

# Razvoj primijenjene programske potpore

---

## 1. Osnove programskog inženjerstva

Životni ciklus razvoja programske podrške.

Planiranje projekta

# Programska potpora

- Programska oprema/podrška/potpora, softver
  - 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:
  - lako se kopira (zajedno s pogreškama)
  - ne troši se, dugo se koristi, ali zastarijeva
  - složenost (problem održavanja, cijena nadogradnje)
- Primijenjena programska potpora = Računalna aplikacija
  - namjenski program, primjenska programska oprema
  - računalom podržano rješenje jednog ili više poslovnih problema ili potreba
- Informacijski sustav = sustav aplikacija za upravljanje ljudskim aktivnostima

# Kako nastaje softver?

- Zanat ili inženjerstvo?
  - Kome je namijenjen softver?
  - Usporedba s proizvodnjom auta, izgradnjom građevina ...
- Tehnike, prakse, alati, predlošci ...
  - područje programskog inženjerstva
- Softver može nastati i bez korištenja tehnika i metoda programskog inženjerstva, ali takav softver je vjerojatno manje pouzdan, a lako moguće u konačnici i skuplji

# Programsko inženjerstvo

*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

# Programsko inženjerstvo – nulti pacijent

- Pojam „programsko inženjerstvo“ nastao 1963. godine
  - Margaret Hamilton (1936 - )
  - Serija NATO-vih konferencija od 1968.g.

Slika:

Margaret Hamilton 1969. godine pored ispisa programskog koda za projekt Apollo.

Izvor: Wikimedia Commons



# Programsko inženjerstvo i sroдna 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 should be engineering*”
  - originalni isječak na  
[https://www.gamasutra.com/view/feature/131817/software\\_engineering\\_is\\_not\\_.php](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.

# Poznavanje korisnikove domene

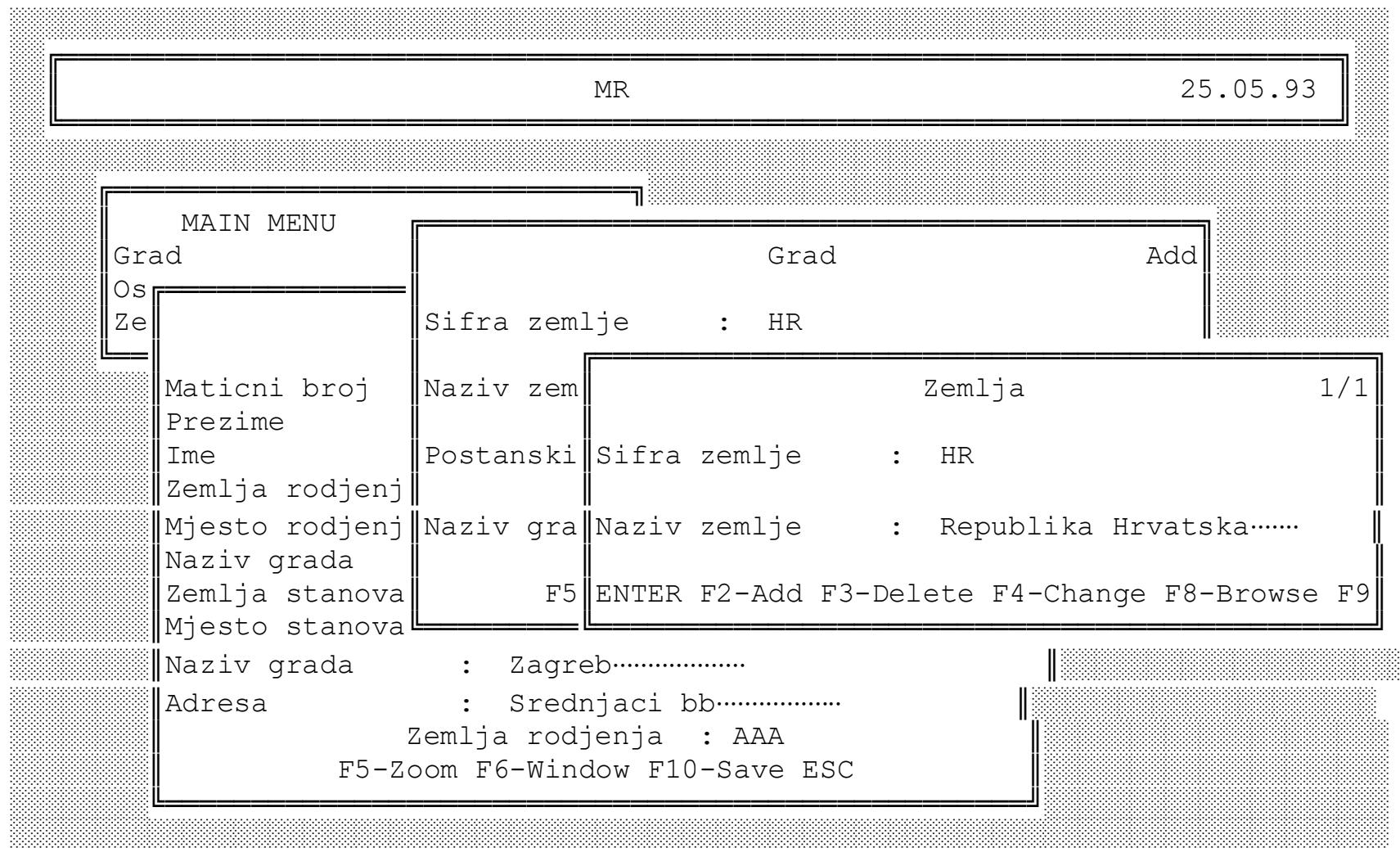
*If your only tool is a hammer then every problem looks like a nail*

<https://quoteinvestigator.com/2014/05/08/hammer-nail/>

- Povećanje potražnje i činjenica da je relativno jednostavno pisati kod posljedično uzrokuje osmišljavanje boljih inženjerskih tehnika...
  - ... pri čemu treba znati kad i koju tehniku upotrijebiti
- Faktori koji utječu na odabir tehnologije i metodologije
  - veličina projekta, namjena, vijek trajanja, broj korisnika, oprema, ...
    - tanki klijent, debeli klijent, višeslojna aplikacija, web, mobilna, *desktop* aplikacija, izgled prilagođen različitim uređajima, ...
- Trendovi i tehnologije se mijenjaju, ali ne i činjenica da je potrebno ovladati jezikom (terminologijom) korisnika

# Jednokorisničke, samostalne aplikacije

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



# Poslužiteljske aplikacije

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

kfertalj (ansi) | PIS - PERSONALNI INFORMACIJSKI SUSTAV | (Sri) 04.12.96

OSOBA: Dohv Sljed Preth Unos Izmj Ostalo Lista Rasp Zap ...  
Postavljanje uvjeta za dohvata zapisa

===== ( LALIĆ MARIJAN ) = (2/3) ===== 120/261

Sprema (71) (Visoka VII/1 )  
Br.svjed. ( ) Dat. ( ) Izdao ( )  
Zanimanje ( 54131) (ORGANIZATORI SISTEMA )

Stamb.stanje ( 3) (Privremeni korisnik)

Krvna grupa ( )  
Zdrav.broj ( )

Ured za obranu ( 0) (\*\*\*\*\* Ne  
Osobni VPD (33207) (SIN-USS I ŠP  
Početak voj.roka ( )  
Vojска voj.roka ( 0) (Nepoznata )  
Razl.prest.voj.roka ( )  
Sudjel. u dom.ratu (D)

Broj dokumenta \_\_\_\_\_ Orig. broj  
Vrsta dokumenta BKA BANKOVNI TZVOD  
Mj. troška/pri. 30 Pregled : radnik  
Konto 99 radnik naziv  
Iznos kuna \_\_\_\_\_  
Dug/Pot D \_\_\_\_\_

Datum dokumenta 05  
Analitika R  
Program \_\_\_\_\_  
Aktivnost \_\_\_\_\_  
Pror. glava \_\_\_\_\_  
Izvor financir. -  
Pozicija \_\_\_\_\_  
Opis \_\_\_\_\_  
Valuta / devize \_\_\_\_\_

traz fm

IZBOR 2 3 4TRAŽI 5 6 7NATRAG 8NAPRIJE 9 0POMOC  
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10

# Klijentske aplikacije

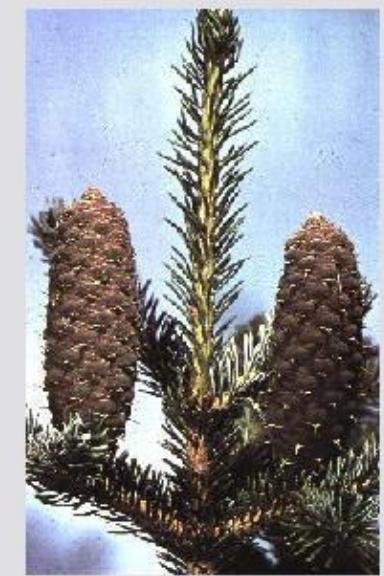
- „debeli“ klijenti, kraj 90-ih, Microsoft Access, Visual Basic, Java, ...

Dokument

Vrsta	Predračun	Broj	102	Datum	09.09.96	Prethodni		Broj	
Partner	Dječji vrtić "Vrapčići"					Br.ulaznog dokumenta			
Datum valute	20.09.96	Dana valute	12	10 % Rabat		Račun PP	<input type="checkbox"/>		
Otprema	HPT	Plaćanje	Virman	Ukupno	432.00	Datum storna			
Ostali troškovi	manipulacija + poštارина		Ček	18.00	Datum ažuriranja				
Slovima	četristopadeset kuna		Gotovina						
Zaglavlje	Napomena								
Šifra	Naziv	Količina							
1	Pokaži što znaš	8.00							
2	Pokušaj nešto novo	8.00							
3	Učimo opažati	8.00							
Stavka		◀◀ 1 od 3 ▶▶							
Dokument		◀◀ 1 od 6 ▶▶							
F3 - Brisanje		F5 - Tablica							

Vrsta: *Abies alba Mill.*

Autor	<input type="text"/>	<input type="button" value="Slika"/>	<input type="button" value="Del"/>
Datum	<input type="text"/>	Thumb <input type="checkbox"/>	<input type="button" value="Sakrij"/>
Tip	<input type="text"/> Slika	<input checked="" type="checkbox"/> Skanirano	<input type="checkbox"/> Javno
Objekt	<input type="text"/> cvat/inflorescence		
Tehnika	<input type="text"/> skanirano iz publikacije/scan from publication		
Pohrana	<input type="text"/>		
Referenca	<input type="text"/> Šilić, Č.		
Godina	<input type="text"/> 1973		
Naslov	<input type="text"/> Atlas drveća i grmlja.		
Lokalitet	<input type="text"/>		
Opaska	<input type="text"/>		



# Mobilne i distribuirane aplikacije (1)

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

Subscribe      1:14      ok

Subscriber: Pretplatnik

Server name: voz

Virtual directory: rep

Publication: gg Osoba: ozura

Database: gg Datum: 7.8.06

Username: sa Projekt: !AktivanJavan

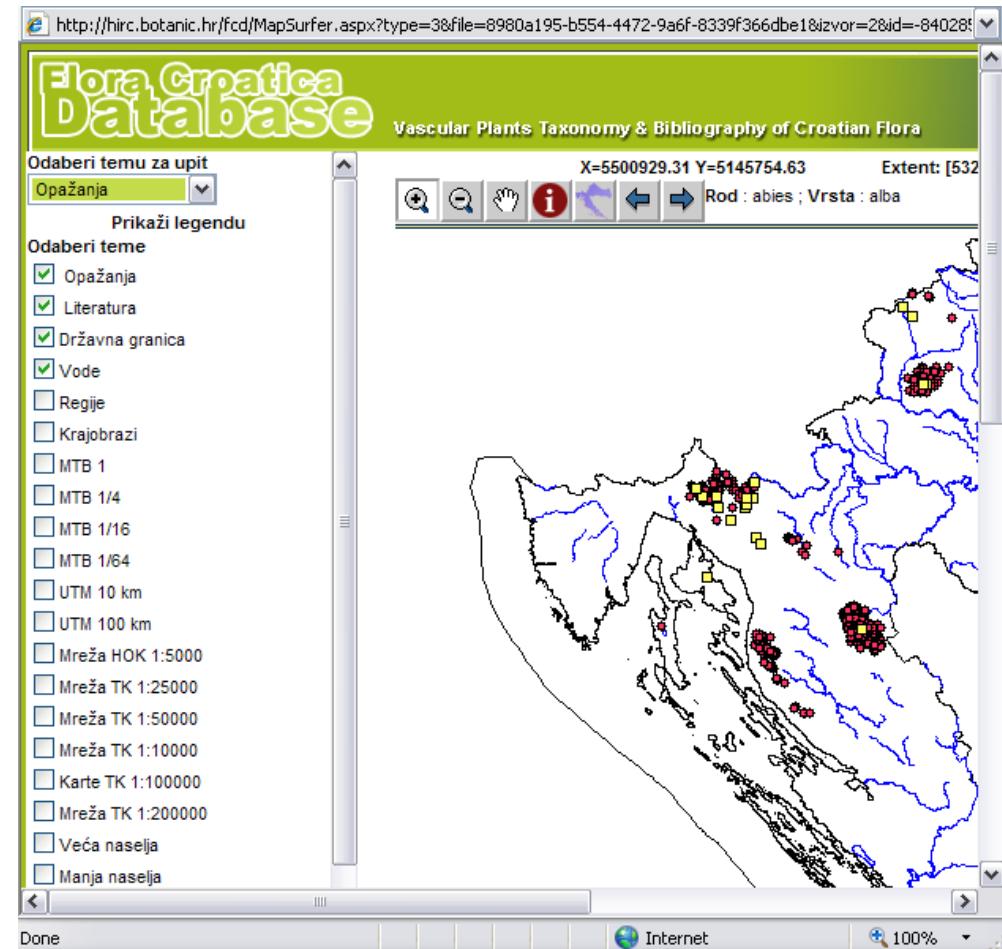
Password: \*\*\* Posao: backup

Database Path: IMY

Rbr	Posa	Osoba	DatPosao	KratP
254662	anaj	21.2.04	[VER]	
254663	duje	14.2.04	111	
254664	tošo	19.2.04	111	
254665	anaj	19.2.04	111	
254666	anaj	21.2.04	ACI	
254667	duje	21.2.04	111	
254668	tošo	12.2.04	111	

Sati Sum 31975,65

View << < > >> + X ✓



# Mobilne i distribuirane aplikacije (2)

- Web servisi + mobilne, web aplikacije, jQuery

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

Count	<a href="#">GET</a>	Vraća ukupni broj podataka u herbaru koji zadovoljavaju traženi filter. Za postavke vidi SearchCollection/{IdZbirke}
Count/{IdZbirke}	<a href="#">GET</a>	Vraća ukupni broj podataka u herbarskoj zbirici koji zadovoljavaju traženi filter. Za postavke vidi SearchCollection/{IdZbirke}
Sabiraci	<a href="#">GET</a>	Vraća popis sabirača koje su zavedene u FCD-u koji počinju određenim nazivom (čvor je za pretraživanje)
Search	<a href="#">GET</a>	Vraća popis herbara koji zadovoljavaju određeni filter (može biti prazan). Opcioni parametri imesvojte, datumsabiranja, inventarnibroj, zbirka, idslike, tdwg, porodica, tip. Filteri: godina, idherbara, idzbirke, inventarni broj (*), nagib opisnalažista (riječi odvojeni zarezom), saslikom (čvor je za pretraživanje). Vrsta: Toni/saslikom=true/opisnalažista=otok Vis/idzbirke=14/godina=2011&sort=godina
SearchCollection/{IdZbirke}	<a href="#">GET</a>	Pretražuje herbarsku zbirku. Vidi Search za detalje.
TDWG	<a href="#">GET</a>	Vraća popis država po TDWG-u. Rezultat je lista parova (oznaka države, naziv države).
TipoviPrimjeraka	<a href="#">GET</a>	Vraća popis tipova primjeraka herbara. Rezultat je lista parova (oznaka tipa, naziv tipa).
Zbirke	<a href="#">GET</a>	Vraća popis zbirki koje su zavedene u FCD-u. Rezultat je lista parova (id zbirke, naziv zbirke).



Menu

Flora Croatica Database Crvena knjiga Bibliografija Korisno bilje Alohtone biljke Galerija Stanija

Opažanja Herbar Korisne poveznice Prikaži praznu kartu

Kako koristiti bazu Prijava korisnika 1.3.2013. 12:23

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)	Označi
<a href="#">31493</a>		Rosaceae	Amelanchier ovalis Medik.	Vrbek, Mirjana	2012	CNHM Herbarium of Croatian Natural History Museum	Croatia	
<a href="#">31492</a>		Boraginaceae	Lithospermum purpurocaeruleum L.	Vrbek, Mirjana	2012	CNHM Herbarium of Croatian Natural History Museum	Croatia	

Dohvaćanje mapa

Ovisno o brzini internetske veze, ovaj proces može potrajati nekoliko minuta.

Datoteka: 26.ozf2

42% 42/100

Mbotanicar

Lilium bulbiferum L.  
Liliales Perleb ,Liliaceae  
Obj: Sp. Pl. 302 (1753)  
S: Lilium aurantiacum Weston, Lilium chaixii Maw, Lilium croceum Chaix, Lilium pubescens Bernh. ex Hornem.  
N: Bulbillentragende Feuerlilie, Giglio di San Giovanni, Giglio rosso, Lis a bulbillés, kruna, lukovičavi ljljan, narančasti krin, orange lily, turška lilija, zlatan, zlatoglav, zvjezdasti lijer, zvjezdasti ljljan, žilj  
Status: VU, S3

Staništa:

C.3.5.3.4. - Travnjaci zmijka i pjegavog jastrebijaka,  
C.5.1.2.5. - Zajednica vlasnatog zmijka i planinske djeteline

# Prilagodljivi dizajn

Kriterij pretrage: Vaskularna flora, Porodica: Liliaceae, Javno: Da  
Rezultati pretrage: Broj fotografija: 1036 , Broj svojti: 33 , Broj autora: 43  
Broj zapisa po stranici

1 2 3 4 >

\* **Danae racemosa (L.) Moench**

 <b>Id:</b> 150022 <b>Autor:</b> Borovečki-Voska, Ljiljana <b>Datum slike:</b> 31.10.2018. <b>Datum unosa:</b> 21.11.2018. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> habitus	 <b>Id:</b> 150023 <b>Autor:</b> Borovečki-Voska, Ljiljana <b>Datum slike:</b> 31.10.2018. <b>Datum unosa:</b> 21.11.2018. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> plod	 <b>Id:</b> 150021 <b>Autor:</b> Borovečki-Voska, Ljiljana <b>Datum slike:</b> 31.10.2018. <b>Datum unosa:</b> 21.11.2018. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> habitus	 <b>Id:</b> 135868 <b>Autor:</b> Hrvatski prirodoslovni muzej, CNHM <b>Datum slike:</b> 4.8.2017. <b>Datum unosa:</b> 25.1.2018. <b>Tehnika:</b> fotografija - digitalni aparat
--	---	--	--

\* **Erythronium dens-canis L.**

 <b>Id:</b> 22479 <b>Autor:</b> Prlj, Dragan <b>Datum slike:</b> 6.4.2011. <b>Datum unosa:</b> 6.4.2011. <b>Tehnika:</b> makro snimak - digitalni aparat <b>Objekt:</b> plod	 <b>Id:</b> 23128 <b>Autor:</b> Čičak, Marina <b>Datum slike:</b> <b>Datum unosa:</b> 23.4.2011. <b>Tehnika:</b> preuzeto iz publikacije <b>Objekt:</b> habitus	 <b>Id:</b> 97252 <b>Autor:</b> Šarić, Šemso <b>Datum slike:</b> <b>Datum unosa:</b> 7.6.2016. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo	 <b>Id:</b> 97253 <b>Autor:</b> Šarić, Šemso <b>Datum slike:</b> <b>Datum unosa:</b> 7.6.2016. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo	 <b>Id:</b> 97255 <b>Autor:</b> Šarić, Šemso <b>Datum slike:</b> <b>Datum unosa:</b> 7.6.2016. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo	 <b>Id:</b> 97256 <b>Autor:</b> Šarić, Šemso <b>Datum slike:</b> <b>Datum unosa:</b> 7.6.2016. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo	 <b>Id:</b> 139829 <b>Autor:</b> Borovečki-Voska, Ljiljana <b>Datum slike:</b> 31.10.2018. <b>Datum unosa:</b> 21.11.2018. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo	 <b>Id:</b> 139840 <b>Autor:</b> Borovečki-Voska, Ljiljana <b>Datum slike:</b> 31.10.2018. <b>Datum unosa:</b> 21.11.2018. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo	 <b>Id:</b> 154623 <b>Autor:</b> Borovečki-Voska, Ljiljana <b>Datum slike:</b> 31.10.2018. <b>Datum unosa:</b> 21.11.2018. <b>Tehnika:</b> fotografija - digitalni aparat <b>Objekt:</b> ostalo
--	---	---	---	---	---	---	---	---

**FER-UNIZG - Razvoj primjenjene programske potpore 2022./2023.**

← → C ⌂ visiani.botanic.hr/fcd-... ⌂ ⌂ ⌂

  
**Flora Croatia Galerija**

**Kriterij pretrage:** Vaskularna flora, Porodica: Liliaceae, Javno: Da  
**Rezultati pretrage:** Broj fotografija: 1036 , Broj svojti: 33 , Broj autora: 43  
Broj zapisa po stranici

1 2 3 4 >

## ► **Danae racemosa (L.) Moench**



 **Id:** 150022

**Autor:** Borovečki-Voska, Ljiljana

**Datum slike:** 31.10.2018.

**Datum unosa:** 21.11.2018.

**Tehnika:** fotografija - digitalni aparat

**Objekt:** habitus



 **Id:** 150023

**Autor:** Borovečki-Voska, Ljiljana

**Datum slike:** 31.10.2018.

**Datum unosa:** 21.11.2018.

**Tehnika:** fotografija - digitalni aparat

**Objekt:** plod



 **Id:** 150021

**Autor:** Borovečki-Voska, Ljiljana

**Datum slike:** 31.10.2018.

**Datum unosa:** 21.11.2018.

**Tehnika:** fotografija - digitalni aparat

**Objekt:** habitus

# Debeli klijenti i višeslojne aplikacije

- Ovisno o namjeni i debeli klijenti i višeslojne aplikacije

The screenshot displays a complex multi-layered application interface titled "IT produkt katalog". The top navigation bar includes tabs for "Produkt katalog", "SeCO", "CCA", "LRIC", "Izvještaji", and "Baza". Below the tabs are several icons and links: "Indikatori proizvoda", "Tipovi naplate", "Indikatori", "Proizvod", "Kategorije", "Klase imovine (vijek imovine)", "FAR", "Stavke Opexa", "FTE", "CAPEX plan", "OPEX", "Capex - Kateg...", "Neaktivirana imovina", "Neaktivirana imovina - Kategorije i proizvodi", "IT produkt podaci detaljno (FAR, FTE, ...)", "Cijene proizvoda", and "Šiframnici", "Matični", "Dokumenti", "PIS - Prebac", "Demo", "Opcije", "Prozor", "Izlaz", and "Izvještaji". A central toolbar features various icons for file operations like save, print, and search.

The main content area is titled "Prologis ERP - Razvoj" and contains a form for "VrstaShemeKnjizenja: Obracun PDV". The form fields include:

- Šifra sheme knjiženja: 702000 (Obracun PDV)
- Vrsta entiteta: 702 Obračun PDV-a
- Kontni plan: 1 RRIF
- Grupa pozicije: 4 obračun PDV i PDV-K
- Vr. poreznog dokumenta: 14 Ulazni račun, R1
- Vr. plaćanja virmanom: 0 Ne plaća se
- Opis: Obracun PDV
- Ime ograničenu primjenu
- Aktivan

Below the form is a section titled "Dovucite stupac kako bi ste grupirali podatke po tom stupcu" (Drag the column header to group by that column). It shows a grid of columns with headers: Rbr, Vrsta pozli..., ..., Ds, Ps, P, K, Z, O, Algoritam, Kontrola, Rbrl, U. The first row of data is:

1	4194	Neupla...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	=P4061+P4066+P4071+P407...	=	1	<input type="checkbox"/>
2	4192	Više u...	0	...	1...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	=P4170+P4160+P4174+P418...	=	2	<input type="checkbox"/>
4	4004	Isporu...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	4	<input type="checkbox"/>
3	4000	Isporu...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	3	<input type="checkbox"/>
5	4002	Isporu...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	5	<input type="checkbox"/>
6	4006	Isporu...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	6	<input type="checkbox"/>
7	4010	R1 isp...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	7	<input type="checkbox"/>
9	4020	R2 isp...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	9	<input type="checkbox"/>
8	4015	Preduj...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	8	<input type="checkbox"/>
10	4025	Nezar...	0	...	1...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	=	=	10	<input type="checkbox"/>

At the bottom left, it says "Prikazano zapisa: 228 / Ukupno zapisa: 228". The footer of the application reads "FER-UNIZG - Razvoj primijenjene programske potpore".

# Sustavi više međusobno povezanih aplikacija

- Različiti izvori podataka (OPC server, baza podataka, konfiguracijske datoteke, ...) + specifičan žargon domene

Control Recipe Editor (CRE)

01 : 005 - Milling

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

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

6-1056 Milling - Local malt intake X

#	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	

Flowchart steps:

- 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

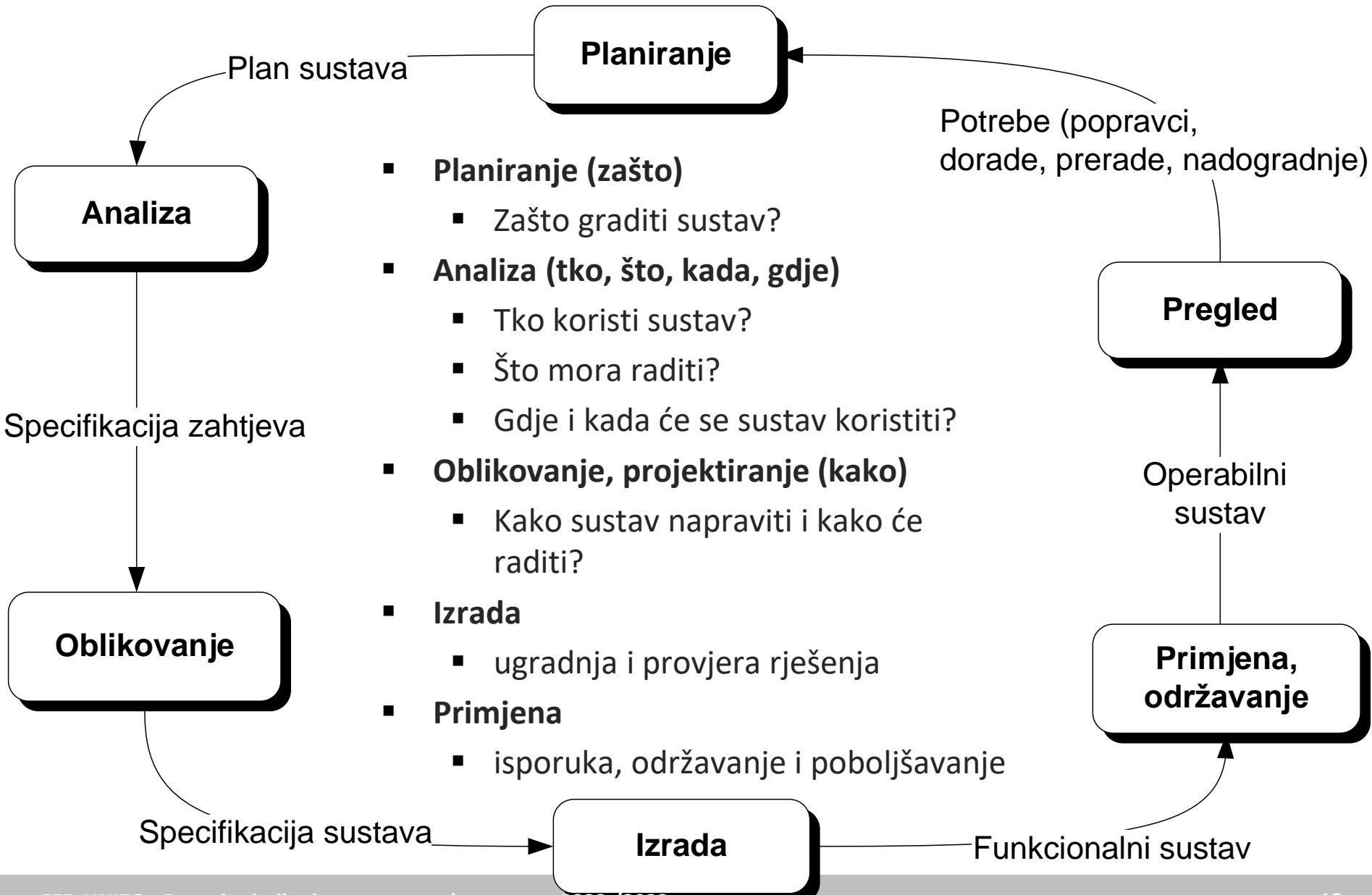
# Fundamentalne aktivnosti razvoja softvera

- Izrada aplikacije nije samo kodiranje.
- Sommerville ističe 4 fundamentalne aktivnosti u svakom procesu razvoja softvera:
  1. Specificiranje
    - Definiranje funkcionalnosti softvera i ograničenja pri radu
  2. Dizajn i implementacija u skladu sa specifikacijama
  3. Validacija
    - ispunjava li proizvod ono što je korisnik htio
    - Validacija (*Are we building the right product*) ≠ Verifikacija (*Are we building the product right*)
  4. Evolucija
    - Izmjene u skladu s korisničkim potrebama.

# Systems/Software Development Life Cycle (SDLC)

- [Dennis, Wixom, Tegarden 2015] definiraju životni ciklus (Systems Development Life Cycle) kroz 4 fundamentalne faze
  1. Planiranje
    - definiranje funkcionalnosti i ograničenja
  2. Analiza
    - analiza postojećeg sustava, identificiranje zahtjeva i mogućnosti unaprjeđenja, izrada koncepta novog sustava (as is → to be)
  3. Dizajn
    - definira kako će sustav raditi, definiranje arhitekture i sučelja
    - sistemska specifikacija
  4. Implementacija
    - konstrukcija, instalacija, poduka, plan podrške

# Životni ciklus razvoja softvera (SDLC)



# 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

# 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.

# Neke tipične 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

# 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
    - dokument koji opisuje sveukupnu organizaciju projekta uključujući hijerarhiju zadataka
    - Vlastiti predložak (u zadaći) ili  
**IEEE Standard for Software Project Management Plans 1058-1998**
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

# Resursi

- Resursi - sredstva
  - Ijudi, oprema i materijal potrebni za obavljanje zadataka
- Vrste resursa:
  - Resursi rada (work resources)
    - Ijudi (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

# 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

# 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

# Izrada plana projekta

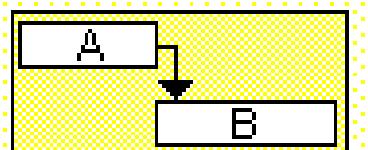
- Koraci izrade plana projekta
  - Izrada liste zadataka
  - Izrada hijerarhije zadataka (work breakdown structure)
  - Procjena trajanja zadataka
  - Izrada ovisnosti među zadacima
  - Dodjela resursa

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

# Međuzavisnost zadataka

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

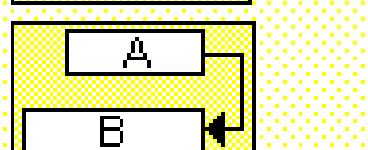
Finish-to-start (FS)



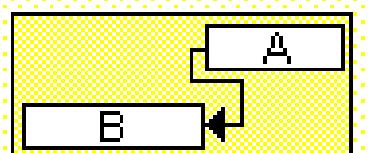
Start-to-start (SS)



Finish-to-finish (FF)



Start-to-finish (SF)



- 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

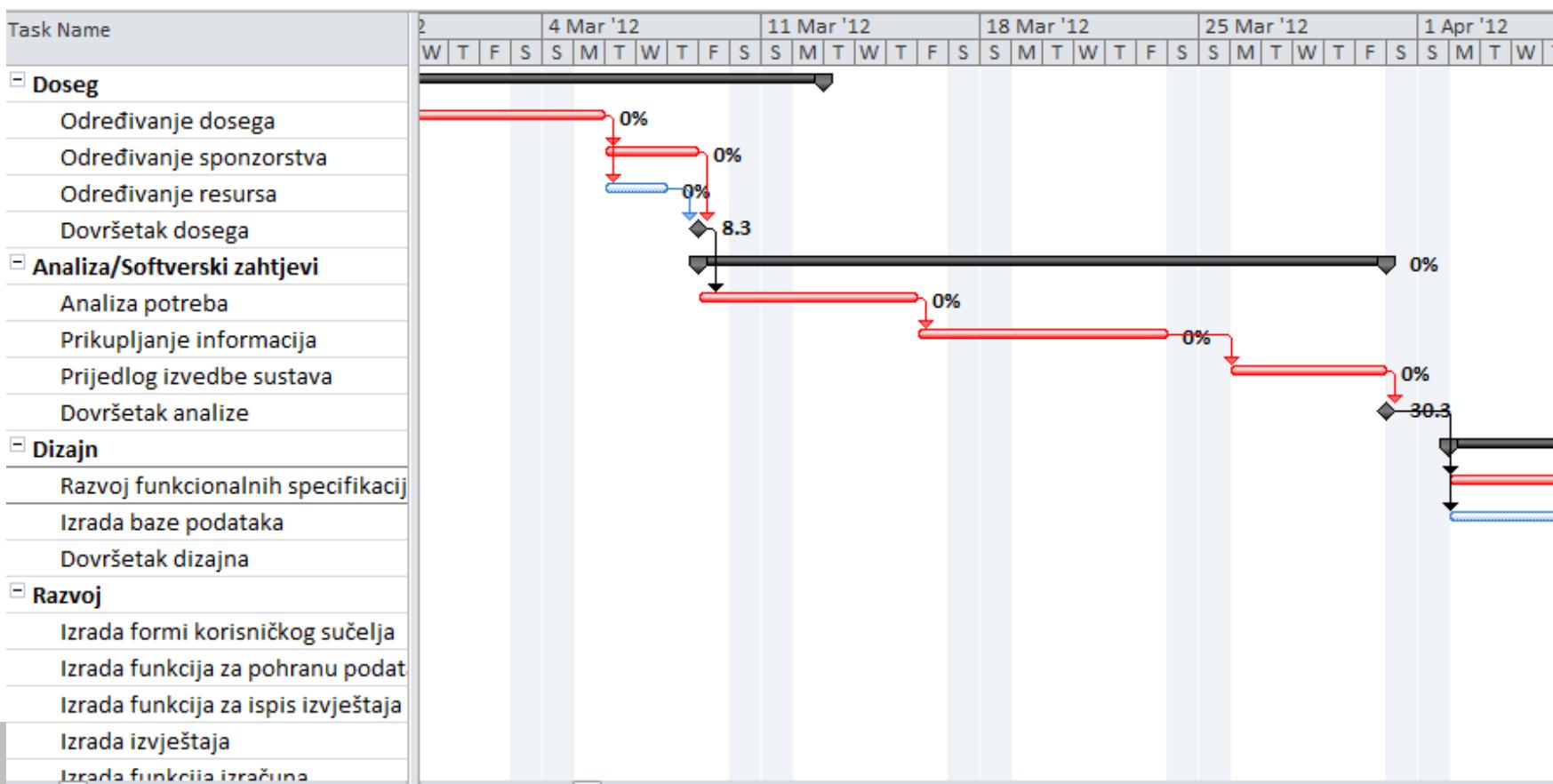
# Procjena trajanja zadataka

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

Zadatak	Trajanje (u daniма)	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	22		Voditelj projekta
5 Dovršetak dosega	03;4		
6 Analiza/Softverski zahtjevi	16		
7 Analiza potreba	55		Sistem analitičar
8 Prikupljanje informacija	67		Sistem analitičar
9 Prijedlog izvedbe sustava	58		Sistem analitičar
10 Dovršetak analize	09		
11 Dizajn	5		
12 Razvoj funkcionalnih specifikacija	510		Sistem analitičar;Projektant
13 Izrada baze podataka	510		Administrator baze podataka
14 Dovršetak dizajna	013;12		

# 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



# Napor i trajanje

- **Razlikovati posao (napor) od vremena (trajanja)**
- Napor (engl. effort, work) – količina radnih jedinica potrebna da bi se dovršio neki zadatak
  - Obično se izražava u čovjek-satima, čovjek-danima, čovjek-mjesecima, ... (engl. man-month)
- Trajanje je vrijeme potrebno da se posao obavi u skladu s kalendarom rada i raspoloživosti resursa koji su potrebni za zadatak
  - Mjeri se u satima, danima, mjesecima, godinama, ...
- Trajanje = Posao / Jedinice (Duration = Work / Units)
  - Teoretski, posao od 6 čm mogu obaviti
    - 2 osobe za 3 mjeseca
    - 6 osoba za 1 mjesec
    - 1 osoba za 6 mjeseci

# Procjena napora

- Napor se procjenjuje
  - u fazi pripreme (prijedloga) projekta
    - neizbjježno spekulativno zbog nepotpunih zahtjeva
  - tijekom početne faze projekta i prikupljanja zahtjeva
  - periodički za vrijeme trajanja projekta
- Tehnike procjene napora
  - bazirane na iskustvu
    - procjene stručnjaka, analogija s prethodnim projektima
  - algoritamski modeli
    - različiti algoritmi bazirani na broju ekrana, tablica, broju zahtjeva, ...

# Raspodjela resursa

- Unos resursa i pratećih podataka (dostupnost, trošak)
- Trajanje kao posljedica procijenjenog napora i pridijeljenih resursa
- Fiksirani napor i vrijeme mogu dovesti da trebamo više od 100% jedinica
- Maksimalne jedinice (max. units)
  - prikazuju vrijednosti raspoloživosti resursa u postocima
  - 100% predstavlja jednog čovjeka punog radnog vremena
  - 300% predstavlja tri čovjeka punog radnog vremena
- 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
- Ovisno o tome što je fiksno, izračun se može „popraviti”
  - pomicanjem rokova (trajanja), dodavanjem novih resursa, prodljivanjem radnog vremena, povećanjem jedinica posla, smanjivanjem količine posla,  
...

# Klasični problemi i moguća rješenja

- Pretjerani optimizam
  - Planirati vremensku zalihu na kraju svake faze, ali ne napuhivati procjene
- Nemogućnost praćenja napretka
  - pratiti napredak / redovno izvještavati o napretku (npr. na tjednoj bazi)
  - Kašnjenje je OK, lažna izvješća nisu
- Neažuriranje rasporeda
  - U slučaju kašnjenja revidirati raspored ili smanjiti funkcionalnosti
- Dodavanje novih osoba na projekt koji kasne
  - Fred Brooks: „*adding manpower to a late software project makes it later*“ The Mythical Man-Month, 1975.
  - Segmentirati dijelove posla kako bi nova osoba mogla pridonijeti napretku

# Upravljanje projektom (Project management)

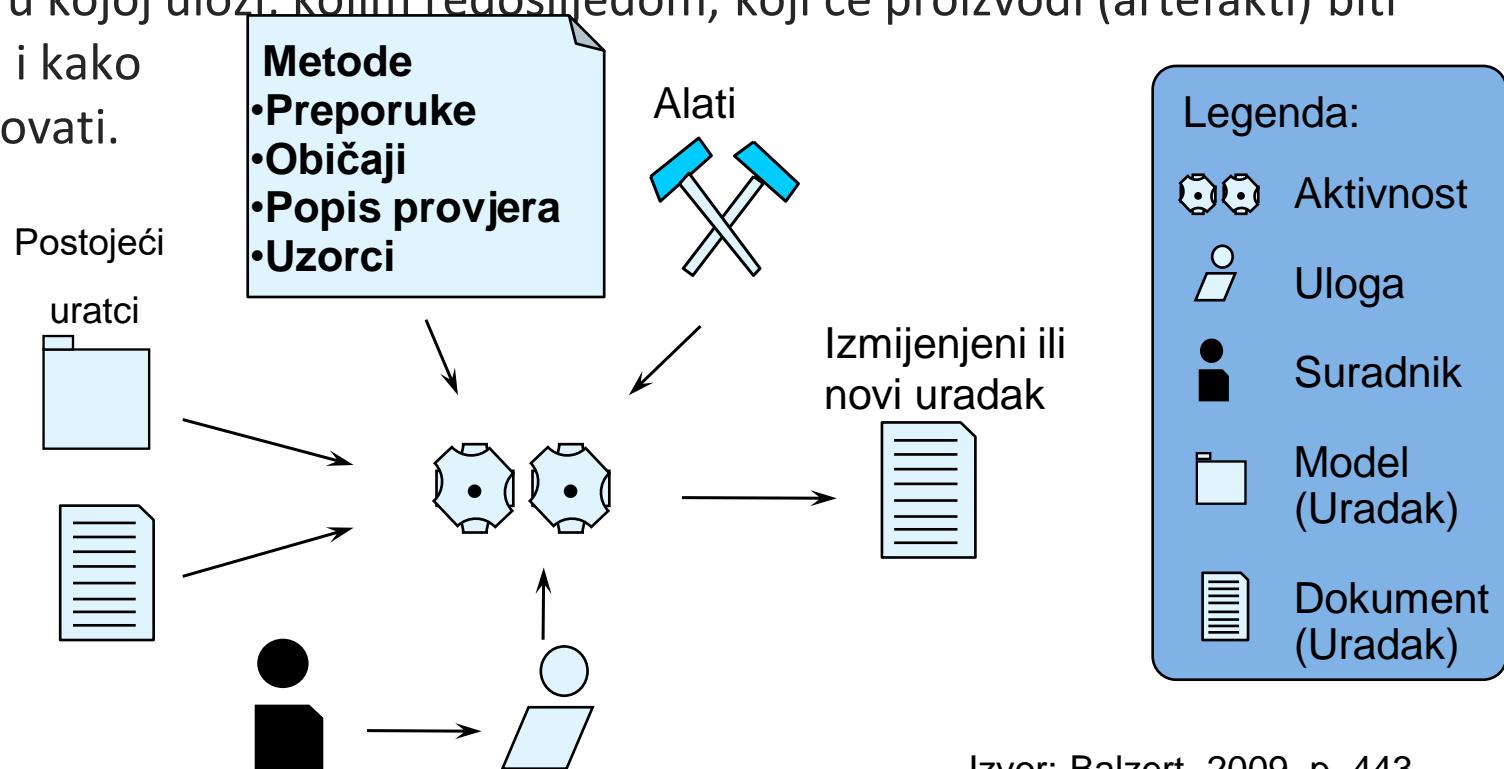
*Upravljanje projekta je primjena znanja, vještina, alata i tehnika u projektnim aktivnostima da bi se ispunili projektni zahtjevi. [PMI]*

- 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

# Modeli i metode razvoja

# Softverski proces

- Proces razvoja sastoji se od skup aktivnosti i uradaka (artefakata)
- Predvidljiv plan prilagođen potrebama omogućava organizaciju aktivnosti i pruža stabilnost i kontrolu
  - 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: koiim redoslijedom, koji će proizvodi (artefakti) biti razvijeni i kako ih vrednovati.



Izvor: Balzert, 2009, p. 443

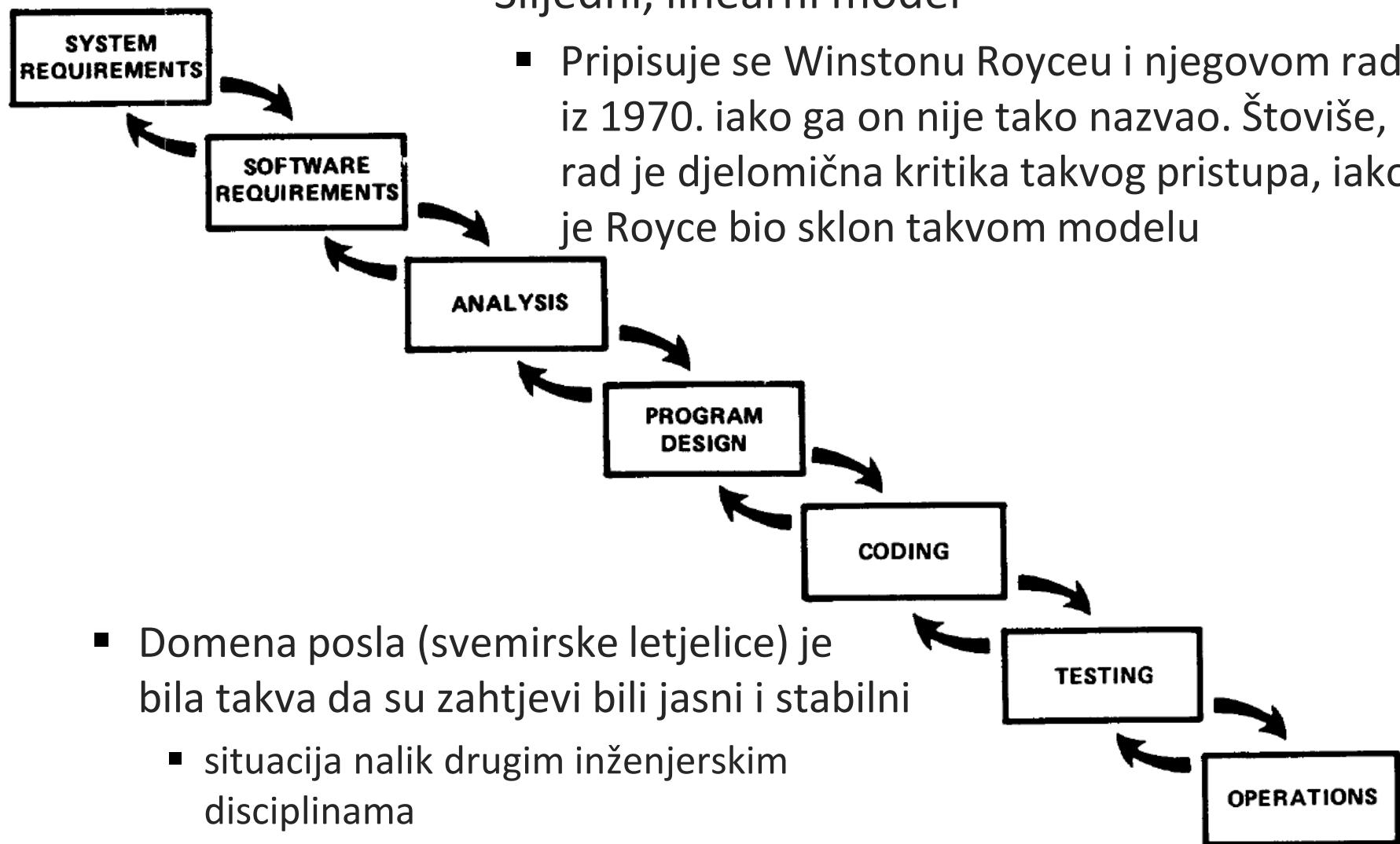
# Zašto je potreban model procesa?

- Bitan za organizaciju projekta
  - U protivnom je upravljanje projektom sporadično i nekoordinirano
  - Omogućava vremensko i finansijsko planiranje te standardizaciju rada
  - Pruža „kontrolne točke“ za praćenje napretka, procjenu postignutih rezultata i donošenje odluka o dalnjim koracima
- Model predstavlja apstrakciju životnog ciklusa razvoja softvera
- Artefakti: dokumenti, modeli, programi, ...
  - Artefakti i ekipe određene pojedinim modelom, alatom i/ili metodologijom rada
- Odabir modela ne isključuje/uključuje pojedine inženjerske prakse
  - Dva isprepletena aspekta: inženjerske prakse i upravljačke tehnike

# Modeli, metode, metodologije, razvojni okviri, paradigme

- engl. *model, method, methodology, framework, paradigm*
  - prilično nekonzistentno upotrebljavano, a ponekad i kontradiktorno
- Model softverskog procesa je apstrakcija (pojednostavljeni prikaz, paradigma) procesa razvoja softvera s ciljem da se propiše strategija razvoja, odnosno način odvijanja fundamentalnih aktivnosti razvoja softvera
  - „što i kojim redoslijedom”
- Metoda razvoja softvera konkretizira pojedini model dalnjom razradom aktivnosti, uvođenjem specifične terminologije, propisivanjem stila i organizacije rada te artefakata koji nastaju u procesu
  - Metoda pruža tehničko rješenje „kako napraviti pojedini korak iz modela”
- Metodologija = metoda + idejni pristup, tj. teoretski okvir za podršku odabranim metodama
  - Hrvatski jezični portal: „znanost o metodama, sustav metoda i načela koji se koriste u nekoj znanstvenoj disciplini, znanosti ili znanstvenoj grani”

# Vodopadni model (engl. waterfall)



# Problemi slijednog (vodopadnog) modela

- Kod ostalih inženjerskih disciplina ili kod razvoja hardvera, ovaj pristup ima smisla, ali korisnik ne može uvijek izraziti sve zahtjeve na početku
- Radna verzija sustava neće biti gotova do samog kraja projekta
  - Pogreška u ranoj fazi zbog krive analize, dizajna mogu imati značajne (pa i katastrofalne) posljedice za projekt uslijed izgubljenog vremena i nastalih troškova
- Manji koraci unatrag mogući, ali zahtijevaju promjenu formalno usvojenih dokumenata i usporavaju cijeli proces
  - Otpor prema promjenama
  - Tko je kriv? Programer koji ne može isprogramirati nešto ili analitičar koji je specificirao nemoguće?

# Kako se nositi s promjenama?

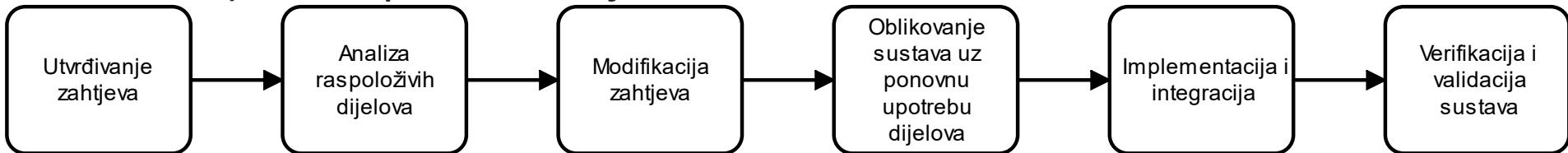
- Često korisnik okvirno zna što želi, ali ne može razraditi detalje
- Razvija se polazna verzija za demonstraciju. Temeljem povratnih informacija polazna verzija se poboljšava, opet pokazuje itd...
  - Analiza, dizajn i implementacija se rade istovremeno, a zahtjevi se otkrivaju i validiraju
- Pokazne verzije mogu biti prototipovi čija je svrha istraživanje zahtjeva – ***prototipiranje***
  - prototip može biti takav da imitira funkcionalnost
    - česta praksa kako bi se ustavio izgled i tok ekrana (*wireframing*)
  - implementacija se može obaviti modelom vodopada
  - Što ako je prototip izgrađen? Odbaciti ili nadograditi? Performance i kvaliteta prototipa? Kompromisi prilikom razvoja?
- Verzije mogu evoluirati u konačni sustav – ***evolucijski razvoj*** koji se naziva ***istraživačko programiranje***

# Model inkrementalnog razvoja

- Prednosti evolucijskog razvoja:
  - Brzi odgovori na zahtjeve korisnika
  - Moguć razvoj s nejasnim početnim zahtjevima
  - Specifikacija zahtjeva se s vremenom dorađuje
- Mane evolucijskog razvoja:
  - Nejasno kad bi sustav mogao biti gotov
  - Težak za naknadno održavanje zbog čestih promjena i dorada, obično loše strukturiran
- ***Model inkrementalnog razvoja*** je sličan evolucijskom razvoju, ali ponavljujući korak ne predstavlja doradu postojećeg sustava, već dodavanje novog dijela – inkrementa
  - Razvoj inkrementa može biti obavljen po različitim modelima
  - Prednosti: lakše praćenje napretka (po inkrementima), korisnici brzo mogu dobiti malu funkcionalnu verziju koja zadovoljava potrebe

# Model usmjeren na ponovnu upotrebu

- Zasniva se na pretpostavci da postoje gotove komponente (dijelovi iz ranijih sustava ili javno dostupne komponente, npr. web-servisi itd.) koje treba spojiti i konfigurirati
- Model se (prema Robert Manger: „Softversko inženjerstvo”, 2016.) može prikazati sljedećim koracima



- Implementacija može uključivati i adaptaciju postojećih komponenti prije integracije
- Prednosti:
  - Brži i jeftiniji razvoj s manjim rizikom
- Mane:
  - Kompromisi u slučaju da sustav ne odgovara izvornim potrebama korisnika te gubitak kontrole nad evolucijom sustava

# Modeli formalnog i grafičkog razvoja

- Model formalnog razvoja (engl. *formal systems development*) oslanja se na matematičku notaciju za izradu specifikacije i transformaciju u konačni program
  - Problem izrade formalne specifikacije: osim za ciljane namjene, izrazito nepraktično, potreba za specifičnim znanjem, nedostatak upotrebljivih jezika
- Model grafičkog razvoja (engl. *model driven engineering*) koristi grafičku notaciju (npr. UML) kojom se mogu opisati dijelovi i ponašanje sustava
  - Strukturni dijagrami prikazuju podatke i statičke odnose u sustavu
    - obično dijagrami klasa, paketa, komponenti i razmještaja
  - Dijagrami ponašanja prikazuju interakciju između dijelova sustava
    - obično dijagrami aktivnosti, slijeda i slučajeva korištenja
- Moguće generirati dijelove programskog koda

# Pristupi razvoju softvera

- Generalna podjela metoda na klasične („planske“) i agilne
- Klasične stavlju naglasak na precizan plan i urednu dokumentaciju svih aktivnosti
- Moderniji pristup – agilne metode
  - Scrum i ekstremno programiranje (XP) među najpoznatijima
  - Prikladne za sustave sa slabo definiranim i/ili čestim promjenama zahtjeva, naglasak na čestim izdanjima i interakciji s korisnicima
  - Aktualni trend i dominantne na tržištu, ali ne bez kritičara
    - za Ivara Jacobsona (UML, Rational Unified Process (RUP)) korak natrag iz inženjerstva prema zanatu
      - Nudi *Essence* kao rješenje
  - Odmak od krutih prethodnih modela
    - Dokumentacija često u obliku kartica s korisničkim pričama uz upotrebu stvarnih ili virtualnih ploča sa statusima
    - Promjene su lakše ako ne mora baš sve biti formalno zapisano

# Agilni razvoj – koncept, a ne model

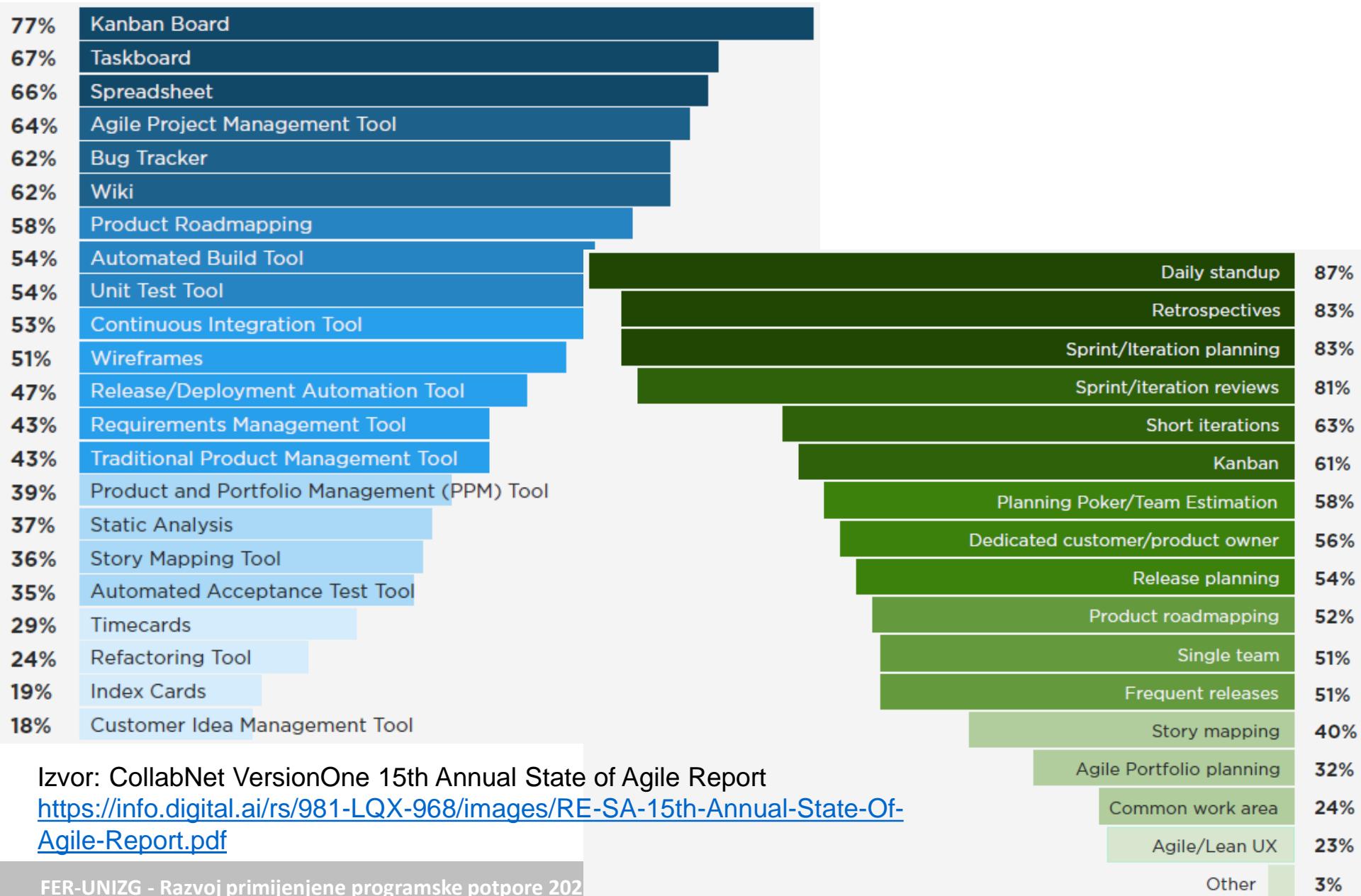
- Posljedica frustracija u 90.-tim godinama uslijed raskoraka između potreba i mogućnosti isporuke kvalitetnog softvera na vrijeme
- Krovni pojam nastao 2001. za različite metode zasnovane na 4 temeljne vrijednosti
  - *Ijudi i njihovi međusobni odnosi su važniji nego procesi i alati*
  - *upotrebljiv softver je važniji od iscrpne dokumentacije*
  - *suradnju s naručiteljem je važnija od pregovaranja oko ugovora*
  - *reagiranje na promjenu je bolje nego ustrajanje na planu*

i 12 principa iza navedenih temeljnih vrijednosti

<https://agilemanifesto.org/principles.html>

- Popis pojmove u agilnom razvoju:  
<https://www.agilealliance.org/agile101/agile-glossary>

# Korištenje agilnih alata i praksi

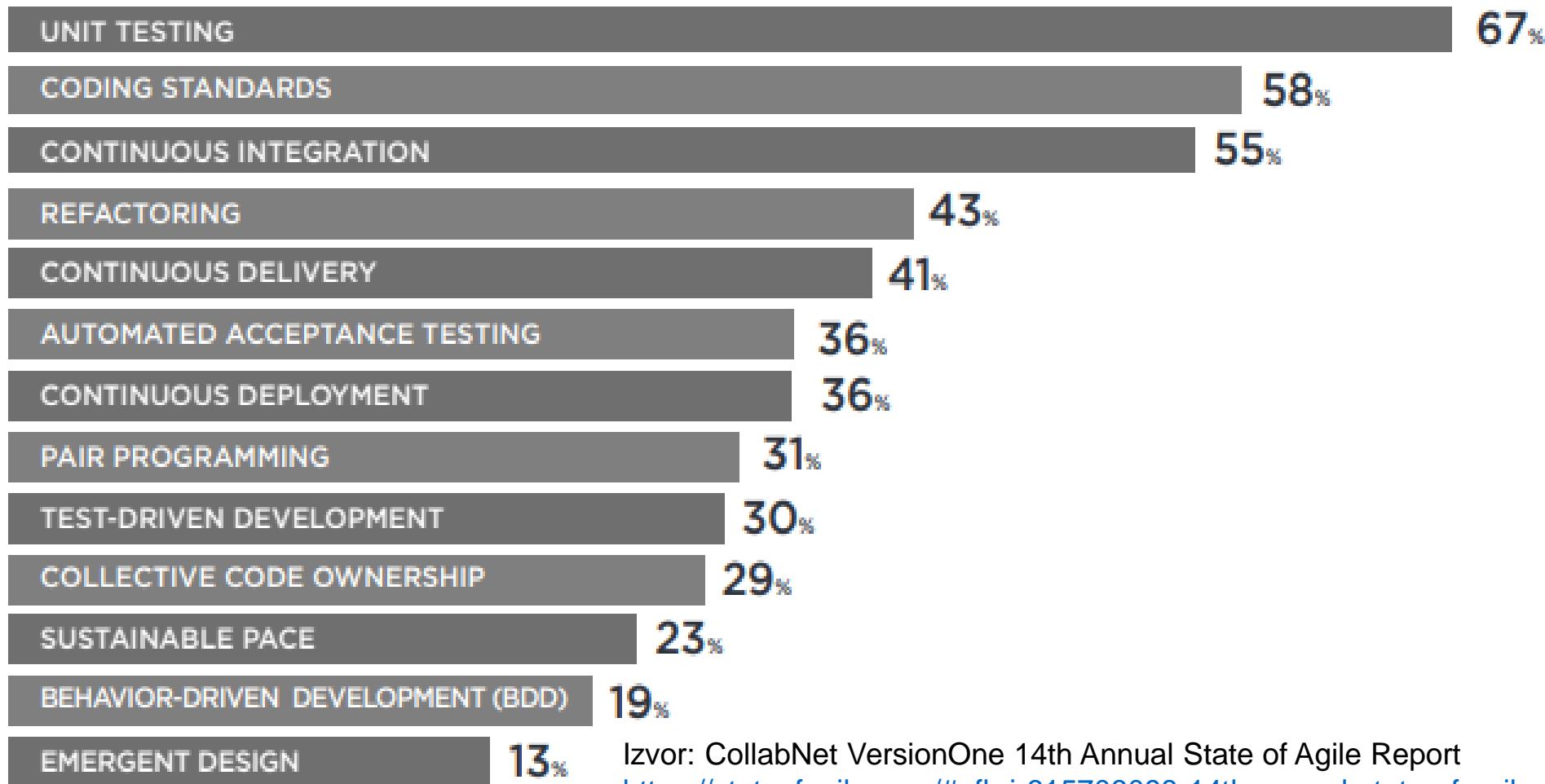


Izvor: CollabNet VersionOne 15th Annual State of Agile Report

<https://info.digital.ai/rs/981-LQX-968/images/RE-SA-15th-Annual-State-Of-Agile-Report.pdf>

# Korištenje inženjerskih praksi

- O nekim od navedenih inženjerskih praksi detaljnije naknadno
  - Promovirane kroz agilne metode, ali nisu nužno vezane za njih



Izvor: CollabNet VersionOne 14th Annual State of Agile Report  
<https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494>

# Ploče sa statusima zadataka

- Kartice predstavljaju elemente podijeljene po stupcima koji predstavljaju određeno stanje
  - Stupci određeni pojedinom metodom/okvirom ili proizvoljno
- Nazivaju se još i kanban ploče po uzoru na metodu u kojoj su originalno primijenjene
- Originalno koncept u kojem se zadaci **preuzimaju** tako da broj aktivnih zadataka (WiP - Work in Progress) nikad ne premaši limit
  - Olakšava vizualizaciju posla, utvrđuje limit aktivnih zadataka i smanjuje vrijeme do isporuke
  - Proces teče glatko, a blokirajuće se aktivnosti lako uoče
- Nije isključivo vezano za pojedinu metodu, niti za iterativni proces
  - U izvornoj verziji proces je kontinuiran, a u pojedinim metodama se odabir vrši po iteracijama

# Ploče sa statusima zadataka (2)

**CaTA** | Add to team | Private | | Invite | Bitbucket | Butler (2 Tips) | ... Show Menu

Ideas

- Korisnici glasaju na koji izlet žele ići
- Korisnik može predlagati ili glasati za redoslijed putovanja
- Generiranje brošure puta
- Integracija s vanjskim kalendarom (npr. Google calendar)
- Grupa korisnika šalje agenciji prijedlog putovanja
- Sustav šalje podsjetnike na putovanja i plaćanja
- Agencija evidentira uplatu
- Konferencija: sudionik, predavač i organizator
- Sudionici mogu vidjeti predavanja na konferenciji, ocijeniti predavanje i dodati podsjetnik u kalendar
- Organizator može unijeti ponude od agencija
- Organizatori mogu glasati za najbolju ponudu
- Omogućiti pregled putovanja po

To Do

- Modul za administraciju korisnika
- Kreirati web-aplikaciju sa web servisom koji vraća trenutno vrijeme
- + Add another card

In Progress

- Agencija može otakzati putovanje
- Agencija može vidjeti popis prijavljenih korisnika za putovanje
- Korisnici pregledavaju i daju nove sugestije oko autobusa i hotela
- Korisnik unosi recenziju puta
- uvedi swagger
- + Add another card

Pull Request

- + Add a card

Testing

- Složiti docker?
- Preparirati Camundu s nekim trivijalnim procesom
- Generirati kod koji puni bazu s nekoliko inicijalnih podataka
- Razdvojiti entitet Putovanje na 2 entiteta (konkretno putovanje s datumom i općenito putovanje)
- Agencija objavljuje putovanja
- Agencija ažurira putovanja
- Ažurirati verziju na herokuapp
- + Add another card

Done

- Kostur angular klijenta word)
- Popraviti .gitignore
- Integrirati s bitbucket
- Korisnik može otakzati
- Korisnik se prijavljuje :
- Korisnik se može prijava formu za prijavu)
- Uspostaviti hosting
- Preparirati bazu podat
- Korisnik pregledava p putovanja
- + Add another card

# Ploče sa statusima zadataka (3)

Backlog Board



New

Approved

0/5

Committed

21/5

Done

+ New item



Na početnoj stranici ne radi  
ažuriranje/brisanje događaja

Definirati build i deploy za web  
aplikaciju



Boris Milašinović

Login preko FER weba



Luka Jukić

Favoriti, sport i znanje prikazati  
na ekranu, a ne kroz opcije  
menija

Obojane zvjezdice za favorite na  
random mjestima

Uvoz podataka iz Excela

Vizualni prikaz rezultata nekog  
sporta

0/1

< Approved

0/5

Committed

21/5

< Done

Stranica s postavkama za izračun  
kako se boduje i organizira koje  
natjecanje



Tomislav Maslač

Bluetooth prijenos



Vjeran Hanžek

Bluetooth prijenos



Toni Bakarčić

Brisati grupu koja je trenutno  
označena, a ne zadnju.



Leon Hrnjak

Omogućiti unos pozicije tima u  
znanju



Luka Jukić

Prikaz rezultata u sportovima



Skratiti prikaz kod uspjeha  
fakulteta



Toni Bakarčić

Prikaz pozicija u rezultatu u  
znanju ne radi



Tomislav Božurić

Dodati mogućnost da se  
pojedinac u znanju natječe  
individualno



Tomislav Božurić

Sinkronizacija s webom sruši  
aplikaciju



Leon Hrnjak

Aktivirati formu za vijesti



Leon Hrnjak

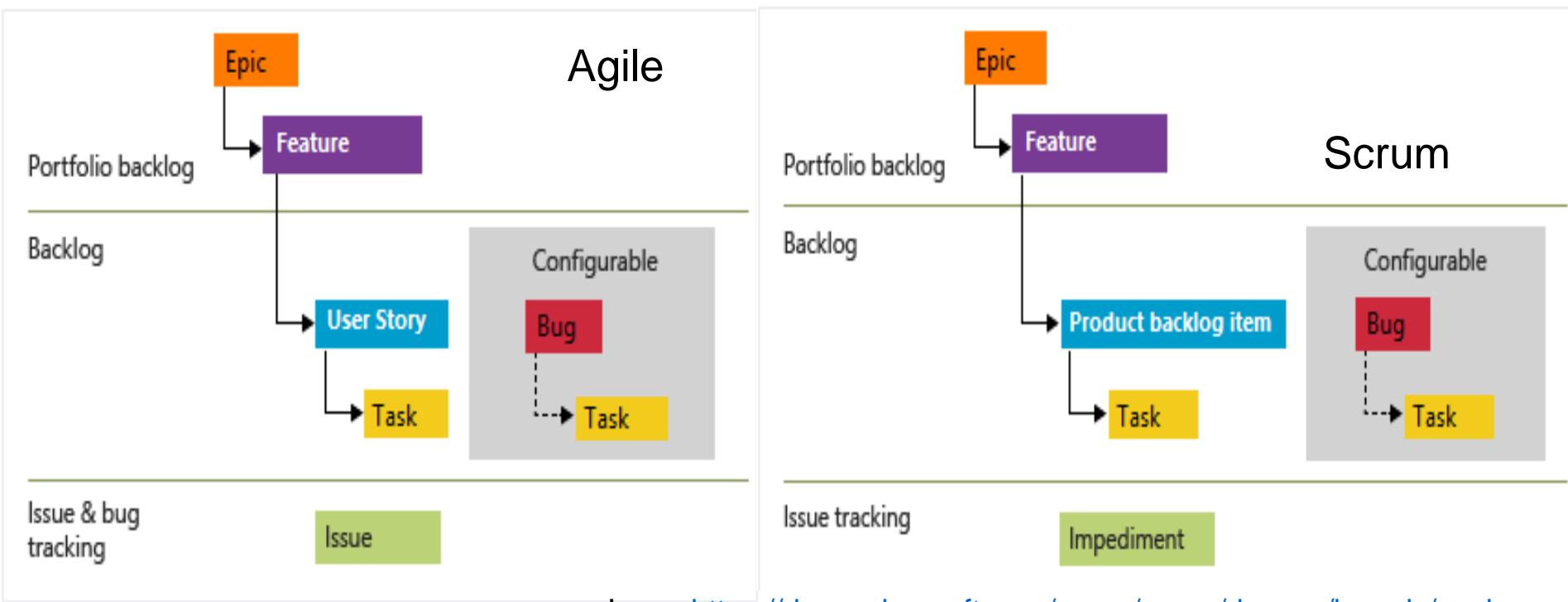
# Stupci na *kanban* pločama

- Podržano u različitim alatima: Jira, Trello, Asana, Azure DevOps, ...
- Stupci određeni pojedinom metodom ili okvirom, npr. na Azure DevOps moguće odabrati
  - Basic: *To Do, Doing, Done*
  - Agile: *New, Active, Resolved, Closed, Removed*
  - Scrum: *New, Approved, Committed, Done, Removed*
  - CMMI: *Proposed, Active, Resolved, Closed*

<https://docs.microsoft.com/en-us/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops>

# Tipovi elemenata na statusnim pločama na primjeru Azure DevOpsa

- Ovisno o odabiru predloška koristi se različita, iako u suština vrlo slična, terminologija i statusi



Izvor: <https://docs.microsoft.com/en-us/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops>

- Backlog = neizvršen rad, preostali posao, lista poželjne funkcionalnosti vidljiva svim dionicima

# Primjer evidencija korisničkih priča, bugova

Backlogs

Queries

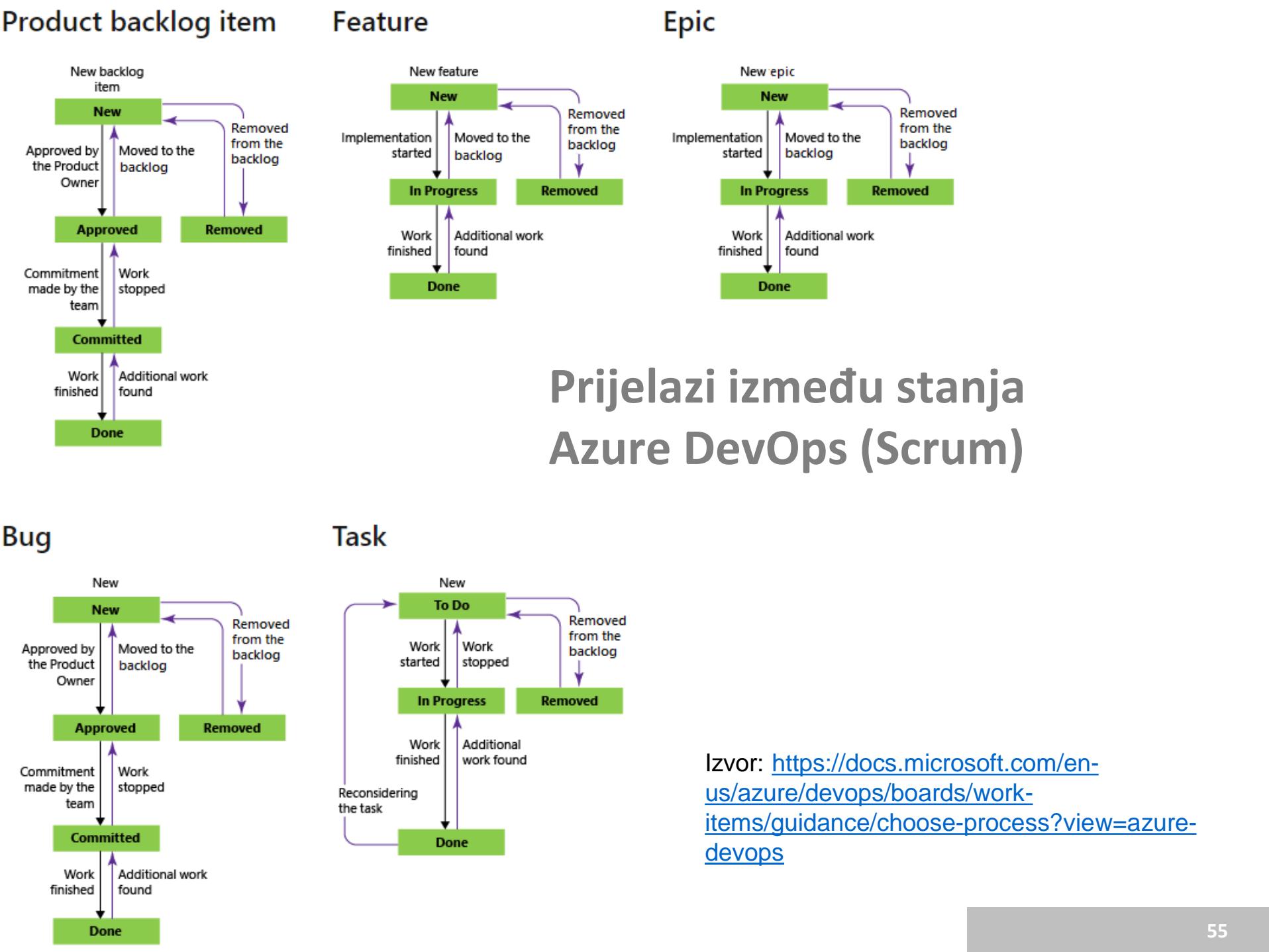
You can now add more backlog levels to this team. Access this setting through the [settings dialog](#).

Features
Backlog items
Past
Sprint 1
Sprint 2
Sprint 3
Bugovi i dorade 1
Current
Bugovi i dorade 2

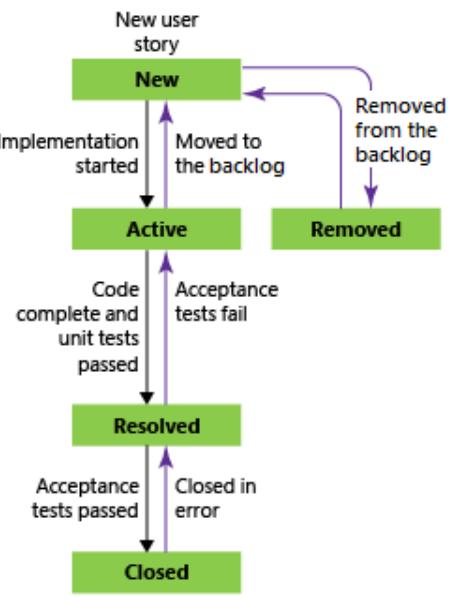
Backlog items

Backlog Board Forecast Off Mapping Off Parents Hide In progress items Show

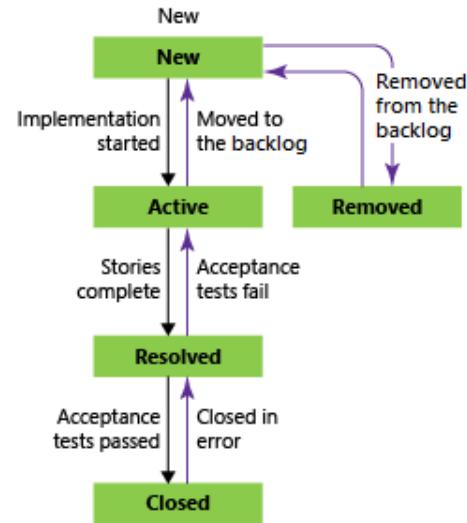
New	Order	Work Item Type	Title	State	Ef Area Path	Iteration Path	Assigned
	19	Bug	Klik na rezultat na počeoj stranici treba ažurirati rezult...	New	Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
	20	Bug	Promjena datuma za natjecanje u znanju 2x pita za dan...	New	Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	
	21	Bug	U postavke za sport dodati koliko bodova nosi pobeda...	Committed	Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrn...
	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 Hrn...
	25	Bug	Dodati u tablici grupne faze broj bodova i po tome sorti...	Committed	Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrn...
	26	Bug	Dodati u postavke sporta tipove komparatora	Committed	Elektrijada\Andr...	Elektrijada\Bugovi i dorade 2	Leon Hrn...
	27	Bug	Ako je nešto u favoritima, onda zvjezdica mora biti obo...	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...



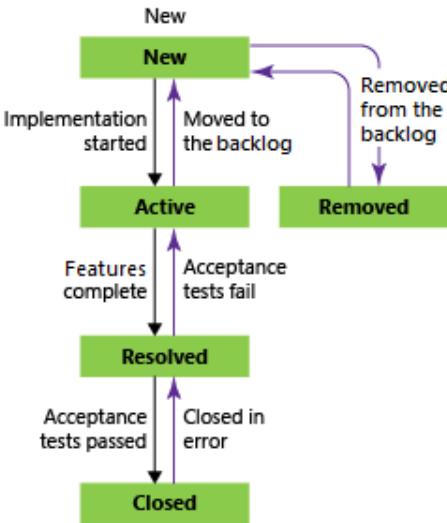
## User story



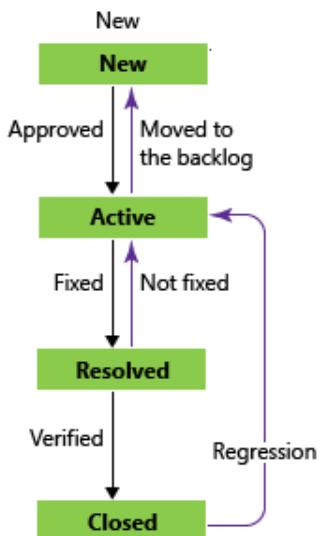
## Feature



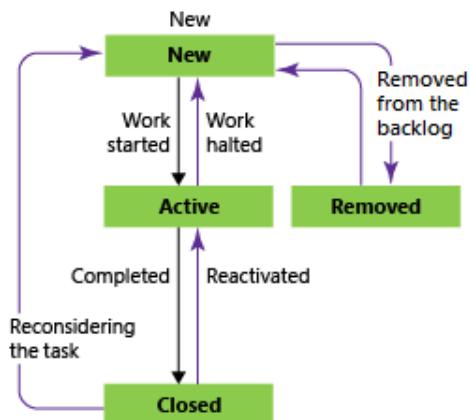
## Epic



## Bug



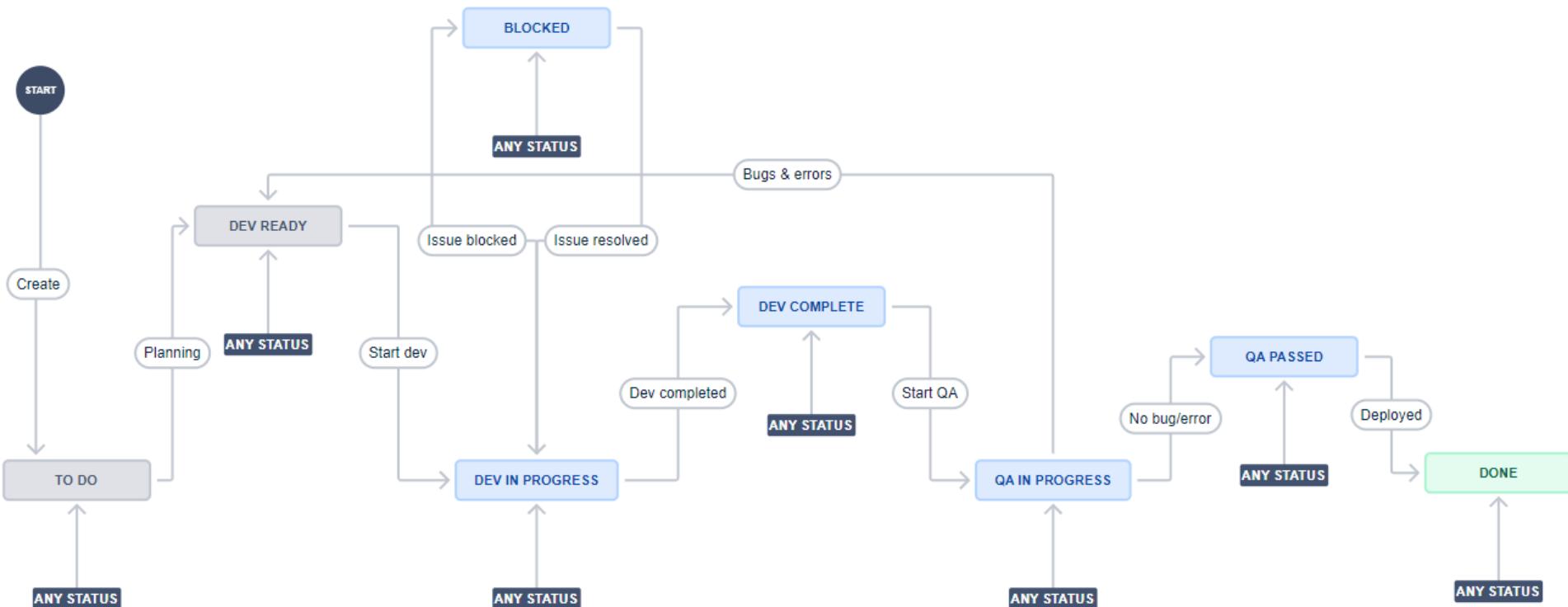
## Task



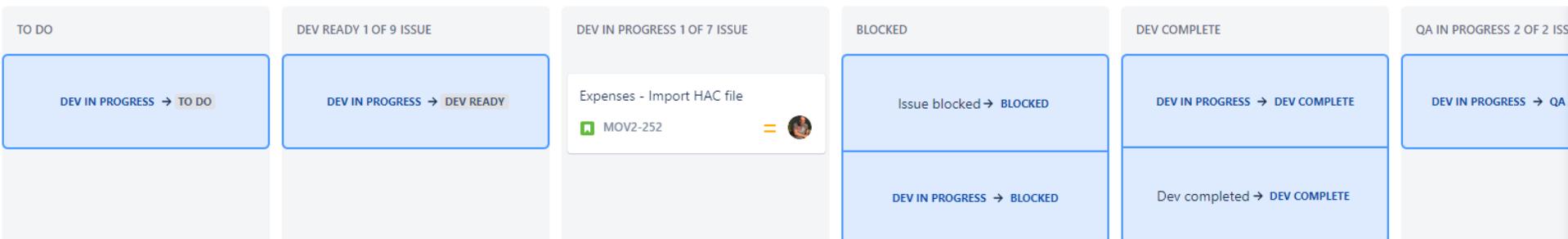
# Prijelazi između stanja Azure DevOps (Agile)

Izvor: <https://docs.microsoft.com/en-us/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops>

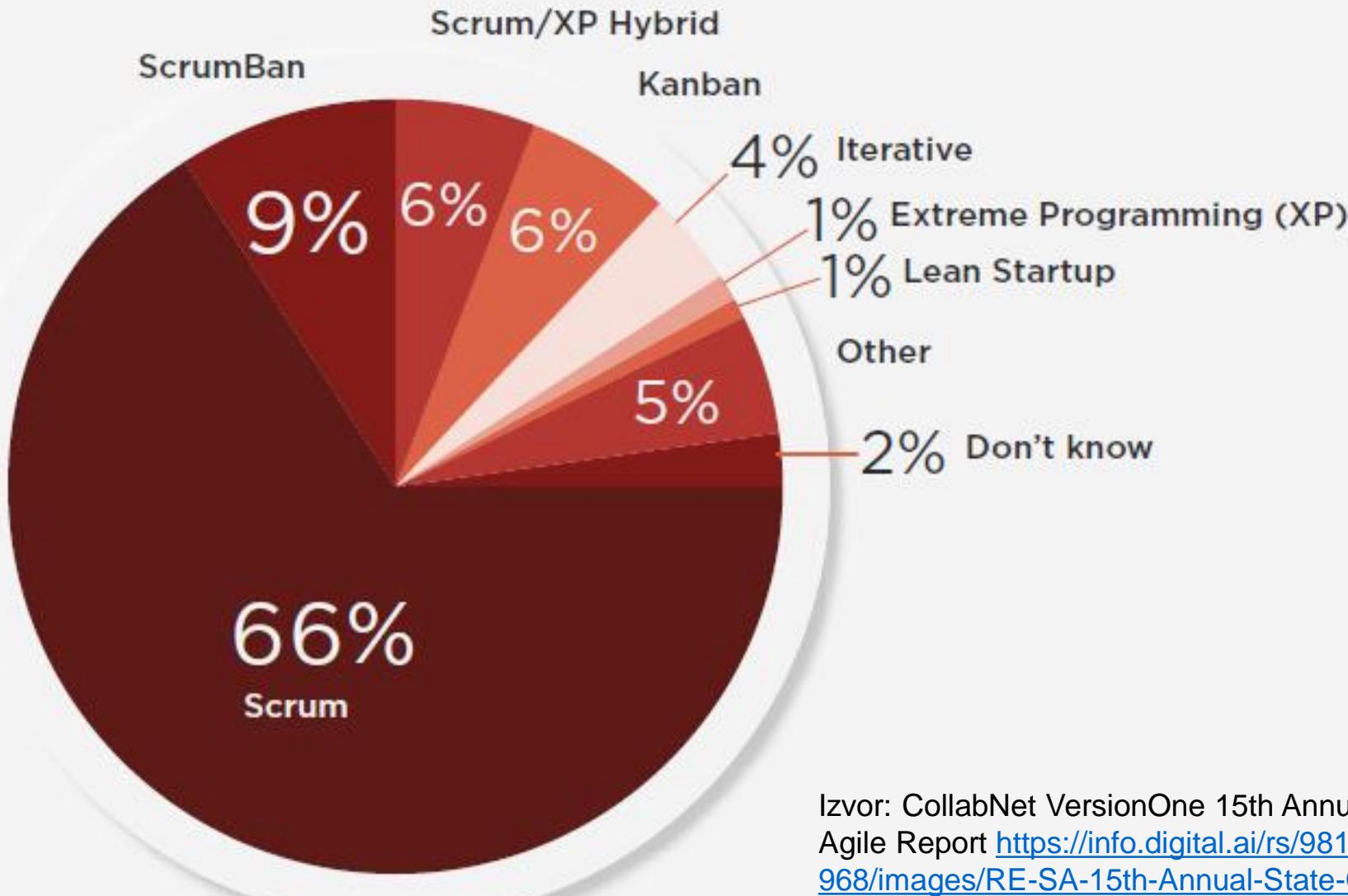
# Definiranje vlastitih pravila prijelaza stanja



- Primjer definiranog radnog toka u Jiri i prijenosa elementa na ploči



# Zastupljenost različitih agilnih metoda



Izvor: CollabNet VersionOne 15th Annual State of Agile Report <https://info.digital.ai/rs/981-LQX-968/images/RE-SA-15th-Annual-State-Of-Agile-Report.pdf>

# Scrum

- Jednostavni upravljački okvir za iterativni i inkrementalni razvoj nastao u ranim devedesetima
  - pristup upravljanju razvojnim procesom, čak i ne nužno razvoju softvera
  - za razliku od npr. XP-a ne propisuje tehničke detalje razvoja
  - cilj je olakšati samoorganizaciju ekipe i adaptaciju pojedinom tipu projekta i promjenama kroz vrijeme
  - iteracija se naziva *sprint*, a rezultira inkrementom proizvoda
- Naziv potječe iz ragbija, gdje se formira skup igrača (engl. *scrum*) za početak igre nakon prekida



Autor slike: Steven Lilley

# Scrum i empirizam

- Scrum temelje zasniva na empirizmu – znanje dolazi iz iskustva, a odluke se donose na temelju ono što je poznato
- Empirizam prema Hrvatskom jezičnom portalu:
  - *shvaćanje da je čovjekov razvoj, osobito psihički, određen prije svega djelovanjem okoline u kojoj živi;*
  - *iskustvo je jedini izvor spoznaje i jedini razlog njegova objektivnog važenja, ničega nema u razumu što prethodno nije bilo u iskustvu*
  - *korištenje empirijskih (iskustvenih) metoda u znanosti*

# Scrum - uloge

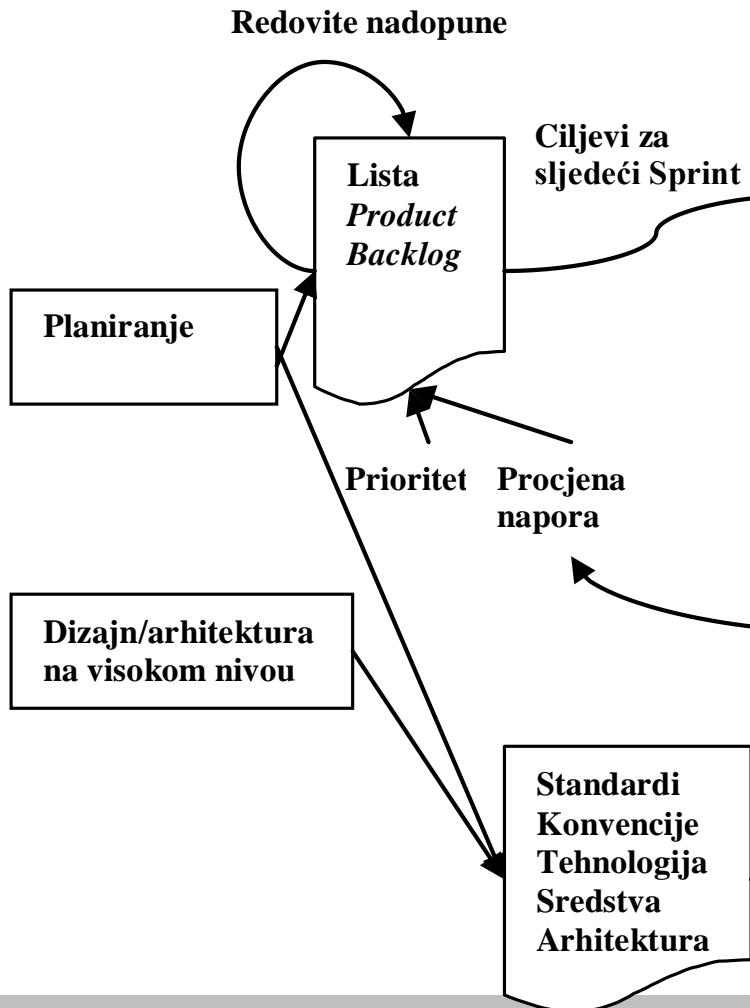
- Vlasnik proizvoda (*Product Owner*)
  - predstavlja sve korisnike
    - razumije domenu i ostvaruje kontakt prema krajnjim korisnicima i sponzorima projekta
  - zadužen za plan, prioritete, troškove i povrat investicije
  - održava *Product Backlog*
- Razvojna ekipa (*Scrum Development Team*, po novijoj terminologiji *Developers*)
  - jedna ili više ekipa od 3 do 9 članova
  - svestrani članovi (*cross-functional*)
  - samoorganizirajuća ekipa (*self-organizing*)
- Majstor (*Scrum Master*)
  - brine o procesu, koordinira, pomaže timu i vlasniku proizvoda u razumijevanju procesa i alata, **ali ne donosi poslovne ni tehničke odluke**

# Scrum - Artefakti

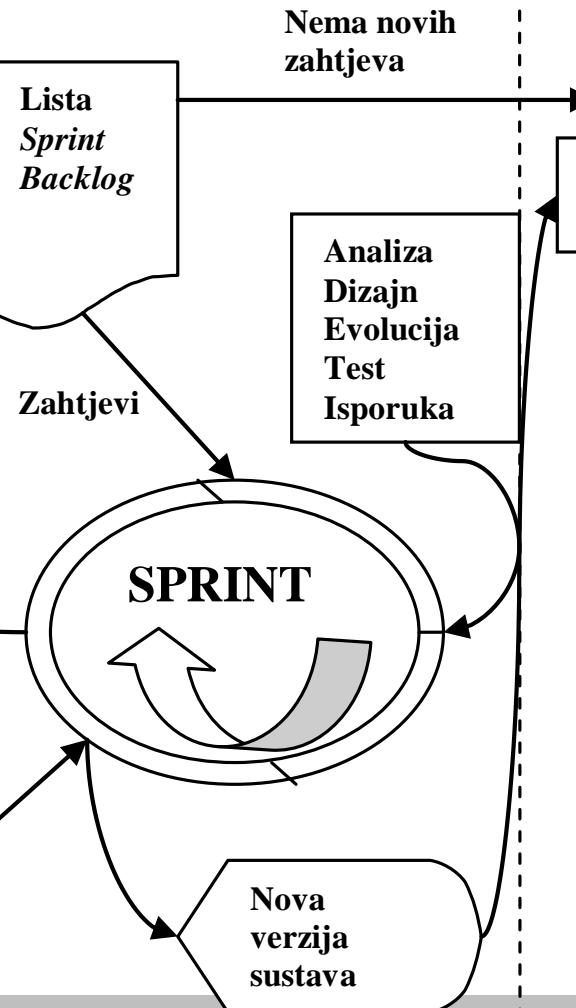
- Product Backlog
  - vidljiv svim dionicima
  - svatko može dodati elemente, ali odgovornost vlasnika proizvoda
- Product Backlog Item - element
  - definira "ŠTO", najčešće kao korisnička priča
  - ima kriterij prihvatljivosti, definiciju „dovršenosti“ – **Definition of done**
    - <https://www.agilealliance.org/glossary/definition-of-done>
  - sadrži više zadataka
  - poslovnu vrijednost odredi Vlasnik
  - napor procijeni Ekipa
- Sprint Backlog
  - popis odabralih priča, razrađenih u zadatke sa statusima
  - ažuriran tokom sprinta
  - uređuje ga samo razvojna ekipa
- Zadatak sprinta (Sprint Task)
  - "KAKO" za PBI "ŠTO"
  - dan posla ili manje
  - preostali napor procjenjuje se dnevno u satima
- Inkrement
  - Suma svih dovršenih elemenata tijekom sprinta
  - Vlasnik odlučuje hoće li se isporučiti ili ne, ali mora moći biti isporučiv

# Životni ciklus Scruma (1)

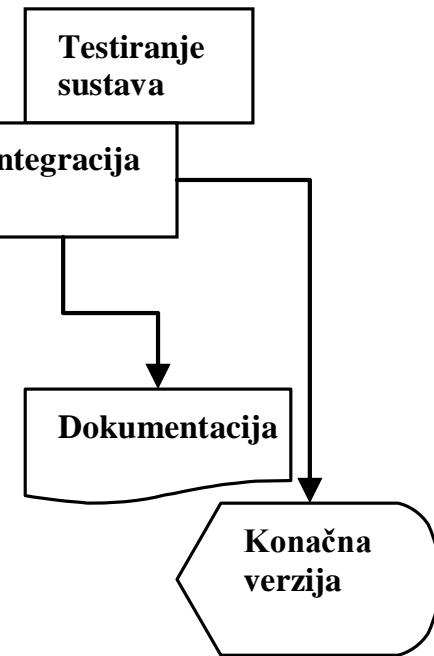
## Prije igre



## Razvoj (Igra)



## Poslije igre

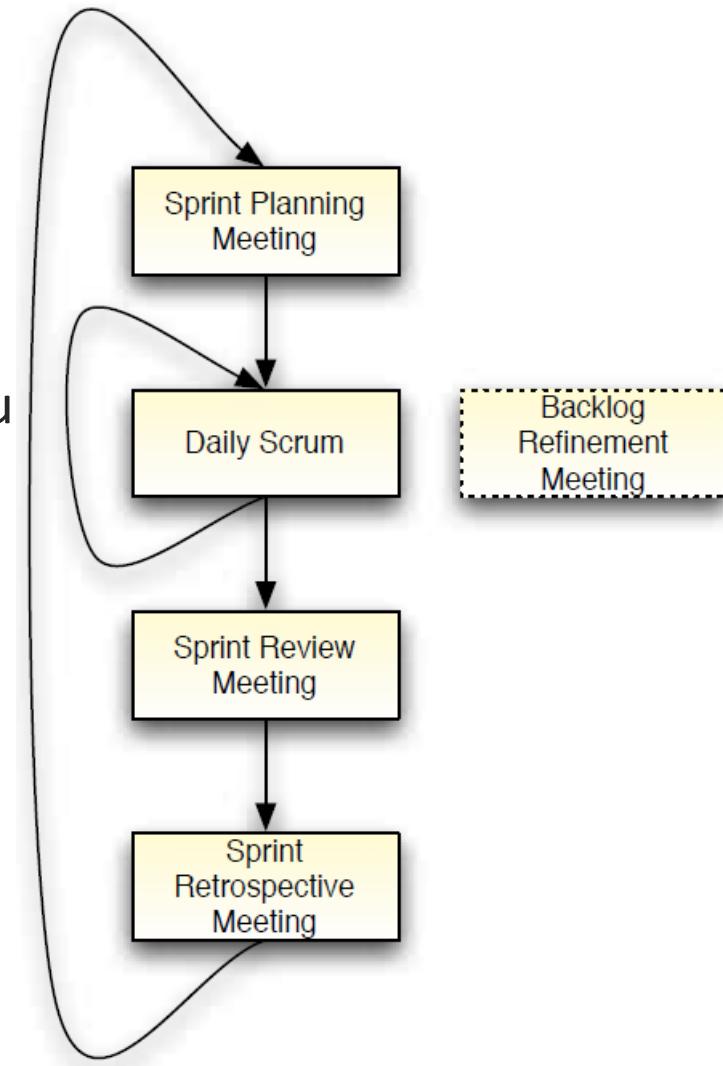


# Životni ciklus Scruma (2)

- Prije igre (pre-game)
  - podfaze: Planiranje i Dizajn/Arhitektura
  - izrađuje se radna lista proizvoda (Product Backlog - PB)
    - u PB se konstantno zapisuju zahtjevi, procjene napora i prioriteti
- Razvoj / "igra" (development / game)
  - razvoj iterativnim ciklusima, takozvanim sprintovima
  - sprintovi - okvirno jednakog trajanja, 30 dana (prema knjizi)
    - tjedan do tri u praksi
  - sprint ima sve faze klasičnog ciklusa
    - zahtjeve, analizu, dizajn, evoluciju, test i isporuku
  - tri do osam sprintova dok sustav ne bude spreman za distribuciju
- Poslije igre (post-game)
  - priprema sustav za izdanje kroz integraciju, testiranje i druge aktivnosti

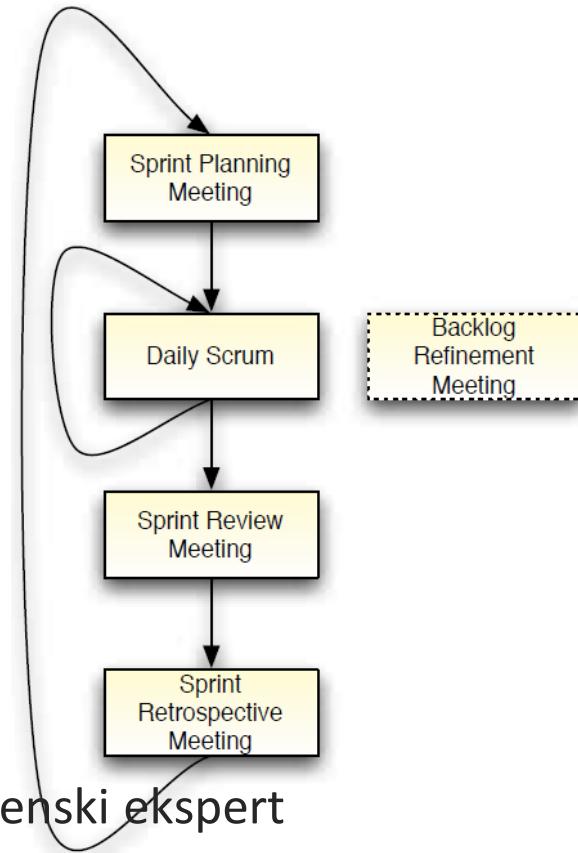
# Sprint i otkazivanje sprinta

- Svaki sprint sadrži 4 vrste sastanaka
  - vremenski ograničeni!
  - povećavaju transparentnost i omogućuju pregled i prilagodbu
  - smanjuje se potreba za sastancima koji nisu predviđeni Scrumom
- Otkazivanje sprinta
  - Rijetko i ima loš efekt na ekipu
  - Vlasnik može otkazati sprint ako njegov cilj postane besmislen (zastario, engl. *obsolete*)
    - Dovršeni poslovi u sprintu se pregledavaju kako bi se vidjelo jesu li iskoristivi
    - Neobavljeni posao se vraća u Product backlog



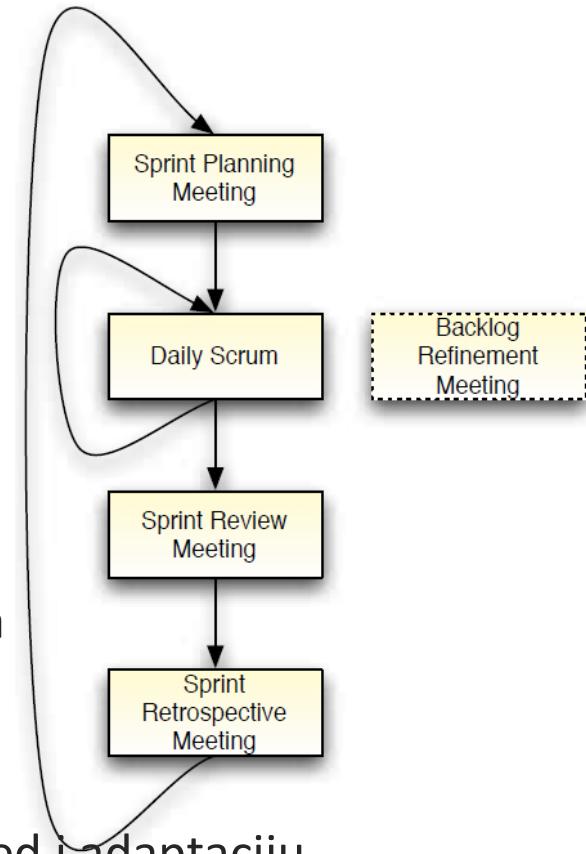
# Planiranje sprinta

- Na početku sprinta odlučuje se što će biti sljedeći inkrement temeljem product backloga (PB), zadnjeg inkrementa, predviđenog radnog kapaciteta i prethodnog učinka razvojne ekipe
- Vlasnik obrazlaže cilj sprinta i navodi stavke PB-a čija bi uspješna implementacija ostvarila cilj
- Razvojna ekipa procjenjuje mogućnosti i bira ciljane elemente iz PB-a i stvara *Sprint Backlog*
  - po potrebi može sudjelovati vanjski tehnički ili domenski ekspert
  - vlasnik radi korekciju ciljanih elemenata ako je premalo ili previše posla
  - plan za početne dane sprinta razlaže se u zadatke (obično) predviđenog trajanja manjeg od 1 dana
- **Majstor osigurava da se sastanak odvija u predviđenom vremenu**
  - Ne duže od 8h za sprintove od 30 dana
- Na kraju sastanka, razvojna ekipa mora biti u stanju objasniti vlasniku i majstoru organizaciju sprinta i način dobivanja inkrementa



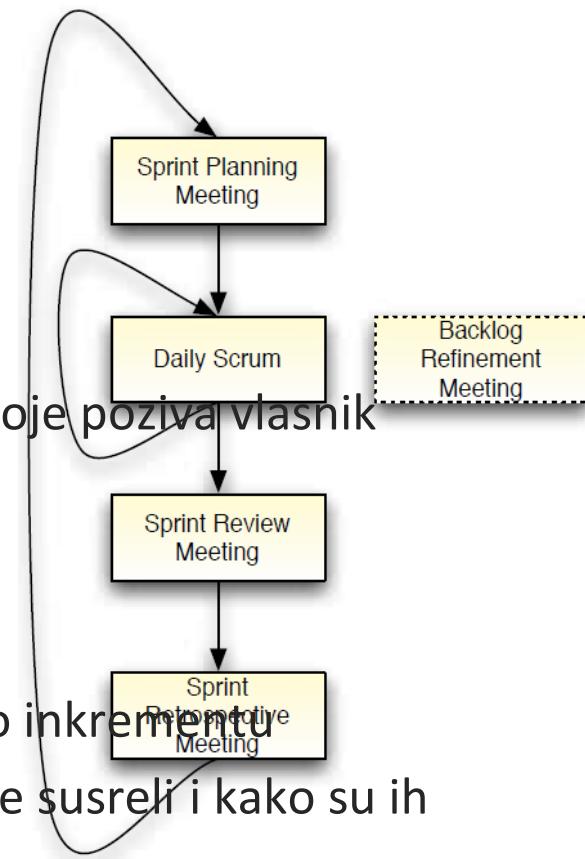
# Dnevni sastanak (Daily Scrum)

- Trajanje: 15 min
  - svaki dan sprinta u isto vrijeme, na istom mjestu , „s nogu“ (*standup meeting*)
  - interni sastanak članova razvojnog tipa
  - majstor osigurava da se sastanak održao i da je vremenski ograničen, ali ne usmjerava tok sastanka
- Služi za pregled dovršetka stavki i napretka prema cilju sprinta i sprinta
  - poboljšava komunikaciju tima, nudi priliku za pregled i adaptaciju
  - ujedno i plan rada za sljedeća 24h
- Obično fokus na pitanja usmjerena prema cilju sprinta
  - Što je napravljeno jučer?
  - Što će biti napravljeno danas?
  - Postoji li smetnja (prepreka, blokirajuća stavka)?
- Detaljnija diskusija po potrebi iza sastanka



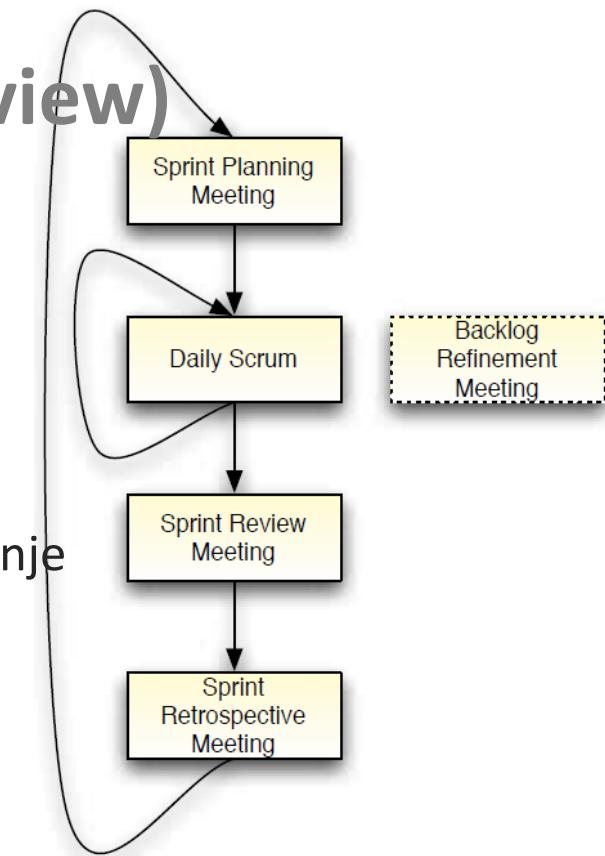
# Pregled sprinta (Sprint Review)

- Demonstracija inkrementa na kraju sprinta
  - Trajanje: ne dulje od 4h za sprintove od 30 dana
  - sudjeluju svi članovi tima i ključni dionici projekta koje poziva vlasnik
- Vlasnik deklarira „dovršeno”
  - ostalo ide u naredni sprint
- Razvojna ekipa
  - demonstrira dovršeni posao i odgovara na pitanja o inkrementu
  - diskutira što je bilo dobro, s kojim problemima su se susreli i kako su ih riješili
- Okvirno se dogovara što će se raditi dalje, odnosno što bi donijelo najveću vrijednost isporukom
  - rezultat je revidirani PB koji definira potencijale elemente za odabir u sljedećem sprintu
  - ujedno ulazna informacija za sljedeće planiranje
- Ažurira se procjena vremena, budžeta i potencijalnog tržišta za sljedeće isporuke



# Retrospektiva sprinta (Sprint Review)

- Samoanaliza procesa
  - Trajanje: ne dulje od 3h za sprintove od 30 dana
  - fokus na ljudima, međusobnim odnosima, procesima i alatima
  - Identificiranje stvari koje su bile dobre, identificiranje potencijalnih problema i njihovih rješenja (ili razloga zašto nisu riješeni)
    - načelni cilj je poboljšanje radnog procesa ili prilagodba definicije „obavljenog“ posla
  - majstor se brine da sastanak bude produktivan i u pozitivnom tonu, ali i kao član zadužen za provođenje Scruma
- Vlasnik može mijenjati PB bilo kad. Po potrebi se radi pročišćavanje preostalog posla (backlog refinement)
  - cijeli tim određuje vrijeme i način izvođenja
  - razrada elemenata, procjena, prioriteti ...
    - procjene uvijek radi razvojna ekipa



# „Scrum, ali...”

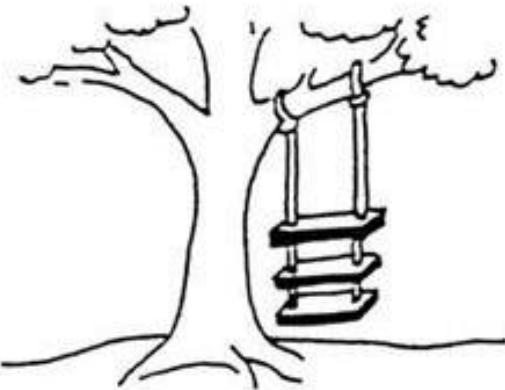
- „Koristimo Scrum, ali ...” <https://www.scrum.org/resources/what-scrumbut>
  - Činjenica je da je često prisutno „krojenje po mjeri” (*method tailoring*) i korištenje dobrih praksi van strogog metodološkog okvira
    - Može dovesti do poboljšanja, ali (češće?) do slabosti
- Načelo Ane Karenjine kaže da ako postoji nedostatak u nekom od ključnih aspekata, obitelj (u ovom slučaju cijeli proces s mnoštvom uvjeta koje treba zadovoljiti) će biti nesretna
  - neiskustvo Scrum majstora, previše specijalizirani timovi, prostorni i vremenski problemi, uključenosti korisnika, skaliranje na veće timove i projekte, ...
- Generalno (ne samo za agilne metode): Nema najboljeg pristupa, ovisi o vrsti i kompleksnosti problema, ekipi, razrađenosti zahtjeva, vremenu za razvoj, potrebi da se precizno odredi raspored i/ili prati napredak

# Razvoj primijenjene programske potpore

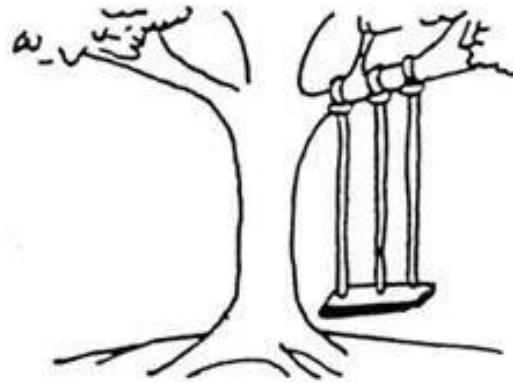
---

## 2. Zahtjevi. Upravljanje konfiguracijom

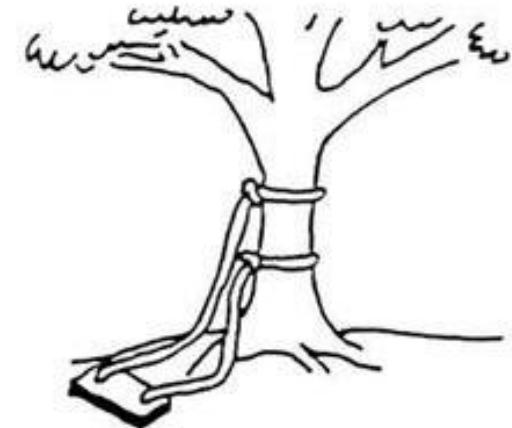
# Pogrešno postavljeni zahtjevi za posljedicu imaju neispunjena očekivanja, a naknadne prepravke su „skupe”



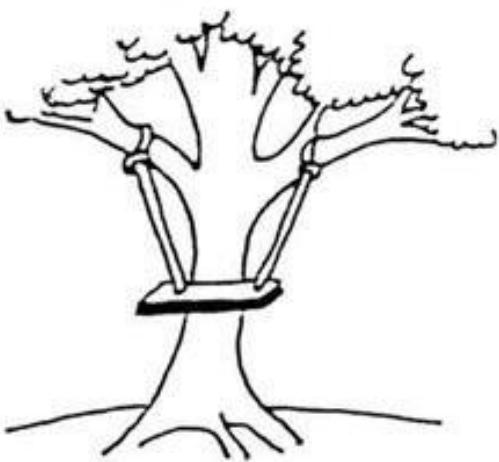
*As proposed by  
the project sponsors*



*As specified in  
the project request*



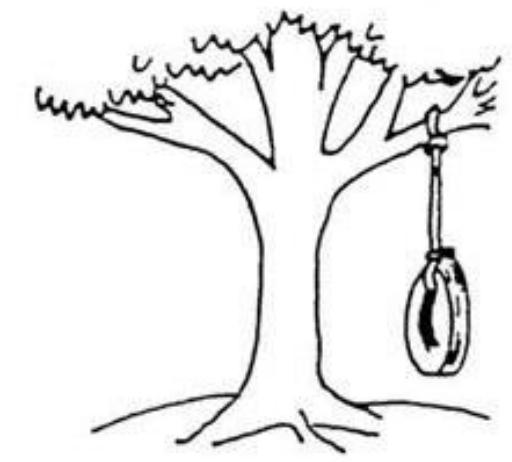
*As designed by  
the senior analyst*



*As produced by  
the programmers*



*As installed at  
the user's site*



*What the user  
wanted*

# 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

# Generalna podjela zahtjeva

- Generalna podjela na funkcionalne i nefunktionalne
- Funkcionalni opisuju funkcionalnost (što softver mora raditi) te daju opise interakcije s korisnicima, drugim softverom ili hardverom (tko, što, koji podaci)
- Nefunktionalni su vezani za svojstva ponašanja
  - performance, sigurnost, atributi kvalitete, ...
  - razna ograničenja (standardi, OS, ...)
  - najčešće se odnose na cijeli sustav, a ne na pojedinačno svojstvo
  - ponekad se nefunktionalni zahtjevi nazivaju i pseudo-zahtjevi

# Analiza (a ne dizajn)

- Zahtjevi odgovaraju na pitanje ŠTO sustav mora raditi, bez ublaženje u implementacijske detalje
  - odnos ulaza i izlaza, sadržaj formi, ali ne izgled formi i razmještaj kontrola
  - slijed operacija, ali ne i detalji implementacije
  - popis provjera ulaznih podataka, ali ne i način prikaza poruka za neispravne podatke
  - što treba poslati vanjskom sučelju, ali ne i kako se šalje
- Rezultat analize služi kao podloga za dizajn
  - Dio odluka prilikom dizajna može biti posljedica rezultata analize, odnosno nefunkcionalnih zahtjeva
    - npr. podržani uređaji, format izvještaja i zapisa, licence, pravila, certifikati ...

# Vrste zahtjeva

- Osim generalne podjele na funkcionalne i nefunktionalne neki autori razlikuju
  - [Sommerville] korisničke i sistemske zahtjeve ovisno o detaljnosti zahtjeva i ciljanoj publici
    - Tko je korisnik? Krajnji korisnik ili vlasnik?
  - [Dennis, Wixom, Tegarden] poslovne i sistemske ovisno o količini tehničkih detalja
  - ...
- Dobro oblikovani zahtjevi bi trebali imati sljedeću podjelu
  - **poslovni**
  - **korisnički**
  - **funkcionalni**
  - **nefunkcionalni**

# 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
  - U idealnom slučaju zahtjevi vlasnika podudaraju se s poslovnim ciljevima!
- Sadržani u dokumentima u kojima se opisuje vizija i opseg projekta

# Primjeri poslovnih zahtjeva (1)

- Sustav za potporu rada udruga
  - evidencija članstva i automatizacija postupka primanja novih članova neke udruge
  - praćenje finansijskih podataka udruge i njenih članova
  - poboljšanje procesa prodaje
  - omogućavanje internetske prodaje
  - podrška organiziranju natjecanja i okupljanja

# Primjeri poslovnih zahtjeva (2)

- Sustav subvencionirane prehrane
  - 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.
  - 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

# 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...“
    - Može (treba) se navesti i cilj, odnosno koristi koje korisnik ima od te akcije - opravdanje zahtjeva

# Primjeri korisničkih zahtjeva

- 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 izbjjeći 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.

# 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 performance
  - ograničenja na dizajn i implementaciju te svojstva kvalitete
    - preciziraju opis proizvoda navodeći karakteristike proizvoda u različitim dimenzija koja su važne ili korisniku, ili razvojniku.
- Primjer
  - U sustavu prehrane nefunkcionalni zahtjevi mogu biti vezani za oblik korisničke kartice, protokol povezivanja, obvezu fiskalizacije itd...

# Kriteriji kvalitete zahtjeva

Zahtjevi moraju odražavati korisnikove želje (**ispravnost**) i sljedeće kriterije

- **Singularnost**
  - Zahtjev ne smije biti kombinacija više zahtjeva
- **Potpunost**
  - opisati sve moguće scenarije, uključujući i one u slučaju pogreške
- **Konzistentnost**
  - zahtjevi ne smiju međusobno biti kontradiktorni
- **Nedvosmislenost**
  - svaki zahtjev se mora moći interpretirati na točno jedan način
- **Izvedivost** (ostvarivost)
  - zahtjevi se moraju moći implementirati
- **Provjerljivost**
  - za zahtjeve se mogu napisati (ponavljači) testovi za provjeru ispravnosti
- **Sljedivost**
  - Zahtjev mora se moći upariti s izvorom, povezanim zahtjevima ali i budućom implementacijom

# Izbjegavati nejasne i općenite izraze

- Primjeri riječi koje treba izbjegavati:
  - najviše, najbolje, lagano za korištenje, efikasno, *user friendly*, visoke kvalitete, on, taj, to, ako je moguće, primjereno, brzo, malo, ...
- Ovakvi izrazi se mogu interpretirati na različite načine i/ili nije moguće provjeriti je li zahtjev ispunjen

# Primjeri metrika za nefunkcionalne zahtjeve

- Brzina
  - broj transakcija u jedinici vremena, vrijeme odziva
- Veličina
  - broj ekrana, komponenti, vrijednost u (kilo/mega/...) bajtovima, ...
- Jednostavnost upotrebe
  - vrijeme treninga, broj pogrešaka prilikom upotrebe
- Pouzdanost
  - prosječno vrijeme prije kvara, postotak dostupnosti
- Robusnost
  - vrijeme oporavka nakon pogreške, postotak događaja koji mogu uzrokovati pogrešku
- Portabilnost
  - broj ili postotak podržanih uređaja (sustava)

# 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?
  - Jesu li „znakovi koji se ne mogu tiskati“ skriveni znakovi, posebne oznake ili kontrolni znakovi? (dvosmisленo, nepotpuno)
- 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 može obaviti određenu akciju, ali nije točno navedeno kakvu (npr. kombinacija tipki, klik miša, potez prsta), što se prepušta dizajnerima.

# Primjer nepotpunog zahtjeva

- „Status pozadinskog posla dostavlja se u redovitim intervalima ne kraćim 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?
- Preciznije i detaljnije bi bilo
  - 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.
  - Ako 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.
  - Ako 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 nepotpunog i neprovjerljivog zahtjeva

- „Parser će brzo generirati izvješće o pogreškama HTML oznaka, koje omogućava brzi ispravak pogrešaka kada program koriste početnici u HTML-u.”
  - Zahtjev je neprovjerljiv! Kako testirati ovaj zahtjev?
    - Pronaći nekoga tko se smatra početnikom u HTML-u i zatim vidjeti kako brzo će, uz pomoć izvješća, ispraviti pogreške?
    - Što znači „brzo”? 5 sekundi ili 5 minuta?
  - Nije definirano što i kada se tvori izvješće i to čini zahtjev nepotpunim.
- Bolje:
  - *„Na zahtjev korisnika sustav će analizirati HTML datoteku te generirati izvješće koje sadrži broj linije i tekst pronađenih HTML pogrešaka, te opis svake pogreške.“*
    - Ako nema pogrešaka prilikom analize, neće se generirati izvješće.

# Nekoliko primjera iz nedavne prakse (1)

- U postojećem sustavu forma je sadržavala polje za unos nalazišta i polje za unos staništa. Korisnik je u pisanom zahtjevu za promjenu naveo sljedeće:  
*„preimenovati opis nalazišta u opis nalazišta i staništa”*
- Treba li novi sustav imati samo jedno zajedničko polje za unos nalazišta i staništa ili 2 polja s nazivima „opis nalazišta i staništa” te „opis staništa”
- Razjašnjenje zahtjeva:
  - „Ostaju dva polja: 1. opis nalazišta i staništa te 2. Opis staništa, ali tako da na printanoj etiketi ne piše zasebno opis staništa već se pridružuje tekstu pod 1.”
  - Ne samo da prvi zahtjev nije bio jasan i nedvosmislen, nego je u sebi krio jedan inicijalno neotkriveni zahtjev

# Nekoliko primjera iz nedavne prakse (2)

- „Dodati opciju za unošenje podzbirki, mora biti moguće dodati nekoliko opcija uz jedan herbarijski primjerak“  
=> „*Uvesti herbarske podzbirke i omogućiti povezivanje herbarskog primjerka s više podzbirki*“
- Može li herbarski primjerak istovremeno pripadati i zbirci i podzbirci?
- Može li herbarski primjerak pripadati podzbirkama različitih nadređenih zbirki?

# Postavljanje prioriteta

- Nužno svojstvo - Da li korisnik nešto stvarno mora imati?
  - Postoji tendencija da se previše zahtjeva proglaši 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!
    - Nužni 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 biti 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.
- Alternativno označavanje prioriteta: numerički, po verzijama, ...

# Dokumentiranje analize (zahtjeva) (1)

- 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

- IEEE standard  
ISO/IEC/IEEE 29148:2011  
Stakeholder Requirements Specification  
StRS

## 1. Introduction

- 1.1 Business purpose
- 1.2 Business scope
- 1.3 Business overview
- 1.4 Definitions
- 1.5 Stakeholders

## 2. References

## 3. Business management requirements

- 3.1 Business environment
- 3.2 Goal and objective
- 3.3 Business model
- 3.4 Information environment

## 4. Business operational requirements

- 4.1 Business processes
- 4.2 Business operational policies and rules
- 4.3 Business operational constraints
- 4.4 Business operational modes
- 4.5 Business operational quality
- 4.6 Business structure

## 5. User requirements

## 6. Concept of proposed system

- 6.1 Operational concept
- 6.2 Operational scenario

## 7 Project Constraints

## 8. Appendix

- 8.1 Acronyms and abbreviations

# Dokumentiranje analize (zahtjeva) (2)

- Specifikacija zahtjeva *Requirements Specification*
    - naziva se 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 (ERD)
  - Software Requirements Specification SRS
    - IEEE standard ISO/IEC/IEEE 29148:2011
- 1. **Introduction**
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Product overview
      - 1.3.1 Product perspective
      - 1.3.2 Product functions
      - 1.3.3 User characteristics
      - 1.3.4 Limitations
    - 1.4 Definitions
  - 2. **References**
  - 3. **Specific requirements**
    - 3.1 External interfaces
    - 3.2 Functions
    - 3.3 Usability Requirements
    - 3.4 Performance requirements
    - 3.5 Logical database requirements
    - 3.6 Design constraints
    - 3.7 Software system attributes
    - 3.8 Supporting information
  - 4. **Verification**
    - (parallel to subsections in Section 3)
  - 5. **Appendices**
    - 5.1 Assumptions and dependencies
    - 5.2 Acronyms and abbreviations

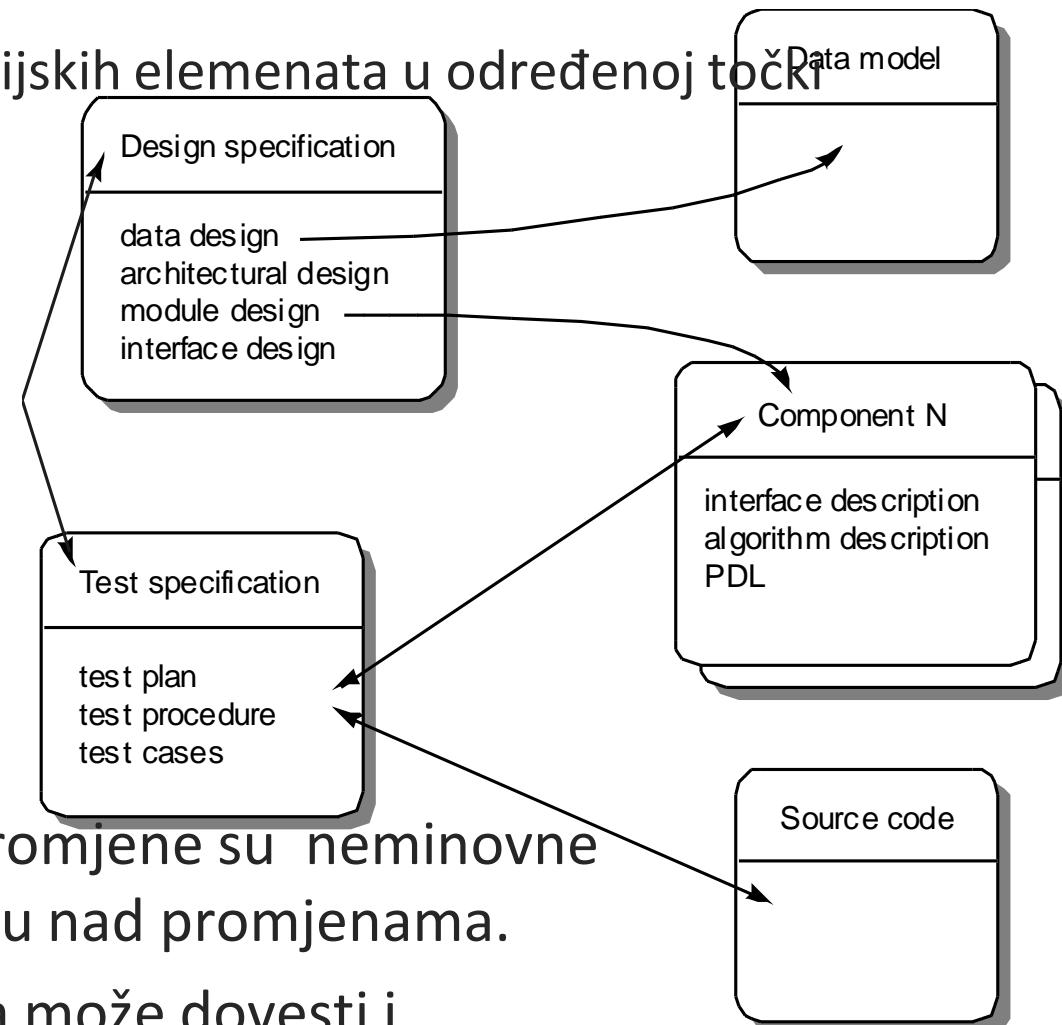
# Dokumentiranje analize (zahtjeva) (3)

- Agilne metode često umjesto formalnog dokumenta evidentiraju zahtjeve u obliku korisničkih priča
  - služe kao podloga za procjenu posla
  - razlažu se u manje zadatke
  - odabiru se za izradu po prioritetu
  - pogodno za sustave s nestabilnim ili nejasnim zahtjevima
- Prednosti:
  - jednostavno uređivanje
  - nije potrebno tehničko predznanje za evidentiranje
- Nedostatak:
  - korisnička priča je samo podsjetnik (obećanje) da će se taj zahtjev razraditi detaljnije
  - preporuča se u nekom trenutku imati jasno definiran dokument sa zahtjevima

# Upravljanje konfiguracijom

# Konfiguracija

- Konfiguracija
  - imenovani skup konfiguracijskih elemenata u određenoj točki životnog ciklusa
- Element konfiguracije
  - agregacija hardvera i/ili softvera koja se tretira kao jedinka u procesu upravljanja konfiguracijom
  - može sadržavati bilo koji artefakt nastao u razvojnem procesu
- Prilikom razvoja softvera, promjene su neminovne te je potrebno imati kontrolu nad promjenama.
- Nekontrolirani niz promjena može dovesti i najbolji softver u kaotično stanje

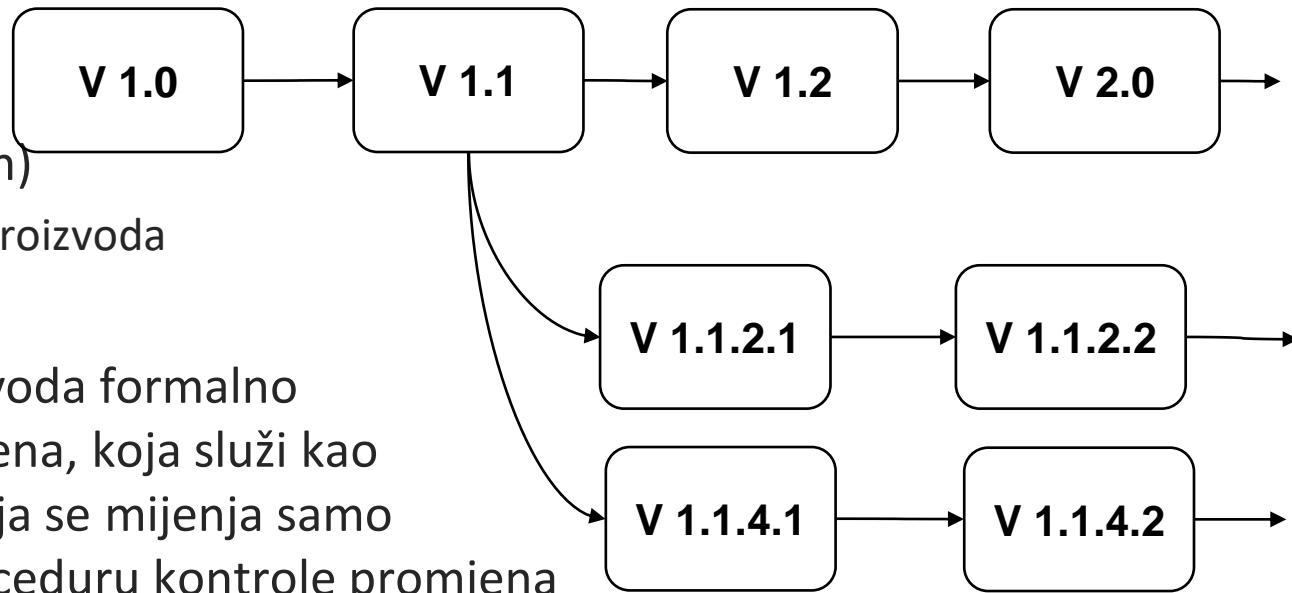


# Upravljanje konfiguracijama

- Upravljanje konfiguracijama (engl. *configuration management*) je proces upravljanja promjenama prilikom razvoja softvera
- Sastoji se o 4 vrste aktivnosti
  - upravljanje zahtjevima za promjenu funkcionalnosti
    - proces procjene i odobravanja zahtjeva
  - upravljanje verzijama
    - evidentiranje različitih elemenata konfiguracije
    - upravljanje verzijama izvornog koda
  - integracija i (automatska) izgradnja sustava
  - upravljanje izdanjima
- Sustavi za podršku upravljanju verzijama i izgradnju obično omogućavaju i
  - praćenje problema (engl. *issues tracking*)
  - praćenje pogreški (engl. *bug tracking*)

# Verzije konfiguracije

- verzija, inačica (version)
  - određeno izdanje proizvoda
- osnovica (Baseline)
  - specifikacija proizvoda formalno provjerena i usvojena, koja služi kao temelj razvoja i koja se mijenja samo kroz formalnu proceduru kontrole promjena
- objava, isporuka (release)
  - verzija isporučena korisnicima
- revizija (revision)
  - služi 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



# Označavanje verzija (1)

- Postupak pridjeljivanja jedinstvene oznake, imena i broja određenoj verziji
  - ne mora biti ista za internu upotrebu i za isporuku
- Šarolika praksa označavanja, ali većinom u praksi dominira oblik koji počinje s *major.minor*
- Major version/release
  - predstavlja veliko izdanje.
  - vrijednost se povećava prilikom znatne promjene funkcionalnosti
    - Može izazvati prekid kompatibilnosti u odnosu na prethodnu verziju
- Minor version
  - malo izdanje
  - promjene mogu biti značajne, ali dodavanjem novih funkcionalnosti i uz zadržavanje kompatibilnosti s prethodnim verzijama

# Označavanje verzija (2)

- Verzija objektne datoteke u .NET Frameworku (assembly) određena je s četiri broja:
  - <major version>.<minor version>.<build number>.<revision>
  - build number - predstavlja ponovno prevodenje 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

# Označavanje verzija (3)

- .NET Core koristi oblik major.minor.patch-sufix
  - Semantic versioning <https://semver.org/>
  - Sufiks može biti proizvoljan
    - alfa, beta, rc-2, ...
  - Nova verzija obično biva označena jednim brojem više na odgovarajućoj poziciji
- Aplikacije za Google Play Store
  - Korisnik vidi oznaku verzije koju razvojni tim odabere (versionName)
  - Interno se koristi pozitivni cijeli broj (versionCode)
    - raste neovisno o tome povećava li se malo ili veliko izdanje
    - služi za uspostavljanje odnosa među verzijama (prethodna ili nadogradnja)

# Označavanje verzija (4)

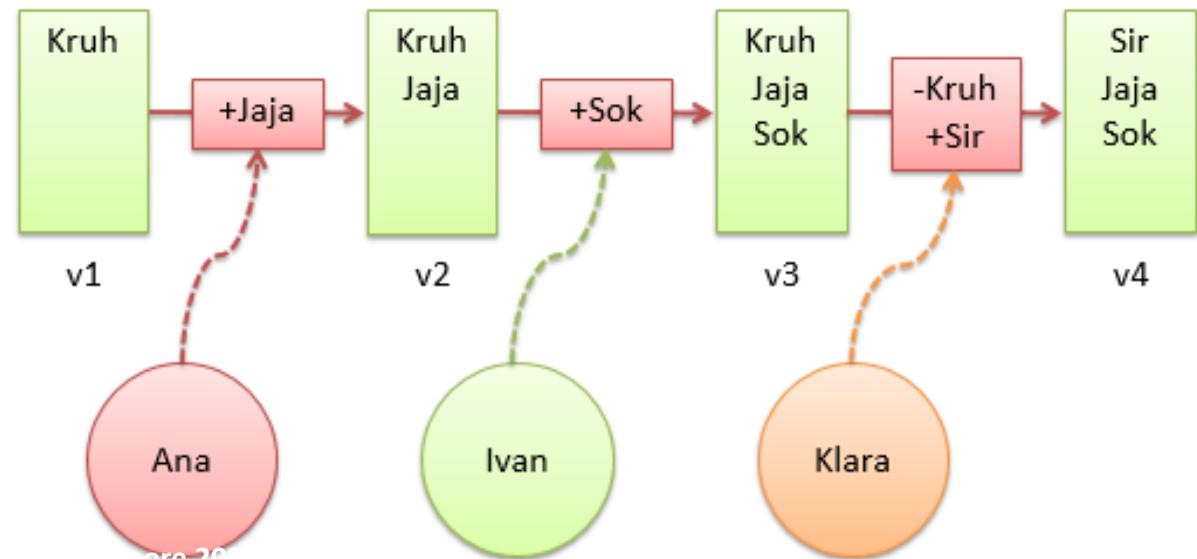
- Tex – trenutna verzija 3.14159265
    - Svaka nova verzija je nova znamenka broja  $\pi$
  - .NET Core 3.\* → .NET 5
    - Preskače se .NET Core 4 da ne izaziva zabunu s .NET Framework 4
    - Mijenja se naziv zbog unificiranja razvoja platformi
  - Java 1.0 → Java 1.1 → ... → Java 1.4 → Java 5 (umjesto 1.5)
    - Iстиче средње знатије верзије
  - Eclipse, Windows 10, Ubuntu, ... upotrebljavaju godinu i mjesec
    - Ubuntu 20.04, 20.10, ...
    - Windows 10 1909, 2004 + build number.update build revision
  - Word 2.0 → Word 6.0 → Word 95 → ...
  - iPhone → iPhone 3G → ... → iPhone 8 → iPhone X
- ... i puno drugih varijanti kao posljedica usklađivanja varijanti, marketinga, izbjegavanja nesretnih brojeva i slično

# 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

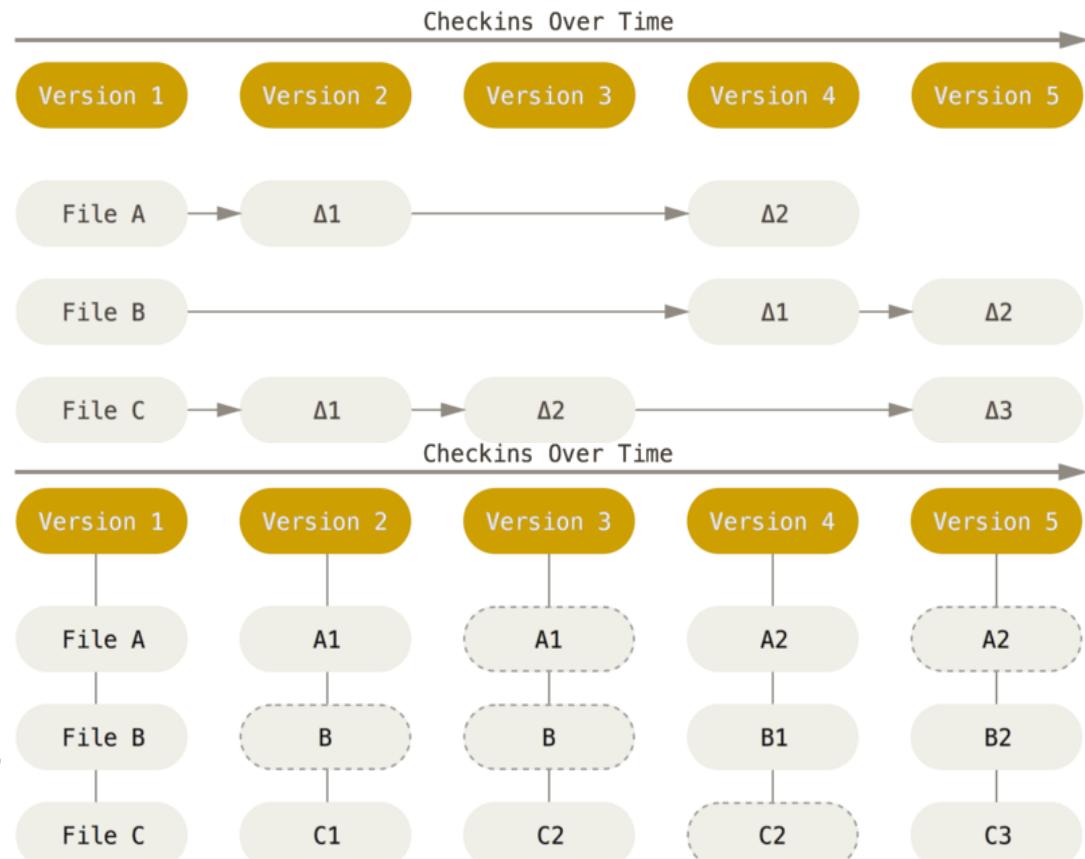
# Upravljanje verzijama (izvornog koda)

- Alati za upravljanje verzijama (*Version control systems*) obično omogućavaju
  - identifikaciju verzije i izdanja
  - bilježenje povijesti promjena
  - podršku za neovisni razvoj više programera nad istim softverom
- Najčešće kontrola promjena izvornog koda i konfiguracijskih datoteka
  - ne evidentirati automatski generirane datoteke!
- Repozitorij sadrži zadnju verziju, ali implementira mehanizam rekonstrukcije određene verzije iz povijesti (npr. evidentirajući razlike)
- Niz verzija nastalih izvođenjem iz prethodnih verzija naziva se *codeline*



# Karakteristike sustava za upravljanje verzijama

- Identifikacija verzija i izdanja
  - Svaka verzija pohranjena u repozitoriju ima jedinstveni identifikator
- Jeden 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.

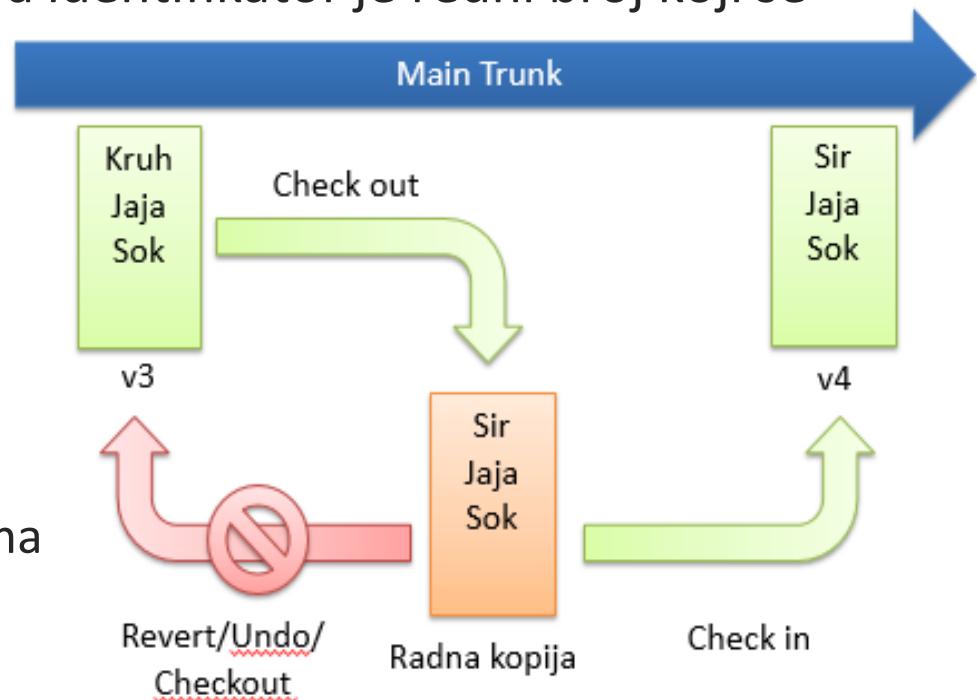
# Centralizirani sustavi za upravljanje verzijama

- 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 *checkoutom*
  - Najava izmjene datoteke
    - Može uključivati prethodni dohvati zadnje verzije datoteke i onemogućavanje ažuriranja te datoteke drugim korisnicima

# Promjene na radnoj kopiji

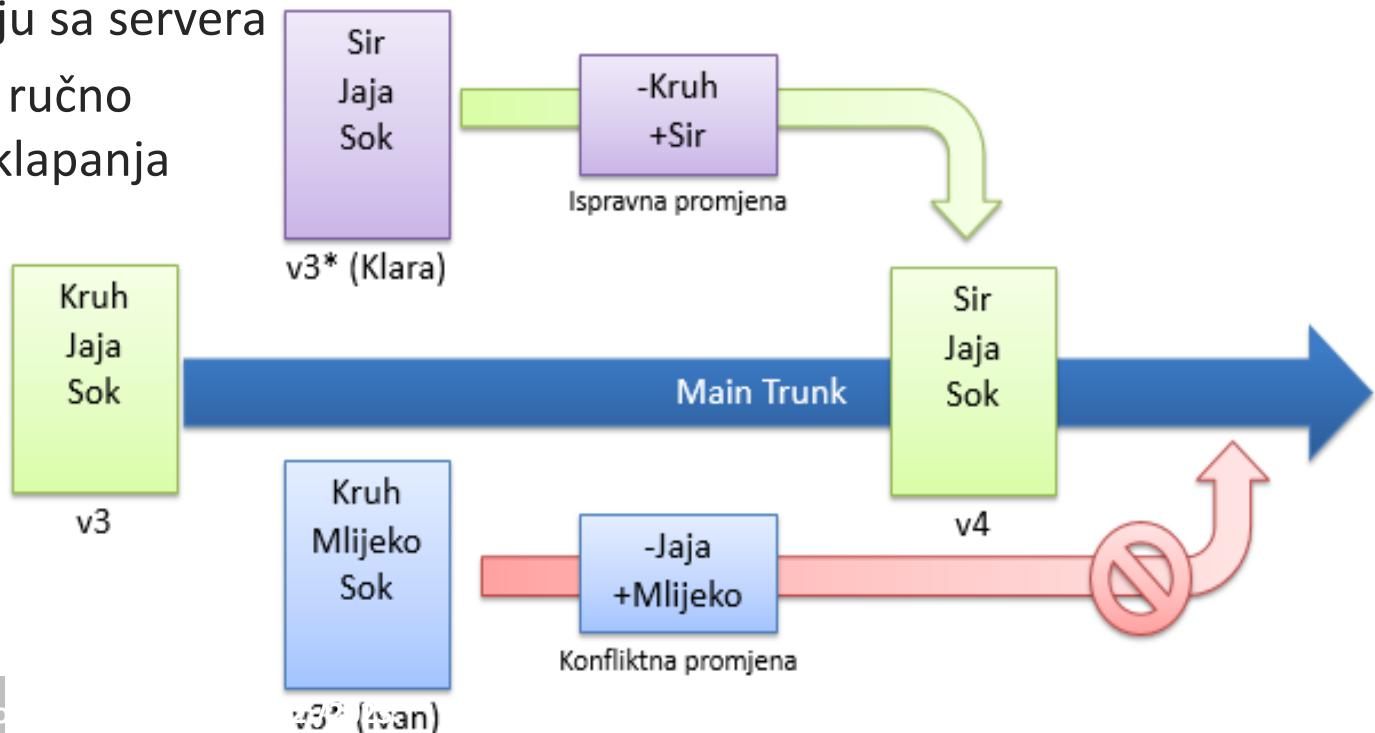
- Korisnik promjene na radnoj kopiji može
  - odbaciti (*undo, revert, checkout*)
  - potvrditi slanjem na centralni repozitorij (*check-in, commit*)
- Skup poslanih promjena naziva se *changeset*
- Svaki skup promjena dobiva jedinstveni identifikator
  - Kod centraliziranih sustava identifikator je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

*Trunk* – jedinstvena linija razvoja koja nije ničija grana



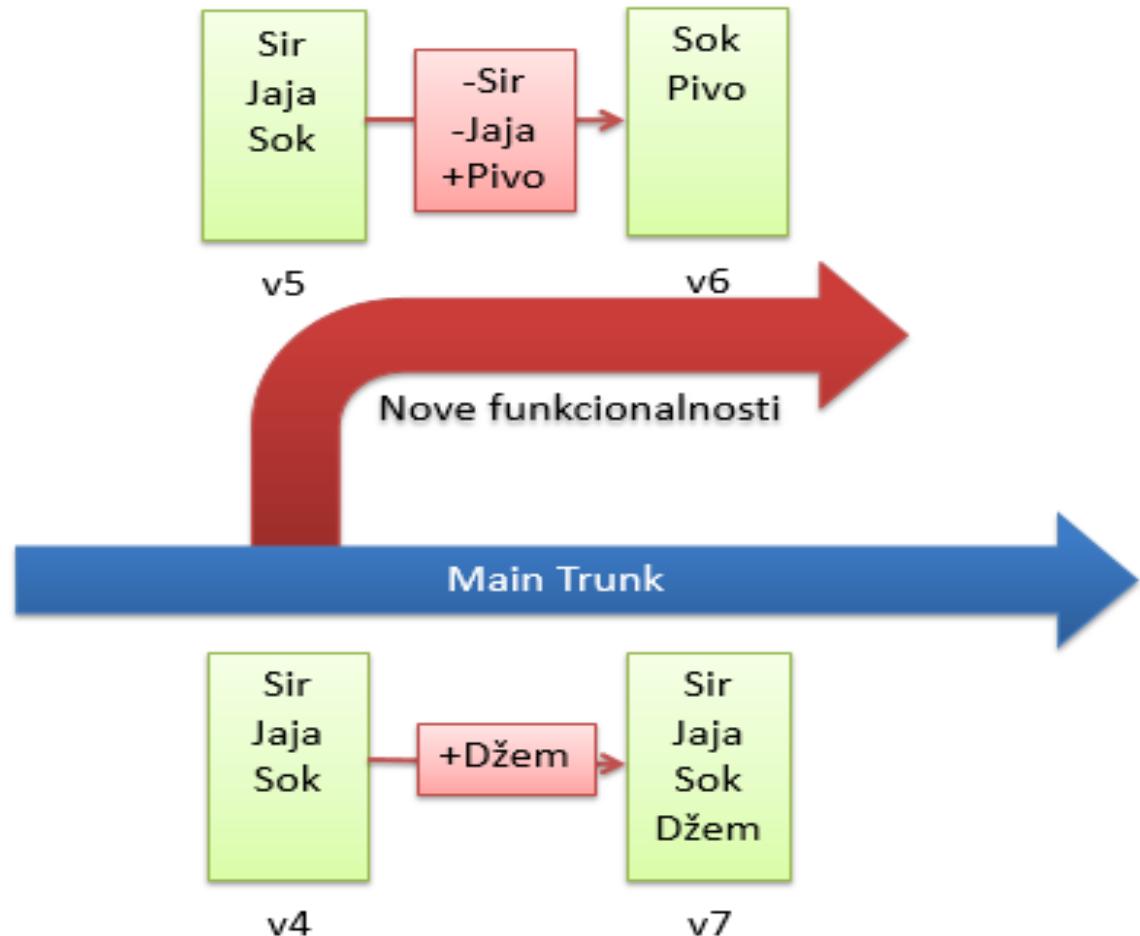
# Konfliktne promjene

- 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



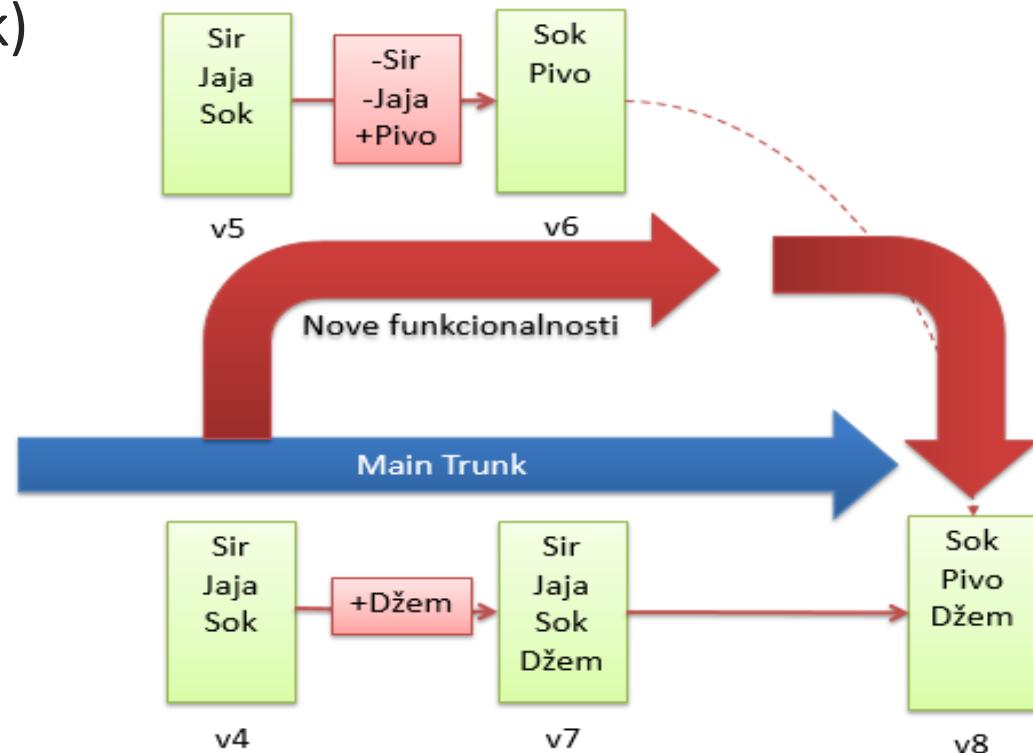
# Paralelne grane

- 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



# 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)

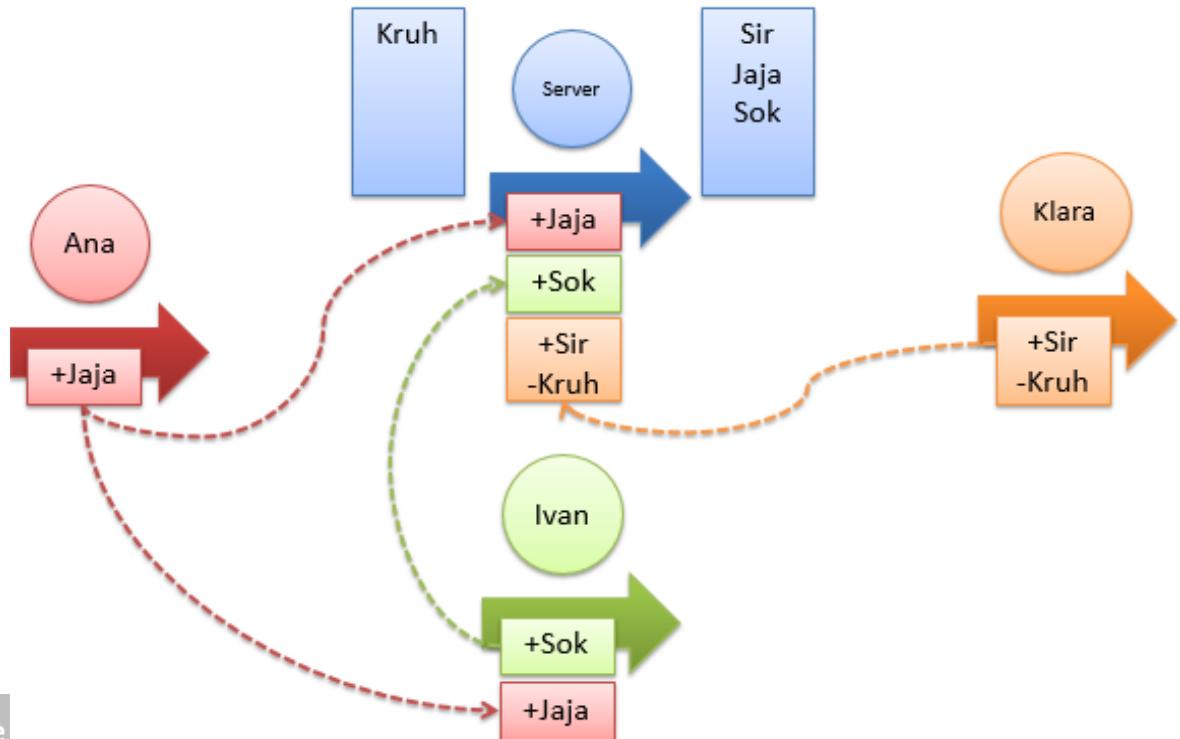


# Distribuirani sustavi

- Git, Mercurial
- Svaki korisnik ima kompletну 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.

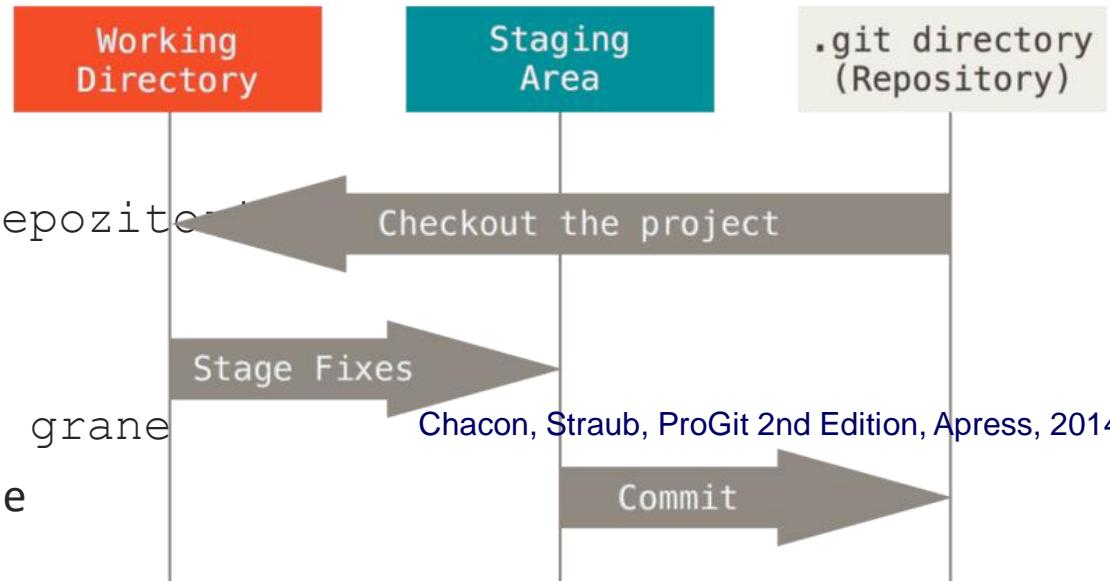
# Pojava konflikta u distribuiranim sustavima

- Kod distribuiranih sustava češće dolazi do konflikta
  - Različiti sadržaji reponzitorija 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)



# Tipični scenarij rada s Git-om

- Kloniranje centralnog ili inicijalizacija samostalnog repozitorija
  - `git clone adresa_repozitorija`
  - `git init`
- Promjena grane
  - Git checkout naziv grane
- Dohvat i primjena zadnje verzije grane centralnog repozitorija
  - `git pull origin naziv grane`
  - *origin* označava udaljeni repozitorij, naziv glavne grane je master ili main
- 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`



# Neki od ostalih pojmova i literatura

- Stash i Unstash
  - Privremeno spremanje izmjena, ali bez potvrđivanja promjena (tj. bez *commita*)
  - Slično kao Shelve/Unshelve kod TFS-a
- Pull request/Merge request
  - Zahtjev nekog korisnika za pregledom njegovog koda i spajanje s nekom drugom (najčešće glavnom) granom
- Literatura za Git
  - <http://tkrajina.github.io/uvod-u-git/git.pdf>
  - <https://git-scm.com/book/en/v2/>
  - <https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>

# Gradnja sustava

- Prikuplja relevantne objekte i proizvodi određenu verziju softvera
  - Dohvat određene verzije iz repozitorija
  - Kompilacija i povezivanje biblioteka
  - Pokretanje testova (ako postoje)
  - Objava verzije
    - Interno
    - Spremno za instalaciju kod korisnika
    - Automatska isporuka korisnicima
- Pokretanje može biti
  - na zahtjev
  - dnevno (npr. *nightly build*)
  - nakon svake promjene koda u repozitoriju
    - Ili nakon nekoliko akumuliranih promjena
  - ...

# Okruženja u procesu razvoja i isporuke (1)

- Development
  - razvojno okruženje pojedinog programera, odnosno ono u kojem se odvija razvoj
- Testing
  - Definiran skup testnih podataka i imitacija ovisnosti u cilju osiguranja predvidivosti i ponavljanja testiranja
  - Može se, odnosno poželjno je automatizirati
    - Lokalno testno okruženje i ono s odgovarajućim alatom za kontinuiranu integraciju

# Okruženja u procesu razvoja i isporuke (2)

- Staging
  - okruženje nalik produkcijskom ili kopija produkcijskog okruženja
  - za detektiranje pogreški koje promaknu u lokalnom razvojnom okruženju, vezano za mrežne postavke, ovisnosti o instaliranim bibliotekama i slično
  - pogodno za testiranje performansi i ponašanja pri većem opterećenju
  - obično namijenjeno za internu upotrebu, ali može poslužiti i za pretpregled novih funkcionalnosti
- Production
- U idealnom slučaju ova okruženja su potpuno odvojena, a dijelovi koda ili ponašanje parametrizirano varijablama okruženja

# Primjer kontinuirane isporuke na RPPP-u do 2020/21 (1)

- Team Foundation Server
- Definiran niz koraka koji služe za automatsku izgradnju i isporuku

Definitions / RPPP01 | Builds

Build Options Repository Variables

Save Queue build... Undo

Repository type: Git

Repository: 01

Default branch: master

Clean: true

Label sources: Don't label sources

Checkout submodules:

Definitions / RPPP01 | Builds

Build Options Repository Variables Triggers General Retention History

Save Queue build... Undo

Add build step...

Run dotnet publish

Tool: dotnet

Arguments: publish -c=\$(BuildConfiguration) -o=\$(build.stagingDirectory)\\$(build.BuildNumber)

Advanced

Control Options

Enabled:

Continue on error:

Always run:

Command Line  
Run dotnet restore

Command Line  
Run dotnet publish

Copy and Publish Build Artifacts  
Copy Publish Artifact: Published

PowerShell  
Objava web-aplikacije i web-servisa

# Primjer kontinuirane isporuke na RPPP-u do 2020/21 (2)

- Izgradnja se pokreće nakon svake promjene u rezervoriju
  - Continuous integration (CI)
    - lako bez testova to je više Continuous build nego integration
  - U definiciji s prethodnog slajda uključena i isporuka (objava web-aplikacije)
  - Continuous delivery (CD)
    - Nova verzija se može odmah dostaviti korisnicima na provjeru u uvjetima nalik proizvodnim
  - Continuous deployment (CD)
    - Nova verzija je odmah objavljena u proizvodnjom okruženju
- Prikazani koncept može biti primjenjen i za druga razvojna okruženja

The screenshot shows a software interface for managing build triggers. At the top, there are tabs: Definitions, RPPP01, and Builds. Below the tabs, there are buttons for Build, Options, Repository, Variables, and Triggers. The Triggers tab is selected. Under the Triggers tab, there are several buttons: Save, Queue build..., Undo, and a refresh icon. A checkbox labeled "Continuous integration (CI)" is checked, with the sub-instruction "Build each check-in." below it. There is also a "Batch changes" checkbox which is checked. Under the "Filters" section, there is a dropdown menu set to "Include" with "master" selected. A "Filters" button is also present. At the bottom, there is a checkbox for "Scheduled".

# Alati za kontinuiranu integraciju i isporuku

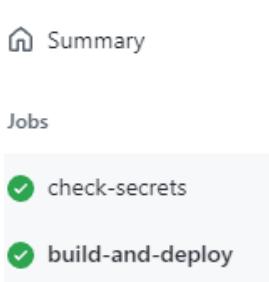
- BitBucket Pipelines, Azure Pipelines, Semaphore CI, GitHub Actions,
- Jenkins, Travis, ...

The screenshot displays a continuous integration pipeline interface. On the left, a sidebar shows branches: 'All branches' (master, big-refactor), 'OTHER BRANCHES' (fix/frontend), and a 'Pipeline' section listing pull requests (#82, #81, #78, #77, #73) with their status, author, commit hash, and branch. A blue vertical bar highlights the 'Pipeline' section. The main area shows a pipeline named 'Build and test' with a status of '32s'. Below it is a 'Deploy Backend to prod...' step with a status of '4m 14s'. To the right, a detailed view of the 'Build' step shows the command: `dpl --provider=heroku --app=$HEROKU_APP_NAME --api-key=$HEROKU_API_KEY`. The build log includes: `apt-get update -qy`, `apt-get install -y ruby-dev`, `gem install dpl`, `dpl --provider=heroku --app=$HEROKU_FRONTEND_APP_NAME --api-key=$HEROKU_API_KEY`, `cd frontend`, and `dpl --provider=heroku --app=$HEROKU_APP_NAME --api-key=$HEROKU_API_KEY`.

Pipeline	Status	Started	Duration
#82 minor fix Ivan Juren 2bd1c2c master	Successful	8 months ago	4 min 48 sec
#81 Merged in add-reviews (pull request #15) Add reviews Ivan Juren 0791112 master	Successful	8 months ago	5 min 24 sec
#78 Merged in feature/user_cancel_trip (pull request #14) add cancel ... Petra Zavrski d3f7838 master	Successful	9 months ago	5 min 6 sec
#77 Merged in feature/user_join_trip (pull request #13) Feature/user j... Petra Zavrski f168df3 master	Successful	9 months ago	5 min 45 sec
#73 create trip fix Ivan Juren 1896de2 master	Successful	9 months ago	5 min 9 sec

# Kontinuirana isporuka na primjeru domaćih zadaća (1)

- Github Actions
  - Niz koraka koji se izvršavaju nakon promjene grane *master*
  - Kombinacija Githubove usluge u oblaku
    - *init Docker Container, ... restore libraries, build, publish, ... i „vlastitog hardvera”*  
(fantom.fer.hr VPS, Srce)



build-and-deploy		Search logs	⚙️
succeeded on 28 Aug in 53s			
>	✓ Set up job	11s	
>	✓ Build kostya-ten/ssh-server-deploy@v4	8s	
>	✓ Build appleboy/ssh-action@master	5s	
>	✓ Run actions/checkout@v2	0s	
>	✓ Setup variables	0s	
>	✓ Print variables	0s	
>	✓ Setup .NET	6s	
>	✓ Restore dependencies	3s	
>	✓ Build	7s	
>	✓ Publish	3s	
>	✓ Change connection string in appsettings.json	0s	
>	✓ Set Kestrel info	0s	
>	✓ Set PathBase	0s	
>	✓ Compress	0s	
>	✓ Copy to destination server	6s	
>	✓ Unpack	3s	
>	✓ Post Run actions/checkout@v2	0s	
>	✓ Complete job	0s	

# Kontinuirana isporuka na primjeru domaćih zadaća (2)

- Primjer:  <https://github.com/FE-R-PPP/HomeworkTemp/late/blob/master/.github/workflows/dotnet.yml>
- Detaljnije prilikom izrade web-aplikacije

```
build-and-deploy:  
  needs: check-secrets  
  if: needs.check-secrets.outputs.data-available == 'true'  
  env:  
    # Leave project folder as-is, except if you main project has been renamed (in that case  
    # PROJECT-FOLDER : RPPP-WebApp  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v2  
    - name: Setup variables  
      id: variables  
      run:  
        echo '::set-output name=group::${{ env.GROUP }}'  
        echo '::set-output name=port::$((50000 + 100 * ${{ env.GROUP }}))"  
        echo '::set-output name=zip-filename::G${{ env.GROUP }}-V$GITHUB_RUN_NUMBER-$(date  
        echo '::set-output name=publish-folder::${{ github.workspace }}/published"  
    - name: Print variables  
      run:  
        echo "Group: ${{ env.GROUP }}"  
        echo "Zip filename: ${{ steps.variables.outputs.zip-filename }}"  
        echo "Publish folder: ${{ steps.variables.outputs.publish-folder }}"  
    - name: Setup .NET  
      uses: actions/setup-dotnet@v1  
      with:  
        dotnet-version: 5.0.x  
    - name: Restore dependencies  
      run:  
        cd ${{ env.PROJECT-FOLDER }}  
        dotnet restore  
    - name: Build  
      run:  
        cd ${{ env.PROJECT-FOLDER }}  
        dotnet build --configuration Release --no-restore
```

# Razvoj primijenjene programske potpore

---

## 3. Kratki pregled posebnosti platforme .NET (Core) i C#-a

Nadogradnja na OOP u Javi

# .NET Framework

- Microsoft .NET Framework
  - nastao s idejom iste osnovice za izradu lokalnih i Internet aplikacija
    - začetak krajem 90-tih, prva verzija 2002. g
    - zadnja verzija 4.8, daljnji razvoj obustavljen
  - neovisnost o jeziku (C++, C#, Visual Basic .NET, F#, ...) i neovisnost o platformi
    - .. Sve dok platforma podržava Common Language Runtime (CLR)
- Suprotno zamisli, .NET je namijenjen uglavnom Windows platformi
  - Mono – (djelomična) implementacija .NET Frameworka za Linux

# .NET (.NET Core)

- Open source varijanta nastala 2014. godine
  - nije podskup .NET Frameworka, ali dijelio dio funkcionalnosti propisane formalnom specifikacijom API-a (.NET Standard)
  - Podjela na modularne pakete dohvatljive korištenjem alata NuGet
- Dostupan za Windows, OSX i razne distribucije Linuxa
  - CoreCLR
  - .NET 6 (preciznije 6.0.10, SDK 6.0.402), listopad 2022.
    - Od verzije 5 umjesto naziva .NET Core, koristi se naziv .NET i napušta koncept .NET standarda
- C#
  - Razvoj započet 1999., prva verzija 2002.
  - trenutna verzija C# 10.0 (studen 2021.)

# .NET status i budućnost

- 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
- Samostalne (desktop i mobilne) aplikacije za
  - Android, iOS, macOS, Windows, Tizen
  - .NET MAUI (.NET Multi-platform App UI)

# Osnovni pojmovi

- IL – Intermediate language – jezik u kojem se kompiliraju viši jezici
- CLR (Common language runtime)
  - Virtualno računalo koje izvršava naredbe naredbe nastale iz IL-a pretvorbom u strojni jezik (JIT – Just-in-time compiler)
- Common Type System (CTS) definira tipove podataka za jezike koji se mogu pretvoriti (kompajlirati) u IL
  - Klase, strukture, enumeracije, sučelja, delegati, modifikatori pristupa, ...
    - <https://docs.microsoft.com/en-us/dotnet/standard/base-types/common-type-system>
- Common Language System (CLS) – podskup CTS-a prisutan u svim jezicima
  - nazivi tipova u pojedinom jeziku ne moraju biti isti
    - npr int vs integer → int32 u IL-u
  - može postojati neki tip u C#-u koji ne postoji u Visual Basicu
- Base Class Library (BCL) – osnovni skup biblioteka

# Stvaranje prvog programa (.NET Core)

- Komanda linija
  - u nekoj mapi pokrenuti dotnet new console –n Naziv
  - programski kod urediti u proizvoljnem 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
  - Potvrди dohvat paketa, a zatim F5
- Visual Studio 2022
  - File → New project → C# Console Application (NET Core)
  - Unijeti naziv i lokaciju projekta
  - F5 (Debug) ili CTRL+F5 (Execute)

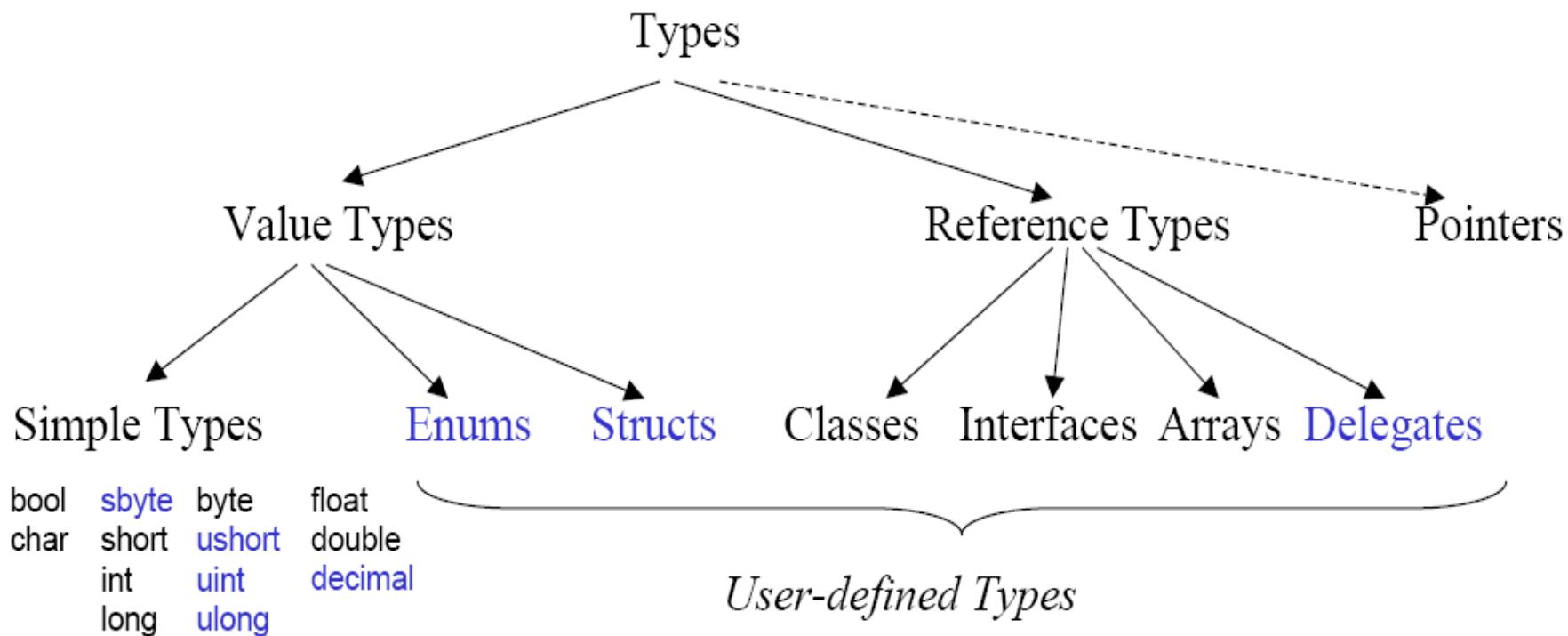
# Java -> C# - prvi koraci

- Ime datoteke ne mora odgovarati nazivu klase, ali je poželjno
  - U istoj datoteci može se nalaziti više klasa
  - Ista klasa se može definirati u više datoteka te se takva klasa mora označiti modifikatorom `partial`
- `package` → namespace      `import` → using
- 1 projekt = 1 dll i/ili exe      (*assembly*)
- Sintaksa nalik C-u i Javi. Ključne riječi (neke ovisne o kontekstu):
  - <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords>
- Standardni ulaz/izlaz
  - `Console.ReadLine`, `Console.WriteLine` ...
  - Više o formatiranju ispisa: <https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings>
- Pravokutna i nazubljena (jagged) polja

# Tipovi podataka

- Svi tipovi izvode se iz osnovnog tipa `System.Object`
  - primitivni tipovi su strukture
  - Moguće npr.: `3.ToString()`; `"abc".ToUpper()`;

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/common-type-system>



# Modifikatori vidljivosti

- 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 naslijedjene razrede
  - private protected – slično kao protected, ali samo za naslijedjene razrede u istom programu/projektu
  - internal – pristup ograničen na program u kojem je razred definiran
  - protected internal - pristup dozvoljen naslijedjenim razredima (bez obzira gdje su definirani) i svima iz programa u kojem je razred definiran

# Prepostavljeni 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
    - do verzije 8 C#-a nije ni bio dozvoljen drugaci modifikator
- Izvedeni razred ne može imati veću dostupnost od baznog razreda

# 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 (u konstruktoru)
- sealed – razred ne može biti naslijeden
- 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)

# Finalizatori (Destruktori)

- Finalizator je postupak (*Finalize*) koji se automatski poziva neposredno prije uništenja objekta od strane sakupljača smeća (Garbage Collector)
- Piše se nalik destruktoru u C++-u - nazivu razreda prefiksiran s ~
  - Ne vraća vrijednost
  - Koristi se u rijetkim slučajevima za brisanje *unmanaged* resursa (ako su korišteni)

```
class Razred {  
    ~Razred(){  
        Console.WriteLine("Finalizer");  
    }  
    ...  
}
```

- Pogledati sadržaj klase s ILDASM (*Intermediate Language Disassembler*)

# Svojstva

- Svojstvo je postupak pristupa zaštićenim varijablama instance
  - Pandan gettteru i setteru u Javi, ali praktičnije sintakse
- 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 dohvati i pridruživanje
  - Za svojstva koja imaju samo get dio kod se može napisati i lambda izrazom
- Primjer  SomeOfCSharpFeatures \ PropertiesIndexersRefOut \ Triple.cs
  - Proučiti sadržaj klase s ILDASM

# Indekseri

- Indekseri
  - omogućavaju korištenje objekta kao niza (pristup elementima operatorom [ ])
  - sintaksa uobičajena za svojstva (get i set)
- Sadržaj su uglatim zagradama može biti proizvoljan

```
public int this[string s, int pos] {  
    get {  
        ...  
    }  
    set { ...  
}
```

```
x["A", 5] = 1 + x["B", 3];
```

- Primjer  SomeOfCSharpFeatures \ PropertiesIndexersRefOut \ Triple.cs
  - Proučiti sadržaj klase s ILDASM

# Ref, out i varijabilni broj argumenata

- ref modifikator – argumenti su reference (*call by reference*)
  - **Prije poziva postupka** argumenti **moraju** biti inicijalizirani
  - Mora se navesti i prilikom poziva postupka
- out modifikator – izlazni argument
  - U trenutku poziva out postupka argumenti ne moraju biti inicijalizirani.
  - **Pri izlasku iz postupka** out argumenti **moraju** biti postavljeni.
- Varijabilni broj argumenata se definira ključnom riječi params i poljem određenog tipa
- Primjer  SomeOfCSharpFeatures \ PropertiesIndexersRefOut \ Program.cs

# Imenovanje argumenata

- Postupci mogu imati opcionalne argumente s prepostavljenim vrijednostima
- Prilikom poziva argumenti se mogu imenovati (umjesto pridruživanja po redoslijedu argumenata)
  - Prilikom poziva navodi se naziv argumenta i vrijednost odvojeni dvotočkom
  - Imenovani argumenti se navode zadnji

# Preopterećenje operatora

- Operatori koji se mogu preopteretiti
  - unarni: + - ! ~ ++ -- true false
  - binarni: + - \* / % & | ^ << >> == != > < >= <=
- Primjer  SomeOfCSharpFeatures \ PropertiesIndexersRefOut \ Triple.cs
- Posljedično sadržaj stringova uspoređujemo s ==

# Enumeracije

- Pobrojani tip (enumerator)
  - Korisnički tip vrijednosti koji nasljeđuje System.Enum
  - Sastoji se od imenovanih konstanti
  - Temeljni tip podataka pobrojanog tipa je int, ali može se promijeniti
  - Može im se pridružiti vrijednost (ali ne mora)
- Korištenja atributa [Flags] dopušta kombinacije vrijednosti

```
[Flags]
public enum DayPeriod
{
    Morning = 1, Evening = 2, Afternoon = 4, Night = 8
}
```

```
DayPeriod p = DayPeriod.Evening | DayPeriod.Night
```

# Nulabilni tipovi

- Varijable mogu imati nedefiniranu vrijednost
- Tip podatka Nullable<T> pri čemu T mora biti vrijednosni tip (*value type*), npr. int
  - Skraćeno se može zapisati T?, npr. int?

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

- Genericsi implementirani u CLR-u
  - ne koristi se brisanje tipa kao u Javi

# Ograničenja na tip parametriziranog razreda

- Ograničenja se postavljaju kontekstualnom ključnom riječi where
- Moguća ograničenja:
  - where T:struct – tip T mora biti value type (*Nullable* nije dozvoljen iako je struktur)
  - where T:class – tip T mora biti reference type
  - where T:new() – tip T mora imati konstruktor bez argumenata
  - 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 izведен iz tipa U pri čemu je U drugi tip po kojem se vrši parametrizacija

# 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

# 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

# Proširenja (eng. extensions)

- Razred za koji se piše proširenje je naveden kao prvi parametar statičke metode u statičkom razredu, prefiksiran s *this*
  - Poziva se kao da se radi o postupku unutar tog razreda (iako nije)
  - Primjer  SomeOfCSharpFeatures \ Extensions \ Extensions.cs

```
public static class Extensions { //može se zvati proizvoljno
    public static V GetOrCreate<K, V>(this Dictionary<K, V> dict,
                                         K key) where V : new() {
        if (!dict.TryGetValue(key, out V value)) {
            value = new V();
            dict[key] = value;
        }
        return value;
    }
}
```

```
var dict = new Dictionary<string, List<int>>;
List<int> list = dict.GetOrCreate("Some string");
```

# Nasljeđivanje i polimorfizam

- Za označavanje nasljeđivanje ili implementiranja nekog sučelja se koristi dvotočka
  - `class DerivedClass: BaseClass, Interface1, Interface2 { ... }`
- Sučelja po standardu imenovanja počinju slovom I
- `sealed` sprječava daljnje nasljeđivanje
- `virtual` deklarira virtualni postupak roditelja koji može biti nadjačan
- `override` deklarira postupak djeteta koji nadjačava, a može koristiti nadjačani postupak
  - Što ako se pokuša nadjačati metoda koja nije virtualna ili ako se definira nova metoda bez nadjačavanja
  - Primjer  SomeOfCSharpFeatures \ Inheritance \ \*.cs

# Odnosi razreda u složenijim tipovima

- Razred `Car` je podrazred razreda `Vehicle`.
- U kojem su odnosu `List<Car>` i `List<Vehicle>` ?
  - Nisu hijerarhijski povezani
- U kojem su odnosu:
  - `IEnumerable<Car>` i `IEnumerable<Vehicle>`
  - `IComparer<Car>` i `IComparer<Vehicle>`
- Odgovor nije očit (jednostavan) !!

# 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  SomeOfCSharpFeatures \ CovarianceContravariance

```
static void PrintVehicles(IEnumerable<Vehicle> vehicles) {  
    foreach (var vehicle in vehicles)  
        Console.WriteLine("\t " + vehicle.Model);  
}
```

- Kao argument moguće je poslati List<Vehicle>, jer List<T> implementira sučelje IEnumerable<T>
- Ali moguće je poslati i List<Car> !!!
  - List<Car> se može pretvoriti u IEnumerable<Car> , a sučelje **IEnumerable<T> je kovarijantno**

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

# Kontravariantnost

- Primjer  SomeOfCSharpFeatures \ CovarianceContravariance

```
void PrintBetterCar(Car a, Car b, IComparer<Car> comparer) {  
    int result = comparer.Compare(a, b);  
    string betterModel = result <= 0 ? a.Model : b.Model;  
    string worseModel = result <= 0 ? b.Model : a.Model;  
    Console.WriteLine($"\\t{betterModel} ... {worseModel}");  
}
```

- Kao komparator moguće je upotrijebiti objekt tipa `IComparer<Vehicle>`, jer je **sučelje `IComparer<T>` kontavariantno**

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

# Delegati (1)

- 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
- Primjer  SomeOfCSharpFeatures \ CovarianceContravariance

```
void PrintBetterCar(Car a, Car b, Comparison<Car> comparer) {  
    int result = comparer(a, b);  
    string betterModel = result <= 0 ? a.Model : b.Model;  
    string worseModel = result > 0 ? b.Model : a.Model;  
    Console.WriteLine($"\\t{betterModel} ... {worseModel}");  
}
```

# Delegati (2)

- Delegat se definira kao varijabla određenog tipa delegata
- Tip delegata definira se sljedećom sintaksom

```
public delegate PovratniTip NazivTipaDelegata (args)
```

- Primjer:

```
public delegate double Tip(int a, string b)
```

bi definirao tip delegata koji omogućava 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 `=, +=, -=`

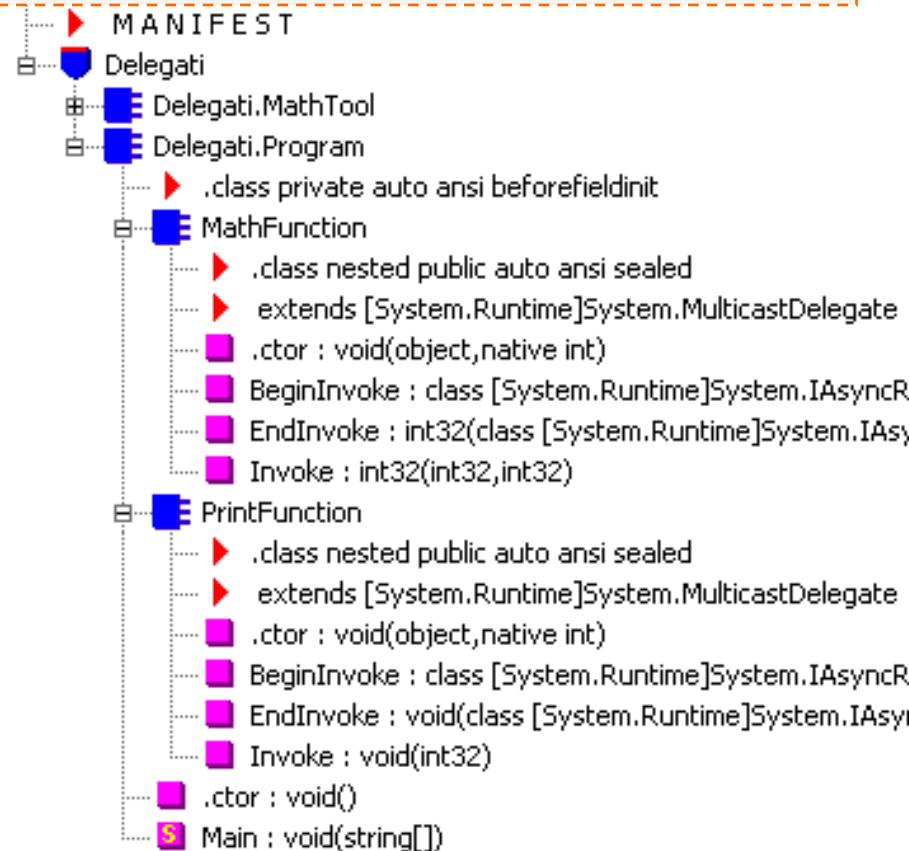
# Primjer tipa delegata

- Primjer SomeOfCSharpFeatures \ Delegates \ 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 n
- U glavnom programu definirane dvije varijable (delegata)

```
MathFunction mf = ...  
PrintFunction pf =
```



# Primjer pridruživanja postupka delegatu (1)

- Primjer SomeOfCSharpFeatures \ Delegates \ 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) {  
        ...  
    }  
}
```

# Primjer pridruživanja postupka delegatu (2)

- Primjer SomeOfCSharpFeatures \ Delegates \ Program.cs
  - Varijabla *mf* je delegat koji sadrži reference na postupke koji 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  SomeOfCSharpFeatures \ Delegates \ Program.cs
  - Delegatu se može pridružiti više postupaka (mogu se i 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, ..., ili 16 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
- U prethodnim primjerima se umjesto *MathFunction* mogao koristiti *Func<int, int, int>*, a umjesto *PrintFunction* *Action<int>*

# 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<Car, Car>* tada se delegatu tog tipa mogu pridružiti reference na postupke npr

- `Car Test(Car a) { ... }`
- `ElectricCar Test(Vehicle a) { ... }`

ali ne i npr.

- `Vehicle Test(Car a) { ... }`
- `Car Test(ElectricCar a) { ... }`
- Navedeno ima smisla, jer ako je

*Func<Car, Car> f = nešto od navedenog*

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

# Razvoj primijenjene programske potpore

---

**4. Defenzivno programiranje,  
konfiguracijske datoteke, praćenje traga rada**

# Defenzivno programiranje

- Zaštita programa od neispravnih podataka, događaja koji se „nikad nisu smjeli dogoditi” i ostalih programerskih pogrešaka
- Defenzivna vožnja automobila temelji se na načelu da vozač nikad ne može biti siguran što će učiniti drugi vozači, pa unaprijed nastoji izbjegći nezgodu za slučaj pogreške drugih vozača
- U defenzivnom programiranju ideja vodilja je da će potprogram s neispravnim podacima „opstati”, odnosno neće izazvati štetu i onda kad su pogrešni podaci posljedica nekog drugog potprograma
  - Pogreške se prvenstveno sprječavaju iterativnim dizajnom, pisanjem pseudo-koda prije kodiranja, pisanjem testova (prije koda), inspekcijom koda, ...
  - Defenzivno programiranje je dodatak na gore navedene tehnike

# Smeća na ulazu - smeće na izlazu

- „Garbage IN, Garbage Out” (GIGO) – 1957.?
  - <https://www.newspapers.com/clip/50687334/the-times/>
  - računalo radi ono što mu kažete, iako je to možda krivo
  - frazu popularizirao George Fuechsel (IBM)
    - <https://www.atlasobscura.com/articles/is-this-the-first-time-anyone-printed-garbage-in-garbage-out>
- Charles Babbage

*On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" ... I am not able rightly to comprehend the kind of confusion of ideas that could provoke such a question.*

(1864.)

# Smeća na ulazu - ? na izlazu

- Dobar program nikad ne smije proizvesti smeće, neovisno o ulaznim podacima. Pristup „smeće unutra, smeće van“ treba zamijeniti sa:
  - „smeće unutra, ništa van“,
  - „smeće unutra, poruka o pogrešci van“
  - „smeću zabranjen ulaz“
  - „počisti smeće“ (*turn garbage input into clean input*)
- 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

# Tehnike obrade pogrešaka (1)

- vratiti neutralnu vrijednost
  - 0, "", NULL
  - Krivi kod boje? Obojati bijelo?
- zamijeniti neispravnu vrijednost sljedećom, moguće ispravnom
  - npr. while (GPSfix != OK) sleep(1/100s) ...
  - Neispravna adresa dostave? Poslati na adresu za dostavu računa?
- vratiti vrijednost vraćenu pri prethodnom pozivu
- zamijeniti neispravnu vrijednost najbližom ispravnom
  - npr. min(max(kut, 0),360)
  - Krivo napisana adresa. Uzeti najsličniju? – ne mora nužno biti dobro
- zapisati poruku o pogrešci u datoteku, kombinirano s ostalim tehnikama

# Tehnike obrade pogrešaka (2)

- vratiti kôd pogreške
  - pogrešku obrađuje neki drugi dio koda – dojava drugom dijelu
  - mogućnosti ovise o programskom jeziku
    - globalna varijabla
    - „zlouporaba” povratne vrijednosti funkcije
    - iznimke
- pozvati „globalnu” (centraliziranu) metodu za obradu pogreške
- odmah prikazati poruku pogreške
- bezuvjetni završetak programa

# Robusnost i ispravnost programa

- Robusnost
  - 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
- Suprotstavljeni pristupi. Što odabrati?
  - Odgovor ovisi o tipu aplikacije i odluci prilikom uspostavljanja arhitekture (dizajna) rješenja
  - Hoće li neispravan rezultat proizvesti npr. zanemarivi artefakt na ekranu ili će npr. krivo izračunati dozu kojom treba ozračiti pacijenta na rendgenu?
  - Što se događa kada tramvaj izgubi vezu ili GPS signal i ne zna na kojoj se stanici približava?

# Iznimke

- 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
  - InnerException – unutarnja (omotana) iznimka
- NullReferenceException, IndexOutOfRangeException, DivideByZeroException, InvalidCastException, ...

# Obrada iznimki

- Obrada iznimki sprječava nepredviđeni prekid izvođenja programa
  - Iznimka se obrađuju tzv. rukovateljem iznimki (exception handler)
  - Obrada pogreške sastoji se try, 1 ili više catch blokova i finally bloka
    - catch ili finally se mogu ispuštiti (ali samo jedan od njih)

```
try {  
    //dio kôda koji može dovesti do iznimke  
}  
catch (ExcepType1 ex0b){  
    //kôd koji se obavlja u slučaju iznimke tipa ExcepType1  
}  
catch (ExcepType2 ex0b) {  
    //kôd koji se obavlja u slučaju iznimke tipa ExcepType2  
}  
...// ostali catch blokovi  
finally {  
    //kôd koji se uvijek obavlja
```

# Preporuke za korištenje iznimki (1)

- Koristiti iznimke za obavijest drugim dijelovima programa o pogreškama koje se ne smiju zanemariti
- Bacati iznimke samo u stanjima koja su stvarno iznimna
  - Ulazne podatke provjeriti što je moguće prije
  - Ne bacati iznimke za pogreške koje se mogu obraditi lokalno
- Izbjegavati bacanje iznimki u konstruktorima i finalizatorima (destruktörima), osim ako ih na istom mjestu i neхватамо
  - Problematično npr. u C++-u, iako nije neuobičajeno imati iznimke u konstruktorima u jezicima sa skupljačem smeća (npr. C# ili Java)
- Izbjegavati prazne blokove za hvatanje iznimki - catch { }
  - Iznimke ne ignorirati – ako ništa bolje ne можемо, onda barem evidentirati

# Preporuke za korištenje iznimki (2)

- Razne tipove pogrešaka obrađivati na konzistentan način kroz čitav kod
  - Razmotriti izradu centraliziranog sustava za dojavu iznimki u kodu
  - Zapisivati trag bačenih iznimki (log)
- Koristiti specifične iznimke (ne samo osnovne iznimke *Exception*) znajući što bacaju vlastite knjižnice
  - U poruci iznimke uključiti sve informacije o kontekstu nastanka iznimke
  - Odrediti ispravan nivo apstrakcije iznimke (na razini sučelja, a ne implementacije)
  - Originalna iznimka se može omotati

# Preporuke za korištenje iznimki (3)

- Hvatati specifične iznimke umjesto Exception
  - možda „uhvatimo“ nešto što nismo trebali
  - više catch blokova ako zahtjeva različito ponašanje
- Očistiti resurse u *finally* bloku ili koristiti podržane konstrukcije koje će to automatski obaviti
  - Ne bacati iznimke u *finally* bloku
  - try-with-resources u Javi
  - *using* u C#-u - vidi sljedeći slajd
- Razmotriti alternative (rukovanju) iznimkama

# Životni vijek objekta – čišćenje resursa

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

```
public class C : IDisposable {  
    public void Dispose() {  
        //zatvaranje datoteke, veze prema bazi  
        // 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.

# *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)

```
C a1 = new C("A1");
using (C b2 = new C("B2")) {
    C c3 = new C("C3");
    throw new Exception("Poruka");
}
a1.Dispose();
```

- *using* blok se može koristiti samo za one razrede koji implementiraju *IDisposable*
  - Koncept sličan try-with-resources u Javi. Pogledati IL kod s ILDASM  
Primjer:  DefensiveProgramming \ Using

# Tvrđnje

- Naredba kojom se program testira tako da određeni izraz mora biti istinit, a inače se izvršavanje programa zaustavlja
  - Obično se sastoji od izraza koji treba provjeriti i poruke koja se prikazuje ako izraz nije istinit
- Tvrđnje se koriste za uklanjanje pogrešaka (debugging) i dokumentiranje ispravnog rada programa
  - Koriste se u fazi kodiranja, naročito u razvoju velikih, komplikiranih programa te u razvoju programa od kojih se zahtijeva visoka pouzdanost
  - Uklanjanju se iz produkcijske verzije
- Pišemo ih na mjestima gdje se pogreške ne očekuju (tj. ne smiju se pojaviti)
  - Ako je tvrdnja istinita, onda znači da program radi kako je zamišljeno

# Tvrđnjama provjeravamo i dokumentiramo prepostavke

- Provjeravamo „nemoguće“ uvjete
  - Jesu li parametri za koje se smatra da su ispravni zaista ispravni?
    - Npr. računamo li opseg trokuta za nešto što nije trokut
      - npr. stranice duljine 2, 3, 5
  - Je li primljena referenca zaista valjana (*ne null*) kao što je trebala biti po dokumentaciji
- Detektiraju pogreške nastale negdje drugdje u kodu, kao posljedicu krivih prepostavki i interpretacija ili naknadnih modifikacija koda
- Tvrđnje ujedno mogu služiti i kao dokumentacija

# Tvrđnje 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:  DefensiveProgramming \ Barricades \ Course.cs

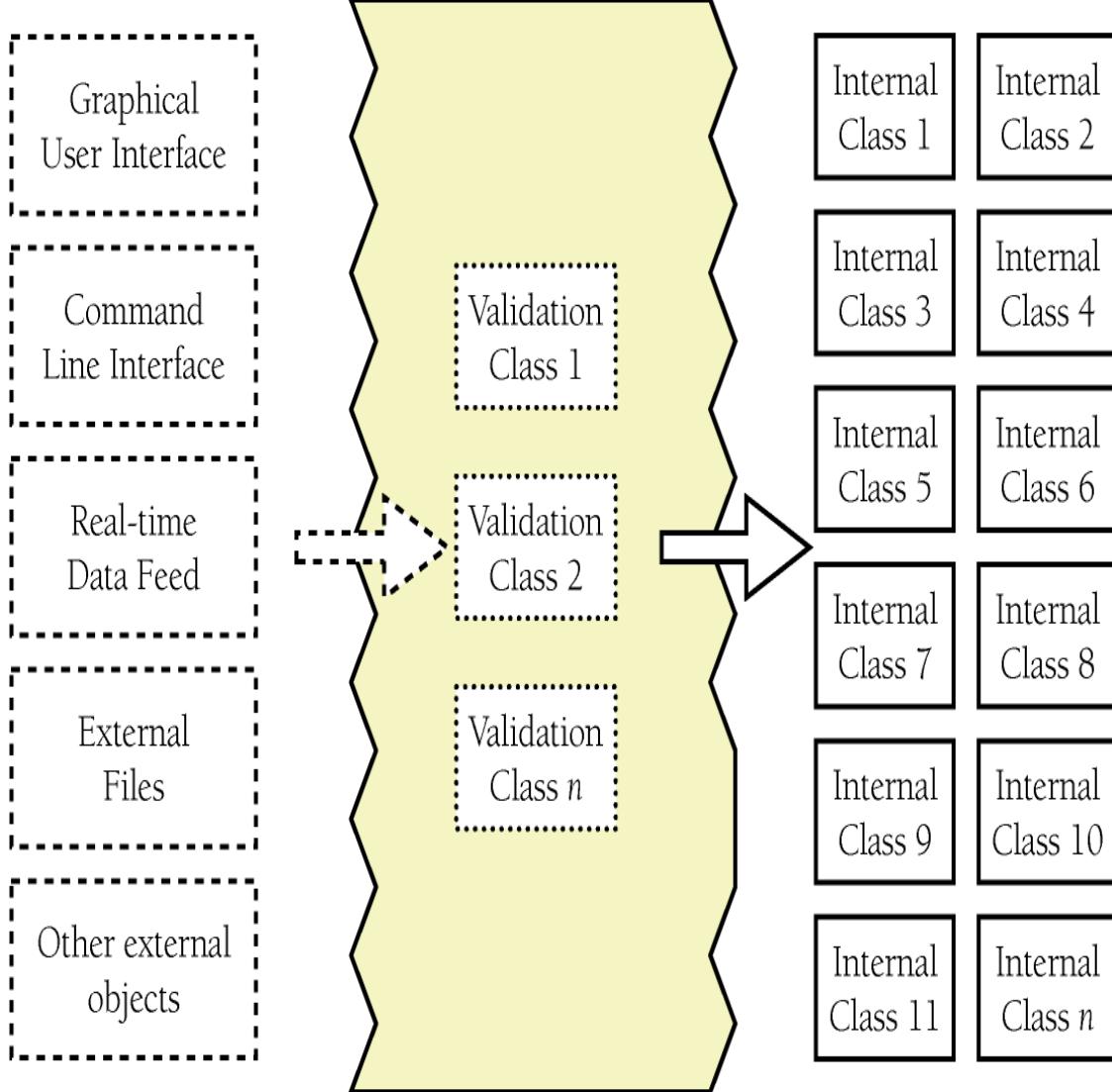
```
public double AverageGrade() {  
    int sum = 0;  
    foreach (var pair in grades) {  
        int grade = pair.Value;  
        Debug.Assert(grade >= 1 && grade <= 5,  
                    $"Invalid grade in dictionary {pair.Key} =  
                    {pair.Value}. This should never happens!");  
        sum += grade;  
    }...
```

# 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 čak i ako je pogreška posljedica neispravne prepostavke
- 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

# 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



Data here is assumed to be dirty and untrusted.

These classes are responsible for cleaning the data. They make up the barricade.

These classes can assume data is clean and trusted.

# Koncept barikada

- Primjer:  DefensiveProgramming \ Barricades \ Course.cs

```
public class Course {  
    private Dictionary<string, int> grades =  
        new Dictionary<string, int>();  
    public int this[string name] {  
        get {  
            //what if there is no student's name in the dictionary?  
            bool exists = grades.TryGetValue(name, out int grade);  
            return exists ? grade : -1;  
        }  
        set {  
            if (value < 1 || value > 5) {  
                throw new ArgumentOutOfRangeException(  
                    $"Invalid grade {value}. It should be from 1 to 5");  
            }  
            grades[name] = 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
    - Ne misle se nužno na *public* metode, već one koje služe za unos podataka
  - 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? Baciti iznimku?

# Otkrivanje pogrešaka

- Uobičajena zabluda programera je da se ograničenja koja se odnose na konačnu verziju softvera odnose i na razvojnu verziju
  - Treba biti spremjan 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:  DefensiveProgramming\Conditional

```
#define DEMO //definiramo simbol  
...  
  
#if (DEMO)  
    Console.WriteLine("Print something..."); //kod za debugiranje  
#endif
```

# 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:  DefensiveProgramming\Conditional

```
...
CheckSomething(); //ne izvodi se ako DEMO nije definiran
...
[Conditional("DEMO")]
static void CheckSomething() {
    ...
}
```

# Količina defenzivnog koda u završnoj verziji

- Previše defenzivnog koda može usporiti izvršavanje i povećati složenosti.
  - 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 „priateljske”
    - Obavijestiti korisnika o “unutarnjoj pogrešci” i navesti e-mail ili broj telefona tako da korisnik ima mogućnost prijaviti pogrešku

# *Checklist: Defensive Programming u Steve McConnell, Code Complete, 2nd Edition (str. 211 i 212)*

## CHECKLIST: Defensive Programming

### General

- Does the routine protect itself from bad input data?
- Have you used assertions to document assumptions, including preconditions and postconditions?
- Have assertions been used only to document conditions that should never occur?
- Does the architecture or high-level design specify a specific set of error-handling techniques?
- Does the architecture or high-level design specify whether error handling should favor robustness or correctness?
- Have barricades been created to contain the damaging effect of errors and reduce the amount of code that has to be concerned about error processing?
- Have debugging aids been used in the code?
- Have debugging aids been installed in such a way that they can be activated or deactivated without a great deal of fuss?
- Is the amount of defensive programming code appropriate—neither too much nor too little?
- Have you used offensive-programming techniques to make errors difficult to overlook during development?

### Exceptions

- Has your project defined a standardized approach to exception handling?
- Have you considered alternatives to using an exception?
- Is the error handled locally rather than throwing a nonlocal exception, if possible?
- Does the code avoid throwing exceptions in constructors and destructors?
- Are all exceptions at the appropriate levels of abstraction for the routines that throw them?
- Does each exception include all relevant exception background information?
- Is the code free of empty *catch* blocks? (Or if an empty *catch* block truly is appropriate, is it documented?)

### Security Issues

- Does the code that checks for bad input data check for attempted buffer overflows, SQL injection, HTML injection, integer overflows, and other malicious inputs?
- Are all error-return codes checked?
- Are all exceptions caught?
- Do error messages avoid providing information that would help an attacker break into the system?

# Postavke (konfiguracijski parametri) aplikacije

- Postavke (konfiguracijske parametre) aplikacije za koje očekujemo da će se mijenjati definirati izvan programa
  - kao varijable okruženja
  - unutar konfiguracijskih datoteka
- U protivnom promjena postavki zahtjeva ponovnu izgradnju programa (nova verzija?) i predstavlja potencijalni sigurnosni problem (npr. ispis u tragu stoga kod iznimke)

# Varijable okruženja (1)

- Varijable okruženje definirane operacijskim sustavom, okruženjem u oblaku ili datotekom *launchSettings.json*
  - Nastane ručno ili kroz Visual Studio
    - desni klik na projekt Debug → Enviroment Variables
- Primjer:  ... / Secrets / Properties / launchSettings.json

```
{  
  "profiles": {  
    "Secrets": {  
      "commandName": "Project",  
      "environmentVariables": {  
        "CustomEnvValue": "Demo"  
      }  
    }  
  }  
}
```

# Varijable okruženja (2)

- Vrijednost varijable okruženja moguće dobiti s *Environment.GetEnvironmentVariable*
- Primjer dohvata svih varijabli okruženja
  - 📁 DefensiveProgramming / Secrets / Program.cs

```
var enumerator = Environment
    .GetEnvironmentVariables()
    .GetEnumerator();
while(enumerator.MoveNext()) {
    Console.WriteLine(
        $"{enumerator.Key} = {enumerator.Value}");
}
```

# Konfiguracijske datoteke (1)

- U .NET Core-u uobičajeno appsettings.json
  - može i nešto drugo (ne mora biti u JSON formatu)
  - U web-aplikacijama će automatski biti uključena u konfiguracijske objekte, u konzolnoj aplikaciji je treba eksplisitno uključiti
- Primjer:  DefensiveProgramming / Secrets / appsettings.json
  - Desni klik → Copy To Output Directory : Copy if newer

```
{  
    "Name": "Bob",  
    "Demo": {  
        "Key0": 123,  
        "Key1": "something that should not be publicly available"  
    }  
}
```

# Konfiguracijske datoteke (2)

- Konfiguracijske datoteke se mogu kombinirati (i s varijablama okruženja) u zajednički objekt tipa *IConfiguration* pri čemu je bitan redoslijed uključivanja.
  - svaki novi izvor nadjačava identične parametre
  - Po potrebi koristiti parametar *optional*
- Primjer:  DefensiveProgramming / Secrets / Program.cs

```
IConfiguration configuration = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json")
    .AddJsonFile("missing_one.json", optional: true)
    .AddEnvironmentVariables()
    ...
    .Build();

Console.WriteLine($"Name = " + configuration["Name"]);
Console.WriteLine($"Key0 = " + configuration["Demo:Key0"]);
```

# Osjetljivi podaci u konfiguracijskim datotekama

- Što ako prilikom razvoja treba javno dijeliti izvorni kod s osjetljivim podacima u konfiguracijskim datotekama?
- 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
    - Mac & Linux:  
~/.microsoft/usersecrets/<userSecretsId>/secrets.json
  - Samo za razvoj! Datoteke nisu kriptirane
    - Prilikom kontinuirane isporuke osmisliti mehanizam zamjene šifre s pravom šifrom na produkciji, jer tajna datoteka možda ne postoji na serveru
- Alternativa: Odvojiti osjetljive podatke u posebne datoteke koje neće biti dio repozitorija

# Postavljanje datoteke s „tajnim” vrijednostima

- Izmijeniti projektnu datoteku tako da sadrži `UserSecretsId` i dodati odgovarajuće pakete (`Microsoft.Extensions.Configuration.UserSecrets`)
  - Primjer:  DefensiveProgramming / Secrets / Secrets.csproj

```
<PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
    <UserSecretsId>RPPP-Secrets</UserSecretsId>
</PropertyGroup>
```

- U naredbenom retku u mapi projekta pokrenuti  
`dotnet user-secrets set "Demo:Key1" "some secret val"`
  - Stvara .../RPPP-Secrets/secrets.json s odgovarajućim ključ
- Moguće i kroz Visual Studio opcijom *Manage User Secrets*
  - Ako `UserSecretsId` nije postavljen, Visual Studio automatski generira jedinstvenu vrijednost

# Uključivanje tajnih vrijednosti

- Primjer:  DefensiveProgramming / Secrets / Program.cs

```
IConfiguration configuration = new ConfigurationBuilder()  
    .AddJsonFile("appsettings.json")  
    .AddEnvironmentVariables()  
    .AddUserSecrets("RPPP-Secrets")  
    .Build();
```

# 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

# 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

# Razine važnosti zapisa (2)

- Information
  - Zapis trajnijeg karaktera koji služi za praćenje toka rada aplikacije
    - npr. informacija o posjetu određenoj stranici ili evidencija postavljenih kriterija pretrage
- 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

# Razine važnosti zapisa (3)

- 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
- Ponekad se kao nivo navodi i *None*
  - Trace < Debug < Information < Warning < Error < Critical < None

# Praćenje traga u .NET Coreu

- Microsoft.Extensions.Logging.Abstractions
  - Podrška za različite alate i odredišta zapisivanja tragova
  - Nekoliko ugrađenih implementacija
    - Npr. konzolni ispis (Microsoft.Extensions.Logging.Console)
  - Moguće dodati vlastitu implementaciju
- Sučelje *ILogger* kao apstrakcija nad različitim implementacijama
  - Može poslužiti za konzistentan i centralizirani način obrade iznimke

# Sučelje ILogger

- `IEnabled(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

# Pokrate za postupak Log

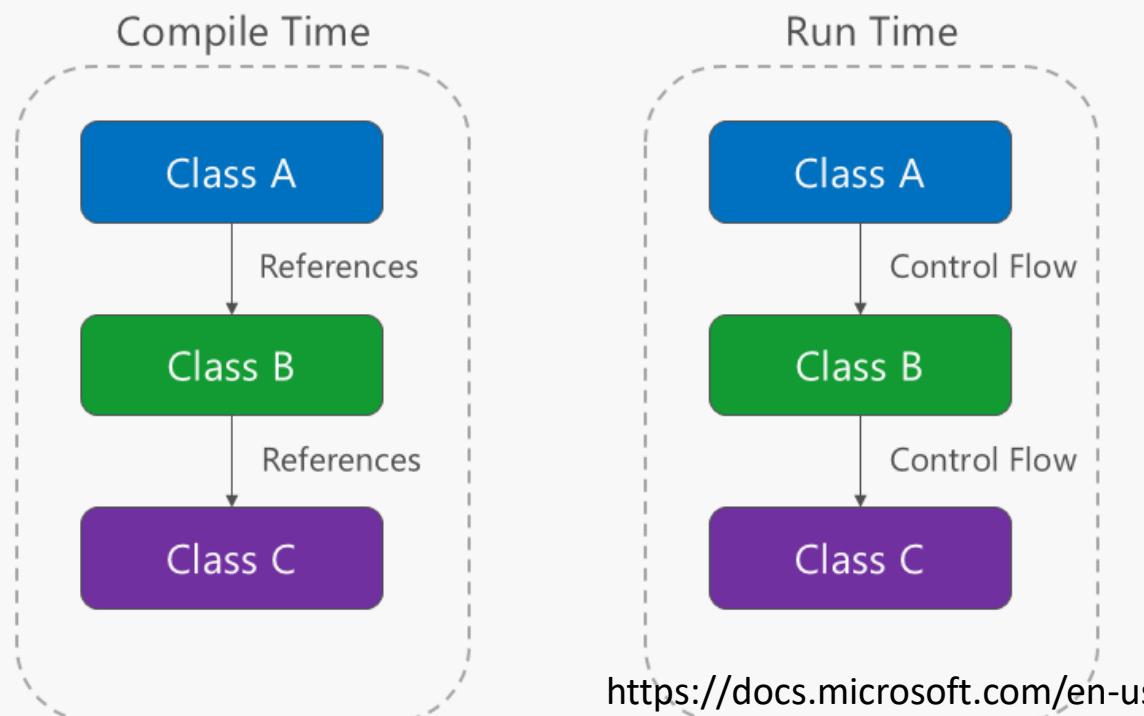
- Statički razred *Microsoft.Extensions.Logging.LoggerExtensions* s proširenjima za sučelje *ILogger*
- Nekoliko preopterećenih postupaka:
  - *LogTrace*
  - *LogDebug*
  - *LogInformation*
  - *.LogError*
  - *LogCritical*

# Primjer za praćenje traga i obradu iznimki

- Primjer:  DefensiveProgramming \ Logging
  - Sučelje *IDataLoader* definirano s jednom metodom za učitavanje parova (osoba, datum rođenja) iz određene datoteke
  - Implementacija sučelja (razred *DataLoader*) prilikom čitanja datoteke želi obraditi pogreške na konzistentan i centraliziran način što npr. *ILogger* omogućava.
  - Kako stvoriti objekt tipa *DataLoader* uz što manje kopčanja (ovisnosti) i omogućiti naknadne promjene
    - Konkretna odredišta se postavljaju prilikom inicijalizacije programa
  - Koristi se tehnika *Dependency Injection*
    - Konkretno u ovom slučaju *ConstructorInjection*
    - *DataLoader* kao argument konstruktora prima referencu tipa *ILogger*

# Uobičajeni način definiranja ovisnosti

## Direct Dependency Graph



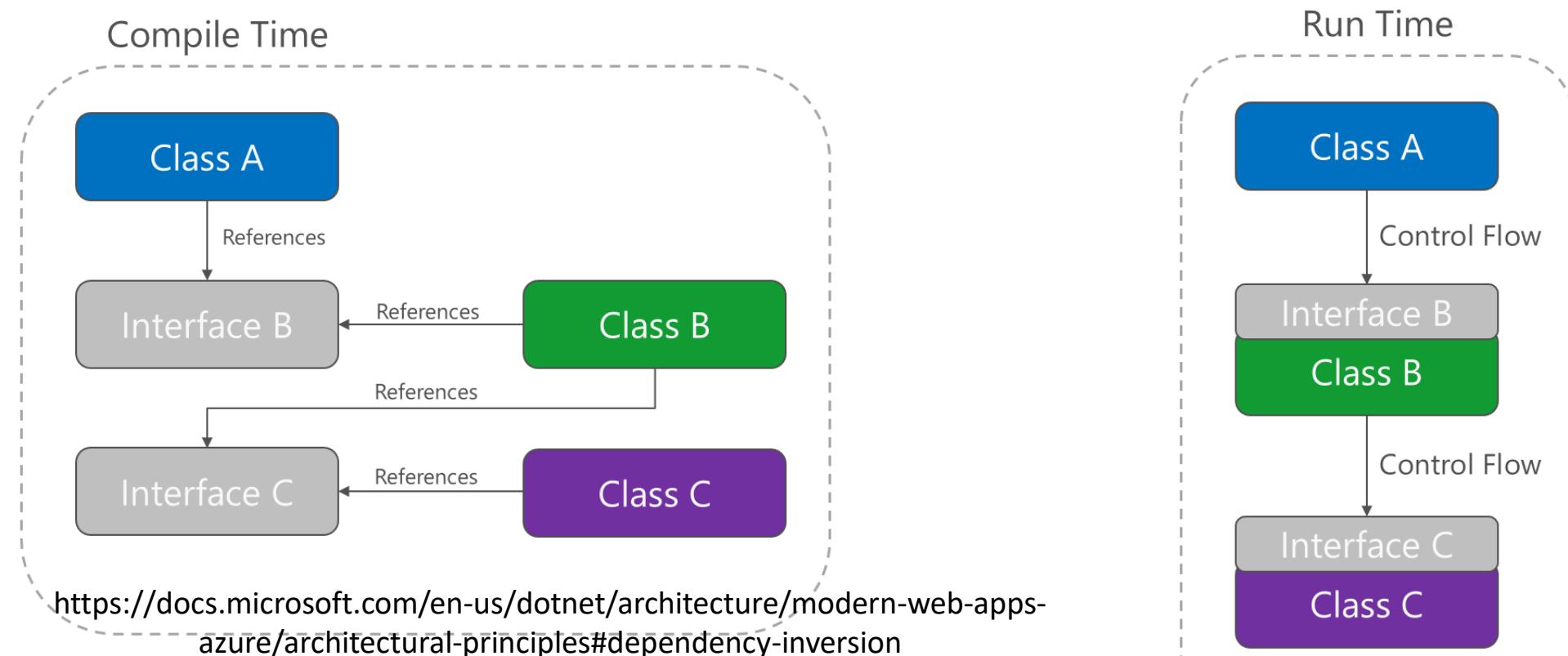
- Ovisnost komponenti odgovara redoslijedu korištenja

<https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#dependency-inversion>

# Dependency Inversion

- Ovisnost o sučelju, a ne o konkretnoj implementaciji
  - Bude povezano (umetnuto) naknadno (*Dependency injection*)

## Inverted Dependency Graph



# Postavljanje lanca ovisnosti

- Za pojedino sučelje ili razred definira se implementacija koja se koristiti kad stvaranje objekta ovisi o tom sučelju (razredu)
  - Primjer  DefensiveProgramming \ Logging \ Program.cs

```
IServiceCollection services = new ServiceCollection();
var provider = services.AddLogging(...) //definira za ILogger
    .AddTransient<IDataLoader, DataLoader>()
    .BuildServiceProvider();
```

- Add[Transient|Singleton|Scope] određuju način stvaranja objekta
  - Transient = svaki put stvori novi primjerak
- Novi objekti se, umjesto s new stvaraju koristeći *GetRequiredService*
  - Uzrokuje kreiranje potrebnih objekata u lancu ovisnosti

```
var dataLoader = serviceProvider.GetRequiredService<IDataLoader>();
```

# Postavljanje minimalne razine praćenja traga (1)

- Minimalna razina praćenja traga određuje hoće li neka poruka uopće biti proslijeđena povezanim *loggerima*
  - Trace < Debug < Information < Warning < Error < Critical < None
  - Pojedini *logger* može imati i zasebnu dodatnu konfiguraciju (za poruke koje uopće stignu do njega)
- Razina se može postaviti u konfiguracijskoj datoteci
  - Primjer  DefensiveProgramming \ Logging \ appsettings.json

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Trace"  
    }  
  }  
}
```

# Postavljanje minimalne razine praćenja traga (2)

- Prilikom podešavanja lanca ovisnosti postavke minimalne razine traga pročitane iz konfiguracijske datoteke
  - Primjer  DefensiveProgramming \ Logging \ Program.cs

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

IServiceCollection services = new ServiceCollection();
var provider = services.AddLogging(configure => {
    configure.AddConfiguration(
        configuration.GetSection("Logging"));
    ...
})
.BuildServiceProvider();
```

# 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
- Uključuje se prilikom konfiguriranja praćenja traga
  - Primjer:  DefensiveProgramming \ Logging \ Program.cs

```
IServiceCollection services = new ServiceCollection();
var provider = services.AddLogging(configure => {
    configure.AddConfiguration(
        configuration.GetSection("Logging"));
    configure.AddConsole();
    configure.AddNLog(new
NLogProviderOptions { RemoveLoggerFactoryFilter = false });
    ).BuildServiceProvider();
```

# Konfiguracijska datoteka za NLog

- Moguće definirati više različitih odredišta, a zatim pravila koja određuju gdje će sve pojedini zapis biti evidentiran
  - Primjer:  DefensiveProgramming \ Logging \ nlog.config

```
<targets>
  <target xsi:type="File" name="allfile"
    fileName="logs\nlog-all-${shortdate}.log"
    layout="${longdate}|${event-
properties:item=EventId_Id:whenEmpty=0}|${logger}|${uppercase:${leve
1}}|${message} ${exception}" />
  <target xsi:type="File" name="ownFile" ... />
  <target xsi:type="Null" name="blackhole" />
</targets>
<rules>
  <logger name="*" minlevel="Trace" writeTo="allfile" />
  <logger name="Microsoft.*" minlevel="Trace" writeTo="blackhole"
        final="true" />
  <logger name="*" minlevel="Trace" writeTo="ownFile" />
</rules>
```

# Razvoj primijenjene programske potpore

---

## 5. Rad s bazom podataka

# Zajednički SQL server

- SQL Server: fantom.fer.hr,3000
- Ogledna baza podataka: Firma
  - SQL Server Authentication: rppp/zaporka (navедено u uputama u repozitoriju na FER webu)
  - Moguće mijenjati podatke u svrhu testiranja
- Baze podataka oblika PPPxx:
  - Baza podataka za grupu XX
  - Podaci za spajanje dostavljeni pojedinoj grupi

Server type:	Database Engine
Server name:	fantom.fer.hr,3000
Authentication:	SQL Server Authentication
Login:	rpppXX
Password:	*****

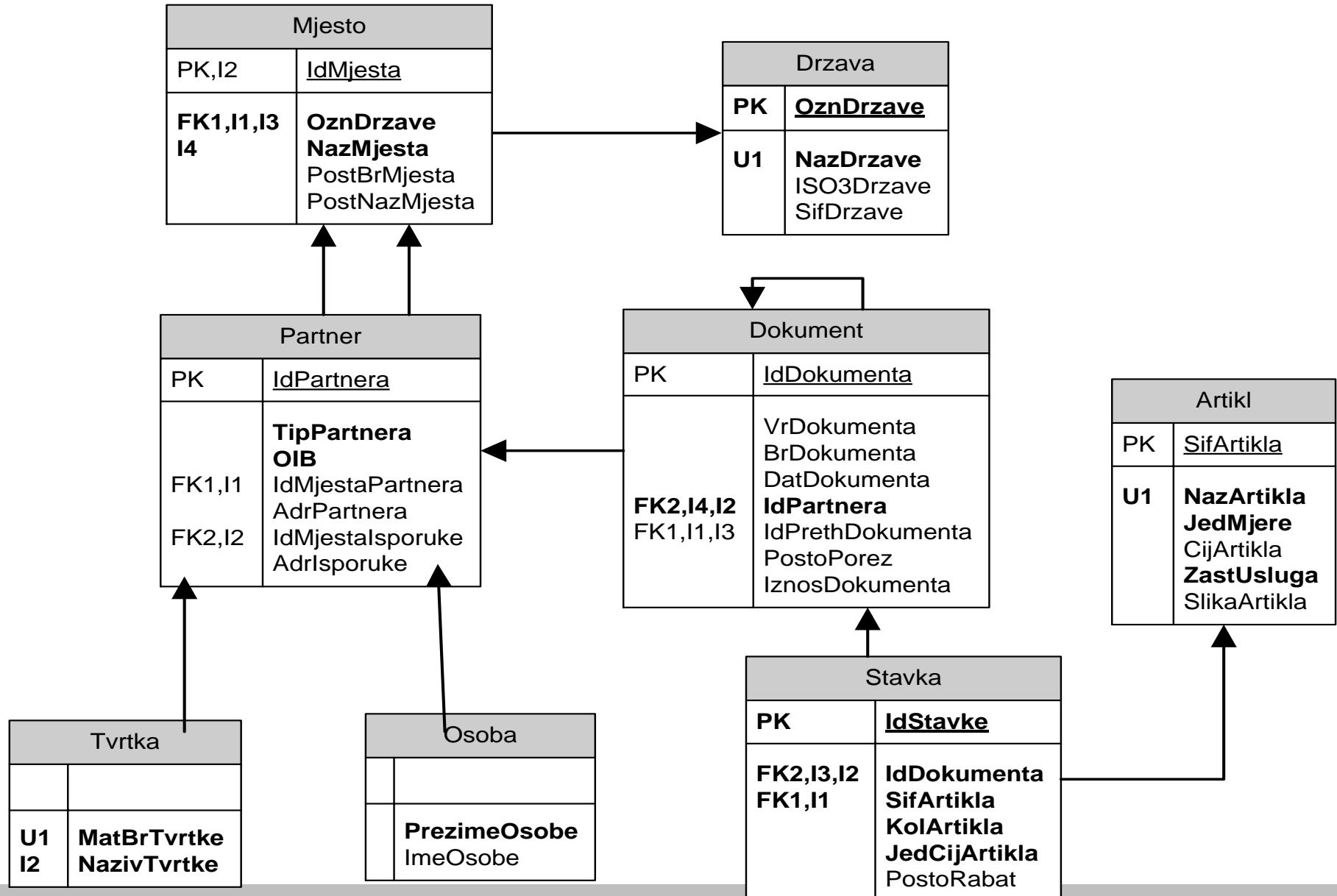
# Lokalna instalacija SQL Servera

- Nije nužna, ali može biti praktična (cca 1.6 GB)
  - Samostalna instalacija
  - Docker
    - [https://hub.docker.com/\\_/microsoft-mssql-server](https://hub.docker.com/_/microsoft-mssql-server)

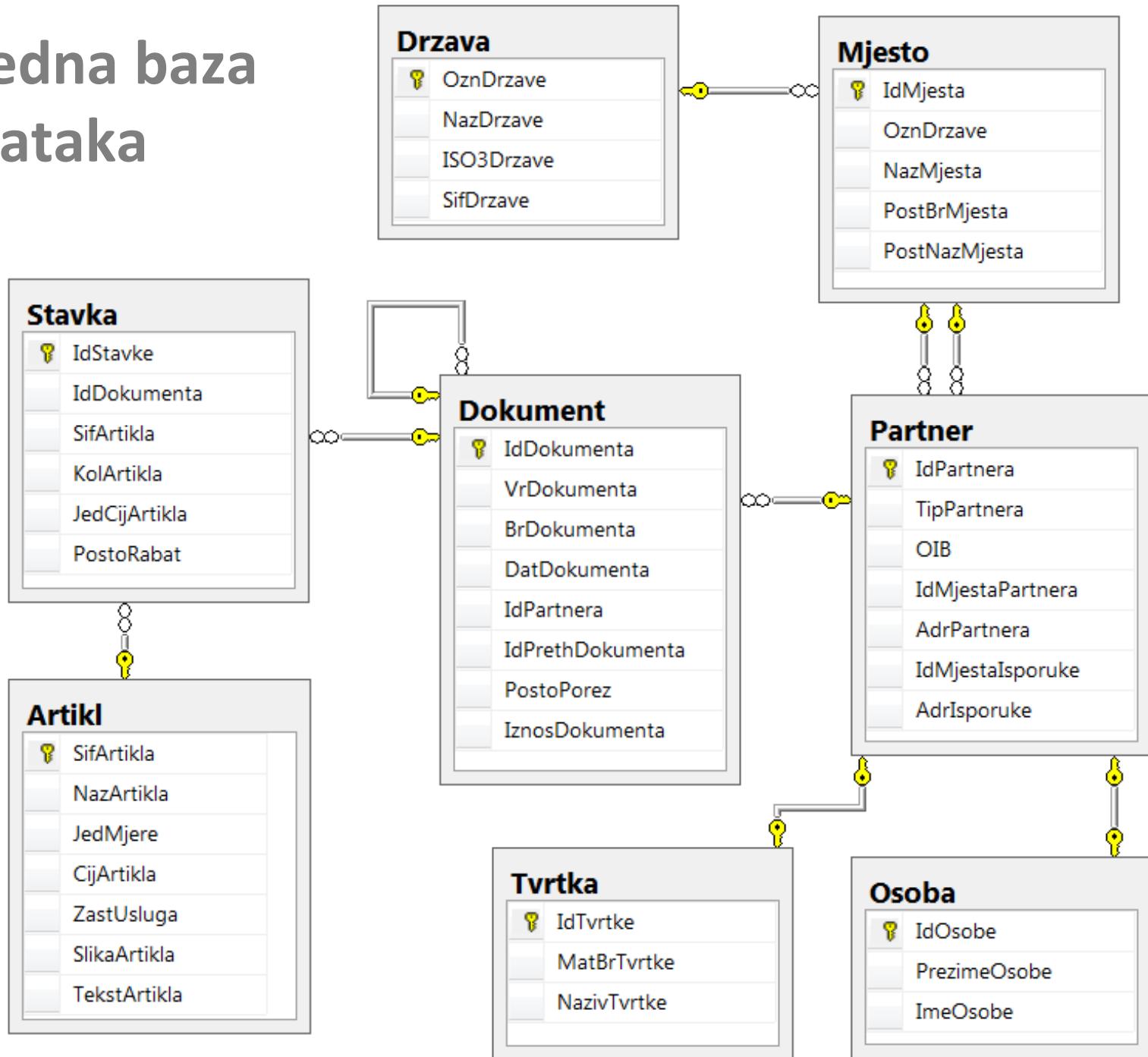
```
docker run -it -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=nestonene.trivialno" -p 1433:1433 --name sql-server-2019 mcr.microsoft.com/mssql/server:2019-latest
```

Server type:	Database Engine
Server name:	localhost,1433
Authentication:	SQL Server Authentication
Login:	sa
Password:	*****

# Ogledna baza podataka (dijagram - MS Visio)

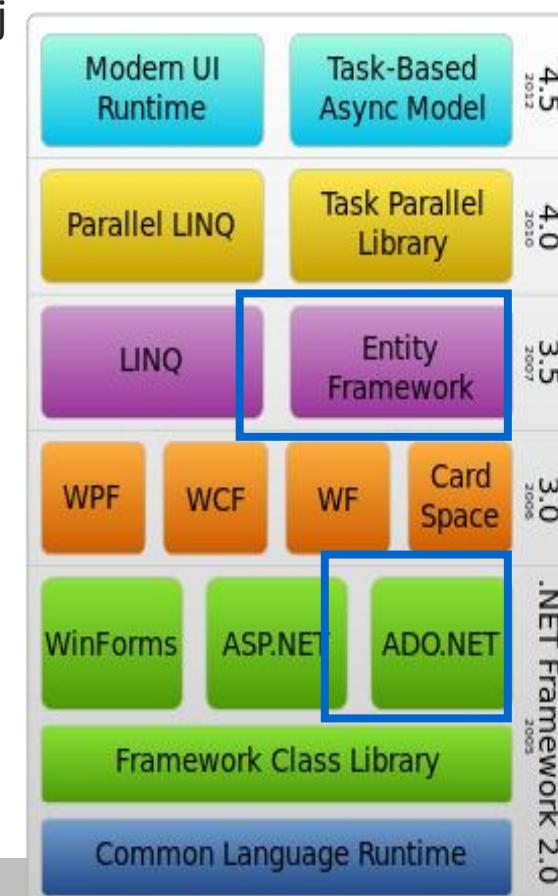


# Ogledna baza podataka



# .NET Framework i ADO.NET

- ActiveX Data Objects .NET (ADO.NET) je tehnologija za rukovanje podacima unutar .NET Frameworka koja 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



# Opskrbljivači (davatelji) podataka

- Davatelji za različite tehnologije (SQL Server, PostgreSQL, SQLite, MongoDB, ...)
  - direktni pristup ili tehnologije s određenom razinom apstrakcije kao što su ORM, npr. Entity Framework Core
    - <https://docs.microsoft.com/hr-hr/ef/core/providers/?tabs=dotnet-core-cli>
    - <https://devblogs.microsoft.com/dotnet/net-core-data-access/>
- Microsoft.Data.SqlClient
  - optimiran za rad s MS SQL Server-om
  - razredi: *SqlCommand*, *SqlConnection*, *SqlDataReader*, ...
- 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

# 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

# Priklučak na bazu podataka

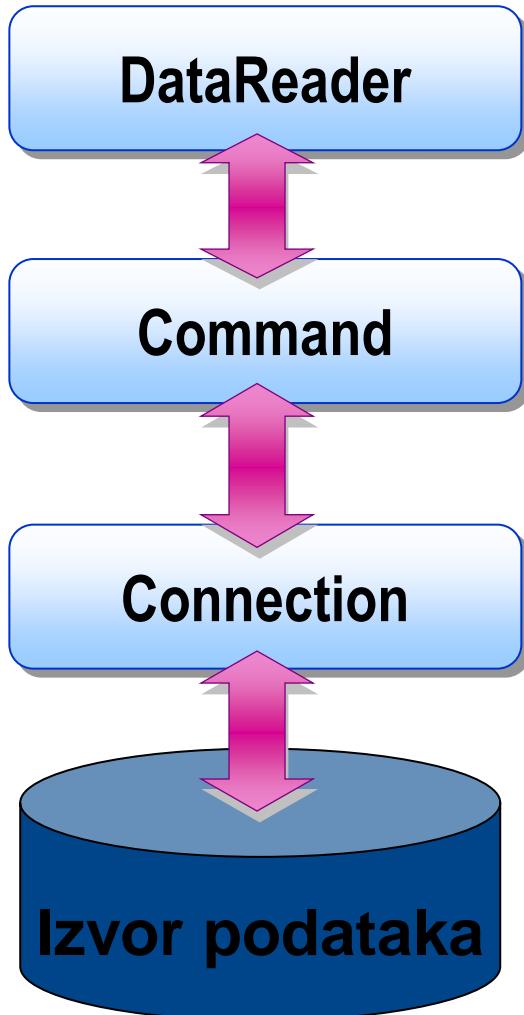
- Priklučak, veza (Connection)
  - otvara i zatvara vezu s fizičkim izvorom podataka
  - omogućuje transakcije i izvršavanje upita nad bazom podataka
- Sučelje *System.Data.IDbConnection*  
i apstraktni razred *System.Data.DbConnection*
- Implementacije: *NpgsqlConnection*, *SqlConnection*, ...
- Važnija 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* – priključivanje na izvor podataka
  - *Close* - uklanjanje s izvora podataka

# 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>

# Izravna obrada podataka

# 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 primjerima koji slijede potrebno uključiti dodatne biblioteke
  - *Manage NuGet Packages* ili ručno ažurirati csproj datoteke
- U primjerima s izravnom obradom ti paketi su
  - Microsoft.Data.SqlClient
  - Microsoft.Extensions.Configuration.UserSecrets

 .NET	<b>Microsoft.Data.SqlClient</b> by Microsoft	5.0.1
	Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS)	
 .NET	<b>Microsoft.Extensions.Configuration.UserSecrets</b> by Microsoft	6.0.1
	User secrets configuration provider implementation for Microsoft.Extensions.Configuration.	

# Skica rješenja izravne obrade podataka

- Kostur rješenja s izravnom obradom, ali bez obrade iznimki

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

# Postavke priključka na bazu podataka

- Postavke staviti u konfiguracijsku datoteku pod *ConnectionString*s
  - Nije nužno koristiti taj naziv, ali postoje postupci (pokrate) koje očekuju postavke upravo pod tim ključem
  - Voditi računa o tome je li izvorni kod javno dostupan
- Primjer:  DataAcess / DataReader / appsettings.json
  - Desni klik → Copy To Output Directory : Copy if newer

```
{  
  "ConnectionStrings": {  
    "Firma": "Data Source=.;Initial Catalog=Firma;Integrated  
Security=true"  
  }  
}
```

- Primjer:  ...UserSecrets/Firma/secrets..json

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

# Dohvat postavki priključka na bazu podataka

- Dohvatljivo iz koda pomoću razreda *ConfigurationBuilder*
- Primjer:  DataAccess / DataReader / Program.cs
  - Metoda proširenja *GetConnectionString* koja u JSON datoteci traži vrijednost ispod elementa *ConnectionStrings*

```
IConfiguration configuration = new ConfigurationBuilder()  
    .AddJsonFile("appsettings.json")  
    .AddUserSecrets<Program>()  
    .Build();  
  
string connString = configuration  
    .GetConnectionString("Firma");
```

- Mapa za tajne vrijednosti određena u csproj datoteci projekta kojem pripada klasa Program

```
<PropertyGroup>  
    <UserSecretsId>Firma</UserSecretsId>  
</PropertyGroup>
```

 DataAccess / DataReader / DataReader.csproj

# 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?
    - Priključak implementira sučelje *IDisposable*.
      - *Dispose* je (u ovom slučaju) ekvivalentan *Close* → koristiti using
    - Primjer  DataAccess \ DataReader \ Programs.cs

```
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 je čitanje preko *reader*a izvan metode u kojoj *reader* nastaje)
  - Automatsko zatvaranje priključka kad se zatvori čitač  
command.ExecuteReader(System.Data.CommandBehavior.CloseConnection)

# 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

# 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 spremi u polje objekata
    - public virtual int GetValues( object[] values );
  - *GetXX* (*GetInt32*, *GetChar*, ...)- dohvaća vrijednost zadanog stupca prepostavljajući određeni tip

# Neovisnost o konkretnoj implementaciji

- Primjeri se mogu poopćiti na način da se za tip reference umjesto konkretnih implementacija koriste sučelja ili 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

# 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 / ParamsAndProc / 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 \ ParamsAndProc \ Program.cs – ParametrizedQueryDemo

```
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`

# Upit s više skupova rezultata

- U slučaju da rezultat upita vraća više skupova rezultata, svaki sljedeći dohvata se postupkom NextResult na čitaču podataka
  - Primjer:  DataAccess \ ParamsAndProc \ Program.cs – ParametrizedQueryDemo

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

# Pozivi pohranjenih procedura

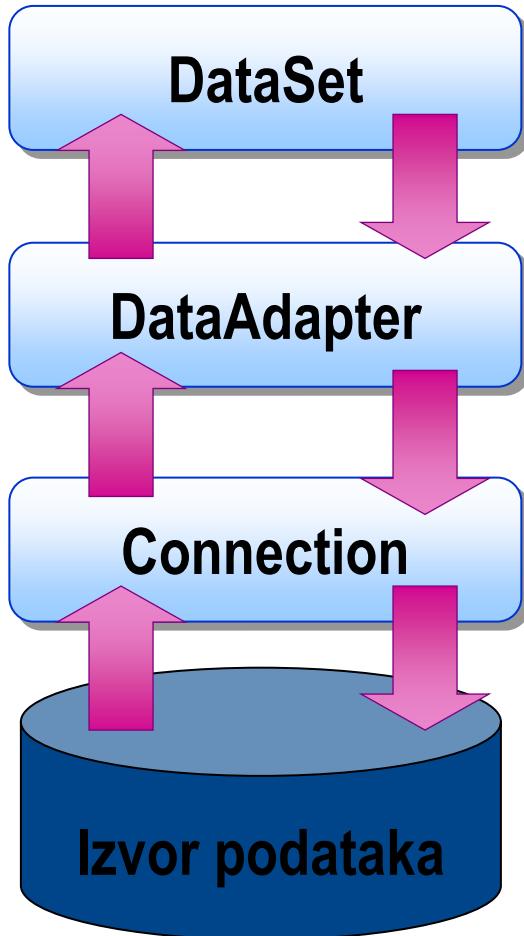
- Primjer:  DataAccess \ ParamsAndProc \ Program.cs – ProcedureDemo
  - Parametri procedure navode se kao i kod parametriziranih upita
  - Svojstvo  *CommandType* na naredbi potrebno je postaviti na *System.Data.CommandType.StoredProcedure*
  - Ako procedura ne vraća skup podataka, koristi se postupak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

```
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
```

# Lokalna obrada podataka

Entity Framework Core

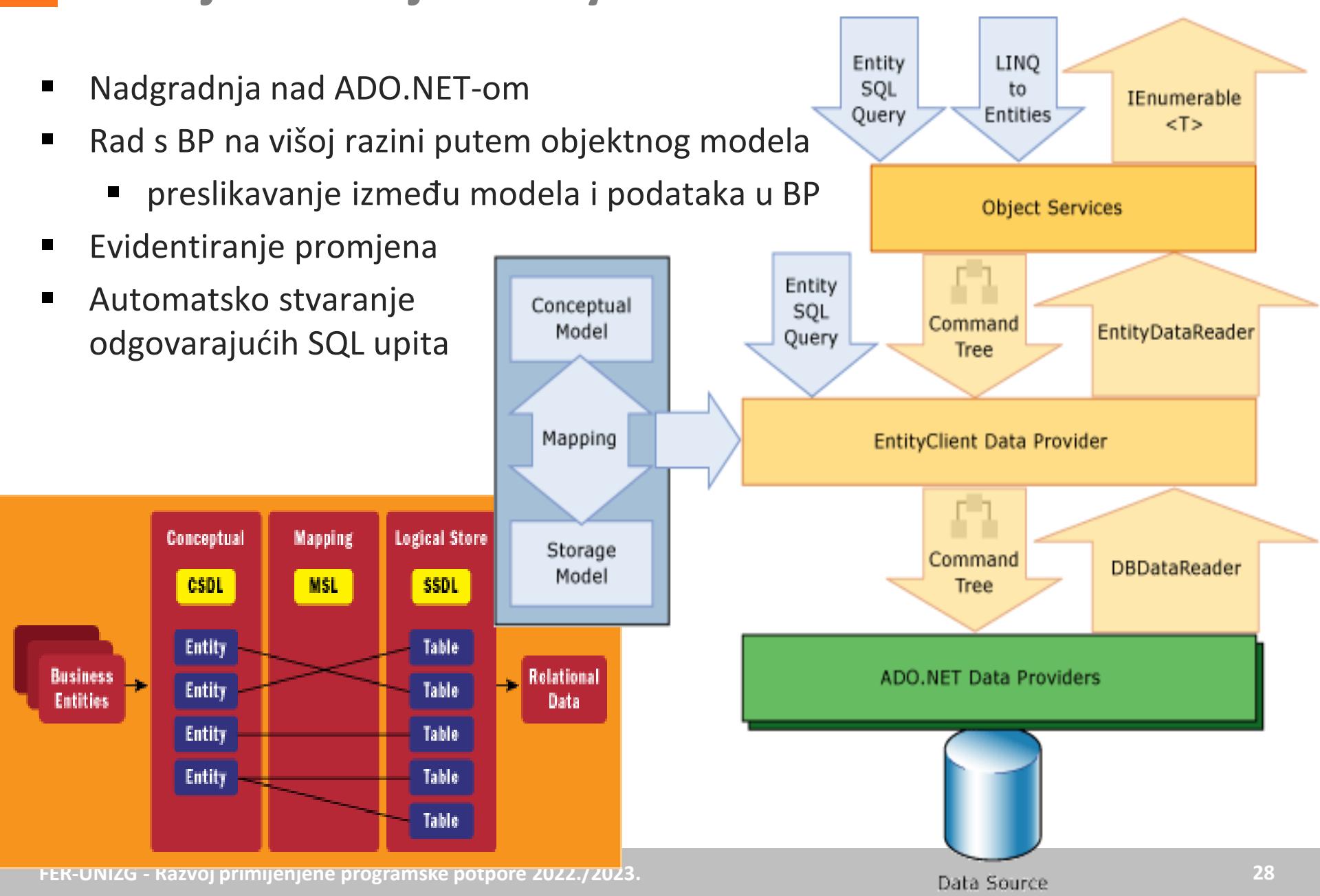
# Lokalna obrada podataka



- Podaci se obrađuju lokalno
  - *DataSet* reprezentira stvarne podatke pohranjene u memoriju
- 1. Otvori priključak
- 2. Napuni DataSet
- 3. Zatvori priključak
- 4. Izmijeni DataSet
- 5. Otvori priključak
- 6. Ažuriraj izvor podataka
- 7. Zatvori priključak
  - Ideja lokalne obrade podataka *DataSetom* „prenesena“ na EntityFramework

# 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



# 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 temeljem napisanih razreda. Izgled 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 se ne stvara nova baza podataka.
  - Razredi se mogu stvoriti ručno ili nekim od generatora.
- *Code First with Migrations*
  - Izvršavaju se posebno definirani postupci (migracije) - Up/Down
- *Entity Framework Core podržava samo Code First varijante*
  - **U primjerima koristimo *Code First from existing database***

# Stvaranje 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
```

1. ili dodati koristeći opciju Manage NuGet Packages

3. U naredbenom retku izvršiti sljedeće dvije naredbe

```
dotnet restore
```

```
dotnet ef dbcontext scaffold
```

```
"Server=fantom.fer.hr,3000;Database=Firma;User
```

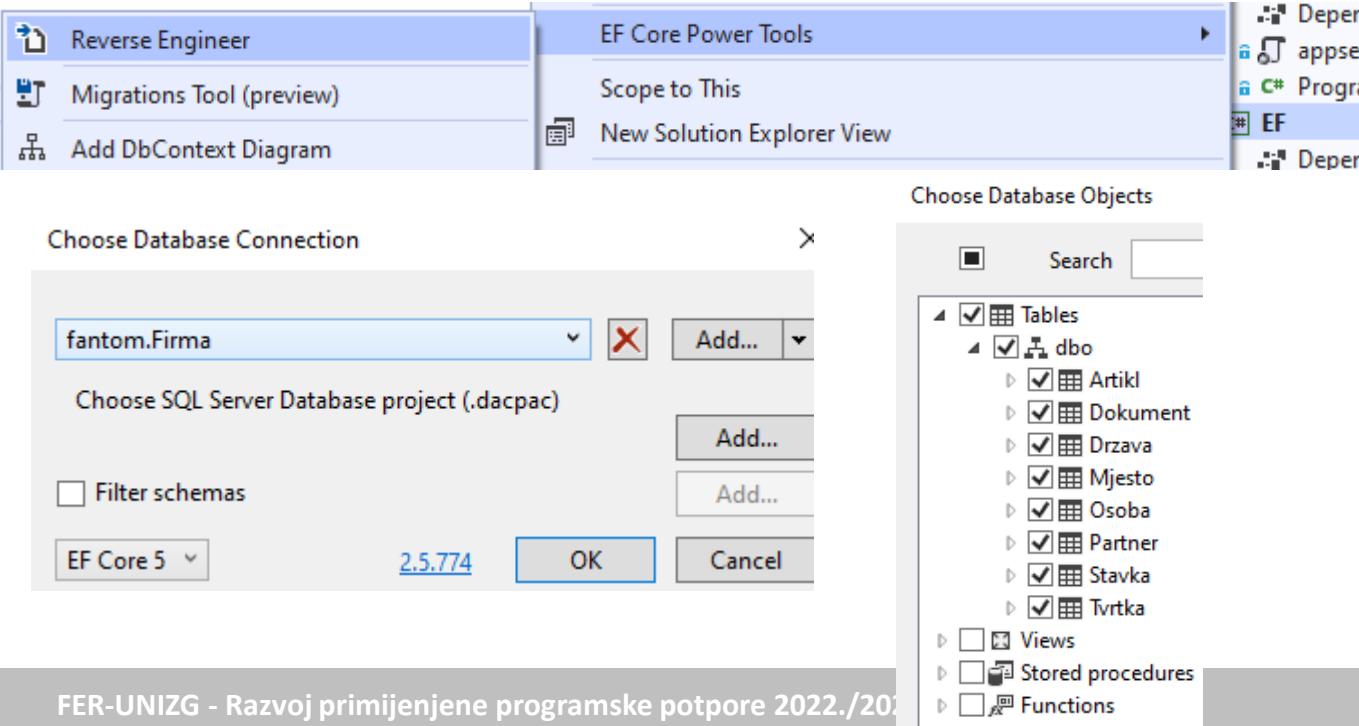
```
Id=rppp;Password=*" Microsoft.EntityFrameworkCore.SqlServer -
```

```
-o Model -t Artikl -t Dokument -t Drzava -t Mjesto -t Osoba -t
```

```
Partner -t Stavka -t Tvrtka --no-pluralize
```

# EF Core Power Tools

- EF Core Power Tools (dodatak za Visual Studio)
  - <https://marketplace.visualstudio.com/items?itemName=ErikEJ.EFCorePowerTools>
  - praktičniji način za stvaranje (i naknadno ažuriranje modela)
    - nudi mogućnost uvoza i procedura i funkcija iz SQL servera
    - Desni klik na projekt → EF Core Power Tools → Reverse Engineer



Generate EF Core Model in Project EF

Context name

Namespace

EntityTypes path (f.ex. Models) - optional Model

EntityTypes sub-namespace (overrides path) - optional

DbContext path (f.ex. Data) - optional

DbContext sub-namespace (overrides path) - optional

What to generate

Naming

Pluralize or singularize generated object names (English)

Use table and column names directly from the database

Use DataAnnotation attributes to configure the model

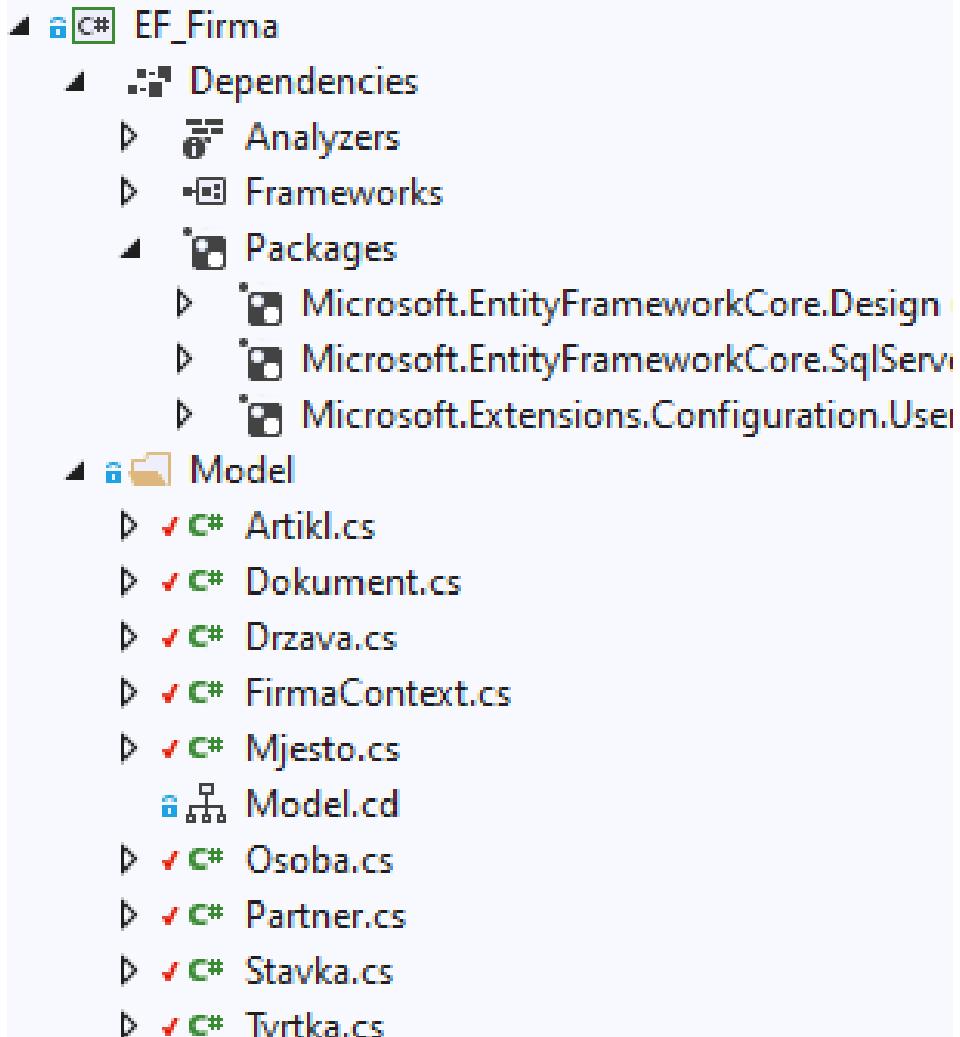
Customize code using Handlebars templates C#

Include connection string in generated code

Install the EF Core provider package in the project

# Model nastao temeljem postojeće BP

- 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 inicialno tvrdo kodirane u FirmaContext.cs
  - **Potrebno ukloniti i prebaciti u konfiguracijsku datoteku**



# Postavke spajanja na BP korištenjem EF-a (1)

- 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 kodom za dohvat konfiguracijskih postavki

```
void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {  
    var config = new ConfigurationBuilder()  
        ...  
        .Build();  
    string connString = config.GetConnectionString("Firma");  
    optionsBuilder.UseSqlServer(connString);
```

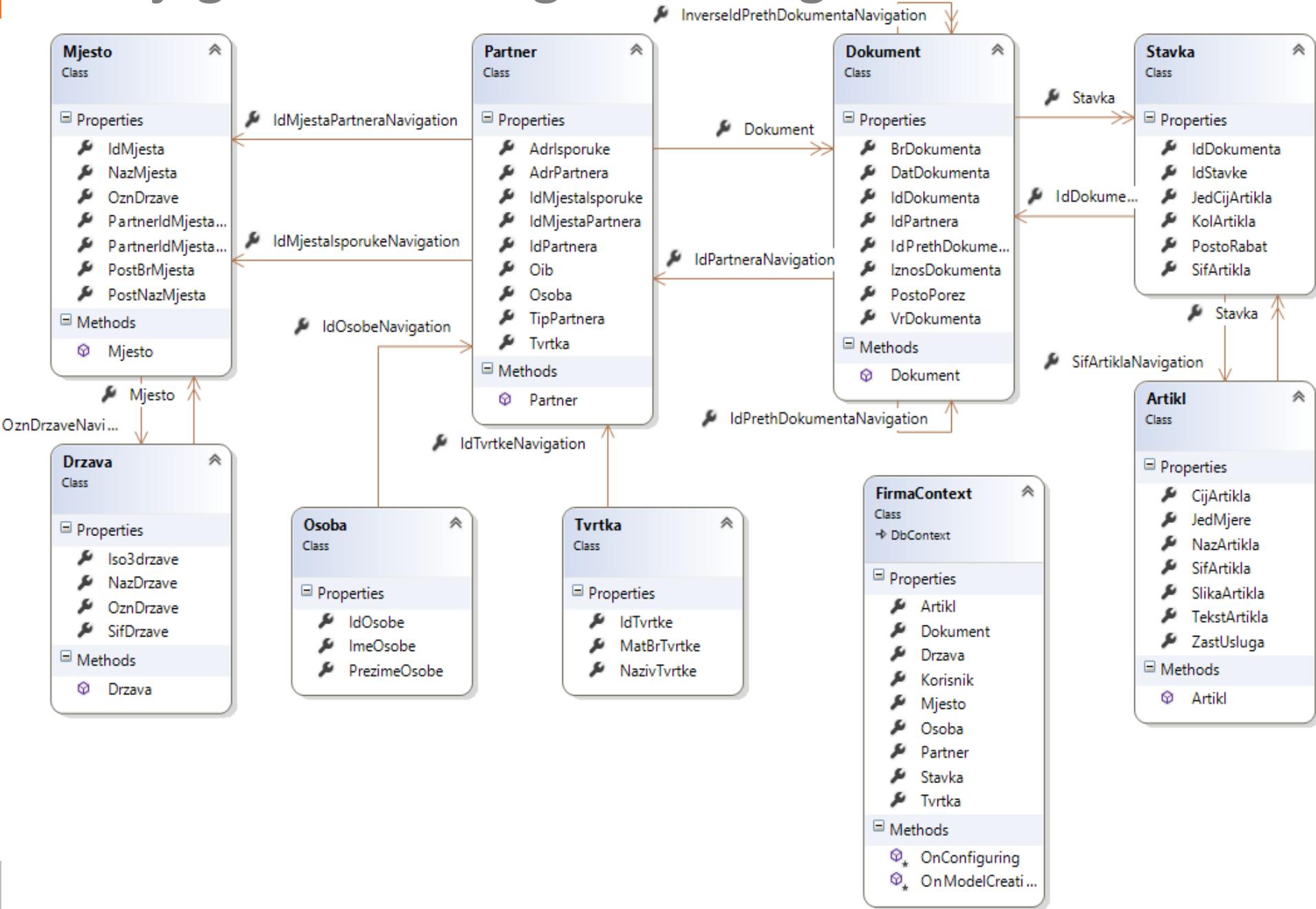
- Nije idealno rješenje, jer model određuje naziv konfiguracijske datoteke i naziv ključa
- Bolje rješenje na sljedećem slajdu

# Postavke spajanja na BP korištenjem EF-a (2)

- Onemogućiti stvaranje *FirmaContexta* direktno
  - Obrisati konstruktor bez argumenata
    - *Connection string* kao argument? → Nije praktično. Potrebno svaki put prije instanciranja dohvatiti *connection string*
  - Posljedično može se obrisati *OnConfiguring*
- Uspostaviti lanac ovisnosti potreban za stvaranje objekta tipa *FirmaContext* koristeći objekt tipa *ServiceProvider*
  - Svaki put potrebno stvoriti novi kontekst (Transient)
  - Primjer:  DataAccess \ EF \ Model \ Program.cs : BuildDI

```
IServiceCollection services = new ServiceCollection();
var provider = services
    .AddDbContext<FirmaContext>(options=>{
        options.UseSqlServer(
            configuration.GetConnectionString("Firma"));
    }, contextLifetime: ServiceLifetime.Transient)
    .BuildServiceProvider();
```

# Dijagram razreda generiranog modela



# 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 \ 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
    - Generirani kontekst sadrži i parcijalne metode

# Preslikavanje između EF modela i BP (1)

- Primjer:  DataAccess \ EF \ 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 razreda 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)

# Važnija svojstva razreda *DbSet*

- *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

# Dodavanje novog zapisa

- Primjer:  DataAccess \ EF \ Program.cs - AddProduct
  - 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())
using (var context =
            serviceProvider.GetService<FirmaContext>()) {
    Artikl artikl = new Artikl() { ... };
    context.Artikl.Add(artikl);
    context.SaveChanges();
}
```

# Ažuriranje postojećeg zapisa

- Primjer:  DataAccess \ EF \ Program.cs – ChangeProductPrice
  - 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 i pohraniti promjene u kontekstu

```
using (var context = ...) {  
    Artikl artikl = context.Artikl.Find(sifraArtikla);  
    //moglo je i context.Find<Artikl>(sifraArtikla);  
    artikl.CijArtikla = 750m;  
    context.SaveChanges();  
}
```

# Brisanje zapisa

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

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

# 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 eksplisitim pozivom postupka *Load (LoadAsync)*
  - Moguće ulančavanje upita (rezultat upita najčešće *IQueryable<T>*)
  - Podaci iz vezane tablice se učitavaju pri svakom dohvatu ili eksplisitno korištenjem postupka *Include* (kreira join upit u sql-u)
- Primjer  `DataAccess \ EF \ Program.cs` – `PrintMostExpensives`
  - Primjer upita za dohvat prvih n najskupljih artikala

```
var query = context.Artikl.Include(a => a.Stavka)
    .AsNoTracking()
    .OrderByDescending(a => a.CijArtikla)
    .Take(n);

foreach (Artikl artikl in query) { ... }
```

# 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
```

# 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
  - Primjer  DataAccess \ EF \ Program.cs – PrintMostExpensivesAnonymous

```
var query = context.Artikl
    .OrderByDescending(a => a.CijArtikla)
    .Select(a => new {
        a.NazArtikla, a.CijArtikla,
        Sales = a.Stavka.Count
    })
    .Take(n);
foreach (var product in query) { ... }
```

# 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
```

# Ostale mogućnosti EF-a

- Nakon uspješnog upita EF automatski vrši dohvati primarnog ključa koji je definiran kao tip *identity*
- U trenutku pisanja ovih materijala EF Core ne podržava dodavanje procedura u model, ali EF Core Power Tools može generirati kod koji to omogućava
- 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 poglavljiju s web-aplikacijama

# Migracije (1)

- Generirani kod vrši promjene u bazi temeljem promjena u modelu
  - Početne migracije ili promjena u odnosu na predhodno stanje
  - Migracije se mogu koristiti neovisno o načinu dobivanja modela (*code first* ili *code first from existing database*)
- Praktično nužno u slučaju da direktni pristup bazi podataka nije moguć, ili ako svaki korisnik ima svoju bazu
- Može biti praktično u slučaju kontinuirane isporuke, jer rješava raskorak između promjene baze podataka i usklađivanja modela
  - Potencijalni problem s više istovremenih instanci
- Migracije se mogu pokretati ručno (*dotnet ef* u naredbenom retku ili *Add-Migration* u *Package Manager Console*) ili kroz kod.

# Migracije (2)

- Promotriti što bi se generiralo s

```
dotnet ef migrations add CreateDb -c FirmaContext -o  
Migrations
```

- Može se promijeniti *connection string* i unijeti postavke za lokalno instalirani SQL server i izvršiti

```
dotnet ef database update -c FirmaContext
```

- Što bi se dogodilo da u EF model dodamo nova svojstva i kreiramo novu migraciju?

# Razvoj primijenjene programske potpore

---

**6. Web-aplikacije**  
ASP.NET Core MVC  
CRUD nad jednostavnom tablicom

# Kakvu web-aplikaciju želimo/trebamo?

- SPA (*single page application*) + web servisi
  - *Client rendered UI*
- „Klasična“ web-aplikacija
  - *Server rendered UI*
  - ne isključuje korištenje web-servisa i dinamičnosti pojedinih stranica
- U oba slučaja „sveti gral“ je razdvajanje ovisnosti komponenti
  - Kako dohvatiti podatke?
  - Kako prikazati podatke?
  - Kako ubrzati prikaz?
  - Koja su (poslovna) pravila i ograničenja podataka i kako ih definirati?
  - Što su podaci i kakva je funkcionalnost aplikacije?

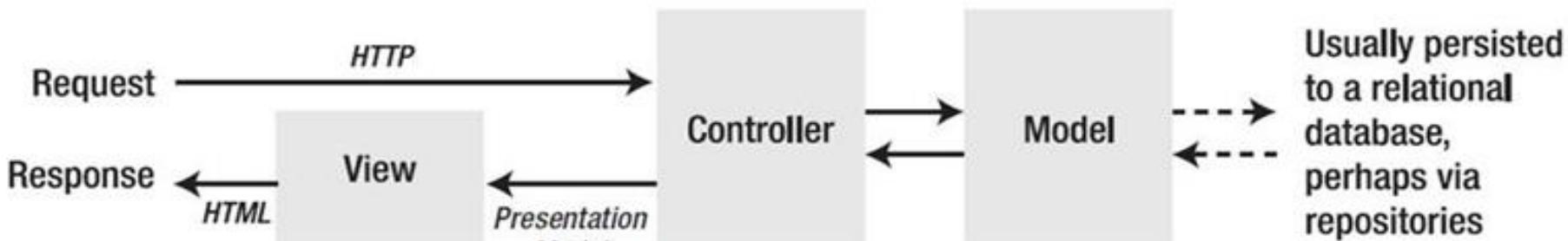
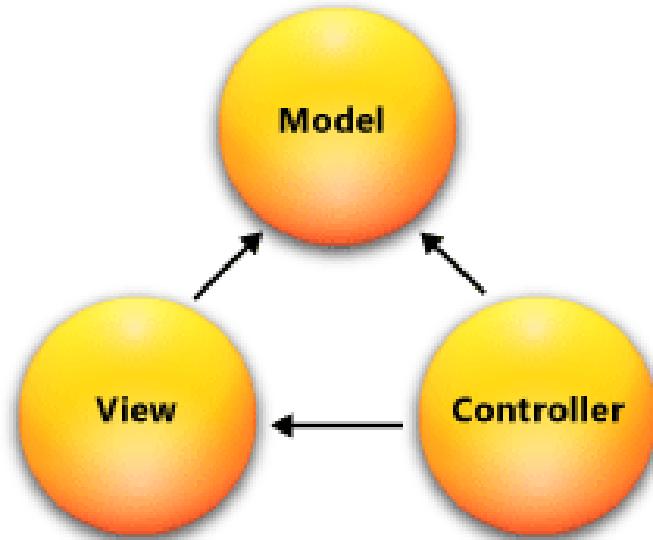
# MVC kao primjer obrasca *Front controller*

- Zahtjevi centralizirani na jedan upravljač (*Front Controller pattern*)
  - *RazorPages i stare WebForme* su *Page Controller pattern*
- MVC kao koncept nastao u kasnim sedamdesetim godinama prošlog stoljeća (Smalltalk projekt unutar Xeroxa)
- Podjela u model, pogled i upravljač omogućuje lakše testiranje
  - ASP.NET MVC objedinjuje iskustva MVC implementacija u drugim jezicima
    - intenzivno korištenje tehnike *Dependency Injection*
    - bogata podrška za interpretiranje/usmjeravanje zahtjeva
    - *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)

\*Par crtica o povijesnom kontekstu u dodatnim slajdovima 6.1

# MVC

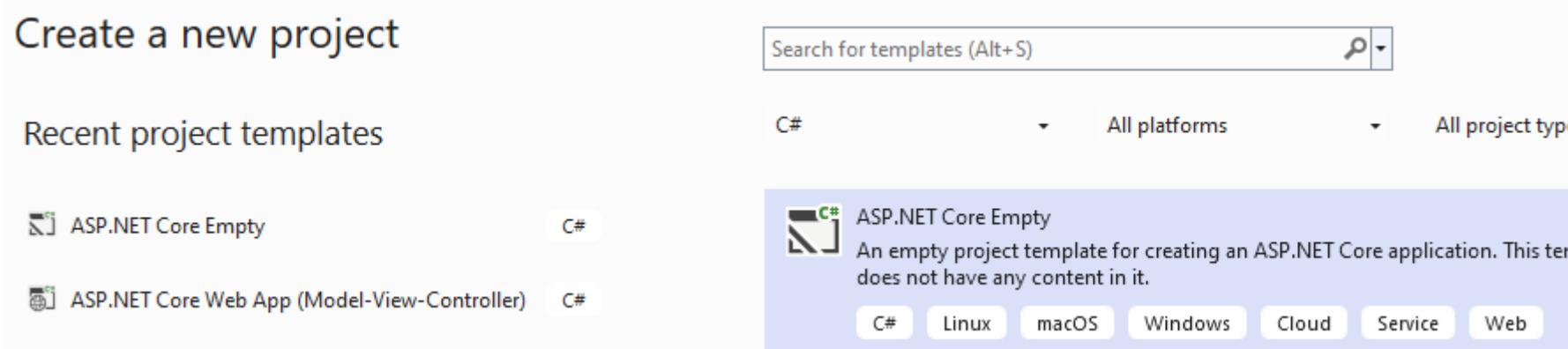
- Model - Pogled - Upravljač
- Detaljnija razrada **prezentacijskog sloja**
  - Pogled definira izgled korisničkog sučelja
  - Obično vezan za objekt iz modela
- **Upravljač predstavlja prezentacijsku logiku**
  - Prima ulaz iz pogleda, obrađuje ga, puni i dohvaća model, poziva postupke iz nižih slojeva 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)

# 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
  - Pokreće web-server i postavlja prepostavljene postavke aplikacije
    - *appsettings.json, appsettings.{aktivno okruzenje}.json, secrets.json* (ako je postavljen *SecretsId* i ako je okruženje *Development*), konfiguracija praćenja traga, ...)
    - Web-aplikacija se izvršava na privremenom web-serveru
      - Može se spojiti s klasičnim web serverom (IIS, Apache, Nginx, ...)
  - Primjer:  BiloKojaPraznaAplikacija \ Program.cs
    - ispisuje *HelloWorld* za svaki zahtjev

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/", () => "Hello World!");
app.Run();
```

- *builder* i *app* nude metode za daljnje konfiguriranje (putanje, lanci ovisnosti, preslikavanja konfiguracijskih objekata, ...)

# Uključivanje podrške za MVC

- Ukloniti odsječak za ispis *Hello World* u svakom zahtjevu
- U popis korištenih servisa dodati podršku za MVC i aktivirati usmjeravanje za MVC
  - Više o rutama i usmjeravanjima naknadno
- Primjer: MVC \ Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

var app = builder.Build();

app.MapGet("/", () => "Hello World!");

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

app.Run();
```

# Način rada MVC aplikacije

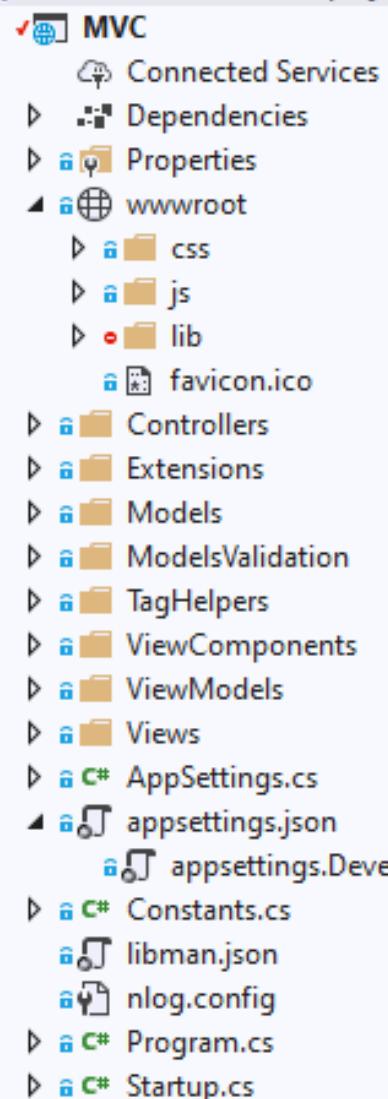
- Iz adrese zahtjeva i postavki usmjeravanja bit će odabran upravljač (razred izveden iz razreda *Controller*) na kojem će se izvesti određena akcija (postupak u odabranom razredu)
  - pozvani upravljač (najčešće) popunjava model i aktivira generiranje HTML-a temeljem pogleda u kojem su umetnute vrijednosti iz modela
- Kako prepoznati koju akciju (i na kojem upravljaču) izvršiti?
  - Inicialno usmjeravanje definira rutu oblika

```
{controller=Home} / {action=Index} / {id?}
```

tj. pretpostavlja da je adresa oblika *controller/action/id* pri čemu su prepostavljene vrijednosti:
    - upravljač: *Home*
    - akcija: *Index*
    - parametar id je opcionalan
  - Napomena: U primjeru s mjestima i dokumentima bit će prikazan način kako definirati drugačije usmjeravanje

# Organizacija mapa unutar projekta

- Uobičajena konvencija
  - Upravljači unutar mape *Controllers* u razredima sa sufiksom *Controller*
    - Primjer:  MVC \ Controllers \ HomeController.cs
  - Pogledi unutar mape *Views* podijeljeni u podmape po nazivima upravljača
    - Naziv datoteke obično odgovara nazivu akcije (postupka u upravljaču)
    - Primjer:  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 (potrebno dodatno podešavanje)
  - Praktično kod velikih aplikacija



# Struktura primjera iz predavanja

- 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 prihvatanje 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 mesta, dizajniraju se zasebno te se uključuju u pojedinim pogledima
  - Konfiguracijske i projektne datoteke
  - Ostali razredi i mape po potrebi

# 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:  `MVC \ Program.cs`

...

```
app.UseStaticFiles();
```

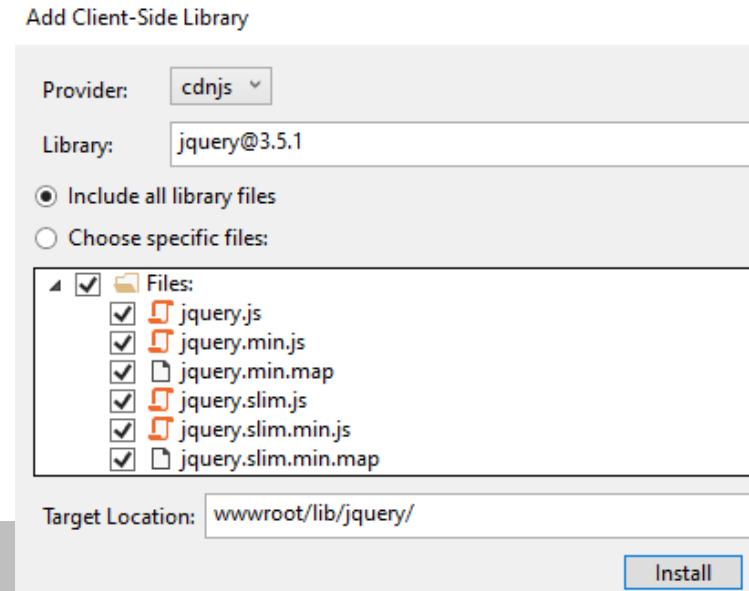
...

- Paziti na redoslijed uključivanja

- <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0#middleware-order>

# Paketi klijentskih biblioteka

- Bootstrap, jQuery, ...
- Način uključivanja
  - Samostalno skinuti distribuciju i staviti na odgovarajuće mjesto
    - uobičajeno wwwroot\lib
  - Uključiti direktno preko CDN-a (**najjednostavnije i preporuča se za zadaće**)
    - npr. <https://code.jquery.com/jquery-3.5.1.js>
    - 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



# 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
  - Primjer:  `Web \ .gitignore`

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

- LibMan automatski obnavlja sadržaj kod svakog korisnika te nakon promjene datoteke `libman.json`
  - Odredište ovisi o postavkama
  - Za automatsko obnavljanje prilikom izgradnje projekta, u projekt dodati NuGet paket *Microsoft.Web.LibraryManager.Build*

# LibMan i paketi klijentskih biblioteka (2)

```
{  
  "version": "1.0", "defaultProvider": "cdnjs",  
  "libraries": [  
    {  
      "library": "bootstrap@5.2.2",  
      "destination": "wwwroot/lib/bootstrap/"  
    }, {  
      "library": "jquery@3.5.1",  
      "destination": "wwwroot/lib/jquery/"  
    }, {  
      "library": "font-awesome@5.15.1",  
      "destination": "wwwroot/lib/font-awesome/"  
    }, {  
      "library": "jquery-validation-unobtrusive@3.2.11",  
      "destination": "wwwroot/lib/jquery-validation-unobtrusive/"  
    }, {  
      "library": "jquery-validate@1.19.2",  
      "destination": "wwwroot/lib/jquery-validate/"  
    }  
  ]  
}
```

- Primjer:  MVC \ libman.json

...

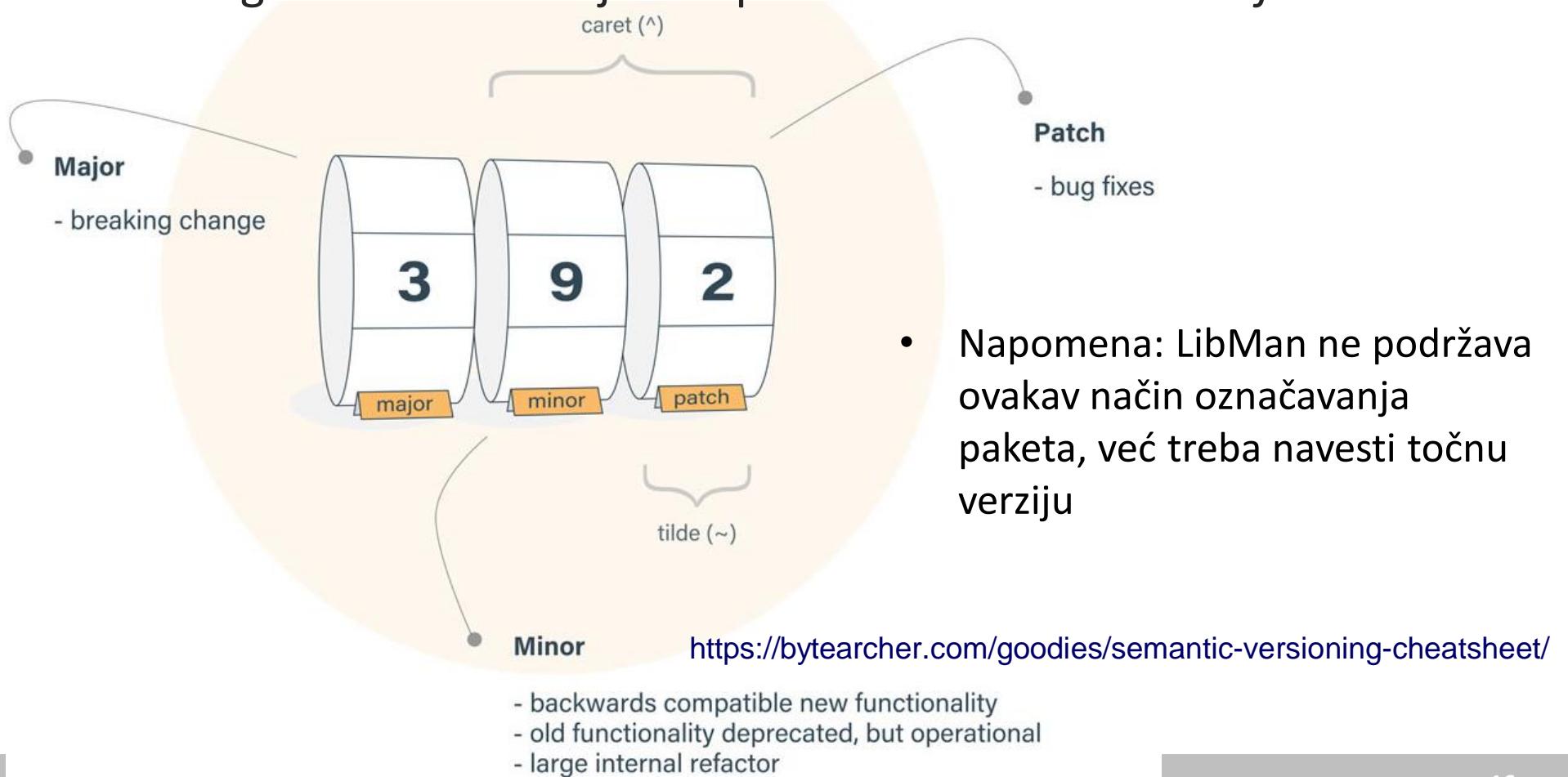
# Primjer konfiguracijske datoteke za neke druge upravljače paketima

- U slučaju da su se koristili *bower*, *yarn* ili *npm* tada bi konfiguracijske datoteke bile *bower.json* ili *package.json* i sadržaj bi mogao biti nalik sljedećem:

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

# Značenja znakova ^ i ~ u konfiguracijskoj datoteci za klijentske biblioteke

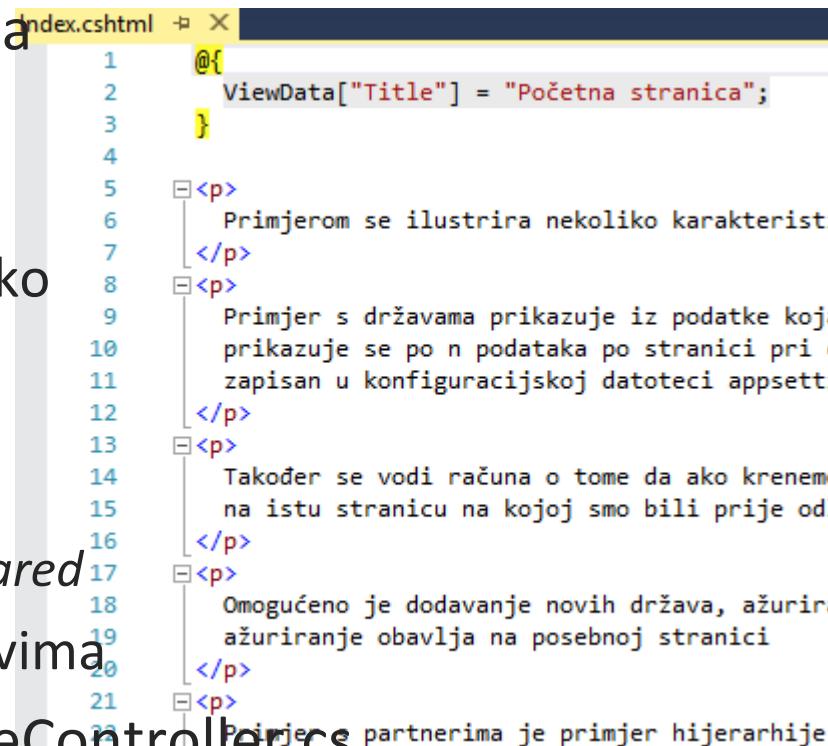
- ^ omogućava instaliranje novije verzije od navedene, sve dok je *major version* isti
- ~ omogućava instaliranje zakrpa navedene *minor verzije*



# 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
- Primjer:  MVC \ Controllers \ HomeController.cs

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



```
Index.cshtml  
1 @{  
2     ViewData["Title"] = "Početna stranica";  
3 }  
4  
5 <p>  
6     Primjerom se ilustrira nekoliko karakterističnih  
7     osobina pogleda.  
8 <p>  
9     Primjer s državama prikazuje iz podatke kojih  
10    prikazuje se po n podataka po stranici pri  
11    zapisan u konfiguracijskoj datoteci appsettings.json.  
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     posljednjeg ažuriranja, pogled će biti ponovo  
17 </p>  
18 <p>  
19     Omogućeno je dodavanje novih država, ažuriranje  
20     ažuriranje obavlja na posebnoj stranici  
21 </p>
```

# 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
UnauthorizedResult	Vraća HTTP kod 401	-
NotFoundResult	Vraća HTTP kod 404	NotFound
StatusCodeResult	Vraća određeni HTTP kod	-
EmptyResult	Bez povratne vrijednosti	-

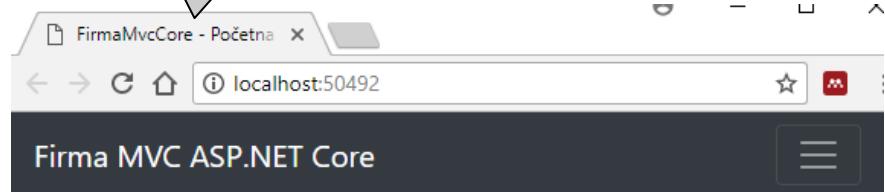
# Sintaksa pogleda

- 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-a i koda koji se izvršava na serveru
  - 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 zaglavje niti koristi glavnu stranicu

```
Index.cshtml + X
1  @{
2      ViewData["Title"] = "Početna stranica";
3  }
4
5  <p>
6      Primjerom se ilustrira nekoliko karakterističnih primjera korištenja MVC-a i ASP.NET Corea.
7  </p>
8  <p>
9      Primjer s državama prikazuje iz podatke koji su prikazuju se po n podataka po stranici pri čemu je podatak o broju elemenata po stranici zapisan u konfiguracijskoj datoteci appsettings.json (novitet iz .NET Corea).
10     Također se vodi računa o tome da ako krenemo u ažuriranje obavlja na posebnoj stranici.
11
12     Omogućeno je dodavanje novih država, ažuriranje i brisanje obavlja na posebnoj stranici.
13
14     Primjer s partnerima je primjer hijerarhije.
15
16
17
18
19
20
21
22
```

- 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



Primjerom se ilustrira nekoliko karakterističnih primjera korištenja MVC-a i ASP.NET Corea.

Primjer s državama prikazuje podatke iz tablice koja nema stranih ključeva. Prilikom pregleda prikazuje se po n podataka po stranici pri čemu je podatak o broju elemenata po stranici zapisan u konfiguracijskoj datoteci appsettings.json (novitet iz .NET Corea).

Također se vodi računa o tome da ako krenemo u ažuriranje neke države da se nakon ažuriranja vratimo na istu stranicu na kojoj smo bili prije odlaska na ažuriranje.

Omogućeno je dodavanje novih država, ažuriranje i brisanje postojećih, pri čemu se ažuriranje obavlja na posebnoj stranici.

Primjer s partnerima je primjer hijerarhije klasa Partner, Tvrtka i Osoba, što je u bazi podataka modelirano u obliku TPT (Table per type). Spojeni podaci dohvaćaju se preko pogleda koji je naknadno ručno dodan u data model.

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

# Glavna stranica

- Prepostavljena glavna stranica za svaki pogled definirana u datoteci Views \ \_ViewStart.cshtml
  - Primjer:  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 prepostavljena 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:  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>
```

# Vlastite komponente (1)

- Za sadržaje koji se pojavljuju na više mesta, primjerice izbornik
- Komponente se mogu stvoriti na više načina, npr. izvođenjem iz apstraktnog razreda *ViewComponent* i implementacijom postupka *Invoke*
  - Rezultat tipa *IViewComponentResult*
  - Odgovarajući „pogled” po sintaksi identičan ostalima
- Primjer:  MVC \ ViewComponents \ NavigationViewComponent.cs
  - Poveznica na trenutni upravljač će biti drugačije označena, pa je u postupku očitana aktualna vrijednost ako postoji
    - `referencia?.član` vraća član objekta na kojeg referencia pokazuje ili null u slučaju da je referencia null
      - `referencia.clan` bi mogao izvazvati `NullReferenceException`
    - O ViewBag-u naknadno

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

# Vlastite komponente (2)

- Primjer:  MVC\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 \*, MVC*

# Preslikavanje konfiguracijske datoteke (1)

- Inicijalna konfiguracija uključuje datoteku *appSettings.json* i podvarijante ovisno o varijabli okruženja te *secrets.json*
- Definira se vlastiti razred koji svojom strukturom prati JSON sadržaj ili neki njegov dio iz konfiguracijske datoteke
- Primjer:  MVC \ AppSettings.cs

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

- Primjer:  MVC \ appsettings.json

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

# Preslikavanje konfiguracijske datoteke (2)

- U početnim postavkama uspostavlja se preslikavanje između dijela konfiguracijske datoteke i vlastitog razreda
  - Primjer:  MVC \ Program.cs

```
var appSection = builder.Configuration.GetSection("AppSettings");
builder.Services.Configure<AppSettings>(appSection);
```

- Omogućava da se u konstruktor pojedinog upravljača umetne *IOptions<AppSettings>*, odnosno *IOptionsSnapshot<AppSettings>*
- Ako to nije dovoljno, argument konstruktora može biti *IConfiguration*

# Podsjetnik - Stvaranje 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
```

1. 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 Models -t Artikl -t Dokument -t Drzava -t Mjesto -t Osoba -
```

```
t Partner -t Stavka -t Tvrтka
```

# Postavke za spajanje na bazu podataka

- Automatski generirani razred za EF kontekst sadrži tvrdo kodirane postavke za spajanje na bazu podataka u postupku *OnConfiguring*
  - Obrisati navedeni postupak i konstruktor bez argumenata
    - ostaviti samo konstruktor s parametrom tipa *DbContextOptions*
    - Konkretni objekt bit će umetnut tehnikom *Dependency Injection*
    - Primjer:  MVC \ Models \ Firma.Context.cs

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

```
public FirmaContext() {}
```

- Primjer:  MVC \ Program.cs

```
builder.Services.AddDbContext<Models.FirmaContext>(
    options => options.UseSqlServer(
        builder.Configuration.GetConnectionString("Firma")));
```

# Akcija dohvata svih država (1)

- Primjer:  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 Program.cs, koristeći tehniku Dependency 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:  MVC \ Controllers \ DrzavaController.cs
  - Upravljač dohvati podatke, napuni model (lista država) i izazove prikaz pogleda s imenom *IndexSimple*
    - Bit će zamijenjeno kasnije s drugom verzijom koja koristi pogled u datoteci *Index.cshtml*

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

# Pregled svih država

- Primjer:  MVC \ Views \ Drzava \ IndexSimple.cshtml
  - Tip modela je *IEnumerable<Drzava>* (može i *List<Drzava>*, ali nepotrebno)
  - 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

```
@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 Libman i ostale paketne alatne
  - aktivira se kodom u jQueryu, nakon što se dokument učita
  - Primjer:  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
    - Prvo se moraju dohvatići svi podaci, pa se prikazuju samo određeni

The screenshot shows a web application interface for managing countries. At the top, there's a navigation bar with links: Firma MVC ASP.NET Core, Početna stranica, Države, Mjesta, Artikli, Partneri, Dokumenti, Izveštaji, Pregled log datoteka, and WebApi dokument. Below the navigation bar, the main title is "Popis država". On the left, there are filters: "Prikaži" dropdown set to 10, "zapisa" dropdown, and a search input field labeled "Pretraga". The main content area is a table listing countries with columns: Oznaka države, Naziv države, Iso3 oznaka, and Šifra države. The table contains 10 rows of data. At the bottom, there's a pagination control with links: Prethodna, 1, ..., 4, 5 (which is highlighted in blue), 6, ..., 23, Sljedeća, and a note indicating 41 - 50 od ukupno 222 zapisa.

Oznaka države	Naziv države	Iso3 oznaka	Šifra države
HR	Croatia	HRV	191
CU	Cuba	CUB	192
CY	Cyprus	CYP	196
CZ	Czech Republic	CZE	203
DK	Denmark	DNK	208
DJ	Djibouti	DJI	262
DM	Dominica	DMA	212
DO	Dominican Republic	DOM	214
EC	Ecuador	ECU	218
EG	Egypt	EGY	818

# Straničenje na serverskoj strani

- Napisati serverske postupke za *DataTables* ili vlastita izvedba?
  - U ovom primjeru vlastita izvedba, kao motivacija za neke druge principe
- Primjer:  MVC \ ViewModels \ PagingInfo.cs
  - 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; }  
}
```

# URL i model za prikaz država na određenoj stranici

- Adresa: /Drzava/Index
  - Upravljač: Drzava(Controller), Akcija (metoda): 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
    - Nazivi parametara odgovarajuću parametrima metode *Index*
- Primjer:  MVC \ ViewModels \ DrzaveViewModel.cs
  - Model za prikaz država sadrži popis država i podatke o trenutnoj stranici, ukupnom broju stranica i redoslijedu sortiranja (*PagingInfo*)

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

# Parametri sortiranja i straničenja

- Primjer:  MVC \ Views \ Drzava \ Index.cshtml
  - Omogućeno sortiranje i straničenje
    - Zaglavljе 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 MVC.ViewModels.DrzaveViewModel
...
<table class="table table-striped">
...
<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> ...

```

# Akcija dohvata podskupa država (1)

- Primjer:  MVC \ Controllers \ DrzavaController.cs
  - Upravljač provjerava postoji li barem jedna država u BP. Ako ne, 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] = "Tablica Država je prazna.";  
            TempData[Constants.ErrorOccurred] = false;  
            return RedirectToAction(nameof(Create)); ...  
        }  
    }  
}
```

# Akcija dohvata podskupa država (2)

- Primjer:  MVC \ Controllers \ DrzavaController.cs
  - Formira se podatak o stranicama i sortiranju te provjerava broj stranice (ako je van raspona, preusmjerava se na zadnju stranicu)
    - Drugi parametar je anonimni objekt za parametre usmjerenja

```
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 < 1 || page > pagingInfo.TotalPages) {  
            return RedirectToAction(nameof(Index),  
                new { page = pagingInfo.TotalPages, sort, ascending});  
        }  
        ...  
    }  
}
```

# Akcija dohvata podskupa država (3)

- Primjer:  MVC \ Controllers \ DrzavaController.cs
  - Upit na tablicu s državama će se proširiti tako da uzme u obzir redoslijed sortiranja
  - Vlastita metoda proširenje (engl. extension)
    - Implementacija skicirana na sljedećem slajdu

```
public class DrzavaController : Controller {  
    ...  
    public IActionResult Index(int page = 1, int sort = 1,  
                            bool ascending = true) {  
        var query = ctx.Drzava.AsNoTracking();  
        int count = query.Count();  
        ...  
        query = query.ApplySort(sort, ascending);  
        ...  
    }  
}
```

# Akcija dohvata podskupa država (4)

- Primjer:  MVC \ Extensions \ Selectors \ DrzavaSort.cs

```
public static IQueryable<Drzava> ApplySort(
    this IQueryable<Drzava> query, int sort, bool ascending) {
    Expression<Func<Drzava, object>> orderSelector = sort switch
    {
        1 => d => d.OznDrzave,
        2 => d => d.NazDrzave,
        3 => d => d.Iso3drzave,
        4 => d => d.SifDrzave,
        _ => null
    };
    if (orderSelector != null)
        query = ascending ? query.OrderBy(orderSelector) :
            query.OrderByDescending(orderSelector);
    return query;
}
```

# Akcija dohvata podskupa država (5)

- Primjer:  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;
        ... query = query.ApplySort(sort, ascending); ...
        var drzave = query
            .Skip((page - 1) * pagesize)
            .Take(pagesize)
            .ToList();
        var model = new DrzaveViewModel {
            Drzave = drzave, PagingInfo = pagingInfo
        };
        return View(model);
    }
}
```

# Dodavanje novog podatka

# Postupci za dodavanje novog podatka

- Primjer:  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, prikaz pogleda
  - prihvati popunjениh podataka (POST zahtjev)
  - određivanje akcije na osnovi atributa *HttpGet* i *HttpPost*
    - Primjer  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:  MVC \ Views \ Drzava \ Create.cshtml

```
@model Drzava      //dodati @using 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" ... >
```

# Tekst pored kontrola za unos novog podatka

- Za svako svojstvo ispisuje se prikladno ime svojstva
  - Primjer:  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:  MVC \ Models \ Drzava.cs

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

# Prihvati podataka

- Parametri u metodi (akciji) upravljača navedeni pojedinačno ili u sklopu nekog razreda.
- Povezivanje, tj. pridjeljivanje vrijednosti se vrši na osnovu svojstva *name* pojedine html kontrole
- **Redoslijed povezivanja (prvi koji bude pronađen):**
  1. `Request.Form` – polja iz forme
  2. `RouteData.Values` – podaci usmjeravanja
  3. `Request.QueryString` – parametri iz *query stringa*
  4. `Request.Files` – nazivi parametara koji služe za slanje datoteke
- 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>

# Kontrole za unos novog podatka

- Atributom *ValidateAntiForgeryToken* provjera se je li zahtjev stigao sa stranice za unos novog podatka ili netko pokušava direktno izvršiti unos s nekog drugog mesta
  - 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:  MVC \ Controllers \ DrzavaController.cs

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

# 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 s dotad upisanim podacima
    - *ModelState* sadrži podatke o pogreškama modela
- Primjer:  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) su zamotane u neke druge iznimke.
  - Potrebno dohvatiti unutarnju iznimku, pa tako rekurzivno dalje
  - Pomoćni postupak napisan u obliku ekstenzije
  - Primjer:  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();  
    }  
}
```

# Prijenos podataka između zahtjeva (1)

- Pogled je vezan za model, a dodatni podaci se mogu prenijeti koristeći *ViewBag* ili *ViewState* (primjeri naknadno)
  - *return View(model)* uzrokuje izvršavanje serverskog koda iz pogleda koji se kombinira sa HTML-om i uklapa u glavnu stranicu
- 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 stavljuju se u *TempData*
  - Podaci iz *TempData* se brišu nakon konzumiranja
    - Interno se koristi sjednica (session), tj. podaci unutar sjednice

# Prijenos podataka između zahtjeva (2)

- U glavnoj stranici uključen parcijalni pogled koji će prikazati odgovarajuće podatke iz *TempData* (ako postoje)

- Primjer:  MVC \ Views \ Shared \ \_Layout.cshtml

```
<partial name="HandleMessage" />
```

- Primjer:  MVC \ Views \ Shared \ HandleMessage.cshtml

```
<script>
@if ( TempData[Constants.Message] != null) {
    bool error = false;
    var obj = TempData[Constants.ErrorOccurred];
    if (obj != null) error = (bool)obj;
    if (error) {
        <text>toastr.error(`@TempData[Constants.Message]` ,
                           'Pogreška'...</text>
    } else {
        <text>toastr.success(`@TempData[Constants.Message]` )</text>
    }
}</script>
```

# Validacija podataka

# 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
- U slučaju razreda Drzava iz primjera, to bi nalikovalo sljedećem

```
public partial class Drzava {  
    [Required(ErrorMessage="Oznaka države je obvezno polje")]  
    public string OznDrzave { get; set; }  
    [MaxLength(3, ErrorMessage="ISO3 ne smije biti dulja od 3 slova")]  
    public string Iso3drzave { get; set; }  
    ...  
}
```

# Validacija korištenjem paketa *Fluent Validation*

- NuGet paketi *FluentValidation* i *FluentValidation.AspNetCore* omogućavaju pisanje validacija u odvojenim datotekama
  - Razredi izvedeni iz *AbstractValidator*, parametrizirani po konkretnom razredu čije podatke treba provjeriti
  - Primjer:  MVC \ ModelsValidation \ DrzavaValidator.cs

```
public class DrzavaValidator : AbstractValidator<Drzava> {  
    public DrzavaValidator() {  
        RuleFor(d => d.OznDrzave)  
            .NotEmpty().WithMessage("Oznaka države je obvezno polje")  
            .MaximumLength(3).WithMessage("Oznaka države ...");  
  
        RuleFor(d => d.NazDrzave)  
            .NotEmpty().WithMessage("Naziv države je obvezno polje");  
  
        ...  
    }  
}
```

# Uključivanje validacije napisane korištenjem paketa Fluent Validation

- Pojedini validacijski razred se može uključiti sa *services.AddTransient* ili *service.AddScoped* ili se mogu automatski pronaći i uključiti svi razredi koji nasljeđuju *AbstractValidator* iz nekog projekta.
  - Primjer:  MVC \ Program.cs

```
using FluentValidation.AspNetCore;  
...  
builder.Services  
    .AddFluentValidationAutoValidation()  
    .AddFluentValidationClientsideAdapters()  
    .AddValidatorsFromAssemblyContaining<DrzavaValidator>();
```

# 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:  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"/></div>  
        <input asp-for="OznDrzave" class="form-control" />  
        ...  
        <span asp-validation-for="NazDrzave" ...>
```

# 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
- Primjer:  MVC \ wwwroot \ css \ site.css
  - neispravna polja budu obrubljena crvenom bojom, a tekst sumarnih pogrešaka prikazan nijansom crvene i podebljano

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

# Validacija na klijentskoj strani

- Dio pogrešaka moguće je 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:  `MVC \ Views \ Shared \ IncludeValidation.cshtml`

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

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

- Potonji parcijalni pogled uključen na mjestima gdje je to potrebno, npr. u `Create.cshtml` i `Edit.cshtml`

# 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:  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 pomoći tag-helpera koristi se ModelState, a ne Model*

# Brisanje podatka

# Forma za brisanje podatka

- Za brisanje treba koristiti POST postupak
  - GET postupak se preporuča za postupke koji ne mijenjaju stanja (objekta)
- 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:  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 ...">
```

# Akcija brisanja podatka

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

[HttpPost]

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

# Potvrda brisanja podatka

- Za svaki klik na kontrolu sa stilom *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:  MVC \ wwwroot \ js \ site.js

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

- Vlastita skripta uključena pri dnu glavne stranice
  - Primjer:  MVC \ Views \ Shared \ \_Layout.cshtml

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

# Ažuriranje jednostavnog podatka

# Poveznica za ažuriranje države

- Primjer:  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 *OznDrzave* 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>...
```

# Akcije ažuriranja podataka

- Primjer:  MVC \ Controllers \ DrzavaController.cs
  - Dva postupka Edit i Update koji se odazivaju na istu akciju
  - *Edit* predstavlja inicialno otvaranje forme (GET zahtjev),
  - *Update* naknadno slanja popunjениh podataka (POST zahtjev)
    - Postupci 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) {  
        ...  
    }  
}
```

# 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:  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:  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);  
        }  
    }  
}
```

# Prihvati vrijednosti za ažuriranje

- Nekoliko tehnika ažuriranja podatka
  - prihvati kompletog podatka pa kopčanje u kontekst i snimanje
  - odabir svojstava koje će se povezati
  - dohvati 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 koristiti će se varijanta dohvata podatka iz baze podataka i ažuriranje samo određenih svojstava
  - Koristi se asinkroni postupak *TryUpdateModelAsync*
  - Preopterećeni postupak
  - Koristiti će se ona varijanta u kojoj se navode svojstva koja se žele izmijeniti temeljem podataka pristiglih s forme
    - Preciznije, umjesto navođenja naziva svojstava navoditi 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 dohvata vrijednosti po završetku vrši se naredbom *await*
  - Kod 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.*

# Prihvati vrijednosti za ažuriranje

- Primjer:  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();
        }
    }
}
```

# Proširenje primjera na popis mjesta

- Rad s mjestima izведен na sličan način kao i rad s državama
- Nekoliko ključnih razlika
  - Koriste se asinkrone metode za rad s bazom podataka
  - Prilikom unosa mjesta potrebno iz padajuće liste odabrati državu umjesto direktnog unosa vrijednosti stranog ključa
  - Kao model za prikaz pojedinog mjesta koristi se vlastiti razred (prezentacijski model) koji uključuje naziv države
  - Ažuriranje i brisanje izvedeno dinamički (detaljnije u slajdovima 6-B)
- Primjer:
  -  MVC \ Controllers \ MjestoController.cs
  -  MVC \ Views \ Mjesto \ \*.cshtml
  -  MVC \ ViewModels \ Mjest\*ViewModel.cs

# Model za pregled svih mesta

- Umjesto entiteta Mjesto za pojedinačno mjesto koristi se novi prezentacijski pogled
  - Umjesto oznake države sadrži naziv države
- Primjer:  MVC \ ViewModels \ MjestaViewModel.cs

```
public class MjestaViewModel {  
    public IEnumerable<MjestoViewModel> Mjesta { get; set; }  
    public PagingInfo PagingInfo { get; set; }  
}
```

- Primjer:  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; }  
}
```

# 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.

## Unos novog mesta

Poštanski broj mesta

Naziv mesta

Poštanski naziv mesta

Država

Odaberite državu

- Odaberite državu
- Croatia
- Andora
- Angola
- Argentina
- Armenia
- Bahamas
- Bangladesh
- Barbados
- Belarus
- Belgium
- Belize

# Priprema podataka za padajuću listu

- Potrebno stvoriti objekt tipa *SelectList*
  - podaci + svojstvo koje predstavlja vrijednost odabranog elementa + svojstvo koje se prikazuje kao tekst u padajućoj listi
  - Stvoreni objekt se pogledu prenosi koristeći *ViewBag* (ili *ViewData*)
  - Primjer:  MVC \ Controllers \ MjestoController.cs

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

# Prikaz padajuće liste u pogledu

- Padajuća lista se koristi unutar HTML oznake *select*, pri čemu se izvor navodi atributom *asp-items*
  - *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 inicialno odabrana
    - Nakon što se jednom promijeniti vrijednost (zbog atributa *disabled*) više se neće moći ponovo odabrati element s porukom
- Primjer:  MVC \ Views \ Mjesto \ Create.cshtml

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

# 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:  MVC \ Controllers \ MjestoController.cs

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

# Dodatne postavke usmjerenja

- Moguće definirati i drugačija usmjerenja (oprezno!)
  - Primjer:  MVC \ Program.cs

```
app.UseEndpoints(endpoints => {
    endpoints.MapControllerRoute("Mjesta i artikli",
        "{action}/{controller:regex(^Mjesto|Artikl)$})" +
        "/Page{page}/Sort{sort:int}/ASC-{ascending:bool}/{id?}",
        new { action = "Index" })
    );
    endpoints.MapDefaultControllerRoute();
});
```

- U tom slučaju ažuriranje mjesta može imati adresu  
<https://.../Edit/Mjesto/Page4/Sort1/ASC-True/7518>,  
a ažuriranje države  
<https://.../Drzava/Edit/CO?page=4&sort=1&ascending=True>

# Razvoj primijenjene programske potpore

---

## 6-A Kreiranje vlastitog tag-helpera

Dodatni materijali

# Napomena vezana za primjer koji slijedi

- Primjeri u ovim predavanjima predstavljaju složene koncepte koji uključuju korištenje CSS stilova i jQuerya
- 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

# 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*



# Podsjetnik na (neke) ugrađene *tag-helpere*

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

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

# 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:  MVC \ TagHelpers \ PagerTagHelper.cs

```
[HtmlTargetElement(Attributes = "page-info")]
public class PagerTagHelper : TagHelper {
    private readonly IUrlHelperFactory urlHelperFactory;
    private readonly AppSettings appData;
    public PagerTagHelper(
        IUrlHelperFactory helperFactory,
        IOptionsSnapshot<AppSettings> options) {
        urlHelperFactory = helperFactory;
        appData = options.Value;
    }
    ...
```

# Uključivanje vlastitog *tag-helpera*

- Može se uključiti u pojedinom pogledu ili za sve poglede navođenjem u `_ViewImports.cshtml`
- Primjer:  `MVC \ Views \ _ViewImports.cshtml`

```
@using MVC
@using MVC.Models
@using MVC.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper MVC.TagHelpers.* , MVC
```

# Svojstva željenog *tag-helpera* (1)

- *Tag-helper* će se primjenjivati na HTML oznaku div i imat će sljedeće vlastite atributе
  - 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:  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:  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; }
```

# 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 *nav*
- 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 iz trenutnog konteksta i objekta tipa *IUrlHelperFactory*
  - Primjer:  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  $\max\{1, \text{trenutna} - n\}$  do  $\min\{\text{ukupno}, \text{trenutna} + n\}$ 
  - 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:  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:  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, ... });
...
...
```

# 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:  MVC \ TagHelpers \ PagerTagHelper.cs

```
$(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:  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'));  
});
```

# 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:  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;  
    }  
}
```

# Razvoj primijenjene programske potpore

---

## 6-B      **ASP.NET Core (Dodatni materijali)**

Parcijalni pogledi i dinamičko ažuriranje

# Ideja primjera

- Prilikom brisanja mjesta umjesto osvježavanja cijele stranice i ponovnog tabličnog prikaza samo ukloniti redak obrisanog mjesta
- Omogućiti ažuriranje podataka o mjestima unutar tabličnog prikaza (bez odlaska na posebnu stranicu)

Poštanski broj	Naziv mjesta	Poštanski naziv mjesta	Država		
53234	Breštane	Udbina	Croatia		
33507	Breštanovci	Crnac	Croatia		
43252	Breza	Prgomelje	Croatia		
51213	Breza	Jurdani	Croatia		

# Akcija brisanja mjesta

- Postupak vraća Json anonimnog razreda s 2 svojstva
  - Primjer:  MVC \ Controllers \ MjestoController.cs
  - Ako mjesto ne postoji vraća se *NotFound*

```
[HttpPost][ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteAjax (int id) {
    var mjesto = await ctx.Mjesto.FindAsync(id);
    if (mjesto != null) {
        try {
            string naziv = mjesto.NazMjesta;
            ctx.Remove(mjesto);
            await ctx.SaveChangesAsync();
            var result = new {
                message = $"Mjesto {naziv} sa šifrom {id} obrisano.",
                successful = true
            };
            return Json(result);
        }
    }
}
```

# Izvedba dinamičkog brisanja na klijentu (1)

- Na postojeći gumb za brisanje dodamo neki vlastiti atribut, npr. `data-delete-ajax` s adresom akcije koja treba obrisati element i vratiti Json.
  - Primjer:  MVC \ Views \ Mjesto \ Index.cshtml

```
<form asp-action="Delete" method="post" ...>
    <button class="btn btn-sm btn-danger delete"
            data-delete-ajax="@Url.Action("DeleteAjax",
            new { id = mjesto.IdMesta })"
            title="Obriši">
        ...
    </button>
</form>
```

- Ovakva izvedba osigurava postojanje *AntiForgeryToken* tokena zbog elementa *form* te osigurava da klasično brisanje radi ako dinamičko brisanje nije implementirano.

# Izvedba dinamičkog brisanja na klijentu (2)

- Postojeći JavaScript kod izmijenjen tako da provjeri je li za *delete* gumb definiran atribut s prethodnog slajda
  - Ako da, poziva se vlastita metoda deleteAjax
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
$function () {  
    $(document).on('click', '.delete', function (event) {  
        if (!confirm("Obrisati zapis?")) {  
            event.preventDefault();  
        }  
        else {  
            const url = $(this).data('delete-ajax');  
            if (url !== undefined && url !== '') {  
                event.preventDefault();  
                deleteAjax($(this), url);  
            }  
        } ...  
    })  
}
```

# Izvedba dinamičkog brisanja na klijentu (3)

- U slučaju uspješnog brisanja uklanja se redak u kojem se nalazio gumba za brisanje
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
function deleteAjax(btn, url) {  
    const tr = btn.parents("tr");//redak kojem zapis pripada  
    ...  
    const token =  
        $('input[name=__RequestVerificationToken"]').first().val();  
    $.post(url, { __RequestVerificationToken:token}, function (data) {  
        if (data.successful) { //data je JSON koji server vrati  
            tr.remove();  
        }  
        ... Ispiši poruku o uspješnom brisanjeu ...  
    }).fail(function (jqXHR) {  
        alert(jqXHR.status + " : " + jqXHR.responseText);  
    }...  
}
```

# Ideja ažuriranja unutar retka

- Klikom na gumb za ažuriranje pozvati akciju na serveru koja vraća parcijalni pogled
- Postojeći redak s podacima sakriti i prikazati HTML iz parcijalnog pogleda
  - Ako korisnik odustane od ažuriranja, odbaciti redak za ažuriranje i otkriti skriveni redak
- Prilikom snimanja podataka pokupiti podatke iz retka, redak za ažuriranje obrisati i pozvati odgovarajuću akciju na serveru
  - Akcija na serveru vraća
    - Status 200 i parcijalni pogled koji sadrži poslane podatke i validacijsku pogrešku
    - Status 204 (No Content) kojim se označava da je spremanje uspješno
      - Skriveni redak ukloniti i ponovo učitati podatke za taj redak
    - Nešto treće – prikazati poruku o pogrešci

# Priprema gumba za ažuriranje

- Na postojeći gumb za ažuriranje dodana 2 atributa koja sadrže
  - adresu akcije koja vraća parcijalni pogled s formom za ažuriranje
  - adresu akcije koja vraća pojedinačni redak s podacima o mjestu
- Primjer:  MVC \ Views \ Mjesto \ Index.cshtml

```
<a asp-action="Edit" ...  
    data-get-ajax="@Url.Action("GetAjax",  
                                new { id = mjesto.IdMesta })"  
    data-edit-ajax="@Url.Action("EditAjax",  
                                new { id = mjesto.IdMesta })"  
    title="Ažuriraj"...
```

# Akcija za prikaz pojedinačnog mjesta

- Primjer:  MVC \ Controllers \ MjestoController.cs

```
[HttpGet]
```

```
public async Task<IActionResult> GetAjax(int id) {
    var mjesto = await ctx.Mjesto
        .Where(m => m.IdMjesta == id)
        .Select(m => new MjestoViewModel {
            IdMjesta = m.IdMjesta, NazivMjesta = m.NazMjesta,
            PostBrojMjesta = m.PostBrMjesta,
            PostNazivMjesta = m.PostNazMjesta,
            NazivDrzave = m.OznDrzaveNavigation.NazDrzave
        })
        .SingleOrDefaultAsync();
    if (mjesto != null)
        return PartialView(mjesto);
    else
        return NotFound($"Neispravan id mjesta: {id}");
}
```

# Parcijalni pogled za prikaz pojedinačnog mjesta

- Izgledom mora odgovarati retku iz Index.cshtml
  - Primjer:  MVC \ Views \ Mjesto \ GetAjax.cshtml

```
@model MjestoViewModel
<tr>
    <td class="text-left">@Model.PostBrojMjesta</td>
    ...
    <td>
        <a asp-action="Edit" asp-route-id="Model.IdMjesta"
            class="btn btn-sm edit"
            data-get-ajax="@Url.Action("GetAjax", new { id =
Model.IdMjesta })"
            data-edit-ajax="@Url.Action("EditAjax", new { id =
Model.IdMjesta })"
            title="Ažuriraj">...
    </td>
    <td>
        <form asp-action="Delete" method="post" ...>
```

# Akcija za prikaz retka za ažuriranje

- Izvedba slična kao kod „normalnog“ ažuriranja
  - pripremiti države (sortiranje i straničenje se ne koristi)
  - Primjer:  MVC \ Controllers \ MjestoController.cs

```
[HttpGet]
public async Task<IActionResult> EditAjax(int id) {
    var mjesto = await ctx.Mjesto
        .AsNoTracking()
        .Where(m => m.IdMjesta == id)
        .SingleOrDefaultAsync();

    if (mjesto != null) {
        await PrepareDropDownLists();
        return PartialView(mjesto);
    }
    else {
        return NotFound($"Neispravan id mjesta: {id}");
    }
}
```

# Parcijalni pogled za ažuriranje mjesta

- Svi elementi za unos označeni vlastitim atributom `data-save`, kao i gumbi za snimanje i odustajanje
  - Primjer:  MVC \ Views \ Mjesto \ EditAjax.cshtml

```
@model Mjesto
<tr> <td class="text-left">
    <input asp-for="NazMjesta" class="form-control" data-save="" />
</td>
...
<td class="text-left">
    <select class="form-control" asp-for="OznDrzave" asp-
items="ViewBag.Drzave" data-save=""> </select>
...
<input type="hidden" asp-for="IdMjesta" data-save="" />
    <button type="submit" class="btn btn-sm btn-primary save" ...>
        ...
    <button class="btn btn-sm cancel" ...>
        ...
    </div>
```

# Akcija za prihvat ažuriranih podataka

- Izvedba slična kao kod „normalnog“ ažuriranja
  - Primjer:  MVC \ Controllers \ MjestoController.cs

```
[HttpPost] [ValidateAntiForgeryToken]
public async Task<IActionResult> EditAjax(int id, Mjesto mjesto) {
    ... provjera ispravnosti ...
    try {
        ctx.Update(mjesto);
        await ctx.SaveChangesAsync();
        return NoContent();
    }
    catch (Exception exc)
    {
        ModelState.AddModelError(...);
        await PrepareDropDownLists();
        return PartialView(mjesto);
    }
}
```

# Izvedba dinamičkog ažuriranja na klijentu (1)

- Za svaki *edit* gumb provjeriti ima li definiran adresu za prikaz retka za ažuriranje
  - Ako da, poziva se vlastita metoda editAjax
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
$(document).on('click', '.edit', function (event) {  
    const editUrl = $(this).data('edit-ajax');  
    if (editUrl !== undefined && editUrl !== '') {  
        const getUrl = $(this).data('get-ajax');  
        event.preventDefault();  
        editAjax($(this), editUrl, getUrl);  
    }  
});
```

# Izvedba dinamičkog ažuriranja na klijentu (2)

- Vrši se dohvati retka za ažuriranje (sa zastavicom pazimo na 2 brza klik – detaljnije u izvornom kodu)
- Dohvaćeni HTML (tj. redak) dodajemo iza retka kojeg smo sakrili
- Definiramo ponašanje za *save* i *cancel* iz dohvaćenog retka
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
function editAjax(btn, editUrl, getUrl) {  
    const tr = btn.parents("tr");//redak kojem zapis pripada  
    $.get(editUrl, {}, function (data) {  
        tr.toggle(); //sakrij trenutni redak  
        var inserted = $(data).insertAfter(tr);  
        setCancelAndSaveBehaviour(tr, inserted, editUrl, getUrl);  
    })  
    .fail(function (jqXHR) {  
        alert(jqXHR.status + " : " + jqXHR.responseText);  
    });  ...  
}
```

# Izvedba dinamičkog ažuriranja na klijentu (3)

- U slučaju odustajanja „redak” za ažuriranje odbacujemo, a vraćamo skriveni redak
  - Preciznije, „redak” za ažuriranje je HTML koji sadrži 2 retka zbog validacije i argument je *insertedData*
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
function setCancelAndSaveBehaviour(hiddenRow, insertedData, editUrl,  
getUrl) {  
  
    //nađi cancel button  
    insertedData.find(".cancel").click(function (event) {  
        insertedData.remove(); //ukloni umetnuti redak  
        hiddenRow.toggle(); //vrati vidljivost originalnom retku  
        hiddenRow.data('active', false);  
    });  
    ...  
}
```

# Izvedba dinamičkog ažuriranja na klijentu (4)

- Prilikom snimanja prikopimo podatke s forme (vlastita metoda `collectData`) i napravimo post zahtjev
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
...
insertedData.find(".save").click(function (event) {
    event.preventDefault();
    var formData = collectData(insertedData);

    $.ajax({
        type: "POST",
        url: editUrl,
        contentType: false,
        processData: false,
        data: formData,
        success: ...
        error: function (jqXHR) {
            alert(jqXHR.status + " : " + jqXHR.responseText);
        }
    });
});
```

# Izvedba dinamičkog ažuriranja na klijentu (5)

- Uspješan status može biti posljedica validacijske pogreške i prikaza HTML-a s pogreškom (status 200) ili uspješno snimanje (status 204)
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
...
success: function (data, textStatus, jqXHR) {
    insertedData.remove();
    if (jqXHR.status == 204) { //podatak ažuriran, učitaj nove podatke
        $.get(getUrl, {}, function (refreshedRow) {
            $(hiddenRow).replaceWith(refreshedRow);
        })
    }
    else { //200
        var inserted = $(data).insertAfter(hiddenRow);
        setCancelAndSaveBehaviour(hiddenRow, inserted, editUrl, getUrl);
    }
}
```

# Izvedba dinamičkog ažuriranja na klijentu (6)

- Podatke s forme prikopimo tražeći sve elemente s atributom data-save, pazeći na interpretaciju (checkbox, file, ...)
  - Primjer:  MVC \ wwwroot \ js \ site.js

```
function collectData(container) {  
    var formData = new FormData();  
    container.find("[data-save]").each(function (index, element) {  
        if ($(element).is(':checkbox'))  
            formData.append($(element).attr('name'),  
                            $(element).is(':checked'));  
        ...  
        else {  
            var val = $.trim($(element).val());  
            if (val != '')  
                formData.append($(element).attr('name'), val);  
        }  
    });  
    ... dodaj i antiforgery token ...  
    return formData;}
```

# Razvoj primijenjene programske potpore

---

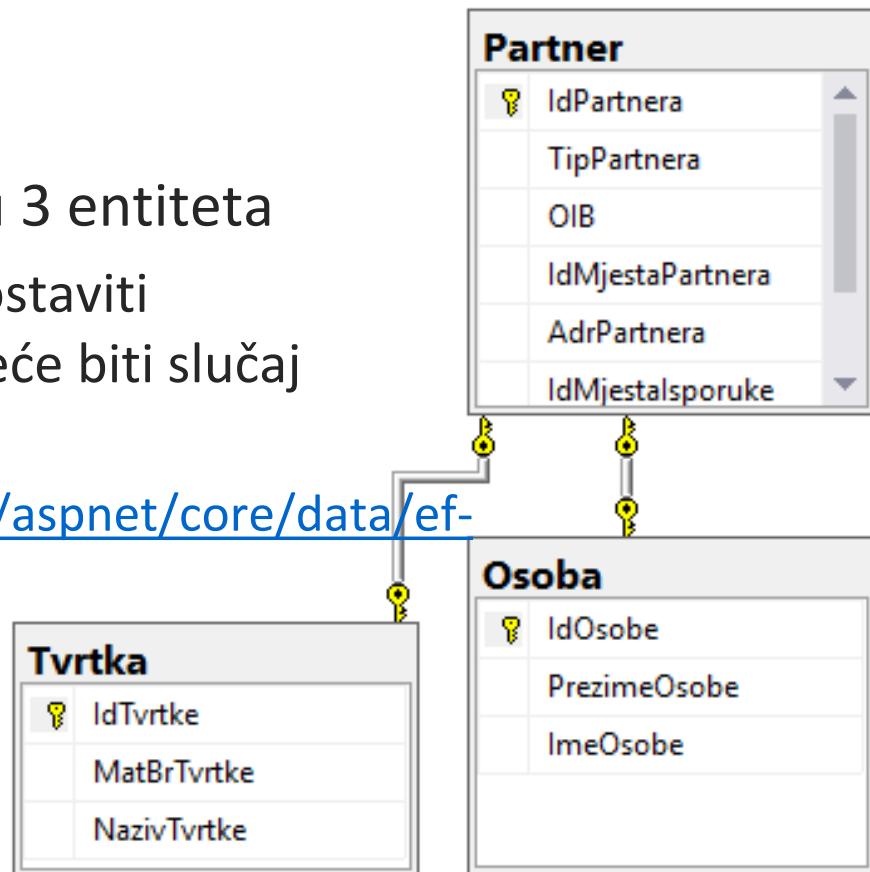
## 7. ASP.NET Core

Specijalizacija i generalizacija

Nadopunjavanje umjesto padajuće liste

# Pohrana generalizacija i specijalizacija

- U oglednom modelu Osoba i Tvrтka su specijalizacije Partnera na principu TPT (Table per type)
  - Alternative
    - TPH (Table per hierarchy)
    - TPC (Table per concrete class)
- EF Core navedene tablice preslika u 3 entiteta
  - U nekim slučajevima, moguće uspostaviti nasljeđivanje u EF modelu, ali to neće biti slučaj u ovom primjeru
    - <https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/inheritance>



# Problem prikaza specijalizacija nekog entiteta

- 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		

# 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:  MVC \ ModelsPartial \ ViewPartner.cs
    - Dodatno napisano svojstvo koje opisno prikazuje tip partnera
    - Smještan u posebnu mapu, ali u isti *namespace* kao i ostatak modela

```
namespace MVC.Models;  
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  
        => TipPartnera == "0" ? "Osoba" : "Tvrtka";  
}
```

# Uključivanje pogleda u EF-model (2)

- Kontekst proširen novim *DbSetom* koji odgovara pogledu
  - naziv svojstva obično odgovara nazivu pogleda, ali nije nužno
    - Može se navesti u `ToView("naziv pogleda")`
    - ili SQL upit koji vraća rezultat traženog tipa, npr.  
`ctx.vw_Partner.FromSqlRaw("SELECT * FROM vw_Partner");`
  - Koristi se kao i drugi *DbSetovi*, ali samo za čitanje
- Primjer:  MVC \ ModelsPartial \ FirmaContext.cs

```
public partial class FirmaContext {  
    public virtual DbSet<ViewPartner> vw_Partner { get; set; }  
    partial void OnModelCreatingPartial(ModelBuilder modelBuilder)  
    {  
        modelBuilder.Entity<ViewPartner>(entity => {  
            entity.HasKey();  
            entity.ToTable("vw_Partner");  
        });  
    }  
}
```

# 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:  MVC \ ViewModels \ PartneriViewModel.cs

```
using MVC.Models;
using System.Collections.Generic;

namespace MVC.ViewModels
{
    public class PartneriViewModel
    {
        public IEnumerable<ViewPartner> Partneri { get; set; }
        public PagingInfo PagingInfo { get; set; }
    }
}
```

# Priprema modela za prikaz svih partnera

- Podaci se pripremaju slično kao u prethodnim primjerima
- Primjer:  MVC \ Controllers \ PartnerController.cs

```
public async Task<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), provjera broja stranica
    var partneri = query
        .Skip((page - 1) * pagesize)
        .Take(pagesize)
        .ToList();
    var model = new PartneriViewModel {
        Partneri = partneri,
        PagingInfo = pagingInfo
    };
    return View(model);
}
```

# Proširenje upita redoslijedom sortiranja

-  MVC \ Extensions \ Selectors \ PartnerSort.cs

```
public static IQueryable<ViewPartner> ApplySort(
    this IQueryable<ViewPartner> query, int sort, bool ascending) {

    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);
    return query;
}
```

# Prikaz svih partnera

- (po uzoru na prethodne)  MVC \Views \ Partner \ Index.cshtml

```
@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-id="@partner.IdPartnera"  
                    ...>  
            </td>  
    </tr>
```

# 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:  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; }
    ...
}
```

# Priprema za unos novog partnera

- Inicijalno postavljeno da se radi o osobi, ali moguće promijeniti prije samog unosa
  - Primjer:  MVC \ Controllers \ PartnerController.cs

```
[HttpGet]
public IActionResult Create()
{
    PartnerViewModel model = new PartnerViewModel
    {
        TipPartnera = "0"
    };
    return View(model);
}
```

# 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:  MVC\ Views \ Partner \ Create.cshtml

```
@model PartnerViewModel  
...  
<form asp-action="Create" method="post">  
    <label class="radio-inline">  
        <input type="radio" asp-for="TipPartnera" value="0">  
        Osoba  
    </label>  
    <label class="radio-inline">  
        <input type="radio" asp-for="TipPartnera" value="T">  
        Tvrтka  
    </label>
```

# Odabir tipa partnera (2)

- Dio kontrola treba prikazati samo ako se unosi nova osoba, odnosno ako se unosi nova tvrtka.
  - Bit će skriveno/otkriveno korištenjem JavaScripta
  - Kontrole pronalazimo po odgovarajućem stilu
    - U Stil koji nema svoje vizualne osobine, već služi za pronalazak takvih kontrola
  - Primjer:  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>  
    ...
```

# Odabir tipa partnera (3)

- Nakon učitavanja stranice te pri svakoj promjeni označenog radiobuttona skrivaju se odnosno prikazuje odgovarajuće kontrole
  - Primjer:  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 == '0') {
                $(".samotvrtka").hide(); $(".samoosoba").show();
            }
            else {
                $(".samoosoba").hide(); $(".samotvrtka").show();
            }
        }
    </script>
}
```

# Validacija prilikom unosa novog partnera (1)

- Model *PartnerViewPartner* sadrži podatke i za osobu i za tvrtku
- Atribut *Required* na imenu osobe (ili *RuleFor(p => p.ImeOsobe).NotEmpty()* koristeći *FluentValidation*) 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 vlastitu provjeru
  - Može se napisati vlastiti atribut
  - Moguće koristiti *FluentValidation* i *When*
    - U oba slučaja moguće (iako ne baš jednostano) moguće napraviti i javascript za klijentsku validaciju
  - Implementirati sučelje *IValidatableObject*
    - Validacija samo na serveru, ali prihvatljivo za primjer

# Validacija prilikom unosa novog partnera (2)

- Implementirati sučelje *IValidableObject*
  - Vraća enumeraciju s opisom pogrešaka (objekti tipa *ValidationResult*)
  - Validacija se odvija samo na serveru
- Primjer:  MVC \ ViewModels \ PartnerViewModel.cs

```
public class PartnerViewModel : IValidableObject {
    public IEnumerable<ValidationResult>
        Validate(ValidationContext validationContext) {
    ...
    new ValidationResult("Potrebno je upisati ime osobe",
        new [] { nameof(ImeOsobe) } );
    ...
}
```

# Validacija prilikom unosa novog partnera (3)

- Primjer:  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 == "0") {
        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 Tvrтka)
  - Vlastita metoda CopyValues prikazana na sljedećem slajdu
  - Primjer:  MVC \ Controllers \ PartnerController.cs

```
public async Task<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);  
            await ctx.SaveChangesAsync();  
            ...  
        }  
    }  
}
```

# Unos novog partnera (2)

- Kopiraju se potrebna svojstva te stvara nova instanca Osobe ili Tvrtke
  - Primjer:  MVC \ Controllers \ PartnerController.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 == "0") {  
        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;  
    }  
}
```

# Strani ključ i *identity* tip podatka

- U postupku CopyValues stvoren novi objekt tipa Osoba ili Tvrta
  - Primjer:  MVC \ Controllers \ PartnerController.cs

```
public async Task<IActionResult> Create(PartnerViewModel model) {  
    ...  
    Partner p = new Partner();  
    ...  
    //CopyValues p.Osoba = new Osoba();  
    ...  
    ctx.Add(p);  
    await ctx.SaveChangesAsync();  
    ...  
}
```

- EF će automatski stvoriti odgovarajuće dvije *Insert* naredbe
- Uz prvu Insert naredbu EF izvršava i upit za dohvati *identity* vrijednosti primarnog ključa koja se koristi kao primarni i strani ključ za tablicu Osoba odnosno Tvrta.

# Nadopunjavanje umjesto padajuće liste (1)

- 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 predstavlјat će naziv mjesta, a dohvaćeni identifikator mjesta će se pohraniti u skriveno polje: podaci su parovi oblika (identifikator, tekst)

# Razred za pohranu rezultata servisa (1)

- Podaci za dinamičku padajuću listu (nadopunjavanje) sastoje se od identifikatora i oznake
  - Primjer:  MVC \ ViewModels \ 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**

# Razred za pohranu rezultata servisa (2)

- Za izbjegavanje nedoumica i potencijalnih problema kod pretvorbe u/iz JSON-a, koristi se atribut JsonPropertyName
  - Primjer:  MVC \ ViewModels \ IdLabel.cs

```
using System.Text.Json.Serialization;
```

```
...
public class IdLabel
{
    [JsonPropertyName("label")]
    public string Label { get; set; }
    [JsonPropertyName("id")]
    public int Id { get; set; }
```

- Alternativno u Startup.cs se može postaviti PropertyNamingPolicy na JsonNamingPolicy.CamelCase ili null (ako se ne želi camel-case)

```
services.AddControllersWithViews()
    .AddJsonOptions(configure =>
        configure.JsonSerializerOptions.PropertyNamingPolicy = ...)
```

# Upravljač za dohvata 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 *IdLabel*
  - Primjer:  MVC \ Controllers \ AutoCompleteController.cs (*Mjesto*)

```
public async Task<IEnumerable<IdLabel>> Mjesto(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 = await query.OrderBy(l => l.Label)  
        .Take(appData.AutoCompleteCount)  
        .ToListAsync();  
  
    return list;
```

# Nadopunjavanje umjesto padajuće liste (2)

- Upravljač i akciju možemo isprobati direktnim pozivom u pregledniku



```
[{"label": "42000 Varaždin", "id": 6245}, {"label": "42204 Varaždin Breg", "id": 6246}, {"label": "42223 Varaždinske Toplice", "id": 6247}]
```

- Na klijentskoj strani koristit će se autocomplete iz *jQuery UI*
  - To je razlog zašto se argument morao zvati term
- Kako prepoznati kontrole kojima treba pridružiti dinamičke padajuće liste i gdje pohraniti identifikator?
  - Te informacije bit će zapisane u *data* atribut oblika *data-naziv*
  - Tekstualno polje za unos teksta i (uobičajeno skrivena) kontrola za pohranu vrijednosti odabira (*id*)

# Priprema i označavanje kontrola za unos

- *autocomplete* ćemo aktivirati na svim kontrolama koje imaju atribut *data-autocomplete*, a vrijednost atributa će biti naziv akcije u upravljaču *AutoComplete*
  - Koristiti ćemo i atribut *data-autocomplete-placeholder-name* za prepoznati (skrivenu) kontrolu u koju ćemo pohraniti odabir
  - Ta kontrola mora definirati atribut *data-autocomplete-placeholder* čija vrijednost odgovara onoj iz *data-autocomplete-placeholder-name*
  - Primjer:  MVC \ Views \ Partner \ Create.cshtml

```
<label asp-for="IdMjestaPartnera"></label>
<input class="form-control" asp-for="NazMjestaPartnera"
       data-autocomplete="mjesto"
       data-autocomplete-placeholder-name="mjestopartnera"
       value="@Model.NazMjestaPartnera" />
<input type="hidden" asp-for="IdMjestaPartnera"
       data-autocomplete-placeholder="mjestopartnera" />
```

# Javascript biblioteke za nadopunjavanje

- jQuery UI (dodati u libman.json) + vlastita skripta
- Ne koristi se svugdje, uključuje se po potrebi
  - Primjer:  MVC \ Views \ Partner \ Create.cshml

```
@section styles{
    <link rel="stylesheet"
    href="~/lib/jqueryui/themes/base/jquery-ui.css" /> }
@section scripts{
    <script src="~/lib/jqueryui/jquery-ui.js"></script>
    <script src="~/js/autocomplete.js"></script>
```

- Potrebno znati putanju do naše aplikacije (može biti npr. oblika (<https://server/apps/my-app> pa trebamo znati korijen, npr. apps))
  - Primjer:  MVC \ Views \ Shared \ \_Layout.cshml

```
<script>
    window.applicationBaseUrl = '@Url.Content("~/")';
</script>
```

# Aktiviranje nadopunjavanja (1)

- Primjer:  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 kojim se prepoznaje kontrola za pohranu dohvaćene vrijednosti
      - Ako se ne navede (bit će praktično kasnije) koristi se naziv akcije
    - ...

```
$("[data-autocomplete]").each(function (index, element) {  
    var action = $(element).data('autocomplete');  
    var resultplaceholder =  
        $(element).data('autocomplete-placeholder-name');  
    if (resultplaceholder === undefined)  
        resultplaceholder = action;  
    ...  
});
```

# Aktiviranje nadopunjavanja (2)

- Primjer:  MVC \ wwwroot \ js \ autocomplete.js
  - Za svaki element koji ima definiran vlastiti atribut data-autocomplete:
    - ...
    - Definiraj sljedeće ponašanje: *Ako se promijeni tekst, obriši pohranjeni identifikator*
    - ...

```
$("[data-autocomplete]").each(function (index, element) {  
    ...  
    $(element).change(function () {  
        var dest = $('[data-autocomplete-placeholder'  
                  ='${resultplaceholder}']');  
        var text = $(element).val();  
        if (text.length === 0  
            || text !== $(dest).data('selected-label')) {  
            $(dest).val('');  
        }  
    }); ...  
});
```

# Aktiviranje nadopunjavanja (3)

- Primjer:  MVC \ wwwroot \ js \ autocomplete.js
  - Za svaki element koji ima definiran vlastiti atribut data-autocomplete:
    - ... Aktiviraj 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 (vidi sljedeći slajd)

```
$("[data-autocomplete]").each(function (index, element) {  
    ...  
    $(element).autocomplete({  
        source: window.applicationBaseUrl + "autocomplete/"  
                + action,  
        autoFocus: true,  
        minLength: 1,  
        select: function (event, ui) {  
            .... Nakon što je odabранo nešto iz liste...  
        }  
    });
```

# Aktiviranje nadopunjavanja (4)

- Primjer:  MVC \ wwwroot \ js \ autocomplete.js
  - Kad se nešto odabere iz padajuće liste, identifikator (vrijednost) se kopira na odredište (skriveno polje)

```
$("[data-autocomplete]").each(function (index, element) {  
    ...  
    $(element).autocomplete({  
        ...  
        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 prilikom snimanja
  - Prethodno povezani podaci su dio modela koji se vraćaju pogledu
  - Naziv mesta je dio modela i koristi se asp-for na odgovarajućem polju, pa ga ne treba ponovo rekonstruirati
  - Primjer:  MVC \ Controllers \ PartnerController.cs

```
try {  
    ctx.Add(p);  
    await ctx.SaveChangesAsync();  
    ...  
}  
catch (Exception exc) {  
    ModelState.AddModelError(string.Empty,  
        exc.CompleteExceptionMessage());  
    return View(model);  
}
```

# Dohvat podataka o partneru prilikom ažuriranja

- Kao model za pogled koristi se isti model kao kod dodavanja
  - Prvo se vrši dohvati zajedničkih podataka iz tablice Partner
  - Ostatak podataka puni se upitom na tablicu Osoba ili Tvrtka
    - Umjesto Find[Async] mogu se koristiti i varijante s Where
  - Primjer:  Mvc \ Controllers \ PartnerController.cs

```
public async Task<IActionResult> Edit(int id, ...) {  
    var partner = ctx.Partner.FindAsync(id);  
    ...  
    PartnerViewModel model = new PartnerViewModel {  
        IdPartnera = partner.IdPartnera,  
        IdMjestaIsporuke = partner.IdMjestaIsporuke,  
        ...  
    };  
    if (model.TipPartnera == "0") {  
        Osoba osoba = ctx.Osoba.Find(model.IdPartnera);  
        model.ImeOsobe = osoba.ImeOsobe;  
        ...  
    }  
}
```

# Ažuriranje podataka o partneru (1)

- Podaci povezani kroz model se provjeravaju na validacijske pogreške (slično kao kod *Create*)
  - Ako je model ispravan, vrši se dohvrat 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:  Mvc \ Controllers \ PartnerController.cs

```
[HttpPost][ValidateAntiForgeryToken]
public ... Edit(PartnerViewModel model...) {
    var partner = ctx.Partner.Find(model.IdPartnera);
    ...
    if (ModelState.IsValid) {
        try
        {
            CopyValues(partner, model);
            ...
        }
    }
}
```

# 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 (Tvrka) jednako null te se instancira novi objekt
  - Primjer:  Mvc \ Controllers \ PartnerController.cs

```
void CopyValues(Partner partner, PartnerViewModel model) {  
    partner.AdrIsporuke = model.AdrIsporuke;  
  
    ...  
    partner.Oib = model.Oib;  
    if (partner.TipPartnera == "0") {  
        partner.Osoba = new Osoba();  
        partner.Osoba.ImeOsobe = model.ImeOsobe;  
        partner.Osoba.PrezimeOsobe = model.PrezimeOsobe;  
    }  
    else {  
        partner.Tvrka = new Tvrka();  
        partner.Tvrka.MatBrTvrte = model.MatBrTvrte;  
        partner.Tvrka.NazivTvrte = model.NazivTvrte; ...  
    }  
}
```

# Snimanje promjena

- 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:  Mvc \ Controllers \ PartnerController.cs

```
public async Task<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) ...  
    await ctx.SaveChangesAsync();
```

# 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 Tvrтka se automatski briše
- Dohvat se može izvršiti s *Where* ili postupkom *Find* (vrijednost PK)
- Primjer:  MVC \ Controllers \ PartnerController.cs

```
public async Task<IActionResult> Delete(int id,...) {  
    var partner = await ctx.Partner.FindAsync(id);  
    if (partner != null) {  
        try {  
            ctx.Remove(partner);  
            await ctx.SaveChangesAsync();  
        }  
        ...  
    }  
}
```

- Umjesto `ctx.Remove` moglo se napisati i

```
    ctx.Entry(partner).State = EntityState.Deleted;
```

# Razvoj primijenjene programske potpore

---

## **8.1 Primjer zaglavije-stavke (1.dio)**

Prikaz svih dokumenata, filtriranje dokumenata,  
pojedinačni prikaz

# Pogled za prikaz svih dokumenata

- Kao i u primjeru s partnerima za dohvati svih dokumenata koristi se pogled iz baze podataka
  - Značajno pojednostavljuje kôd za dohvati 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 '0' 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:  MVC \ ModelsPartial \ 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  
}
```

# (Podsjetnik) Dodavanje pogleda u EF model (2)

- U definiciju konteksta dodati novi DbSet za pogled
  - Primjer:  MVC \ ModelsPartial \ FirmaContext.cs

```
namespace MVC.Models
public partial class FirmaContext {
    ...
    public virtual DbSet<ViewDokumentInfo> vw_Dokumenti { get; set; }

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder) {
        ...
        modelBuilder.Entity<ViewDokumentInfo>(entity => {
            entity.HasKey();
            //entity.ToTable("vw_Dokumenti");
                //u slučaju da se DbSet svojstvo zove drugačije
        });
    }
}
```

# Filtriranje podataka

- Popis dokumenata moguće filtrirati po partneru, iznosu ili datumu.
- Za odabir partnera nadopunjavanje, a za odabir datuma kalendar
  - Kalendar po automatizmu (zbog tipa *DateTime* i *input asp-for*)
  - Atribut *[DataType(DataType.Date)]* uklanja odabir vremena
- 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

```
<partial name="KriterijPretrage"
model="Model.Filter" />
```

The screenshot shows a user interface for filtering data. At the top, there is a text input for 'IdPartnera' with value '0' and a placeholder 'Zadovoljni korisnik (01234567988)'. Below it are two input fields for 'Iznos': one with '300' and another with '500'. Further down is a date input 'Datum' with '01.05.2015.' and a calendar dropdown showing 'svibanj, 2017' with a selected date '10.05.2017.'. To the right of the calendar are icons for filter and delete. At the bottom, there is a date input 'Datum dokumenta' with '12.01.2014.' and a small calendar icon.



# 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 pronašlaskom 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
    - *IPageFilter* - vlastito sučelje kao „zajednički nazivnik“ za pager
  - Primjer:  MVC \ ViewModels \ DokumentFilter.cs

```
public class DokumentFilter : IPageFilter {  
    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;  
    }  
}
```

# Razred za informacije o filtru (2)

- Uneseni podaci mogu se spojiti u string te rekonstruirati iz stringa
  - Primjer:  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();  
        string[] arr = s.Split('-', 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;  
    }  
}
```

# Razred za informacije o filtru (3)

- Upit za dohvat svih dokumenata filtrira se po odabranim kriterijima
  - Primjer:  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 nadopunjavanje)

- Izvedeno korištenjem pogleda korištenog za pregled svih partnera
  - Primjer:  MVC \ Controllers \ AutoComplete.cs

```
public async Task<IEnumerable<IdLabel>> Partner(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 = await query.OrderBy(l => l.Label)  
        .ThenBy(l => l.Id)  
        .Take(appData.AutoCompleteCount)  
        .ToListAsync();  
}
```

# Digresija: Dohvat svih partnera upitom bez pogleda

- Bez pogleda, kôd za upit korištenjem EF-a bi bio puno složeniji

```
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();
```

# Aktiviranje nadopunjavanja za partnere

- Izvedeno slično kao kod padajuće liste za mjesta prilikom dodavanja novog partnera
  - Vlastita skripta aktivira nadopunjavanje za sve kontrole koje imaju definiran atribut *data-autocomplete* pri čemu vrijednost tog atributa predstavlja relativnu adresu 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:  MVC \ Views \ Dokument \ KriterijPretrage.cshtml

```
<input asp-for="IdPartnera" readonly="readonly" class="form-control"  
       data-autocomplete-placeholder="partner" />
```

```
<input class="form-control" data-autocomplete="partner"  
      asp-for="NazPartnera" />
```

# 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:  MVC \ Controllers \ DokumentController.cs

```
public async Task<IActionResult> Index(string filter, ...  
...  
DokumentFilter df = DokumentFilter.FromString(filter);  
if (!df.IsEmpty()) {  
    if (df.IdPartnera.HasValue) {  
        df.NazPartnera = await ctx.vw_Partner  
            .Where(p => p.IdPartnera == df.IdPartnera)  
            .Select(vp => vp.Naziv)  
            .FirstOrDefaultAsync();  
    }  
    query = df.Apply(query);  
} ...
```

# Model za rad s dokumentima (1)

- Prezentacijski model s validacijskim atributima (bit će korišteni kod ažuriranja)
  - Primjer:  MVC \ ViewModels \ DokumentViewModel.cs

```
public class DokumentViewModel {  
    public int IdDokumenta { get; set; }  
  
    ...  
    [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); }  
    }  
    ...
```

# Model za rad s dokumentima (2)

- Osim atributa koji će u konačnici završiti u tablici Dokument, model sadrži i kolekciju stavki
  - Primjer:  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

# Model za rad sa stavkama

- Novi razred kao model za rad sa stavkama da se izbjegnu problemi s vezom prema tablici Dokument
  - Primjer:  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);  
        }  
    }  
}
```

# Prikaz dokumenta (1)

- Prvo dohvati dokumenta, a potom i njegovih stavki
  - Podaci o prethodnom dokumentu i partneru se dohvaćaju naknadnu preko odgovarajućih pogleda (relativno jednostavan kod, ali prevelik za slajdove)
  - Primjer:  MVC \ Controllers \ DokumentController.cs
    - Naziv pogleda je argument ove metode, jer se isti kod koristi i za pripremu dokumenta za ažuriranje

```
public async Task<IActionResult> Show(int id, ....,  
                                         string viewName = nameof(Show)) {  
  
    var dokument = await ctx.Dokument  
        .Where(d => d.IdDokumenta == id)  
        .Select(d => new DokumentViewModel {  
            BrDokumenta = d.BrDokumenta,  
            DatDokumenta = d.DatDokumenta,  
            ...  
            VrDokumenta = d.VrDokumenta  
        })  
        .FirstOrDefaultAsync();
```

# Prikaz dokumenta (2)

- Nakon dohvata dokumenta, dohvatimo sve njegove stavke
  - Primjer:  MVC \ Controllers \ DocumentController.cs

```
public async Task<IActionResult> Show(int id, ...,
                                         string viewName = nameof(Show)) {
    ...
    var stavke = await ctx.Stavka
        .Where(s => s.IdDokumenta ==
                    dokument.IdDokumenta)
        .OrderBy(s => s.IdStavke)
        .Select(s => new StavkaViewModel {
            IdStavke = s.IdStavke,
            NazArtikla = s.SifArtiklaNavigation
                .NazArtikla,
            ...
        })
        .ToListAsync();
    dokument.Stavke = stavke;
```

## Prikaz dokumenta (2)

- U zaglavlju prikazani podaci dokumenta, a nakon toga tablično prikazane pojedinačne stavke
- Primjer:  MVC \ Views \ Dokument \ Show.cshtml

# Određivanje sljedbenika i prethodnika (1)

- Na stranici za prikaz 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
  - Inicialno pridijeljeno prilikom dohvata dokumenata
  - Primjer:  MVC \ Controllers \ DokumentController.cs

```
public async Task<IActionResult> Index(string filter,...  
... Dohvati dokumente u ovisnosti o filtru i sortu ...  
  
for(int i=0; i<dokumenti.Count; i++)  
    dokumenti[i].Position = (page - 1) * pagesize + i;
```

# Određivanje sljedbenika i prethodnika (2)

- Za prikaz dokumenta potrebno imati njegov id, a ne samo poziciju
- Formira se upit s istim filterom (sort je nebitan) i dohvate identifikatori prethodnika i sljedbenika
- Primjer:  MVC \ Controllers \ DokumentController.cs

```
private async Task SetPreviousAndNext(int position, string filter,
                                     int sort, bool ascending) {
    var query = ctx.vw_Dokumenti.AsQueryable();
    ... primjeni filter nad upitom
    if (position > 0)
        ViewBag.Previous = await query.Skip(position - 1)
            .Select(d => d.IdDokumenta)
            .FirstAsync();
    if (position < await query.CountAsync() - 1)
        ViewBag.Next = await query.Skip(position + 1)
            .Select(d => d.IdDokumenta)
            .FirstAsync();
```

# Razvoj primijenjene programske potpore

---

## 8.2 Primjer zaglavije-stavke (2.dio)

### Ažuriranje podataka

# 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 (zaglavlj + 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



# 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
- Slično kao kod prikaza pojedinačnog složenog podatka
  - Dohvat podatka je isti, samo se koristi drugačiji pogled za prikaz
  - Primjer:  MVC \ Controllers \ DokumentController.cs

```
[HttpGet]
```

```
public Task<IActionResult> Edit(int id,
                                int? position, string filter, int page = 1,
                                int sort = 1, bool ascending = true) {
    return Show(id, position, filter, page, sort, ascending,
                viewName: nameof(Edit));
}
```

# Priprema dokumenta za ažuriranje (2)

- Odabir partnera i prethodnog dokumenta vrši se nadopunjavanjem, pa ne treba pripremati podatke za padajuće liste
  - Primjer:  MVC \ Controllers \ AutoCompleteController.cs : postupci Dokument i Partner
- Potrebno dohvatiti nazine partnera i dokumenta kako bi se prikazali korisniku
  - Za partnera provjeriti tip i dohvatiti ime i prezime, odnosno naziv tvrtke
  - Slično za dokument, pri čemu je potrebno koristiti isti format koji se koristi u upravljaču koji služi kao izvor za nadopune
  - Programski kod prevelik za slajd, ali relativno jednostavan...
  - Primjer:  MVC \ Controllers \ DokumentController.cs

```
[HttpGet]
```

```
public async Task<IActionResult> Show(int id, ...  
...  
dokument.NazPartnera = ...  
if (dokument.IdPrethDokumenta.HasValue) {  
    dokument.NazPrethodnogDokumenta = ...
```

# 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 uključen parcijalni pogled za stavke dokumenta
  - Primjer:  MVC \ Views \ Dokument \ Edit.cshtml

```
@model DokumentViewModel  
...  
<form id="form" method="post" asp-action="Edit" ... >  
    <input type="hidden" asp-for="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 više kontrola s istim atributom *name*, unesene vrijednosti se na upravljaču prihvataju u istoimenom argumentu koji je polje određenog tipa
- 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) elemente naziva *Stavka.Index* čija je vrijednost jednaka korištenoj vrijednosti u uglatim zagradama

```
<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* pri čemu se kao indeks koristi šifra artikla
  - Primjer:  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"/>
    ...
}
```

# Izmjena dokumenta i njegovih stavki (1)

- Prilikom dohvata dokumenta potrebno dohvatiti njegove stavke
- Kao prvi korak vrši se kopiranje podataka dokumenta iz povezanog modela u objekt koji je dohvaćen iz BP
  - Primjer:  MVC \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(DokumentViewModel model, ...
    var dokument = await ctx.Dokument
        .Include(d => d.Stavka)
        .Where(d => d.IdDokumenta == model.IdDokumenta)
        .FirstOrDefaultAsync();

...
    ... Kopiraj podatke o dokumentu iz modela u objekt dokument
    ... dokument.PostoPorez = model.StopaPoreza / 100m;
    dokument.VrDokumenta = model.VrDokumenta;
```

# 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:  MVC \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<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 ili nove stavke ili neke od postojećih koje treba ažurirati
  - Postojeće treba dohvatiti iz objekta prethodno dohvaćenog iz baze podataka
  - Nove treba stvoriti i dodati u dokument
  - U oba slučaja u novi objekt kopirati svojstva iz povezanog modela
- Primjer:  MVC \ Controllers \ DokumentController.cs

```
foreach (var stavka in model.Stavke) {  
    Stavka novaStavka; //potpuno nova ili ona koju treba izmijeniti  
    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
```

# Izmjena dokumenta i njegovih stavki (4)

- Potrebno izračunati iznos dokumenta i spremiti promjene
  - Primjer:  MVC \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(DokumentViewModel model, ...
...
dokument.IznosDokumenta = (1 + dokument.PostoPorez)
    *
    dokument.Stavka
        .Sum(s =>
            s.KolArtikla
            * (1 - s.PostoRabat)
            * s.JedCijArtikla
        );
await ctx.SaveChangesAsync();
```

# Odabir artikla za novu stavku

- Redak na dnu postojećih stavki u kojem se nadopunjavanjem bira artikel
  - Povratni podaci sa servera sadrže šifru, naziv i cijenu artikla
  - Cijena artikla se kopira u odgovarajuće polje
  - Primjer  MVC \ Views \ Dokument \ Stavke.cshtml

```
<tr>
    <td>
        <input id="artikl-sifra" type="hidden" data-autocomplete-
placeholder="artikl" readonly="readonly" />
        <input id="artikl-naziv" type="text" data-autocomplete="artikl" />
    </td>
    <td class="text-center col-sm-1">
        <input id="artikl-kolicina" type="text" />
    </td> ...
```

- Pogledati programski kôd u sljedećim datotekama:
  -  MVC \ ViewModels \ AutoCompleteArtikl.cs
  -  MVC \ Controllers \ AutoCompleteController.cs : Artikel
  -  MVC \ wwwroot \ js \ autocomplete.js

# Dodavanje nove stavke

- Skrivena tablica s identifikatorom *template* koja služi za dodavanje novog retka na temelju vrijednosti iz posebnog retka na dnu stavki (opisano na prethodnom slajdu)
  - 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:
  -  MVC \ Views \ Dokument \ NovaStavkaTemplate.cshtml
  -  MVC \ wwwroot \ js \ dokumenti.js
    - Postupak *dodajArtikl* i kod koji poziva postupak

# Brisanje stavke

- Gumb za brisanje stavke (prepoznaje se po stilu *deleterow*) uklanja stavku iz strukture HTML dokumenta.
  - Primjer:  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 ...  
});
```

- Posljedično bit će izbrisana iz dokumenta nakon klika na gumb za snimanje

# 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:  MVC \ Controllers \ DokumentController.cs

```
[HttpGet]
```

```
public async Task<IActionResult> Create() {
    int maxbr = await ctx.Dokument.Max(d => d.BrDokumenta) + 1;
    var dokument = new DokumentViewModel {
        DatDokumenta = DateTime.Now,
        BrDokumenta = maxbr
    };
    return View(dokument);
}
```

- Ostatak izведен slično kao kod ažuriranja (razlika u tome što su sve stavke nove).

# Brisanje dokumenta

- U skladu s prethodnim primjerima.
- U BP definirano kaskadno brisanje, pa je dovoljno dohvatiti dokument i obrisati ga
  - Ne treba brisati pojedinačno svaku stavku.
  - Primjer:  MVC \ Controllers \ DokumentController.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(int IdDokumenta, ...) {
    var dokument = await ctx.Dokument
        .Where(d => d.IdDokumenta == IdDokumenta)
        .SingleOrDefaultAsync();

    if (dokument != null) {
        ...
        ctx.Remove(dokument);
        await ctx.SaveChangesAsync();
    }
}
```

# Razvoj primijenjene programske potpore

---

## 9. Web-aplikacije

### Slanje datoteke i prikaz slike

# Organizacija primjera

- Primjer izведен s dinamičkim ažuriranjem i brisanjem kao kod primjera s mjestima
- Ključne razlike
  - Prikaz pojedinačnog retka u posebnom (parcijalnom) pogledu
    - Primjer:  MVC \ Views \ Artikel \ Index.cshtml

```
@model ArtikliViewModel  
...  
@foreach (var artikel in Model.Artikli) {  
    <partial name="Get" model="artikel" />  
}  
}
```

- Prilikom ažuriranja artiklu se može pridružiti nova slika, obrisati postojeća ili ostaviti podatak o slici neizmijenjenim

# Slike artikala

- Slika artikla pohranjena u tablici Artikl → Što sadrži model za prikaz pojedinog artikla
  - Preuzeti sliku zajedno s ostalim podacima o artiklu?  
→ sporiji dohvati 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?*

# Problem promjene slike artikla

- Dohvat slike se vrši po šifri artikla, a ne po identifikatoru slike
- Što ako preglednik sliku pospremi u cache, a slika se promijeni?
- Varijanta 1:
  - Nova slika = nova adresa!
- Kako na serveru znati da je slika nova?
  - Svakoj slici pridijeliti jedinstveni broj?
  - Koristiti neku varijantu sažetka?
- U tablicu Artikel je dodano izračunato polje *CHECKSUM(SlikaArtikla)*

RPPP2\SQL2012.Firma - dbo.Artikel			
	Column Name	Data Type	Allow Nulls
	SifArtikla	int	<input type="checkbox"/>
	NazArtikla	nvarchar(255)	<input type="checkbox"/>
	JedMjere	nvarchar(5)	<input type="checkbox"/>
	CijArtikla	money	<input type="checkbox"/>
	ZastUsluga	bit	<input type="checkbox"/>
	TekstArtikla	nvarchar(MAX)	<input checked="" type="checkbox"/>
	SlikaArtikla	varbinary(MAX)	<input checked="" type="checkbox"/>
	SlikaChecksum		<input checked="" type="checkbox"/>

Column Properties

Computed Column Specification (Formula) `(checksum([SlikaArtikla]))`

Is Persisted No

# Model za prikaz artikala

- Primjer:  MVC \ ViewModels \ ArtikliViewModel.cs

```
public class ArtikliViewModel
{
    public IEnumerable<ArtikliViewModel> Artikli { get; set; }
    public PagingInfo PagingInfo { get; set; }
}
```

- Model za pojedinačni artikl je razred *ArtiklViewModel*

- Primjer:  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

- Projekcija na prezentacijski model bez dohvata slike
  - Umjesto (sadržaja) slike, evidentira se postoji li slika
- Primjer:  MVC \ Controllers \ ArtikelController.cshtml

```
var artikli = ctx.Artikel
    .Select(a => new ArtikelViewModel {
        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 bili sigurni da preglednik zna da se radi o novoj slici
- Primjer:  MVC \ Views \ Artikl \ Get.cshtml

```
@if (Model.ImaSliku) {  
      
}
```

# Akcija za dohvati sliku

- Polje bajtova koje predstavlja sliku artikla dohvati se EF upitom
  - Rezultat postupka je *IActionResult*,
    - u slučaju da slika postoji vraća se binarni sadržaj pozivom postupka *File* iz upravljača.
    - Ako slike nema, vraća se status 404 s *NotFound*
  - Primjer:  MVC \ Controllers \ ArtiklController.cs

```
public async Task<IActionResult> GetImage(int id) {  
    ...  
    byte[] image = await ctx.Artikl  
        .Where(a => a.SifArtikla == id)  
        .Select(a => a.SlikaArtikla)  
        .SingleOrDefaultAsync();  
    return File(image, "image/jpeg" ...  
    ...
```

# Kako sugerirati pregledniku da spremi sliku u cache? (1)

- Druga (i bolja) varijanta rješenja (problema) *cacheiranja* slika u pregledniku
- Koristi se *entity tag* (ETag)
  - Proizvoljni identifikator resursa kojim preglednik utvrđuje ima li aktualnu verziju resursa. Ako postoji preglednik ga šalje prilikom zahtjeva u zaglavlju *If-None-Match*
  - U slučaju da se tagovi poklapaju, vraća se status 304 (Not Modified)
- Primjer:  MVC \ Controllers \ ArtikelController.cs

```
public async Task<IActionResult> GetImage(int id) {  
    int? checksum = await ctx.Artikel  
        .Where(a => a.SifArtikla == id)  
        .Select(a => a.SlikaChecksum)  
        .SingleOrDefaultAsync();  
  
    if (checksum == null) return NotFound(); //slika ne postoji  
    string responseETag = "\"" + checksum.Value + "\"";  
    if (Request.Headers.TryGetValue(HeaderNames.IfNoneMatch,  
        out var requestETag) && requestETag == responseETag) {  
        return StatusCode((int) HttpStatusCode.NotModified);  
    }  
}
```

# Kako sugerirati pregledniku da spremi sliku u cache? (2)

- Kako je preglednik dobio ETag?
  - U primjeru se to dogodilo korištenjem jedne od varijanti postupka File što postavlja zaglavlj ETag u odgovoru
- Detaljnije na  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-None-Match>
- Primjer:  MVC \ Controllers \ ArtiklController.cs

```
public async Task<IActionResult> GetImage(int id) {  
    int? checksum = await ctx.Artikl  
        .Where(a => a.SifArtikla == id)  
        .Select(a => a.SlikaChecksum)  
        .SingleOrDefaultAsync();  
  
    ...  
    return File(image, "image/jpeg",  
               lastModified: DateTime.Now,  
               entityTag: new EntityTagHeaderValue(responseETag));
```

▶ GET https://localhost:44395/Artikl/GetImage/4409?hash=876721973

Status	200 OK ⓘ
Version	HTTP/2
Transferred	24.02 KB (23.79 KB size)
▼ Response Headers (239 B)	
content-length:	24360
content-type:	image/jpeg
date:	Thu, 16 Dec 2021 21:37:40 GMT
etag:	"876721973"
last-modified:	Thu, 16 Dec 2021 21:37:40 GMT

# Forma za slanje datoteke

- Prilikom unosa novog artikla može se poslati datoteka sa slikom
- Za unos se koristi HTML *input* kontrola tipa *file*
  - Naziv proizvoljan, ali mora odgovarati argumentu u akciji upravljača
- Forma mora imati atribut *enctype* postavljen na *multipart/form-data*
  - Primjer:  MVC \ Views \ Artikel \ 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>
```

# Prihvati 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:  MVC \ Controllers \ ArtikelController.cs

```
[HttpPost][ValidateAntiForgeryToken]
... async Task<IActionResult> Create(Artikl artikel, IFormFile slika){
    ...
    if (slika != null && slika.Length > 0) {
        using (MemoryStream stream = new MemoryStream()) {
            await slika.CopyToAsync(stream);
            byte[] image = stream.ToArray();
            artikel.SlikaArtikla = image; //smanji prije pridruživanja
        }
    }
    ctx.Add(artikel); await ctx.SaveChangesAsync(); ...
```

# Udaljena validacija na klijentskoj strani

- Provjerava postoji li već artikl s navedenom šifrom
  - sprječava se postavljanje upita koji će sigurno biti neuspješan
- Kako bi se korisniku ta informacija pružila i prije slanja forme koristi se tzv. udaljena validacija
  - Generira se javascript kod za poziv postupka na serveru (true/false)
  - Primjer:  MVC \ Models \ Artikl.cs

```
public partial class Artikl {  
    [Required]  
    [Remote(action: nameof(ArtiklController.ProvjeriSifruArtikla),  
            controller: "Artikl", ErrorMessage = "Šifra već postoji")]  
    public int SifArtikla { get; set; }  
}
```

- Primjer:  MVC \ Controllers \ ArtiklController.cs

```
public async Task<bool> ProvjeriSifruArtikla(int SifArtikla) {  
    return !await ctx.Artikl.AnyAsync(a => a.SifArtikla == SifArtikla);  
}
```

# 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?#custom-attributes>

# 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 prenesena u pogled → dohvati artikl iz b.p. i ažurirati potrebno
  - Primjer:  MVC \ Controllers \ ArtikelController.cs

```
public async Task<IActionResult> Edit(Artikel artikel,
                                         IFormFile slika, bool obrisiSliku) {
    Artikel dbArtikel = await ctx.Artikel.FindAsync(artikel.SifArtikla);
    ...
    dbArtikel.JedMjere = artikel.JedMjere;
    ...
    if (slika != null && slika.Length > 0) {
        ... izvuci byte[] image iz stremama
        dbArtikel.SlikaArtikla = image;
    }
    else if (obrisiSliku) dbArtikel.SlikaArtikla = null;
    await ctx.SaveChangesAsync();
}
```

# Alati za smanjivanje slike

- Ugrađena podrška u CoreCompact.System.Drawing
- Neki od paketa za rad sa slikama
  - <https://andrewlock.net/using-imagesharpenum-to-resize-images-in-asp-.net-core-a-comparison-with-corecompat-system-drawing>
  - <https://devblogs.microsoft.com/dotnet/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](https://kleisauke.github.io/net-vips/tutorial/getting_started.html)

# Uključivanje paketa NetVips

- Uključiti odgovarajuću verziju NetVipsa
- Ako na određenom računalu libvips nije instaliran i nisu postavljene varijable okruženja, može se uključiti u implementaciju
  - Potrebno uključiti za željenu platformu (npr. 64 bitne Windows)
  - Primjer:  MVC \ MVC.csproj

```
<PackageReference Include="NetVips" Version="1.2.4" />
<PackageReference Include="NetVips.Native.win-x64" Version="8.10.1" />
```

# Dodatne aplikacijske postavke

- Konfiguracijska datoteka proširena s visinom smanjene slike
  - Primjer:  MVC \ AppSettings.cs

```
public class AppSettings {  
    ...  
    public ImageSettingsData ImageSettings { get; set; }  
    public class ImageSettingsData {  
        public int ThumbnailHeight { get; set; } = 100;  
    }  
}
```

- Primjer:  MVC \ appsettings.json

```
{  
    "AppSettings": {  
        "PageSize": 10, ...  
        "ImageSettings": {  
            "ThumbnailHeight": 100,  
        } ...  
    } ...
```

# Izrada smanjene slike

- Primjer:  MVC \ Util \ ImageUtil.cs
  - Postavimo ciljanu visinu ili širinu, a NetVips sačuva omjer
  - Druga dimenzija postavljena na neku vrijednost koja je veća od stvarne

```
const int VIPS_MAX_COORD = 10000000;
public static byte[] CreateThumbnail(byte[] image,
                                     int? maxwidth = null, int? maxheight = null) {
    if (maxwidth == null && maxheight == null)
        throw new ArgumentException(
            "Maximum size for at least one of axis must be specified");
    using (var thumbnailImage = Image.ThumbnailBuffer(image,
                                                       maxwidth ?? VIPS_MAX_COORD,
                                                       height: maxheight ?? VIPS_MAX_COORD)) {
        byte[] thumbnail = thumbnailImage.WriteToBuffer(".jpg");
        return thumbnail;
    }
}
```

# Razvoj primijenjene programske potpore

---

**10. Izrada izvješća**

# 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

v1.4.3

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 orientacija
  - Primjer:  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  
        });  
    })
```

# 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:  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);  
    })  
}
```

# Zaglavlje i podnožje jednostavnih izvješća

- Jednostavna izvješća u zaglavljima imaju naslov, a u podnožju trenutni datum
  - Potrebno postaviti smjer teksta na LeftToRight
  - Primjer:  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:  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  
        .Where(a => a.SlikaArtikla != null)  
        .OrderByDescending(a => a.CijArtikla)  
        .Select(a => new {  
            a.SifArtikla, a.NazArtikla,  
            a.CijArtikla, a.SlikaArtikla  
        })  
        .Take(10).ToListAsync();
```

# 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:  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:  MVC \ Controllers \ ReportController.cs

```
report.MainTableColumns(columns => {
    columns.AddColumn(column => {
        column.IsRowNumber(true);
        column.CellsHorizontalAlignment(HorizontalAlignment.Right);
        column.Visible(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");
    });
});
```

# 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 dodavanjem zaglavlja *content-disposition:inline*
  - Preglednik nudi odabir mape za snimanje datoteke
  - Primjer:  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:  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 međusume za svaku stranicu s *PageSummarySettings*
  - Primjer:  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 zaglavje 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

10 najvećih kupnji							
<b>Id dokumenta:</b>	2237	<b>Datum dokumenta:</b>	11.02.2014	<b>Iznos dokumenta:</b>	2.178.811,73 kn		
<b>Partner:</b>	KBC	#	Naziv artika	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 crtica, 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 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)	5,00	403,00 kn	1,00%	1.994,85 kn
		12	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	4.799,00 kn	7,00%	17.852,28 kn
		13	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	5.192,00 kn	8,00%	19.106,56 kn
		14	RAM, 1 GB, SSSR 2, PC-5300, 667 MHz, Torba za notebook Marasst MFL30	10,00	178,00 kn	2,00%	1.744,40 kn
		15	Torbica za Strukkle triPOD trio nano 3-1, kožna, siva	4,00	141,00 kn	8,00%	518,88 kn
		16	TV LCD 82 cm, SHARP LC32WD1E, 16:9, stereo, HD Ready, 2 x HDMI, DVB-T	3,00	114,00 kn	4,00%	328,32 kn
		17	TV LCD 94 cm, PHILIP MORRIS 37PFL9732, FULL HD, DVB-T	1,00	7.399,00 kn	9,00%	6.733,09 kn
		18	TV LCD 94 cm, PHILIP MORRIS 37PFL9732, FULL HD, DVB-T	100,00	16.999,00 kn	1,00%	1.682.901,00 kn
		19					Ukupno 763.166,63 kn

# Oblik podataka za izvješća s grupiranjem

- Potrebno koristiti denormalizirane podatke
  - Primjer:  MVC \ Models \ 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* predstavljaće adresu dokumenta
  - Formirat će se nakon dohvata na osnovi usmjeravanja i *IdDokumenta*
  - Ne postoji u bazi podataka pa se mora označiti s *NotMapped*

# Funkcija za dohvat n najboljih kupnji

- Ispis n dokumenata s najvećim iznosom
  - Složen upit izведен korištenjem funkcije na BP
  - Kao i u primjeru pogleda u kontekst potrebno dodati *DbSet<StavkaDenorm>* i 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
  - Izostavljanje upita bi uzrokovalo upit *SELECT \* FROM StavkaDenorm*
- Upit u primjeru sadrži parametre
- Podaci moraju biti poredani po stupcima po kojima se želi izvršiti grupiranje
  - Poredak nakon toga po želji
- Primjer:  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();
```

# 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:  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:  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;
                });
        });
    });
}
```

# Stupac s automatskim izračunom vrijednosti

- Vrijednost stupca s ukupnom cijenom neke stavke računa se na osnovu ostalih vrijednosti iz trenutnog retka
  - Primjer:  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:  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:  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 (1)

- Vrijednosti grupe mogu se dobiti iz ulaznog argumenta tipa `IList<CellData>`
  - Primjer:  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");  
    }  
}
```

# Predložak za zaglavje grupe (2)

- Zaglavje grupe definirano kao tablica s 4 stupca s podacima grupe
  - Svaka ćelija može sadržavati fiksni tekst ili biti vezana za neku vrijednost
  - Primjer:  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");  
            });  
}
```

# Razvoj primijenjene programske potpore

---

## 11. Refleksija

# 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 prozor *Properties*)
  - dinamičko uključivanje dodataka (*pluginova*)

# 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 definiranim 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 izведен, informacije o tipovima koji su korišteni za parametrizaciju, ...)
    - služi za dohvat informacija o članovima razreda (atributi, svojstva, postupci, ...)

# 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");
```

# 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.Collections.Generic.Dictionary`2");`
  - Postupak `Type.GetType` je preopterećen te po želji omogućuje ignoriranje velikih i malih slova

# Dinamičko učitavanje asemblija

- Asemblij (dll ili exe) se može dinamički učitati unutar programa
- Učitavanje vrši korištenjem postupka *LoadFromAssemblyPath* nad objektom tipa *AssemblyLoadContext* (iz prostora imena System.Runtime.Loader)
- 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:  Reflection \ Reflection \ Program.cs

```
string assemblyLocation = " ... LottoImplementation.dll";
AssemblyLoadContext loadctx = AssemblyLoadContext.Default;
Assembly asm = loadctx.LoadFromAssemblyPath(
    Path.GetFullPath(assemblyLocation));
```

```
Type type = asm.GetType("LottoImplementation.Lotto");
...
...
```

# 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:  Reflection \ Reflection \ Program.cs

```
Type type = asm.GetType("LottoImplementation.Lotto");
object obj = Activator.CreateInstance(type, 7, 39);
```

```
MethodInfo info = type.GetMethod("DrawNumbers");
object result = info.Invoke(obj, new object[] {false});
```

```
string print = string.Join(", ", (List<int>)result);
Console.WriteLine(print);
```

- Bolja varijanta: definirati sučelje koje dinamički stvoreni tip implementira (ILotto u primjeru)

# Izvoz podataka u Excel (1)

- Prije još jedne demonstracije upotrebe refleksije slijedi primjer izvoza podataka u Excel



**EPPlus** by EPPlus Software AB

v5.5.0

A spreadsheet library for .NET framework and .NET core

- Potrebno instalirati paket EPPlus i u appsettings.json označiti da se koristi u ne-komercijalne svrhe
- Primjer: MVC \ appsettings.json

```
"EPPlus": {  
    "ExcelPackage": {  
        "LicenseContext": "NonCommercial"  
    }  
}
```

# Izvoz podataka u Excel (2)

- Primjer:  MVC \ Controllers \ ReportController.cs

```
const string ExcelContentType =
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";
public async Task<IActionResult> DrzaveExcel() {
    var drzave = ... //lista s podacima
    byte[] content;
    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");
}
```

# 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:  MVC \ ReportController.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");  
}
```

# Korištenje refleksije za izvoz u Excel (2)

- Nazivi stupaca jednaki nazivima (jednostavnih) svojstava
  - Primjer  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;  
    }  
    ...
```

# Korištenje refleksije za izvoz u Excel (3)

- Vrijednost nekog svojstva nekog objekta dobije se iz objekta tipa  *PropertyInfo*
  - Primjer  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 u glatim 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* primjenjenog na asemblrij, član razreda, ...
  - GetCustomAttributes – vraća polje atributa
  - IsDefined – određuje je li ijedan atribut zadano tipa *Type* definiran

# 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  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;  
    }  
}
```

# 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{...}
```

# Primjer vlastitog atributa

- Primjer  MVC \ Util \ ExcelFormatAttribute.cs
  - Atribut kojim će se definirati format čelije 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;
    }
}
```

# Primjer korištenja vlastitog atributa (1)

- Primjer  MVC\ViewModels \ 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  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;
```

# Razvoj primijenjene programske potpore

---

## 12. Testiranje

# Općenito o testiranju

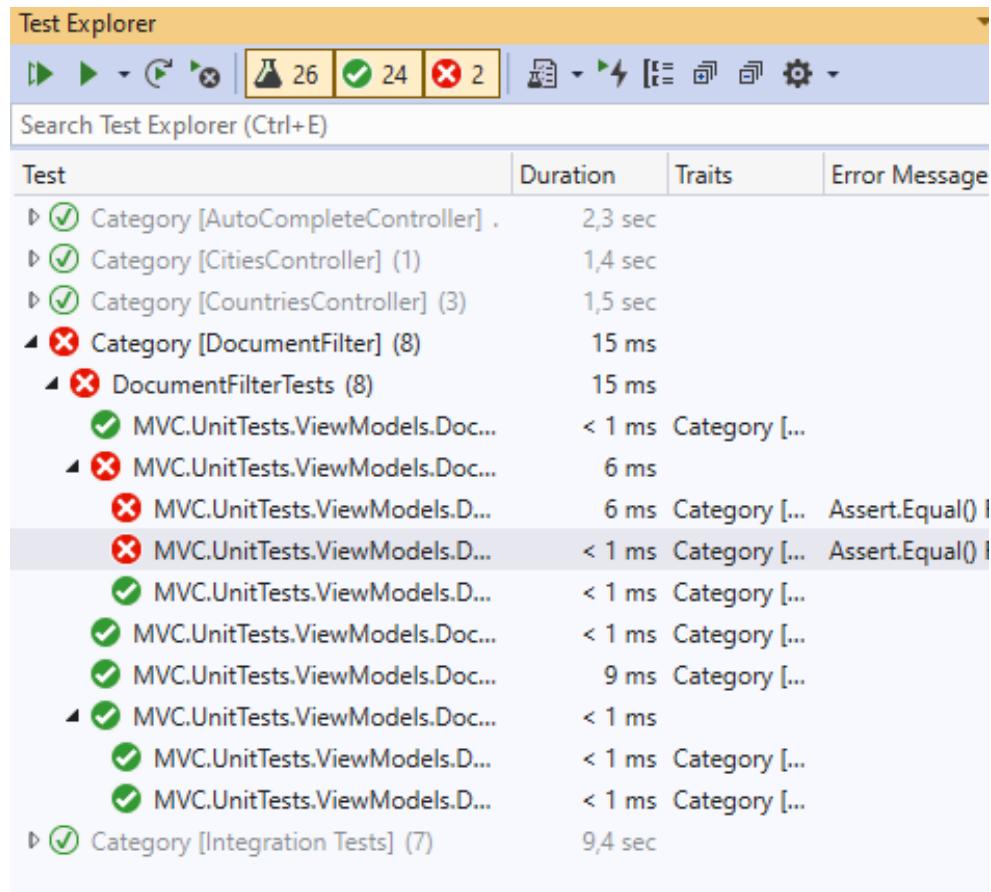
- „*Unit testing is a tool and not a religion and only you know how much testing you require*”  
Adam Freeman: Pro ASP.NET Core MVC, Sixth Edition, Apress
- „*Testing can only show the presence of bugs, not their absence*”  
Edsger W. Dijkstra
- Praktični aspekti testiranja
  - regresijsko testiranje – testiranjem se može brzo provjeriti da promjenama nije narušen prethodno ispravni kod koji je već bio testiran, tj da nije došlo do regresije
  - predstavlja jedan vid dokumentacije kojom se opisuje očekivano ponašanje sustava
  - automatizirati testiranje dijelova koje je nepraktično ili dugotrajno za ručno provjeriti
    - fokusirati se na komplikirane i važnije dijelove, a ne na trivijalnosti - pokrivenost koda testovima je mjera koja može zavarati

# Vrste testiranja korištene u primjerima

- Jedinično testiranje
  - Testiraju se dijelovi koda, tj. razredi i njihova definirana ponašanja
  - Jedinični test
    - mora biti brz (mjereno u milisekundama)
    - mora se moći pokrenuti izolirano od vanjskih ovisnosti
    - mora se moći ponoviti i davati konzistentne rezultate
- Integracijsko testiranje
  - Testiraju se (prethodno jedinično testirani) elementi, odnosno ispravnost njihove međusobne integracije
- U primjerima se ilustrira upotreba ovih tehnika za MVC aplikaciju
  - Ostale vrste testiranja, tehnike odabira reprezentativnih testnih podataka itd... izlaze van okvira ovih predavanja i nisu predmet razmatranja, što ne znači da ih ne treba uzeti u obzir.

# Radni okviri za testiranje u .NET okruženju

- *xUnit, NUnit, MSBuild*
  - <https://docs.microsoft.com/en-us/dotnet/core/testing/#testing-tools>
- Podržani kroz *TestExplorer* u *Visual Studio*
- U primjerima koji slijedi koristi se *xUnit*, ali koncepti su isti
  - <https://xunit.net/docs/comparisons>
  - <https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/>
- Redoslijed u testu:  
*Arrange – Act – Assert*



The screenshot shows the Visual Studio Test Explorer window. At the top, there are icons for running, stopping, and filtering tests, followed by counters for 26 total tests, 24 passed, and 2 failed. Below the header is a search bar labeled "Search Test Explorer (Ctrl+E)". The main area is a table with columns: "Test", "Duration", "Traits", and "Error Message". The table lists various test categories and their details:

Test	Duration	Traits	Error Message
► Category [AutoCompleteController] .	2,3 sec		
► Category [CitiesController] (1)	1,4 sec		
► Category [CountriesController] (3)	1,5 sec		
► Category [DocumentFilter] (8)	15 ms		
► DocumentFilterTests (8)	15 ms		
✓ MVC.UnitTests.ViewModels.Doc...	< 1 ms	Category [...]	
✗ MVC.UnitTests.ViewModels.Doc...	6 ms		
✗ MVC.UnitTests.ViewModels.D...	6 ms	Category [...]	Assert.Equal()
✗ MVC.UnitTests.ViewModels.D...	< 1 ms	Category [...]	Assert.Equal()
✓ MVC.UnitTests.ViewModels.D...	< 1 ms	Category [...]	
✓ MVC.UnitTests.ViewModels.D...	< 1 ms	Category [...]	
✓ MVC.UnitTests.ViewModels.D...	9 ms	Category [...]	
► MVC.UnitTests.ViewModels.Doc...	< 1 ms		
✓ MVC.UnitTests.ViewModels.D...	< 1 ms	Category [...]	
✓ MVC.UnitTests.ViewModels.D...	< 1 ms	Category [...]	
► Category [Integration Tests] (7)	9,4 sec		

**Test Detail Summary**

✗ MVC.UnitTests.ViewModels.DocumentFilterTests.FilterString\_Is\_WellFormed

- Source: [DocumentFilterTests.cs](#) line 51
- Duration: < 1 ms

Message:  
Assert.Equal() Failure  
↓ (pos 16)  
Expected: 1-15.12.2015-07.5.2021-300-500  
Actual: 1-15.12.2015-07.05.2021-300-500  
↑ (pos 16)

Stack Trace:  
[DocumentFilterTests.FilterString\\_Is\\_WellFormed\(String\\_expec...](#)

# Imenovanje testova

- Različite prakse i preporuke ovisno o izvoru
  - <https://dzone.com/articles/7-popular-unit-test-naming>
    - *MethodName\_StateUnderTest\_ExpectedBehavior*
    - *MethodName\_Scenario\_ExpectedBehavior*  
<https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices>
    - *MethodName\_ExpectedBehavior\_StateUnderTest*
    - *test[Feature being tested]*
    - *Feature to be tested*
    - *Should\_ExpectedBehavior\_When\_StateUnderTest*
    - *When\_StateUnderTest\_Expect\_ExpectedBehavior*
    - *Given\_Preconditions\_When\_StateUnderTest\_Then\_ExpectedBehavior*
  - Organizacija i imenovanje razreda i prostora imena
    - <https://ardalis.com/unit-test-naming-convention/>

# Primjer jediničnog testiranja

- Skupom testnih postupaka testira se ispravnost pretvorbe aktivnog filtra u string kod primjera s dokumentima i obrnuto
- Test je ispravan ako su sve tvrdnje zadovoljene
  - Različiti postupci iz razreda *Assert: IsTrue, Equal, Contains, IsType, ...*
  - Testni postupak označen atributom *Fact*, a atributom *Trait* može se dodatno opisati što može biti korisno kod grupiranja u prikazu svih testova
  - Primjer:  Testing \ MVC.UnitTests \ ViewModels \ DocumentFilterTests

```
[Trait("Category", "DocumentFilter")]
[Fact]
public void EmptyFilter_Returns_4_Slashes() {
    string expected = "----";
    DokumentFilter filter = new();
    string actual = filter.ToString();
    Assert.Equal(expected, actual);
}
```

# Provjera bacanja iznimke

- Ponekad očekivano ponašanje treba biti bacanje iznimke
- Umjesto pisanja try-catch blokova, koristi se Assert.Throws u kombinaciji s akcijom koja treba proizvesti iznimku određenog tipa
  - Primjer:  Testing \ MVC.UnitTests \ ViewModels \ DocumentFilterTests

[Fact]

```
public void InvalidDate_Throws_FormatException() {  
    string filterString = "-15.12.2015-15.13.2021--";  
    Assert.Throws<FormatException>(() =>  
        DokumentFilter.FromString(filterString));  
}
```

- Napomena: Test s *Throws* neće proći ako je generirana iznimka izvedena iz tražene, npr. sljedeća tvrdnja ne bi bila zadovoljena

```
Assert.Throws<Exception>(() =>  
    DokumentFilter.FromString(filterString));
```

- Za tu namjenu postoji *ThrowsAny*

# Ponavljanje istog testa s različitim podacima

- Ponekad je isti test potrebno ponoviti s različitim argumentima, pa se umjesto s [Fact] test može označiti s [Theory], a u [InlineData] navesti vrijednosti koje će se pridružiti ulaznim argumentima testnog postupka
  - Primjer:  Testing \ MVC.UnitTests \ ViewModels \ DocumentFilterTests

[Theory]

```
[InlineData("1-15.12.2015-15.12.2021-300")]
[InlineData("1-2-3-4-5-6")]
public void Not_4_Slashes_IsEmptyFilter(string filterString) {
    DokumentFilter filter = DokumentFilter.FromString(filterString);
    Assert.True(filter.IsEmpty());
}
```

- Broj argumenata može biti proizvoljan, ali se u ***InlineData*** mogu navoditi **samo konstante**.
  - string, int, double, ... ali ne i decimal, float i slično (bitno ako je ulazni argument nulabilan) – vidi testni postupak *FilterString\_Is\_WellFormed* u istom razredu

# Dodatni alati za pisanje tvrdnji

- Fluent Assertions <https://fluentassertions.com/>
  - NuGet paket *FluentAssertions*
  - Omogućava pisanje provjera koje izgledaju prirodnije ljudskom jeziku
    - `variable.Should().Be(value)`
    - `variable.Should().BeGreaterThan(value,  
because: "some reason...");`
    - `variable.Should().  
.NotBeNull()  
.And.BeOfType<List<int>>  
.Which.Count().Should().Be(value)`
  - Bit će korišteni u primjerima koji slijede

# Kreiranje instance upravljača kod testiranja (1)

- Zahtijevane ovisnosti se (uobičajeno) navode u konstruktoru pojedinog upravljača
  - U skladu s principom *Explicit Dependencies Principle*  
<https://deviq.com/principles/explicit-dependencies-principle>
  - ASP.NET Core automatski umetne potrebne ovisnosti kod instanciranja upravljača s ciljem izvršavanja akcije
  - U testovima upravljač instanciramo sami, pa moramo i sami pripremiti objekte o kojima upravljač ovisi
    - Primjer:  ...\\MVC.UnitTests\\Controllers\\AutoComplete\\Mjesto.cs

```
public async Task ReturnsCity_UsingCaseInsensitiveSubstring(
    string term, string expectedCity, string unexpectedCity) {

    using var ctx = new FirmaContext(dbContextBuilder.Options);
    var controller = new AutoCompleteController(ctx, options);
    IEnumerable<IdLabel> result = await controller.Mjesto(term)
    ...
}
```

# Kreiranje instance upravljača kod testiranja (2)

- U konstruktoru podesimo postavke za instanciranje konteksta
  - Definiran konfiguracijski objekt s postavkama za spajanje na bazu podataka
  - Primjer:  ...UnitTests\Controllers\AutoComplete\Mjesto.cs

```
public class Mjesto{  
    private DbContextOptionsBuilder<FirmaContext> dbContextBuilder;  
    public Mjesto() {  
        //Arrange  
        var builder = new ConfigurationBuilder()  
            ...  
        var configuration = builder.Build();  
        dbContextBuilder = new DbContextOptionsBuilder<FirmaContext>()  
            .UseSqlServer(configuration.GetConnectionString("Firma"));  
        ...  
    }  
}
```

# Kreiranje instance upravljača kod testiranja (3)

- Što ako trebamo samo dio objekta o kojem ovisimo? Npr. u aplikaciji koju testiramo uspostavili smo preslikavanje između dijela konfiguracijske datoteke
  - U dijelu koji testiramo, koristimo samo informaciju o max. broju rezultata
  - Možemo koristiti imitacije objekata (*Mock*) i definirati vlastito ponašanje samo pojedinih postupaka
    - NuGet paket *Moq*
  - Primjer:  ...UnitTests\Controllers\AutoComplete\Mjesto.cs

```
public class Mjesto{  
    private readonly IOptionsSnapshot<AppSettings> options;  
    public Mjesto() { ...  
        var mockOptions = new Mock<IOptionsSnapshot<AppSettings>>();  
        var appSettings = new AppSettings {  
            AutoCompleteCount = 10  
        };  
        mockOptions.SetupGet(appsettings => appsettings.Value)  
            .Returns(appSettings);  
        options = mockOptions.Object
```

# Testna baza podataka u memoriji

- Upravljač *Drzava* pri izvođenju akcije *Index* treba preusmjeriti korisnika na akciju *Create* ako u bazi podataka nema podataka
  - Praznu bazu podataka ćemo osigurati time što ćemo stvoriti EF bazu podataka u memoriji
  - Više testova može dijeliti istu bazu podataka u memoriji, pa jedinstvenost osiguramo imenom i postavljamo u testnom postupku, a ne u konstruktoru
  - Primjer:  Testing \ MVC.UnitTests \ ViewModels \ DocumentFilterTests

```
public void RedirectsToCreate_WhenNoCountries() {  
    ...  
    var dbOptions = new DbContextOptionsBuilder<FirmaContext>()  
        .UseInMemoryDatabase(  
            databaseName: nameof(RedirectsToCreate_WhenNoCountries))  
        .Options;  
    using var context = new FirmaContext(dbOptions);  
    ...
```

# Testiranje upravljača i *TempData*

- Upravljač *Drzava* pri izvođenju akcije *Index* treba preusmjeriti korisnika na akciju *Create* ako u bazi podataka nema podataka
  - Upravljač zapisuje informaciju u *TempData* koji je posljedica postojanja sjednice na serveru.
  - U jediničnom testiranju *TempData* treba zamijeniti proizvoljnom implementacijom ili imitirati
  - Primjer:  ...UnitTests\Controllers\Drzava\Index

```
public Index() {  
    var tempDataMock = new Mock<ITempDataDictionary>();  
    tempData = tempDataMock.Object;  
    ...  
  
    public void RedirectsToCreate_WhenNoCountries() {  
        ...  
        var controller = new DrzavaController(context, ...);  
        controller.TempData = tempData;
```

# Provjera tipa rezultata akcije upravljača

- Upravljač *Drzava* pri izvođenju akcije *Index* treba preusmjeriti korisnika na akciju *Create* ako u bazi podataka nema podataka
  - Provjeravamo je li rezultat akcije očekivan
  - Primjer:  ...UnitTests\Controllers\Drzava\Index

```
var controller = new DrzavaController(context, options,
                                         mockLogger.Object);
controller.TempData = tempData;

var result = controller.Index();

var redirectToActionResult =
    Assert.IsType<RedirectToActionResult>(result);
Assert.Equal("Create", redirectToActionResult.ActionName);
```

# Potvrda interakcije s imitiranim objektima

- Upravljač *Drzava* pri izvođenju akcije *Index* treba preusmjeriti korisnika na akciju *Create* ako u bazi podataka nema podataka i zapisati o tome informaciju koristeći umetnuti *ILogger*
  - Provjeramo je li prilikom izvršavanja akcija pozvana odgovarajuća metoda i to s očekivanim parametrima (konkretno *LogInformation*)
    - *LogInformation* je pokrata za poziv Log i razinom poruke *Information*
  - Primjer:  ...UnitTests\Controllers\Drzava\Index

```
public void RedirectsToCreate_WhenNoCountries() {  
    ...  
    var result = controller.Index();  
    mockLogger.Verify(l => l.Log(LogLevel.Information,  
        It.IsAny<EventId>(),  
        It.IsAny<object>(),  
        It.IsAny<Exception>(),  
        (Func<object, Exception, string>)It.IsAny<object>()),  
        Times.Once());
```

# Na upravljaču se može testirati npr. i...

- ... je li upravljač vratio ispravan model i točne podatke
  - Podatke možemo pripremiti s memorijskom bazom podataka ili nekom drugom testnom bazom podataka
  - Primjer:  ...UnitTests\Controllers\Drzava\Index - Returns\_CorrectPageData
- ... je li upravljač pripremio podatke za padajuću listu tako što ih je stavio u ViewBag/ ViewData
  - Primjer:  ...UnitTests\Controllers\Mjesto\Create - PreparesViewBag

# Integracijsko testiranje

- U prethodnim primjerima testirani su upravljači, a ovisnosti su bile eksplicitno definirane, a često i imitirane.
- Integracijskim testiranjem se provjerava rade li upravljači kad se integriraju s ostalim komponentama u web-server
  - Odazivaju li se na pravoj adresi?
  - Primaju li ispravno postavke za spajanje na bazu podataka?
  - Vraćaju li podatke u ispravnom formatu?
- U testu se simulira web-server korištenjem NuGet paketa *Microsoft.AspNetCore.Mvc.Testing*
  - Web server nastaje korištenjem objekta tipa *WebApplicationFactory<TEntryPoint>*
    - koristi se konfiguracijski razred iz testiranog projekta, pa je to *WebApplicationFactory<Startup>*

# xUnit i *IClassFixture*

- „Implementiranjem” sučelja *IClassFixture* (sučelje bez metoda) osigurava se zajednički objekt za sve testne postupke u tom razredu te je on umetnut u konstruktoru.
  - U primjeru je to primjerak `WebApplicationFactory<Startup>` koji omogućava stvaranje `HttpClient`a za pozive prema testnom serveru
  - Primjer:  ...IntegrationTests\AutoCompleteShould.cs

```
public class HomeControllerShould :  
IClassFixture<WebApplicationFactory<Startup>> {  
    private readonly WebApplicationFactory<Startup> factory;  
  
    public HomeControllerShould(  
        WebApplicationFactory<Startup> factory) {  
        this.factory = factory;  
    }  
    ... //kasnije HttpClient = factory.CreateClient();
```

# Početna provjera integriranosti komponenti

- Odaziva li se aplikacija na početnoj stranici i vraća li HTML u kojem se nalazi riječ PPP?
  - Primjer:  ...IntegrationTests\AutoCompleteShould.cs
  - Primijetiti da se ne poziva upravljač kroz kod, već otvara HTTP veza prema određenoj adresi i ispituje odgovor

```
[Theory]
[InlineData("Home/Index")]
[InlineData("")]
public async Task ServeHomePage(string url) {
    var client = factory.CreateClient();
    var response = await client.GetAsync(url);
    response.StatusCode.Should().Be(HttpStatusCode.OK);
    string content = await response.Content.ReadAsStringAsync();
    content.Should().Contain("PPP");
}
```

# Provjera serijalizacije

- Integracijsko testiranje bi npr. moglo otkriti da json koji nastane iz popisa gradova ima npr. neispravan *casing*, pa potencijalno deserijalizacija ne bi bila ispravna.
  - Primjer:  ...IntegrationTests\AutoCompleteShould.cs

```
[Theory] [InlineData("velika gor", 10410, 10380)]
public async Task ReturnCities(string input, params int[] postcodes) {
    string url = "/AutoComplete/Mjesto?term=" + input;
    var client = factory.CreateClient();
    var response = await client.GetAsync(url);
    response.StatusCode.Should().Be(HttpStatusCode.OK);
    var stream = await response.Content.ReadAsStreamAsync();
    var data = await
        JsonSerializer.DeserializeAsync<IEnumerable<IdLabel>>(stream);
    ...
    //imamo li barem 2 zapisa u data i to one koji sadrže
    // navedene nazive i poštanske brojeve
```

- Ukloniti *JsonPropertyName* iz *IdLabel*, pa pokrenuti ponovo test

# Korištenje memorijske baze podataka u integracijskom testiranju (1)

- Integracijsko testiranje koristi stvarne komponentne koje su postavljene u aplikaciji, ali moguće je neke ukloniti/promijeniti nadjačavanjem postupaka iz *WebApplicationFactory*
  - Primjer:  ...IntegrationTests\CustomWebApplicationFactory.cs

```
public class CustomWebApplicationBuilder<TTest> :  
    WebApplicationBuilder<Startup> {  
...    override void ConfigureWebHost(IWebHostBuilder builder) {  
    builder.ConfigureServices(services => {  
        var descriptor = services.SingleOrDefault(  
            d => d.ServiceType == typeof(DbContextOptions<FirmaContext>));  
        services.Remove(descriptor);  
  
        services.AddDbContext<FirmaContext>(options => {  
            options.UseInMemoryDatabase("Firma-" +  
                typeof(TTest).Name); //database per test collection  
        });  
    }};
```

# Korištenje memorijske baze podataka u integracijskom testiranju (2) ...

- ... nam omogućava da lako pripremimo situaciju kojom možemo provjeriti ispravnost pojedinih scenarija, npr. preusmjeravanja s Index na Create
- Primjer:  ...IntegrationTests\CountryControllerShould.cs

```
public class CountryControllerShould :  
IClassFixture<CustomWebApplicationFactory<CountryControllerShould>>{  
...  
[Fact]  
public async Task Redirect_When_DbContainsNoCountries() {  
    string url = "/Drzava/Index";  
    var client = factory.CreateClient(new  
        WebApplicationFactoryClientOptions {  
            AllowAutoRedirect = false  
        });  
    var response = await client.GetAsync(url);  
    response.StatusCode.Should().Be(HttpStatusCode.Redirect);  
    response.Headers.Location.Should().NotBeNull()  
        .And.Be("/Drzava/Create");  
}
```

# Napomene uz EF i memorijske baze podataka

- Treba imati na umu da se memorijska baza podataka za Entity Framework ne mora ponašati kao ostale relacijske baze podataka
    - case sensitive
    - ne podržava transakcije
    - ne podržava direktne SQL upite
- <https://docs.microsoft.com/en-us/ef/core/testing/#approach-3-the-ef-core-in-memory-database>
- Poželjno izbjegavati za integracijsko testiranje, odnosno zamijeniti s SQLite
    - <https://jimmybogard.com/avoid-in-memory-databases-for-tests/>
    - <https://docs.microsoft.com/en-us/ef/core/testing/in-memory>

# Razvoj primijenjene programske potpore

---

## 13. Web-servisi

# 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
- Pristup omogućen heterogenim klijentima
- Implementacija kao crna kutija
  - Promjena implementacije ne utječe na klijente
- Obično govorimo o web-servisima, ali servis predstavlja općenitiji pojam, ne nužno baziran na HTTP protokolu
  - Različiti protokoli, standardi, koncepti:
    - HTTP, XML, JSON, SOAP, OData, GraphQL, gRPC ...

# Remote Procedure Call

- Temelji u RPC-u (Remote Procedure Call)
  - *Remote Method Invocation* u kontekstu objektno orijentiranih jezika
- Proces na jednom računalu (klijent) poziva proceduru/metodu koja se izvršava na drugom računalu (server)
  - Ulazni parametri i povratna vrijednost prenose se mrežom.  
Postupak pakiranja u poruku prikladnu za prijenos preko mreže naziva se *marshalling*.
  - Iz perspektive programera takav udaljeni poziv se ne razlikuje od poziva lokalne procedure
    - pakiranje, slanje i primanje odradjuju namjenske biblioteke – *stubs*
- Prve konkretne implementacije u Xeroxu početkom 1980.-tih
  - doktorat Brucea Jaya Nelsona te rad zajedno s Andrewom Birellom

<https://www.cs.cmu.edu/~dga/15-712/F07/papers/birrell842.pdf>

[http://www.bitsaves.org/pdf/xerox/parc/techReports/CSL-81-9\\_Remote\\_Procedure\\_Call.pdf](http://www.bitsaves.org/pdf/xerox/parc/techReports/CSL-81-9_Remote_Procedure_Call.pdf)

# 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
    - kako izložiti servis, koristiti ponuđene servise, katalogizirati ih ...
- Distribuirani sustav u kojem sudjeluje više autonomnih servisa međusobno šaljući poruke preko granica
  - Granice mogu biti određene procesom, mrežom, ...
  - Otvoreni standardi i generičke poruke koje nisu specifične za pojedinu platformu ili pojedini jezik

# Četiri stupa servisno orijentirane arhitekture

- Jasno određene granice
  - Jasno iskazana funkcionalnost i struktura podataka, što manja to bolja
  - Implementacija je crna kutija, a cilj što lakše korištenje
- 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 i ne utječu na shemu
- Semantika, a ne samo sintaksa
  - Logička kategorizacija servisa, smisleno imenovanje postupaka

# Problemi prilikom izradu servisa

- Problem višenitnosti, skalabilnost, brzina obrade postupka
- Sigurnost komunikacije
- Pouzdanost i robusnost servisa
  - Klijent treba znati je li servis primio poruku.
  - Pogreške u servisu treba obraditi
- Konzistentnost stanja
  - Neuspjeh prilikom izvršavanja servisa ne smije ostaviti sustav u stanju pogreške
- **Interoperabilnost**
  - Tko sve može pozvati servis?

# Različiti aspekti interoperabilnosti

- Komunikacijski protokol i vrsta komunikacije
  - HTTP, HTTP/2, TCP, ...
  - Sinkrona, asinkrona, streaming, ...
- Format poruke
  - XML, JSON, binarni format, ...
- Struktura poruke
  - zaglavlja, sadržaj, poredak podataka, ...
- Preslikavanje podataka među tipovima podataka u različitim jezicima
- *Interface Definition Language (IDL)* – generički jezik (neovisan o konkretnom programskom jeziku) kojim se opisuju sučelja i strukture podataka (i preslikavanja u konkretnе jezike)
  - različite formati: CORBA, WSDL, protocol buffers, OpenAPI, ...

# 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)
    - **Unutar XML-a nalazi se informacija koji postupak web-servisa treba pozvati**
- 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

# REST - Representational State Transfer (1)

- Arhitekturni obrazac za izradu mrežnih aplikacija koji je osmislio Roy Fielding u doktorskoj disertaciji 2000.
- Općeniti koncept formalno nevezan za HTTP, ali isprepleten s nastankom HTTP/1.1 (npr. uvođenje metoda PUT, PATCH, DELETE)
  - Fielding je ujedno i koautor specifikacije za HTTP/1.1
- Postavlja nekoliko ograničenja koje sustav morao zadovoljiti, pri čemu treba istaknuti koncept jedinstvenog sučelja (engl. uniform interface), odnosno jedinstvenog identifikatora resursa
- Detaljnije na
  - <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
  - <https://codewords.recurse.com/issues/five/what-restful-actually-means>

# REST - Representational State Transfer (2)

- Poslužio kao temelj za novi koncept izrade web-servisa orientiran na resurse (engl. *resource driven*), a ne na akcije kao SOAP
  - adresa servisa jednoznačno određuje resurs
  - akcije nad resursom određene vrstom HTTP zahtjeva, npr.
    - GET – dohvati podatke
    - POST – stvaranje novog podatka
    - PUT i PATCH – izmjene cijelog ili dijela podatka
      - Možemo li dodati podatak s PUT? Ni HTML ni REST to ne određuju.
    - DELETE – brisanje podatka
  - SOA → ROA (Resource oriented architecture)?
- Istovremeno otvorio Pandorinu kutiju zloupotreba pojma REST
  - npr. usvajanje pojedinačnih elemenata REST-a, ali ne u cijelosti, uz zadržavanje pojma REST
    - Kako nazivati takve servise? WebAPI? Nije li i SOAP servis isto neka vrsta web-API-a? Više na slajdu Richardsonov model zrelosti.

# REST\* vs SOAP (OpenAPI vs SOAP)

\* = REST ili bazirano na odabranim elementima REST-a. (WebAPI?)

- Slanjem SOAP poruke na pristupnu točku provjerava se sadržaj i na osnovu sadržaja se određuje postupak koji se treba izvršiti
- Kod REST\* web-servisa 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 i manje poruke
    - POX (Plain old XML) – XML poruke bez SOAP zaglavja
    - JSON – (JavaScript Object Notation)
    - Dovoljna je podrška za HTTP - nije potrebno implementirati složene WS-\* standarde
  - Lakše cacheiranje
- SOAP koristi WSDL, REST\* servisi koriste OpenAPI (Swagger)

# WebAPI i HTTP metode

- GET: 2 metode za dohvat podataka koje vraćaju podatke ili 404
  - Dohvat svih ili podskupa podataka
  - Dohvat jednog elementa temeljem primarnog ključa (identifikatora)
- POST: Kreiranje novog podatka
  - Za uspješno stvoreni podatak vraća HTTP status 201 (te često lokaciju stvorenog resursa i sam resurs)
- PUT/PATCH: Ažuriranje cijelog (PUT) ili dijela (PATCH) podatka
  - Nakon uspješnog ažuriranja vraća HTTP status 200 ili 204
  - Nigdje nije propisano da je CREATE=POST. Može i PUT, ali nije često
- DELETE: Brisanje podatka
  - Ako je podatak uspješno obrisan vraća 200 ili 204
  - Ako je podatak već ranije bio obrisan, ili je identifikator neispravan vraća HTTP status 404
    - U kontradikciji s činjenicom da bi DELETE trebao biti idempotentan

# Kostur Web API upravljača u .NET-u

```
[ApiController] [Route("[controller]")]
public class ValuesController : BaseController {
    // GET: api/values
    [HttpGet]
    public IEnumerable<string> Get() {
        return new string[] { ... };
    }
    // GET api/values/5
    [HttpGet("{id}")]
    public string Get(int id) { ... }
    // 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) { ... }
```

# MVC – Web API : Razlike u *Startup.cs*

- Web API

```
public void ConfigureServices(IServiceCollection services) {  
    services.AddControllers() ...  
  
    public void Configure(... {  
        app.UseEndpoints(endpoints => {  
            endpoints.MapControllers();  
        });
```

- MVC

```
public void ConfigureServices(IServiceCollection services) {  
    services.AddControllersWithViews()  
  
    public void Configure(... {  
        app.UseEndpoints(endpoints => {  
            endpoints.MapDefaultControllerRoute();  
        });
```

# Primjer Web API servisa – Web API za mjesta

- Definira se slično kao i upravljači u MVC-u, ali ima definirano vlastito usmjeravanje, ne vraća pogled već podatke i/ili statusni kod
  - Nasljeđuje ControllerBase
- Označen atributom *ApiController*
  - **Automatska provjera valjanost modela. Ako model nije valjan rezultat je status 400 - BadRequest**
- Ovisnosti o sučeljima i razredima u konstruktoru kao i kod MVC-a
  - Primjer:  WebServices \ ... WebAPI \ Controllers \ MjestoController.cs

```
[ApiController]
[Route("[controller]")]
public class MjestoController : ControllerBase {
    public MjestoController(FirmaContext ctx) {
        this.ctx = ctx;
    }
    ...
}
```

# Web API – dohvati svih mesta (1)

- Naziv postupka nebitan – postupak se određuje prema atributu `HttpGet` i usmjeravanju upravljača
  - Konkretno u ovom primjeru `/mjesto`
- Klijentu isporučiti prezentacijski model neovisno o sličnosti s modelom iz EF-a
  - Omogućava naknadno neovisnu izmjenu podatkovnog sloja
- Primjer:  ... \ WebApi \ Controllers \ MjestoController.cs
  - Napomena: EF model u zasebnom projektu, jer se koristi i u drugim primjerima

```
[HttpGet(Name = "DohvatiMjesta")]
public async Task<List<MjestoViewModel>> GetAll(...) {
    var query = ctx.Mjesto.AsQueryable(); // .AsNoTracking()?
    ...
    var list = await query.Select(m => new MjestoViewModel {
        ...
    })
    ...
    .ToListAsync();
    return list;
}
```

# Web API – dohvati svih mesta (2)

- U slučaju velikog broja mesta poželjno definirati parametre za dohvat podskupa elemenata
  - Parametri nisu standardizirani, već često ovise o ciljanom klijentu

```
public ... GetAll([FromQuery] LoadParams loadParams)
```

- Primjer:  ...WebApi\ViewModels\LoadParams.cs
- U konkretnom primjeru parametri prilagođeni za jTable

```
public class LoadParams {  
    [FromQuery(Name = "jtStartIndex")]  
    public int StartIndex { get; set; }  
    [FromQuery(Name = "jtPageSize")]  
    public int Rows { get; set; }  
    [FromQuery(Name = "jtSorting")]  
    public string Sort { get; set; }  
    ...  
}
```

# Izvori podataka za argumente Web API servisa

- Ispred tipa i naziva argumenta postupka Web API servisa može se navesti neki od atributa
  - [FromBody], [FromForm], [FromHeader], [FromQuery], [FromRoute], [FromServices]
- Za upravljače označene s [ApiController] ne vrijede pravila, tj. redoslijed kao kod upravljača u MVC-u. Ako nije naveden niti jedan, onda se koriste sljedeća pravila
  - jednostavnii tipovi (int, string, ...) trebaju biti dio *query stringa*
  - složeni tipovi se nalaze u tijelu zahtjeva
  - Datoteke su dio podataka forme
- **Postupak može imati samo jedan složeni podatak čiji su podaci iz tijela zahtjeva.**
- Detaljnije na <https://docs.microsoft.com/en-us/aspnet/core/web-api/#binding-source-parameter-inference>

# Web API – dohvati svih mjesta (3)

- Otvaranjem adrese

<https://.../mjesto?jtStartIndex=0&jtPageSize=10&jtSorting=NazivMjesta> dobit će se JSON sadržaj s popisom prvih 10 mjesta poredanih po nazivu mjesta

- Popis serijaliziran u JSON
- Konfiguracijskom datotekom može se postaviti i drugi format (npr. XML)



```
{"IdMjesta":1,"PostBrojMjesta":3121,"NazivMjesta":"Ada","PostNazivMjesta":"Laslovo","OznDrzave":"HR","NazivDrzave":"Croatia"
{"IdMjesta":8569,"PostBrojMjesta":24430,"NazivMjesta":"ADA","PostNazivMjesta":null,"OznDrzave":"CS","NazivDrzave":"Serbia an
 {"IdMjesta":2,"PostBrojMjesta":10363,"NazivMjesta":"Adamovec","PostNazivMjesta":"BeloVAR","OznDrzave":"HR","NazivDrzave":"Cr
 {"IdMjesta":8404,"PostBrojMjesta":22244,"NazivMjesta":"ADAŠEVCI","PostNazivMjesta":null,"OznDrzave":"CS","NazivDrzave":"Serb
 Montenegro"}, {"IdMjesta":8375,"PostBrojMjesta":21251,"NazivMjesta":"ADICE","PostNazivMjesta":null,"OznDrzave":"CS","NazivDrz
 Montenegro"}, {"IdMjesta":7856,"PostBrojMjesta":8341,"NazivMjesta":"ADLEŠIĆ","PostNazivMjesta":null,"OznDrzave":"SI","NazivDrzave":
```

- Ne odredi li se drugačije (serijalizacijskim atributima ili postavkama u Startup.cs), koristit će se camelCase
  - Primjer: ... WebApi \ Startup.cs koristi PascalCase

```
services.AddControllers()
    .AddJsonOptions(configure =>
        configure.JsonSerializerOptions.PropertyNamingPolicy = null);
```

# Web API – dohvati određenog mjesta

- Putanja oblika /mjesto/idmesta (npr. /mjesto/123)
  - Usmjeravanje imenovano posebnim nazivom kao bi se moglo naknadno referencirati (vidi primjer za stvaranje novog mjesta)
- Povratna vrijednost može biti konkretno mjesto ili status 404
  - Rezultat je ActionResult<MjestoViewModel>
  - Primjer:  ... WebApi \ Controllers \ MjestoController.cs

```
[HttpGet("{id}", Name = "DohvatiMjesto")]
public async Task<ActionResult<MjestoViewModel>> Get(int id) {
    var mjesto = await ctx.Mjesto.Where(m => m.IdMesta == id)
        .Select(m => new MjestoViewModel { ... })
        .FirstOrDefaultAsync();

    if (mjesto == null)
        return Problem(statusCode: StatusCodes.Status404NotFound,
                      detail: $"No data for id = {id}");
    else
        return mjesto; ...
```

# ActionResult i ProblemDetails

- U prethodnom primjeru postupak je mogao vratiti
  - konkretno mjesto (objekt tipa *MjestoViewModel*) – posljedično status 200
  - Status + poruku da mjesto ne postoji
- U takvim slučajevima koristi se povratni tip *ActionResult<T>*
  - Definira implicitnu konverziju iz T u *ActionResult<T>*
- Poruke o pogrešci bi trebale biti u skladu s RFC 7807
  - <https://tools.ietf.org/html/rfc7807>
- U prethodnom primjeru može se koristiti
  - *NotFound* - obratiti pažnju da treba koristiti varijantu bez argumenata)
  - *Problem* koji vraća *ProblemDetails* u skladu s RFC 7807

# Web API – dodavanje mjesta

- Postupak POST proizvoljnog imena
  - U slučaju uspjeha vraća se status 201, podatak i njegova adresa
    - Adresa je dio zaglavlja odgovora, podatak je u tijelu
  - Neispravni model uzrokuje statusnu poruku 400 (BadRequest)
  - Primjer:  ... WebApi \ Controllers \ MjestoController.cs

```
[HttpPost(Name = "DodajMjesto")]
public async Task<IActionResult> Create(MjestoViewModel model) {
    Mjesto mjesto = new Mjesto {
        NazMjesta = model.NazivMjesta, OznDrzave = model.OznDrzave,
        PostBrMjesta = model.PostBrojMjesta,
        PostNazMjesta = model.PostNazivMjesta
    };
    ctx.Add(mjesto);
    await ctx.SaveChangesAsync();
    var addedItem = await Get(mjesto.IdMjesta);
    return CreatedAtAction(nameof(Get), new { id = mjesto.IdMjesta },
        addedItem.Value);
}
```

# Kako isprobati POST i ostale zahtjeve? (1)

- Postman – API klijent/alat (<https://www.postman.com/downloads>)
  - Alternativa Fiddler ili slični alati, vlastita konzolna aplikacija, generirani klijent preko Swaggera (o tome malo kasnije)
  - Postaviti Content-Type na application/json i poslati odgovarajući JSON

The screenshot shows the Postman application interface with the following details:

- Request Method:** POST
- Request URL:** https://localhost:44385/mjesto
- Headers (1):** Content-Type: application/json
- Body:** Raw JSON content:

```
1 {  
2     "PostBrojMjesta": 123, "NazivMjesta" : "probno mjesto",  
3     "OznDrzave" : "HR", "PostNazivMjesta" : "probno mjesto"  
4 }
```

# Kako isprobati POST i ostale zahtjeve? (2)

Body    Cookies    Headers (6)    Test Results    Status: 201 Created

Pretty    Raw    Preview    JSON ▾   

```
1 {  
2     "IdMjesta": 36460,  
3     "PostBrojMjesta": 123,  
4     "NazivMjesta": "probno mjesto",  
5     "PostNazivMjesta": "probno mjesto",  
6     "OznDrzave": "HR",  
7     "NazivDrzave": "Croatia"  
8 }
```

Body    Cookies    **Headers (6)**    Test Resi

**Content-Type** → application/json; charset=utf-8

**Date** → Mon, 11 Jan 2021 10:26:36 GMT

**Location** → <https://localhost:44385/Mjesto/36460>

# Kako isprobati POST i ostale zahtjeve? (3)

- U slučaju validacijske pogreške, odgovor sadrži poruku o pogrešci i statusni kod 400 (*Bad Request*)
  - Posljedica validacijskih atributa (ili *FluentValidation*a) te atributa [ApiController] na upravljaču

Body Cookies Headers (5) Test Results Status: 400 Bad Request

Pretty Raw Preview JSON ↴

```
1 {  
2     "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  
3     "title": "One or more validation errors occurred.",  
4     "status": 400,  
5     "traceId": "|91f38987-4e468597ec15583c.",  
6     "errors": {  
7         "PostBrojMjesta": [  
8             "Dozvoljeni raspon: 10-60000"  
9     ]  
1 }
```

# Web API – ažuriranje mjesta

- Postupak PUT proizvoljnog imena
  - U primjeru se šalje cijeli model, a ne samo parcijalne promjene (PATCH)
  - Identifikator mjesta iz zahtjeva i onaj iz modela moraju biti jednaki
  - Uspješna promjena podatka vraća statusnu poruku 204
  - Neispravni podaci: 400 ili 404 (ako mjesto ne postoji)
  - Primjer:  ... WebApi \ Controllers \ MjestoController.cs

```
[HttpPut("{id}")]
public async Task<IActionResult> Update(int id,
                                         MjestoViewModel model) {
    var mjesto = await ctx.Mjesto.FindAsync(id);
    if (mjesto == null)
        return Problem(statusCode: StatusCodes.Status404NotFound,
                      detail: $"Invalid id = {id}");
    mjesto.NazMjesta = model.NazivMjesta;
    mjesto.OznDrzave = model.OznDrzave; ... //ostala pridruživanja
    await ctx.SaveChangesAsync();
    return NoContent();
```

# Web API – brisanje mjesta

- Postupak DELETE proizvoljnog imena sa šifrom mjesta u adresi
- Uspješno brisanje vraća statusnu poruku 204, a za nepostojeće mjesto vraća se 404
- Primjer:  ... WebServices \ Controllers \ MjestoController.cs

```
[HttpDelete("{id}", Name = "Obrisimjesto")]
public async Task<IActionResult> Delete(int id) {
    var mjesto = await ctx.Mjesto.FindAsync(id);
    if (mjesto == null)
        return Problem(statusCode: StatusCodes.Status404NotFound,
                      detail: $"Invalid id = {id}");
    else {
        ctx.Remove(mjesto);
        await ctx.SaveChangesAsync();
        return NoContent();
    }
}
```

# Primjer poziva WebAPI-a iz konzolne aplikacije (1)

- Primjer:  ...WebApi \ ConsoleClientApps \ ConsoleClient \ Program.cs
  - Razred *HttpClient* za interakciju s web stranicama/servisima
  - Paket Microsoft.AspNet.WebApi.Client za proširenje *ReadAsAsync<T>*
  - Razred *Mjesto* definiran da po strukturi odgovara isporučenom Jsonu (nastalom iz razreda *MjestoViewModel*)

```
private static async Task DohvatiMjesto(int id) {
    using (var client = new HttpClient()) {
        var response = await client.GetAsync($"{url}/{id}");
        if (response.IsSuccessStatusCode) {
            var mjesto = await response.Content.ReadAsAsync<Mjesto>();
            Console.WriteLine($"Mjesto s id-om {id} je : ");
            Console.WriteLine(mjesto.ToString());
        }
        else
            Console.WriteLine($"Neuspješan dohvat mjesta {id}");
    }
}
```



**Microsoft.AspNet.WebApi.Client** by Microsoft v5.2.7

This package adds support for formatting and content negotiation to System.Net.Http.

# Primjer poziva WebAPI-a iz konzolne aplikacije (2)

- Primjer:  WebApi \ ... \ ConsoleClient \Program.cs : DodajMjesto
  - Potrebno postaviti tip sadržaja na application/json
  - Poziva se postupak *PostAsJsonAsync*
    - Slično u primjerima koji slijede PutAsJsonAsync, DeleteAsync, ...

```
using (var client = new HttpClient()) {
    var response = await client.PostAsJsonAsync<Mjesto>(url, mjesto);
    if (response.IsSuccessStatusCode) {
        Console.WriteLine($"Mjesto dodano i može se dohvatiti na adresi
                           {response.Headers.Location}");
        mjesto = await response.Content
                    .ReadAsStringAsync<ContentResultValue<Mjesto>>();
        Console.WriteLine(mjesto);
        return mjesto.IdMjesta;
    }
    else {
        string content = await response.Content.ReadAsStringAsync();
        // response.StatusCode, response.ReasonPhrase, ...
    }
}
```

# 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
  - Nešto treće?

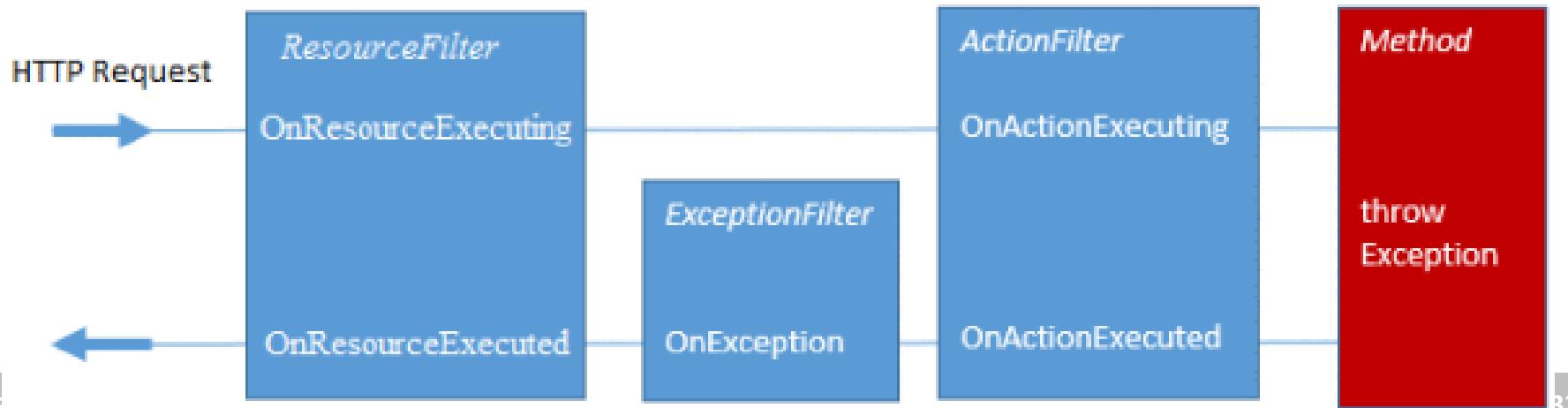
# Primjer korištenja servisa koristeći jTable

- Proučiti sljedeće sadržaje
  -  WebServices \ ... \ WebApi \ wwwroot \ mjesta.html  
<https://localhost:44385/mjesta.html>
  -  WebServices \ ... \ WebApi \ Controllers \ jTable \ JTableController.cs
  -  WebServices \ ... \ WebApi \ Controllers \ jTable \ MjestoJTableController.cs
  -  WebServices \ ... \ WebApi \ Controllers \ jTable \ LookupController.cs
  -  WebServices \ ... \ WebApi \ ViewModels \ jTable \ \*
- jTable ne radi s WebAPI servisima direktno, već koristi POST, pa je potreban *wrapper*.
  - jTable koristi koncept da je poziv uvijek uspješan (status 200) uz poruku je li akcija uspjela ili ne
    - U slučaju validacijske pogreške *wrapper* ne vraća 400, već 200 uz (primjerice) sadržaj

```
{ "Result": "ERROR",      "Message":  
          "PostBrojMjesta: Dozvoljeni raspon: 10-60000; "}
```

# Obrada iznimke korištenjem atributa

- Umjesto napornog pisanja try-catch blokova u svakom postupku (a i za nepredviđene iznimke), obrada iznimke se centralizira posebnim atributima
  - Vlastiti atribut izveden iz `ExceptionFilterAttribute` ili implementacijom `IExceptionFilter`
  - Zbog DI-a ne navodi se direktno, nego koristeći atribut `TypeFilter`, npr. `[TypeFilter(typeof(ErrorStatusTo200WithErrorMessage))]`
  - Primjeri:  ... WebApi \ Controllers \ MjestoController.cs  
 ... WebApi \ Controllers \ Jtable \ JTableController.cs  
 ... WebApi \ Util \ ExceptionFilters \ ProblemDetailsForSqlException.cs  
 ... ExceptionFilters \ ErrorStatusTo200WithErrorMessage.cs
- Ako se više atributa primjeni na upravljač, bitan je njihov poredak (ili redom, ili eksplicitno naveden). Za obradu iznimke poredak je obrnut.



# Dokumentacija za web servise - OpenAPI

- Za dokumentiranje Web API servisa koristi se alat Swagger
- Osim informacija o tipovima podataka i rezultata omogućava i pozivanje servisa

Swagger  
Supported by SMARTBEAR

Select a definition RPPP Firma WebAPI

## WebServices 1.0 OAS3

/swagger/v1/swagger.json

### Mjesto

**GET** /Mjesto/count Vraća broj svih mesta filtriran prema nazivu mesta

**GET** /Mjesto Dohvat mjesta (opcionalno filtrirano po nazivu mjesta). Broj mjesta, poredak, početna pozicija određeni s loadPa

**POST** /Mjesto Stvara novo mjesto opisom poslanim modelom

**GET** /Mjesto/{id} Vraća grad čiji je IdMjesta jednak vrijednosti parametra id

**DELETE** /Mjesto/{id} Brisanje mesta određenog s id

**PUT** /Mjesto/{id} Ažurira mjesto

# Aktivacija Swaggera (1)

- Dodati NuGet paket Swashbuckle.AspNetCore
- U Startup.cs aktivirati Swagger
  - Swagger automatski pronađi sve upravljače i atribute Http[Get|Post|...]
    - S ApiExplorerSettings(IgnoreApi = true) moguće izbaciti željene upravljače
  - U postavkama projekta uključiti kreiranje XML dokumentacije
  - Navesti putanje do svih xml datoteka koje treba uključiti
  - Primjer  ... WebApi \ Startup.cs

```
public void ConfigureServices(...  
{  
    ...  
    services.AddSwaggerGen(c => {  
        var xmlFile = $"{Assembly.GetExecutingAssembly()  
                           .GetName().Name}.xml";  
        var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);  
        c.IncludeXmlComments(xmlPath);  
        ...  
    });  
}
```

# Aktivacija Swaggera (2)

- Primjer  ... WebApi \ Startup.cs

```
public void Configure(...  
...  
    app.UseSwagger();  
    app.UseSwaggerUI(c => {  
        c.SwaggerEndpoint("/swagger/v1/swagger.json",  
                         "RPPP Firma WebAPI");  
        c.RoutePrefix = "docs";  
    });  
...
```

- Nakon navedenih postavki automatski generirana dokumentacija za WebApi servise je dostupna na <http://.../route-prefix/>
  - Npr. <https://localhost:44377/docs>

# Informacije o statusnim porukama

- Ako upravljači nisu označeni s [APIConventions] Swagger prepostavlja da svi postupci vraćaju status 200
- U slučaju da nije tako, potrebno eksplicitno navesti moguće vrijednosti iznad postupka
  - Koristi se atribut *ProducesResponseType*
  - Primjer  ... \ WebApi \ Controller \ MjestoController.cs

```
[HttpPut("{id}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> Update(
    int id, [FromBody] MjestoViewModel model)
```

- Napomena: Ovo ne znači da se ne može pojaviti neki drugi status (npr. 500 – Internal Server Error), već da se takav rezultat ne bi trebao pojaviti.

# Informacije o povratnim porukama

- Swagger određuje povratne tipove na temelju potpisa metode, npr. ako je to `ActionResult<povratni_tip>`, kolekcija, ...
- `IAsyncResult` je općeniti rezultat, pa *Swagger* nema automatski tu informaciju. U tom slučaju, koriste se atributi `ProducesResponseType` i `SwaggerResponseAttribute` (*izvedeni razred iz ProducesResponseType s mogućnošću dodavanja opisa*)

```
[[HttpGet("{oznDrzave}", Name = "DohvatiDrzavu")]
[ProducesResponseType(typeof(Drzava),
                (int) HttpStatusCode.OK)]
[ProducesResponseType((int) HttpStatusCode.NotFound)]
public async Task<IActionResult> Get(string oznDrzave)
...
[HttpGet]
[SwaggerResponseAttribute((int) HttpStatusCode.OK,
                           typeof(IEnumerable<Drzava>),
                           Description = "Vraća enumeraciju država")]
public async Task<IEnumerable<DrzavaApiModel>> Get()
```

# Generiranje klijenta na osnovi Swagger dokumentacije (1)

- Na temelju Swaggerove json datoteke i alata NSwag moguće generirati klijente razrede (kroz Visual Studio ili samostalno)
  - Desni klik na projekt -> Add Service Reference -> OpenAPI
    - Lokalno kopira datoteke swagger.json
    - Generira swaggerClient.cs u podmapi *obj* s kodom koji sadrži
      - metode za poziv servisa
      - parcijalne razrede za podatke koji služe kao ulazne i izlazne vrijednosti servisa (npr. *MjestoViewModel*)
- 
- Add new OpenAPI service reference  
Select a file or URL  
 File  
 URL  
https://localhost:44377/swagger/v1/swagger.json|  
Provide the namespace for the generated code  
OpenAPIClients  
Provide the class name for the generated code  
RPPPClient  
Code generation language  
C#

# 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  ... \ ConsoleClientApps \ StronglyTypedClient \ Program.cs

```
using(HttpClient client = new HttpClient()) {  
    var apiClient = new RPPPClient(url, client);  
    await apiClient.DodajMjestoAsync(model);
```

- Primjer  ... WebApi \ Controllers \ MjestoController.cs

```
[HttpPost(Name = "DodajMjesto")]  
public async Task<IActionResult> Create(MjestoViewModel model))
```

# Richardsonov model zrelosti

- 4 nivoa zrelosti Web API-a prema ograničenjima REST-a
  - Leonard Richardson 2008.
  - <https://martinfowler.com/articles/richardsonMaturityModel.html>
- Nivo 0: RPC/RPI (Remote Procedure Call/Invocation) preko HTTP-a
- Nivo 1: Izlaganje resursa umjesto metoda
  - interakcija vezana za konkretni resurs (URL identificira resurs)
- Nivo 2: Pravilna upotreba vrsta HTTP zahtjeva, statusa i sadržaja odgovara (npr. lokacija dodanog resursa)
  - Napomena: primjer s mjestima (ali ne i wrapper za jTable) pripada ovom nivou
- Nivo 3: Odgovor sadrži hipermedijske elemente (npr. poveznice na ostale resurse i akcije koje se mogu napraviti s tim resursom)
  - Hypermedia as the Engine of Application State (HATEOAS)
  - omogućavaju samostalno otkrivanje ostalih resursa/mogućnost servisa
- Kreator REST-a, Roy Fielding smatra da je nivo 3 preuvjet da bi se nešto nazivalo REST servisom
  - <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

# 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
  - Nudi informacije o entitetima s kojima pojedini servis radi
  - Podrška za različite programske jezike na serverskoj i klijentskoj strani
- Za detalje o standardu pogledati na <http://www.odata.org>

# Primjeri poziva po OData standardu (1)

- Metapodaci servisa:
  - [https://localhost:44343/odata/\\$metadata](https://localhost:44343/odata/$metadata)
- Dohvat prvih 13 mjesta
  - [https://localhost:44343/odata/Mjesto?\\$top=13&\\$count=true](https://localhost:44343/odata/Mjesto?$top=13&$count=true)
  - [https://localhost:44343/odata/Mjesto?\\$top=13&\\$count=true&\\$format=application/json;odata.metadata=full](https://localhost:44343/odata/Mjesto?$top=13&$count=true&$format=application/json;odata.metadata=full) (promotriti rezultat)
- Dohvat prvih 10 mjesta u Hrvatskoj koji u svom nazivu sadrže riječ Velika, poredanih po nazivu mjesta pri čemu nas u rezultatu zanima samo poštanski broj i naziv mjesta
  - [https://localhost:44343/odata/Mjesto?\\$top=10&\\$count=true&\\$filter=contains\(NazivMjesta,'Velika'\)%20and%20NazivDrzave%20eq%20'Croatia'&\\$orderby=PostBrojMjesta&\\$select=PostBrojMjesta,NazivMjesta](https://localhost:44343/odata/Mjesto?$top=10&$count=true&$filter=contains(NazivMjesta,'Velika')%20and%20NazivDrzave%20eq%20'Croatia'&$orderby=PostBrojMjesta&$select=PostBrojMjesta,NazivMjesta)
  - Za varijantu s linkovima u rezultatu dodati na kraju *&\$format=application/json;odata.metadata=full*
- Podaci se pripremaju na serveru

# Izrada REST servisa prema standardu OData (1)

- Struktura projekta kao i u primjeru za WebApi
  - Dodati potrebne ovisnosti za Dependency Injection
  - Definirati rutu (tj. prefiks) za Odata i definiciju modela
    - Potreban NuGet paket *Microsoft.AspNetCore.OData*
  - Primjer  ... \ ODataApi \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {  
    ...  
    services.AddDbContext<EFModel.FirmaContext>(options => ...  
  
    services.AddControllers()  
        .AddOData(opt => opt.AddRouteComponents("odata",  
            FirmaODataModelBuilder.GetEdmModel()));
```

- Definicija modela u vlastitom postupku *GetEdmModel*

# Izrada REST servisa prema standardu OData (2)

- Definirati entitet, adresu na kojoj je dostupan i dozvoljene postupke
  - Primjer  ... \ ODataApi \FirmaODataModelBuilder.cs

```
using DTOs = ODataApi.Contract;  
...  
public class FirmaODataModelBuilder {  
    public IEdmModel GetEdmModel() {  
        var builder = new ODataConventionModelBuilder();  
        builder.EntitySet<DTOs.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;
```

- U ovom slučaju koristi se Mjesto iz zasebnog projekta, a ne iz EF-a

# Izrada REST servisa prema standardu OData (3)

- Upravljač OData web-servisa izведен iz razred ODataController
- Postupci slični kao kod u prethodnom primjeru uz dodatak
  - Patch za promjenu dijela podatka prima objekt tipa Delta<T>
  - Get sadrži atribut *EnableQuery* za pripremu rezultata ovisno o zahtjevu
    - **Metoda u tom slučaju mora vratiti IQueryable**
  - Primjer  ODataAPI \ Controllers \ MjestoController.cs

```
public class MjestoController : ODataController {  
    ...  
    [EnableQuery]  
    public IQueryable<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)
```

# OData, EnableQuery i IQueryable

- *EnableQuery* zahtjeva *IQueryable* kao rezultat
- Iako je korišten zasebni razred za model i dalje ovisimo o Entity Framework
  - Samostalna implementacija sučelja *IQueryable* je prilično komplikirana i može biti problematična
- Može se implementirati i bez atributa *EnableQuery*, ali je onda potrebno dodati argument tipa *ODataQueryOptions* i samostalno obraditi parametre
- U nastavku će za povratne vrijednosti biti korišteni razredi iz EF modela kako bi se demonstrirale ostale mogućnosti za OData
  - Daleko od idealnog rješenja, jer narušava ideju servisa kao crne kutije

# Izrada REST servisa prema standardu OData (4)

- Moguće definirati i veze među pojedinim entitetima
  - npr. definira se mogućnost uključivanja stavki uz pojedini dokument
  - Primjer  ... ODataAPI \ FirmaODataModelBuilder.cs

```
public class FirmaODataModelBuilder {  
    public IEdmModel GetEdmModel() {  
        var builder = new ODataConventionModelBuilder();  
        ...  
        builder.EntitySet<ODataDTOs.Dokument>("Dokument")  
            .EntityType  
            .Filter() // Allow for the $filter Command  
        ...  
            .Select() // Allow for the $select Command  
            .HasMany(x => x.Stavke)  
            .Expand();  
        builder.EntityType<EFModel.Stavka>().HasKey(s => s.IdStavke);  
    }  
}
```

- Svaki entitet mora imati atribut [Key] ili eksplicitno naveden ključ prilikom definicije

# Izrada REST servisa prema standardu OData (4)

- Uz korištenje EF-a implementacija dohvata je trivijalna
  - Primjer  ... ODataAPI \ DokumentController.cs

```
[EnableQuery(PageSize = 50)]
public IQueryable<Dokument> Get()      {
    return ctx.Dokument.AsNoTracking();
}

[EnableQuery]
public async Task<IActionResult> Get(int key) {
    var query = ctx.Dokument.AsNoTracking()
        .Where(d => d.IdDokumenta == key);
    if (await query.AnyAsync())
        return Ok(query);
    else
        return NotFound("Traženi dokument ne postoji");
}
```

# Primjeri poziva po OData standardu (2)

- Želimo prva 3 dokumenta starija od 1.1.2016. osobe koja u prezimenu sadrži riječ *Stipan* s iznosom manjim od 10 000, sortirano silazno po iznosu, pri čemu u rezultat želimo uključiti ime i prezime osobe
  - [https://localhost:44343/odata/Dokument?\\$filter=startswith%28IdPartneraNavigation%2FOsoba%2FPrezimeOsobe%2C%27Stipan%27%29%20and%20DatDokumenta%20lt%20202016-01-01T00%3A00%3A00%2B01%3A00%20and%20IznosDokumenta%20gt%2010000&\\$top=3&\\$expand=IdPartneraNavigation\(\\$expand=Osoba\(\\$select=ImeOsobe,PrezimeOsobe\)\)&\\$select=IdDokumenta,DatDokumenta,IznosDokumenta&\\$orderby=IznosDokumenta%20desc](https://localhost:44343/odata/Dokument?$filter=startswith%28IdPartneraNavigation%2FOsoba%2FPrezimeOsobe%2C%27Stipan%27%29%20and%20DatDokumenta%20lt%20202016-01-01T00%3A00%3A00%2B01%3A00%20and%20IznosDokumenta%20gt%2010000&$top=3&$expand=IdPartneraNavigation($expand=Osoba($select=ImeOsobe,PrezimeOsobe))&$select=IdDokumenta,DatDokumenta,IznosDokumenta&$orderby=IznosDokumenta%20desc)
- Ako uz svaki od tih dokumenata želimo uključiti i podatke o stavkama, možemo upotrijebiti \$expand=Stavke
  - [https://localhost:44343/odata/Dokument?\\$filter=startswith%28IdPartneraNavigation%2FOsoba%2FPrezimeOsobe%2C%27Stipan%27%29%20and%20DatDokumenta%20lt%20202016-01-01T00%3A00%3A00%2B01%3A00%20and%20IznosDokumenta%20gt%2010000&\\$top=3&\\$expand=Stavka&\\$select=IdDokumenta,DatDokumenta&\\$orderby=IznosDokumenta%20desc](https://localhost:44343/odata/Dokument?$filter=startswith%28IdPartneraNavigation%2FOsoba%2FPrezimeOsobe%2C%27Stipan%27%29%20and%20DatDokumenta%20lt%20202016-01-01T00%3A00%3A00%2B01%3A00%20and%20IznosDokumenta%20gt%2010000&$top=3&$expand=Stavka&$select=IdDokumenta,DatDokumenta&$orderby=IznosDokumenta%20desc)

# Pozivanje OData servisa iz konzolne aplikacije (1)

- Korištenjem nekih od standardnih klijenata
  - <https://www.odata.org/getting-started/understand-odata-in-6-steps/>
  - U primjeru korišten NuGet paket Simple.OData.V4.Client
- Primjer  OData \ ConsoleODataClient \ Program.cs

```
var settings = new ODataClientSettings(new Uri(url));
var client = new ODataClient(settings);
...
var mjesta = await client.For<Mjesto>()
    .Filter(m => m.OznDrzave == "HR")
    .Filter(m => m.PostBrojMjesta >= 10000
                && m.PostBrojMjesta < 10100)
    .Filter(m => !m.NazivMjesta.Contains("Zagreb"))
    .Select(m => new { m.PostBrojMjesta, m.NazivMjesta })
    .OrderBy(m => m.PostBrojMjesta)
    .FindEntriesAsync();
```

- Na sličan način možemo raditi upite i za dokumente (postupak DemoDokumenti)

# Pozivanje OData servisa iz konzolne aplikacije (2)

- Primjer dodavanja novog mjesta

📁 ...ODataApi \ ConsoleODataClient \ Program.cs

```
...
var novoMjesto = new Mjesto {
    PostBrojMjesta = 59999, OznDrzave = "HR",
    NazivMjesta = "Test mjesto", PostNazivMjesta = "Test mjesto"
};
var mjesto = await client.For<Mjesto>()
    .Set(novoMjesto)
    .InsertEntryAsync();
```

- Proučiti i ostatak primjera (ažuriranje, brisanje, složeni upit za dokumente, ...)

# Generalni problemi REST servisa

- Prekomjerno preuzimanje (engl. *overfetching*)
  - skup povratnih vrijednosti određen je prezentacijskim modelom
  - rezultat možda sadrži i više podataka nego što klijentu treba
    - može uzrokovati nepotrebne join upite i povećati veličinu rezultata
- Nedovoljno preuzimanje (engl. *underfetching*)
  - FT1P („fali ti jedan podatak“)
  - npr. ako rezultat s mjestima ne sadrži podatak u nazivu države, to će možda uzrokovati još jedan poziv za popis država
  - generalno vodi ka n+1 problemu
    - Zamislimo upit u kojem trebamo države i sva njihova mjesta. Upit za popis država + n upita za mjesta u pojedinoj državi
      - Zaobilazno rješenje je raditi upit s mjestima uz uključenu državu
      - Što s upitim na još jednu razinu niže
- OData pokušava riješiti i jedno i drugo, ali je komplikirano u odnosu na neke druge tehnike – npr. GraphQL

# GraphQL

- <https://graphql.org/>
  - Facebook 2012., javno objavljen 2015. godine
  - Podrška za različite programske jezike
    - <https://graphql.org/code>
- Sasvim drugačiji koncept u odnosu na REST s primarnim ciljem da se izbjegne prekomjerno ili nedovoljno preuzimanje
- GraphQL server izlaže (jednu) pristupnu točku kojoj se može poslati jedan ili više
  - upita (engl. *query*)
  - zahtjeva za promjenom (engl. *mutation*)
  - zahtjeva za pretplatom (engl. *subscription*)
  - POST zahtjevi + JSON, iako ovisno o postavkama može biti i GET pri čemu *query stringu* ima ključ *query*, *mutation* ili *subscription*

# Primjer implementacije GraphQL servera (1)

- Korištenjem EF-a i paketa *HotChocolate* omogućava brzu i jednostavnu implementaciju
- Primjer  GraphQL \ GraphQLServer \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
    services.AddDbContext<FirmaContext>(options => ...);
    services.AddGraphQLServer()
        .SetPagingOptions(new PagingOptions
        {
            MaxPageSize = 1000, IncludeTotalCount = true, ...
        })
        .AddProjections()
        .AddFiltering()
        .AddSorting()
        .AddQueryType<Queries>()
        .AddMutationType<Mutations>();
}
```

# Primjer implementacije GraphQL servera (2)

- Korištenjem EF-a i paketa *HotChocolate* omogućava brzu i jednostavnu implementaciju
  - Primjer  ...GraphQLServer \ SetupGraphQL \ Queries.cs

```
[UsePaging]
[UseProjection]
[UseFiltering]
[UseSorting]
public IQueryable<Dokument> GetDocuments([Service] FirmaContext ctx)
=> ctx.Dokument.AsNoTracking();

[UseFiltering]
[UseSorting]
public IQueryable<Mjesto> GetCitiesForCountry(
    [Service] FirmaContext ctx, string country)
=> ctx.Mjesto.AsNoTracking()
    .Where(m => m.OznDrzave == country);
```

# Primjeri GraphQL upita (1)

- Dostupni su nam uključeni upiti pri čemu se Get iz naziva može izostaviti
- <https://localhost:44320/graphql/> (automatski uključen alat za interakciju)
- Želimo sva mjesta u Hrvatskoj koja sadrže riječ *Velika* sortirano po poštanskom broju i zanima nas samo poštanski broj i naziv mjesta

```
{  
  citiesForCountry (  
    country: "HR"  
    where: {nazMjesta : {contains:"Velika"}}  
    order : {postBrMjesta : ASC}  
  ) {  
    postBrMjesta  
    nazMjesta,  
  }  
}
```

- Definirani upit ne omogućava projekcije (npr. dohvata naziva države i zahtjeva naziv države s ključem *country*)

# Primjeri GraphQL upita (2)

- Upiti koji omogućavaju straničenje, smještaju vrijednosti unutar ključa nodes (ili edges, pa node)

```
{  
  cities (  
    where: {nazMjesta : {contains:"Velika"  
      , and: {ncontains: "Lovre"} }, oznDrzave : {neq : "HR"}},  
    order : {nazMjesta : ASC}, first:5, after: "NA==" ) {  
    totalCount, pageInfo {  
      hasPreviousPage, hasNextPage, startCursor, endCursor  
    },  
    nodes {  
      postBrMjesta  
      nazMjesta  
      oznDrzaveNavigation {  
        nazDrzave  
      }  
    }  
  }  
}
```

# Primjeri GraphQL upita (3)

- Zanima nas zadnji dokument osobe čije prezime počinje sa *Stipan*
  - Želimo dohvatiti i stavke te i nazine artikala u pojedinoj stavci

```
{  
  documents (first:1, order:{datDokumenta:DESC}  
    where: {idPartneraNavigation : {osoba : { prezimeOsobe :  
{startsWith : "Stipan"}}}}) {  
    nodes {  
      idDokumenta, iznosDokumenta, datDokumenta  
      idPartneraNavigation {  
        osoba {  
          imeOsobe, prezimeOsobe  
        }  
      }  
      stavka {  
        kolArtikla  
        sifArtiklaNavigation {  
          nazArtikla  
        }  
      }  
    }  
  }  
}
```

# Mutacije

- Mutaciju čine vlastiti postupci i njihovi ulazno/izlazni podaci
- Primjer  ...GraphQLServer \ SetupGraphQL \ MjestoMutations.cs  
 ...GraphQLServer \ SetupGraphQL \ MjestoInput.cs

```
mutation {
    addCity(input: {
        postBrMjesta : 44433
        nazMjesta: "Demo mjesto"
        postNazMjesta: "nebitno"
        oznDrzave: "_A2"
    }) {
        idMjesta
        postBrMjesta
        nazMjesta
        postNazMjesta
        oznDrzave
    }
}
```

```
public partial class Mutations {
    public async Task<Mjesto>
    AddCity([Service] FirmaContext
    ctx, MjestoInput input) ...

    public async Task<bool>
    DeleteCity([Service] FirmaContext
    ctx, int id) {
        ...
}

mutation {
    deleteCity(id: cijeli broj)
}
```

# Opaske oko imenovanja svojstava

- Naziv pojedinog atributa, pa tako i navigacijskog svojstva moguće je promijeniti dodavanjem atributa *GraphQLName*
- Moguće isključiti svojstva, ili definirati da se projekcije moraju eksplicitno definirati i slično
- Detaljnije na  
<https://chillicream.com/docs/hotchocolate/defining-a-schema/object-types>

# Razvoj primijenjene programske potpore

---

## 13.1 Izrada programske dokumentacije

Video upute: <https://bit.ly/2NjyDDx>

# 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:  WebServices\WebApi\WebApi\Util\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
    /// dohvaca </param>
    /// <returns>String s porukama svih unutarnjih iznimki.
    /// Poruka svake iznimmke dodana je u novi redak</returns>
    public static string CompleteExceptionMessage(this Exception exc ...)
```

# 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
    - Bolja varijanta – dodati sljedeći sadržaj u odgovarajući csproj
- Prevođenje proizvede *NazivAsemblja.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

# 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
  - <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

# 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, nabranje elemenata
  - <para> - paragraf teksta
  - <see> - omogućava stvaranje linka
  - <seealso> - link u sekciji *See also*
  - ...
- 
- Za detaljniji opis oznaka pogledati:
    - <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/recommended-tags>

# Primjer generirane XML datoteke

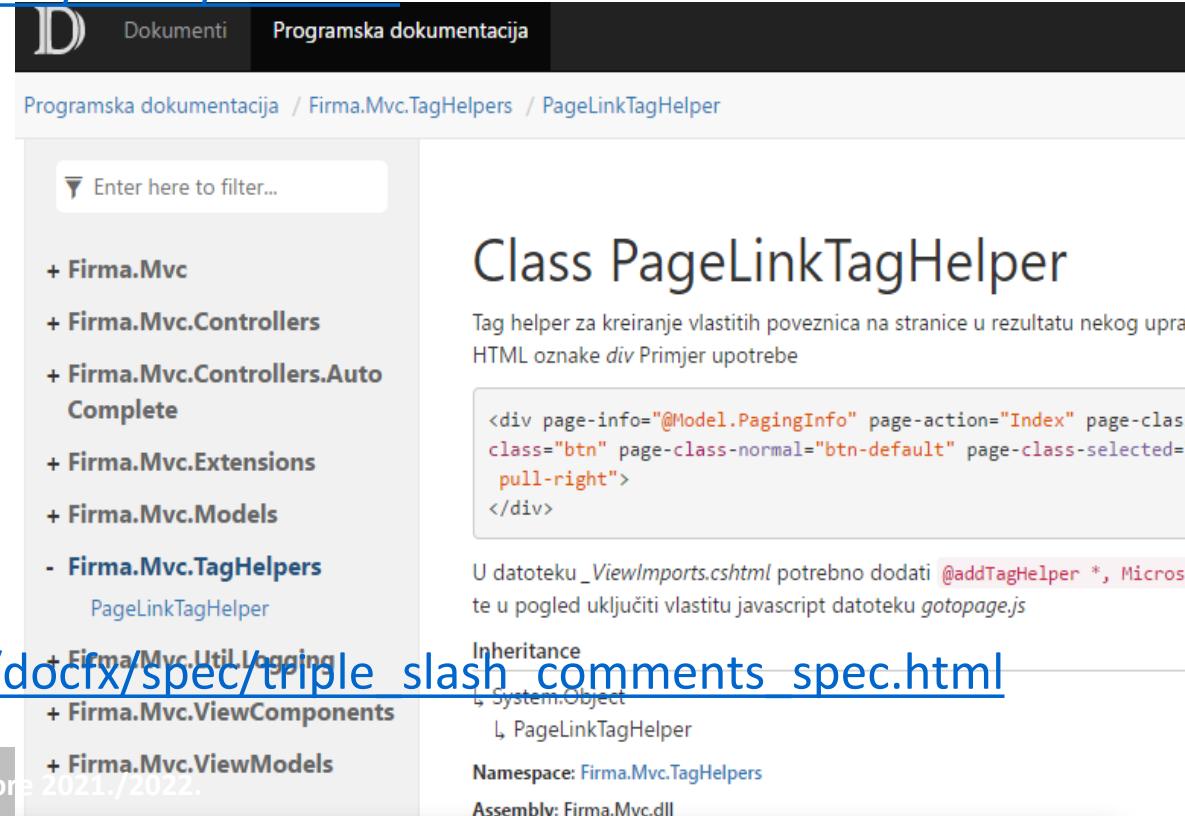
- Primjer: 

WebServices\..\WebApi\bin  
\Debug\net5.0\  
WebApi.dll

```
13 <returns></returns>
14 <member>
15   <summary>
16     Lookup controller prilagoden za jTable
17   </summary>
18 </member>
19 <member name="T:WebServices.Controllers.JTable.LookupController">
20   <summary>
21     Web API servis za rad s mjestima
22   </summary>
23 </member>
24 <member name="M:WebServices.Controllers.MjestoController.Count(System.String filter)">
25   <summary>
26     Vraća broj svih mesta filtriran prema nazivu mesta
27   </summary>
28   <param name="filter">opcionalni filter za naziv mesta</param>
29   <returns></returns>
30 </member>
31 <member name="M:WebServices.Controllers.MjestoController.GetAll(WebServices.Models.Mjesto loadParams)">
32   <summary>
33     Dohvat mesta (opcionalno filtrirano po nazivu mesta).
34     Broj mesta, poredak, početna pozicija odredeni s loadParams.
35   </summary>
36   <param name="loadParams">Postavke za straničenje i filter</param>
37   <returns></returns>
38 </member>
39 <member name="M:WebServices.Controllers.MjestoController.Get(System.Int32 id)">
40   <summary>
41     Vraća grad čiji je IdMjesta jednak vrijednosti parametra id
42   </summary>
43   <param name="id">IdMjesta</param>
44   <returns></returns>
45 </member>
46 <member name="M:WebServices.Controllers.MjestoController.Delete(System.Int32 id)">
47   <summary>
48     Brisanje mesta određenog s id
49   </summary>
50   <param name="id">Vrijednost primarnog ključa (Id mesta)</param>
51   <returns></returns>
52   <response code="204">Ako je mjesto uspješno obrisano</response>
53   <response code="404">Ako mjesto s poslanim id-om ne postoji</response>
```

# Alat za generiranje programske dokumentacije

- XML datoteka korisna unutar razvojnog sučelja, ali nije prikladna za čitanje
- Za izradu programske dokumentacije namijenjene za čitanje koristi se 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 Studio 2019 instalirati NuGet paket docfx.Console
- Napomena: podržava podskup oznaka iz XML dokumentacije
  - [http://dotnet.github.io/docfx/spec/triple\\_slash\\_comments\\_spec.html](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;
    dependencyChecker = 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
values (?,?)"; //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 interface](#)

>[Repository implementation](#)

>[Repository implementation](#)

[ActionResponse as a result](#)

[Data for comboboxes](#)

[Saving changes](#)

[Register repository](#)

[Create View Model](#)

[Choosing appropriate repository](#)

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.ContextMenuOpeningCommand, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" />
        </i:EventTrigger>
        <i:EventTrigger EventName = "ContextMenuClosing">
            <mvvm:CustomCommandInvoker Command = "{Binding Path=DataContext.ContextMenuClosingCommand, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
    ...

```

# Neki primjeri korištenja DocFX-a iz prakse (2)

## 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	

# Generiranje programske dokumentacije

- Upute koje slijede prilagođene su upute originalnih uputa  
[https://dotnet.github.io/docfx/tutorial/docfx\\_getting\\_started.html](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*

FER-RPPP > examples > WebServices > WebApi	
Name	Date
.vs	4.1
ConsoleClientApps	4.1
WebApi	8.1
WebApi.sln	4.1

WebServices > WebApi > Docs	
Name	
api	
apidoc	
articles	
images	
src	
.gitignore	
docfx.json	
index.md	
toc.yml	

# 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 programsку dokumentaciju
    - Izmijeniti sadržaj dodatne dokumentacije po uzoru na  WebServices \ WebApi \ 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  WebServices \ WebApi \ Docs \ index.md i toc.yml

WebServices > WebApi > Docs

Name
 api
 apidoc
 articles
 images
 src
 .gitignore
 docfx.json
 index.md
 toc.yml

# Uključivanje web-aplikacije u izradu dokumentacije

- Primjer WebServices\WebApi\Docs\docfx.json

```
"metadata": [
{
  "src": [
    {
      "files": [
        "src/**.csproj"
      ],
    }
  ],
  "dest": "api"
}
], ...
```

```
"metadata": [
  {
    "src": [
      {
        "files": ["*.csproj"],
        "src": "../WebApi",
      }
    ],
    "dest": "api"
  }
],
```

Name
.vs
ConsoleClientApps
Docs
WebApi
WebApi.sln

# Generiranje dokumentacije

- U mapi Docs pokrenuti docfx
  - Generira podmapu \_site s html stranicama generiranimi 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

WebServices	>	Docs	>	_site	>
<hr/>					
	▲	Name			▲
		api			
		articles			
		fonts			
		styles			
	D	favicon.ico			
		index.html			
		logo.svg			
	L	manifest.json			
	L	search-stopwords.json			
		toc.html			
		xrefmap.yml			

# Izrada PDF-a s generiranom dokumentacijom

- 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  WebApi \ 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  WebApi \ Docs \ docfx.json  
[https://dotnet.github.io/docfx/tutorial/walkthrough/walkthrough\\_generate\\_pdf.html](https://dotnet.github.io/docfx/tutorial/walkthrough/walkthrough_generate_pdf.html)
  - Pokrenuti docfx pdf
    - Nakon uspješne izgradnje pdf se nalazi u podmapi \_site\_pdf

# Razvoj primijenjene programske potpore

---

## 14. Command-Query Separation

# Uslojavanje programskog koda (1)

- Primjer:  MVC \ Controllers \ \*Controller.cs
  - Upravljači iz primjera vrše prihvati ulaznih argumenata, dohvati podataka i pripremu modela
  - Problem je u načinu dohvatu podataka
    - umjesto na „što”, upravljač u 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?
    - Što ako želimo dodati neko zajedničko ponašanje na svim mjestima (validacija prije snimanja, praćenje traga i slično)
    - ...

```
public IActionResult Index(int page = 1, int sort = 1, string filter = null)
{
    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 = 1 });
    }

    System.Linq.Expressions.Expression<Func<Artikl, object>> orderSelector;
    switch (sort)
    {
        case 1:
            orderSelector = a => a.SlikaArtikla; //ima smisla da je prvi
            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,
        Cijena = a.CijArtikla,
        DatumObjave = a.DatumObjave,
        DatumUsluge = a.DatumUsluge,
        SlikaArtikla = a.SlikaArtikla,
        JednostMjere = a.JedMjere,
        Zastavljeno = a.ZastUsluga
    });
}
```

# Uslojavanje programskog koda (2)

- Dio problema riješen
  - odvajanjem koda za sortiranje u posebne metode, ali i dalje ostaje problem vezanosti za tehnologiju
  - pogledima koji koriste prezentacijske modele
- ... ali ostaje problem vezanosti za tehnologiju (EF i relacijske baze podataka)
- Ideja: Apstrahirati pristup podacima tako da aplikacija ovisi o sučeljima kojima se definira interakcija sa slojem pristupa podacima
  - Izdvojiti poslovna pravila i složenu validaciju

# 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 mijenja 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, a konkretna implementacija se umetne tehnikom *Dependency Injection*
  - u ovom slučaju rješava se samo dio problema
    - skriva način perzistencije objektnog modela u relacijsku bazu i djelomično olakšava testiranje programa
    - moguće napisati testnu implementaciju repozitorija

# 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?
    - IQueryble<T> bi omogućio daljnje upite (filtriranje, sortiranje, ...)
      - Iz čega je nastao taj IQueryble? 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 dodatnim metodama
    - 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

# Odvajanje upita od naredbi

- Više upita u istom sučelju narušava SOLID principe
  - *Single Responsibility, Open/Closed i Interface Segregation Principle*
- *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 rezervorije

→ Svaki upit predstavljen jednim sučeljem

# Opisi upita

- „Upit” specificira tip rezultata tog upita za neke ulazne podatke
  - ne mora nužno imati argumente (svojstva)
  - može se opisati generičkim sučeljem
    - Primjer:  CommandQueryCore \ IQuery.cs
- Primjeri opisa upita

 CommandQuerySample \ Contract\ Queries \ ...

- opis upita koji treba vratiti broj mjesta ovisno o traženom tekstu

```
public class MjestoCountQuery : IQuery<int> {  
    public string SearchText { get; set; }  
}
```

- opis upita koji vraća podatke o mjestu s određenim identifikatorom

```
public class MjestoQuery : IQuery<DTOs.Mjesto> {  
    public int Id { get; set; }  
}
```

# Objekti koji „putuju” kroz slojeve

- DTO – Data Transfer Objects
- Čisti, podatkovni razredi koji služe za prijenos podataka između slojeva
- U jednostavnim primjerima dolazi do dupliciranja istih razreda, ali potrebno zbog potencijalnih promjena u budućnosti
  - promjene naziva u bazi podataka ili vrste spremišta ne smije utjecati na opise podataka koji su dogovoren s konzumentima web-servisa
  - može se koristiti *AutoMapper* ili neki drugi alat za automatsko kopiranje vrijednosti iz objekta jednog tipa u objekt drugog tipa
    - posebno praktično ako su nazivi svojstava isti
    - Preslikavanja postavljena u zasebnim klasama (pri inicijalizaciji AutoMappera dovoljno navesti neku klasu iz projekta)

```
services.AddAutoMapper(typeof(Startup),  
    typeof(Util.ApiModelsMappingProfile));
```

# Primjer složenijeg upita

- Primjeri opisa upita  Contact \ Queries \ ...
  - opis upita koji treba vratiti podskup mesta poredanih po određenim atributima i u ovisnosti o postavljenjem tekstu pretrage

```
public class MjestaQuery : IQuery<IEnumerable<DTOs.Mjesto>> {  
    public string SearchText { get; set; }  
    public int? From { get; set; }  
    public int? Count { get; set; }  
    public SortInfo Sort { get; set; }  
}
```

```
public class SortInfo {  
    public enum Order {  
        ASCENDING, DESCENDING  
    }  
    public List<KeyValuePair<string, Order>> ColumnOrder  
    { get; set; } = new List<KeyValuePair<string, Order>>();  
    ...  
}
```

# 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
  - Primjer:  CommandQueryCore \ IQueryHandler.cs

```
public interface IQueryHandler<TQuery, TResult>
{
    where TQuery : IQuery<TResult>
    Task<TResult> Handle (TQuery query);
}
```

- Upit predstavlja opis upita (ulazne podatke i povratnu vrijednost), a rukovatelj izvršava tako opisani upit
  - preciznije, sučelje propisuje implementaciju rukovatelja određenim upitom

# Primjeri rukovatelja upitom

- Primjeri sučelja u  Contract \ QueryHandlers \ ...
- Opis rukovatelja upitom za dohvati broja mjesta
  - obrađuje upit postavljen razredom MjestoCountQuery i mora isporučiti cijeli broj kao rezultat

```
public interface IMjestoCountQueryHandler :  
    IQueryHandler<MjestoCountQuery, int> { }
```

- Opis rukovatelja upitom za dohvati mjesta s određenom oznakom
  - obrađuje upit postavljen razredom MjestoQuery i mora isporučiti podatkovni objekt s podacima o traženom mjestu

```
public interface IMjestoQueryHandler :  
    IQueryHandler<MjestoQuery, DTOs.Mjesto> { }
```

# Primjeri implementacije rukovatelja upitom

- Primjer sučelja  DAL \ QueryHandlers \ ...
  - dohvaća konkretno mjesto i vraća odgovarajući DTO

```
public class MjestaQueryHandler : IMjestaQueryHandler {  
    private readonly FirmaContext ctx;  
    public MjestaQueryHandler(FirmaContext ctx) {  
        this.ctx = ctx;  
    }  
    public async Task<DTOs.Mjesto> Handle(MjestoQuery query)  
    {  
        var mjesto = await ctx.Mjesto  
            .Where(m => m.IdMjesta == query.Id)  
            .Select(m => new DTOs.Mjesto {  
                IdMjesta = m.IdMjesta ...  
            })  
            .FirstOrDefaultAsync();  
        return mjesto;  
    ...  
}
```

# Korištenje rukovatelja upitom iz upravljača

- Upravljač ovisi o sučelju, a konkretnu implementaciju rukovatelja upitom prima u konstruktoru preko DI-a

- Primjer  ... \ WebServices \ Controllers \ MjestoController.cs

```
public class MjestoController ...  
    public MjestoController(IMjestoQueryHandler mjestoQueryHandler,  
                           IMjestaQueryHandler mjestaQueryHandler,  
                           IMjestaCountQueryHandler mjestaCountQueryHandler,  
                           ...  
    this.mjestoQueryHandler = mjestoQueryHandler;  
    ...  
    public async Task<ActionResult<Mjesto>> Get(int id) {  
        var query = new MjestoQuery { Id = id };  
        var mjesto = await mjestoQueryHandler.Handle(query);  
        if (mjesto == null)  
            return Problem(statusCode: StatusCodes.Status404NotFound,  
                           detail: $"No data for id = {id}");  
        else  
            return mjesto;
```

# Postavljanje ovisnosti

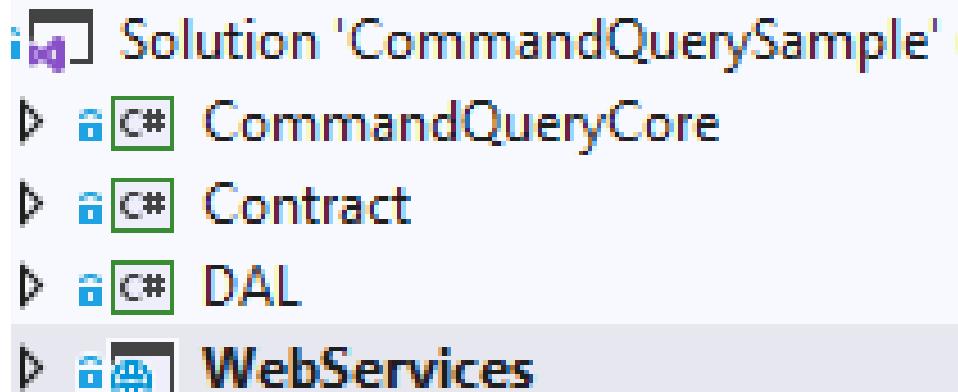
- Povezivanje za DI postavljano u razredu Startup web-aplikacije
  - Primjer  CommandQuerySample \ WebServices \ Startup.cs

```
public class Startup {  
    public void ConfigureServices(IServiceCollection services) {  
        ...  
        services.AddTransient<IMjestaQueryHandler,  
                           MjestaQueryHandler>();  
        services.AddTransient<IMjestoQueryHandler,  
                           MjestoQueryHandler>();  
        services.AddTransient<IMjestoCountQueryHandler,  
                           MjestoCountQueryHandler>();  
        ...  
    }  
}
```

- Umjesto repetitivnog pisanja može se riješiti refleksijom tražeći i registrirajući sve klase koje implementiraju neki *IQueryHandler<,>* iz *typeof(ArtiklQueryHandler).Assembly*

# Međusobne ovisnosti projekata

- Smisao pojedinog projekta
  - *CommandQueryCore* – generička sučelja za upit, naredbu i njihove rukovatelje
  - *Contract* – opis svih upita koji se koriste u rješenju i pomoći razreda koji služe za razmjenu podataka s pozivateljem
    - Data transfer objekti (Value object) – sadrže samo vrijednosti
  - *DAL* – implementacija postupaka iz Firma.DataContract
- Nijedan element web-aplikacije (upravljači, pogledi, ...) osim razreda Startup ne zna za projekt DAL i ovise samo o projektu Contract
  - Startup.cs postavlja Dependency Injection pa stoga mora postojati referenca iz projekta WebServices na DAL



# Naredbe (1)

- „Naredba” predstavlja podatke koje neki rukovatelj akcijom treba zaprimiti i odraditi
  - rukovatelja se može se opisati generičkim sučeljem
    - Primjer:  CommandQueryCore \ ICommandHandler.cs

```
public interface ICommandHandler<TCommand> {  
    Task Handle(TCommand command);  
}
```

- Sama naredba je podatkovni objekt, a ne neka akcija
  - Primjer:  Contract \ Commands \ UpdateMjesto.cs

```
public class UpdateMjesto {  
    public int IdMjesta { get; set; }  
    public string NazivMjesta { get; set; }  
    public int PostBrojMjesta { get; set; }  
    public string OznDrzave { get; set; }  
    public string PostNazivMjesta { get; set; }  
}
```

# Naredbe (2)

- Smije li naredba vraćati vrijednost? – predmet diskusija
  - Primjer:  CommandQueryCore \ ICommandHandler.cs

```
public interface ICommandHandler<TCommand, TKey> {  
    Task<TKey> Handle(TCommand command);  
}
```

- Primjer:  Contract \ Commands \ AddMjesto.cs

```
public class AddMjesto {  
    public string NazivMjesta { get; set; }  
    public int PostBrojMjesta { get; set; }  
    public string OznDrzave { get; set; }  
    public string PostNazivMjesta { get; set; }  
}
```

- Kasnije definirana referenca tipa  
`ICommandHandler<AddMjesto, int>`

# Implementacija rukovatelja naredbom

- Implementacija naredbe u podatkovnom sloju
  - Primjer  DAL \ CommandHandlers \ MjestoCommandHandler.cs
    - Može se razdvojiti na 3 rukovatelja, ali nije praktično

```
public class MjestoCommandHandler : ICommandHandler<DeleteMjesto>,
ICommandHandler<AddMjesto, int>, ICommandHandler<UpdateMjesto> {
    private readonly FirmaContext ctx;
    public MjestoCommandHandler(FirmaContext ctx) {
        this.ctx = ctx;
    }
    public async Task<int> Handle(AddMjesto command) {
        var mjesto = new Mjesto {
            NazMjesta = command.NazivMjesta,
            ...
        };
        ctx.Add(mjesto);
        await ctx.SaveChangesAsync();
        return mjesto.IdMjesta; ...
    }
}
```

# Korištenje rukovatelja upitom iz upravljača

- Upravljač ovisi o sučelju, a konkretnu implementaciju rukovatelja upitom prima u konstruktoru preko DI-a
  - Primjer  ... \ WebServices \ Controllers \ MjestoController.cs

```
public class MjestoController ...  
    public MjestoController(IMjestoQueryHandler mjestoQueryHandler,  
                           ICommandHandler<AddMjesto, int> addMjestoCommandHandler,  
                           ...) {  
  
    public async Task<IActionResult> Create(Mjesto model) {  
        AddMjesto command = mapper.Map<AddMjesto>(model);  
        int id = await addMjestoCommandHandler.Handle(command);  
  
        var addedItem = await mjestoQueryHandler.Handle(  
            new MjestoQuery { Id = id });  
  
        return CreatedAtAction(nameof(Get), new { id }, addedItem);  
    }  
}
```

# Dodatne zanimljivosti u projektu

- Primijetiti da je izvedena validacija naredbi za dodavanje i ažuriranje mjesta
  - Dodatno u odnosu na validaciju DTO-a
  - Ovom validacijom se provjera jedinstvenost para (pbr, država) neovisno o načinu izvedbe podatkovnog sloja
- Razred ValidateCommandBeforeHandle služi kao dekorator postojećeg rukovatelja nekom naredbom na način da prvo izvrši validaciju naredbe
  - Upravljači ovoga nisu svjesni – ovise samo u sučelju koje opisuje traženi rukovatelj naredbom

```
services.AddTransient< ICommandHandler<AddMjesto, int>,
    ValidateCommandBeforeHandle<AddMjesto, int,
        MjestoCommandHandler>>();
```

# Nedostatci pristupa

- Zamorno registriranje ovisnosti velikog broj rukovatelja
- Previše složeno za male projekte
- Paradoksalno, ali ponekad teško za testiranje
- Constructor over-injection
  - Razred radi previše toga?
  - Koristiti umetanje u pojedinoj akciji s [FromServices]?
  - Rješenje: obrazac *Medijator*
    - <https://github.com/jbogard/MediatR>
    - Konstruktori ovise o medijatoru, a trivijalno se registriraju svi rukovatelji iz nekog projekta
    - Primjer  CqsWithMediator \ \*
    - Moguće definirati akcije koje treba izvesti prije ili poslije rukovanja upitom/naredbom
      - Primjer  ... \ Contract \ Validation \ ValidationPipeline.cs