NC SA

Postavljanje tvrdnji korisniku koji se želi prijaviti

- **Nakon uspješne provjere korisnika postavljaju se tvrdnje slično kao u primjeru s jednostavnom prijavom**
  - Primjer: 🗁 SingleSignOn \ Controllers \ AccountController.cs

```
var claims = new List<Claim> {
  new Claim(JwtClaimTypes.Subject, model.Username),
  new Claim(JwtClaimTypes.Name, user.Naziv),
  new Claim(JwtClaimTypes.AuthenticationTime,
              DateTime.UtcNow.ToEpochTime().ToString())
};
if (user.IsAdmin) {
   claims.Add(new Claim(JwtClaimTypes.Role, "Admin"));
}
if (user.IdPartnera.HasValue) {
    claims.Add(new Claim(CustomClaims.IdPartnera,
                         user.IdPartnera.ToString()));
}
```

CC BY NC SA

Prijava korisnika

- **Nakon postavljenjih tvrdnji pokreće se postupak prijave korisnika**
  - Primjer: 🗁 SingleSignOn \ Controllers \ AccountController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginInputModel model... {
  ...
  var ci = new ClaimsIdentity(claims, "password",
              JwtClaimTypes.Name, JwtClaimTypes.Role);

  var cp = new ClaimsPrincipal(ci);

  await HttpContext.SignInAsync(cp);

  if (interaction.IsValidReturnUrl(model.ReturnUrl) ||
                    Url.IsLocalUrl(model.ReturnUrl)) {
      return Redirect(model.ReturnUrl);
  }
```

Provjera korisničkog tokena

ASPNETCORE_ENVIRONMENT=OpenIdConnect

Zaštićeni web servis

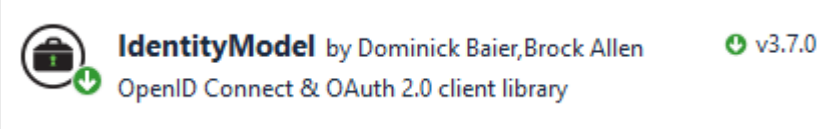
- **Primjer: 🗁 WebApp \ Controllers \ Api \ DemoApi.cs**

```
[ApiController]
[Route("api/demo/[action]")]
public class DemoApi : ControllerBase {
   public IEnumerable<string> Public() {
      var rijeci = "Ovo je javni servis".Split(" ");
      return rijeci;
   }

  [Authorize(AuthenticationSchemes =
                        JwtBearerDefaults.AuthenticationScheme)]
   public IEnumerable<string> Private() {
      var rijeci = "Ovu poruku mogu vidjeti samo logirani
korisnici".Split(" ");
      return rijeci;
    }
    [Authorize(Policy = "AdminForApi", AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme)]
    public IEnumerable<string> AdminsOnly()...
```

BY NC SA

Dohvat pristupnog tokena iz konzolne aplikacije

- **Iz konzolne aplikacije se poziva zaštićeni web servis.**
- **Potrebno je obaviti prijavu na SSO server i dohvatiti pristupni token**
  - Instalirati NuGet paket IdentityModel
    - automatsko otkrivanje adrese zahtjeva na osnovi discovery dokumenta
  - Primjer: 🗁 ConsoleApp \ Program.cs




```
static async Task<string> GetToken(string url, string username) {
      var disco = await DiscoveryClient.GetAsync(url);
      if (disco.IsError) {
        Console.WriteLine(disco.Error);
        return null;
      }
      var tokenClient = new TokenClient(
                        disco.TokenEndpoint, "client", "psst...");
      var tokenResponse = await
              tokenClient.RequestResourceOwnerPasswordAsync(
                        username, username, scope: "demo_api");
      return tokenResponse.AccessToken;
}
```

CC BY NC SA

Pristupni token

- **Sadržaj nekog tokena može se vidjeti na https://jwt.io/**

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsImtpZCI6IjMwMDk4OGYz
NmY2ZWY1NDkzN2I2OGRjNTBiNjhiYTY3IiwidHlw
IjoiSldUIn0.eyJuYmYiOjE1MjkwMTY0MTQsImV4
cCI6MTUyOTAyMDAxNCwiaXNzIjoiaHR0cHM6Ly9s
b2NhbGhvc3Q6NDQzMTQiLCJhdWQiOlsiaHR0cHM6
Ly9sb2NhbGhvc3Q6NDQzMTQvcmVzb3VyY2VzIiwi
ZGVtb19hcGkiXSwiY2xpZW50X2lkIjoiY2xpZW50
Iiwic3ViIjoiMTIzNDU2Nzg5MTAiLCJhdXRoX3Rp
bWUiOjE1MjkwMTY0MTQsImlkcCI6ImxvY2FsIiwi
aHR0cDovL3JwcHAuZmVyLmhyL2NsYWltcy9pZF9w
YXJ0bmVyYSI6IjEwIiwic2NvcGUiOlsiZGVtb19h
cGkiXSwiYW1yIjpbInB3ZCJdfQ.MfdgiTWEdD3cZ
iB4_yBUuUn4FisRKMGk3h2HmNvYjHiMd_wO5uJQ4
PH2sFcdkqSlGEhlJJ6xyuZgJ_NtZ3YQKaJa80upl
3hf9HtVvHQmwqF2QTr7DtSZvEFtAV9UCyDai6f3X
YDSHfaR3Sb0OmZNXKpLTIL9fEEsJWduKot_cozM0
vE12MtgmQEeUGgjovPMfulv3L36tMuzVXt2qU55I
9S5u7l625x93cb8OL3JSpqJxCE-
87QgSOhUTHdB_JAPnDmMauqL0EAoC-
GZNQL4j2fkHfV9rMU-
K6wMdIp17RCBdpNF_7C3BnAtTZbcXRjh5oQBBDGp
5TByyMctGng9Ow
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "kid": "300988f36f6ef54937b68dc50b68ba67",
  "typ": "JWT"
}
```


PAYLOAD: DATA


```
{
  "nbf": 1529016414,
  "exp": 1529020014,
  "iss": "https://localhost:44314",
  "aud": [
    "https://localhost:44314/resources",
    "demo_api"
  ],
  "client_id": "client",
  "sub": "12345678910",
  "auth_time": 1529016414,
  "idp": "local",
  "http://rppp.fer.hr/claims/id_partnera": "10",
  "scope": [
    "demo_api"
  ],
  "amr": [
    "pwd"
  ]
}
```


BY NC SA

Dohvat pristupnog tokena iz konzolne aplikacije

- **Pristupni token se stavlja pomoću ekstenzije *SetBearerToken* iz paketa IdentityModel**
  - Primjer: 🗁 ConsoleApp \ Program.cs

```
static async Task CallApi(string url, string access_token = null) {
   HttpClientHandler handler = new HttpClientHandler() {
        AllowAutoRedirect = false
   };
   using (var client = new HttpClient(handler)) {
        if (!string.IsNullOrWhiteSpace(access_token)) {
          client.SetBearerToken(access_token);
        }
        var response = await client.GetAsync(url);
        Console.WriteLine($"{response.StatusCode} :
                            {response.ReasonPhrase}");
        if (response.IsSuccessStatusCode)
        {
              ...
```

CC BY NC SA

Literatura

- **ASP.NET Core i autorizacija i autentifikacija**
  - https://docs.microsoft.com/en-us/aspnet/core/security/authentication/
  - https://docs.microsoft.com/en-us/aspnet/core/security/authorization/
- **Identity server, OAuth2 i slično**
  - https://identityserver4.readthedocs.io/
  - https://auth0.com/docs/api-auth/which-oauth-flow-to-use
  - https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-v2-protocols-oauth-code
  - https://www.scottbrady91.com/Identity-Server/Getting-Started-with-IdentityServer-4
  - http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/
- **Command-query separation**
  - https://cuttingedge.it/blogs/steven/pivot/entry.php?id=92
  - https://www.future-processing.pl/blog/cqrs-simple-architecture/
  - https://martinfowler.com/bliki/CQRS.html

BY NC SA