

# Web-aplikacije

ASP.NET Core MVC - CRUD nad jednostavnom tablicom

2020/21.07

## ➤ Što je web aplikacija?

- Programska aplikacija kojoj se pristupa preko internetskog preglednika ili drugog programa koji implementira HTTP (Hyper Text Transfer Protocol)

## ➤ Da li je skup web stranica ujedno i web aplikacija?

- Aplikacija koristi programsku logiku da bi prikazala sadržaj korisniku
- Kod web aplikacija obično se izvršava neki programski kod na serveru
- Primjer: statički cjenik nije aplikacija; dinamički cjenik je aplikacija

## ➤ Primjer web-aplikacije: Edgar <http://edgar.fer.hr>

- Identifikacija i autorizacija korisnika, pisanje provjere, pregled rezultata, ...
- Baza podataka + programska logika za generiranje web stranica

## ➤ Elementi web-stranice

- HTML (Hyper Text Markup Language) - osnovni jezik za definiranje web-stranica
- JavaScript: jezik za klijentske skripte koje izvodi preglednik
- Kôd u nekom drugom jeziku koji se izvodi na poslužitelju

# Karakteristike web-aplikacija

3

- Izvršava se i na poslužitelju i na klijentu
  - Klijentski dio mora biti napisan u nekom od jezika koji Web preglednik podržava (HTML, JavaScript).
    - Ograničen pristup resursima na strani klijenta (npr. JavaScript koji se izvršava unutar preglednika ne može čitati s korisnikovog diska)
- Prednosti
  - Korisnik može biti bilo tko s pristupom Internetu
    - nema instalacijske procedure na strani klijenta
    - koriste se na bilo kojem OS, koji ima internetski preglednik
  - Jednostavno održavanje i nadogradnja na novu verziju
- Nedostatci
  - Složenija izrada u odnosu na samostojne klijentske aplikacije
    - Potreba za posebno dizajniranim sučeljem (Web design)
    - Mogući problemi pri prikazu u različitim preglednicima
    - Potrebna prilagodba regionalnim posebnostima korisnika
  - Sigurnosni problemi (neovlašten pristup aplikaciji i poslužitelju, zatrpavanje prometom)

# Kratka povijest razvoja web-aplikacija

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

# Interaktivnost na klijentskoj strani

5

- „Duga” povijest, ali šira upotreba tek u nekoliko zadnjih godina
- JavaScript 1995. godine
  - Nakadno se pojavljuju (s više ili manje uspjeha) Flash, Java Applets i Silverlight
- jQuery – prva verzija 2006. godine
  - Verzija 1.5: 2011.godine
  - Verzija 2.0: 2013. godine
  - Verzija 3.1: srpanj 2016.
  - Verzija 3.4.1: travanj 2019.
  - Verzija 3.6.0: ožujak 2021.
- HTML 5 2014. godine
- CSS predložen 1994., objavljen prvi put 1996.
  - Aktualna verzija CSS 3 – daljnji razvoj po modulima

# CGI skripte

6

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

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

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

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

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

# Pokušaji odvajanja prezentacijskih elemenata

8

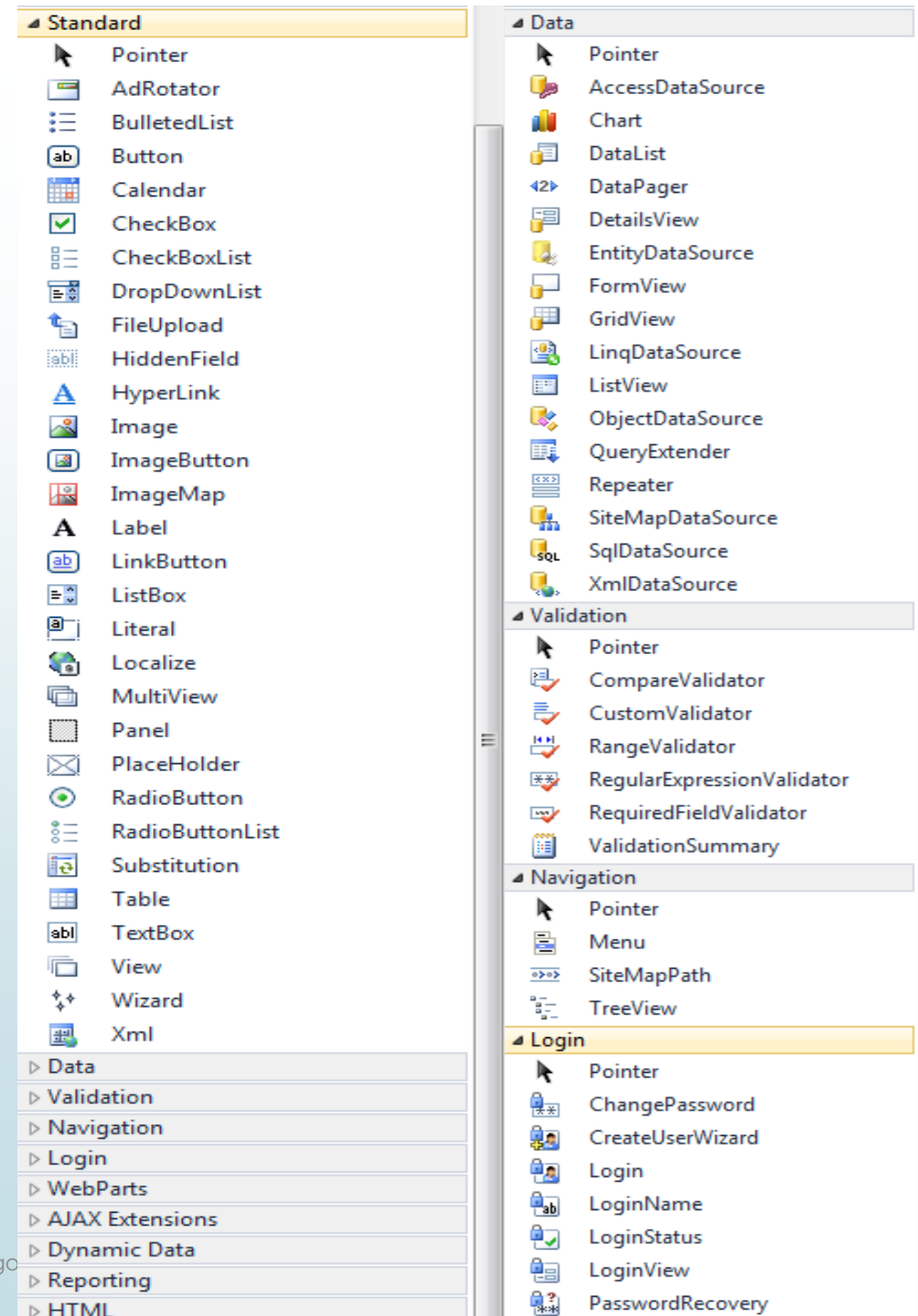
- Problem kako odvojiti prezentacijski dio od dohvata i pripreme podataka
  - Izbjeći miješanje koda koji se izvršava na serveru i HTML-a koji se prikazuje korisniku.
- Kod PHP-a se počinju koristiti predlošci (npr. *Smarty*)
  - u posebnim datotekama kreiraju se predlošci s posebno označenim varijablama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



# ASP.NET Web Forms

9

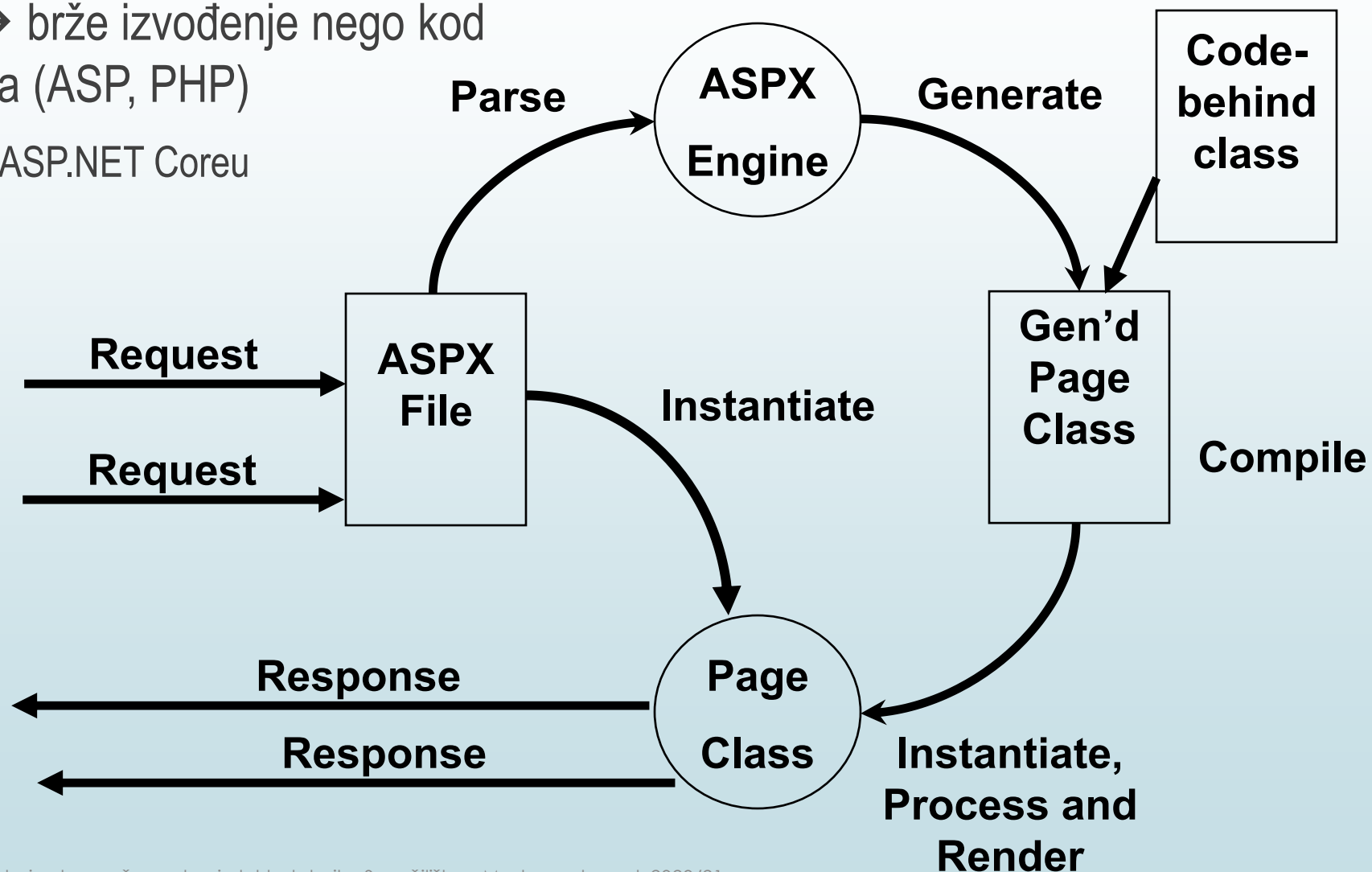
- cca 60 unaprijed definiranih serverskih kontrola
  - simulira se izrada web aplikacija nalik windows formama s ograničenjima web tehnologije
- Realizacija
  - Stranica.aspx + Stranica.aspx.cs + Stranica.aspx.designer.cs
  - .NET Framework vrši transformaciju kontrola u HTML
  - kontrole imaju mogućnosti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



# Prevođenje i prikaz stranica

10

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



# WebForms - Odvajanje dizajna od koda

11

## ➤ Datoteka s kontrolama - Default.aspx

- ASPX stranica uobičajeno sadrži HTML, serverske kontrole, JavaScript i ostale prezentacijske elemente
- Source i/ili Design pogled na istu stranicu

The screenshot displays the Visual Studio IDE with the Source view of the Default.aspx file. The code is as follows:

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

The Client Objects & Events window on the left shows "(No Events)". The Solution Explorer on the right shows the project structure:

- Default.aspx
- Default.aspx.cs
- Default.aspx.designer.cs

The bottom of the window shows the Design, Source, and <html> tabs, with the Source tab selected.

# Odvajanje dizajna od koda

12

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

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

- Klikom na gumb izvodi se postupak Page\_Load (ako postoji), a zatim slijede obrade događaja
  - Rekonstrukcija sadržaja kontrola pri svakom zahtjevu
  - Npr. U gornjem primjeru klikom na gumb se na postojeći tekst (sa svim prethodnim nadopisivanjima) nadopiše trenutno vrijeme

# Ključne karakteristike Web Formi

13

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

# Nedostatci ASP.NET web formi

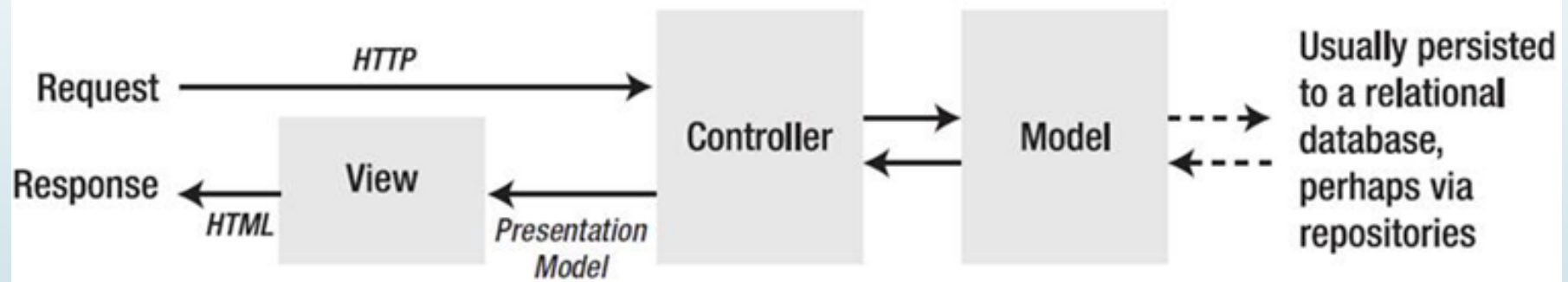
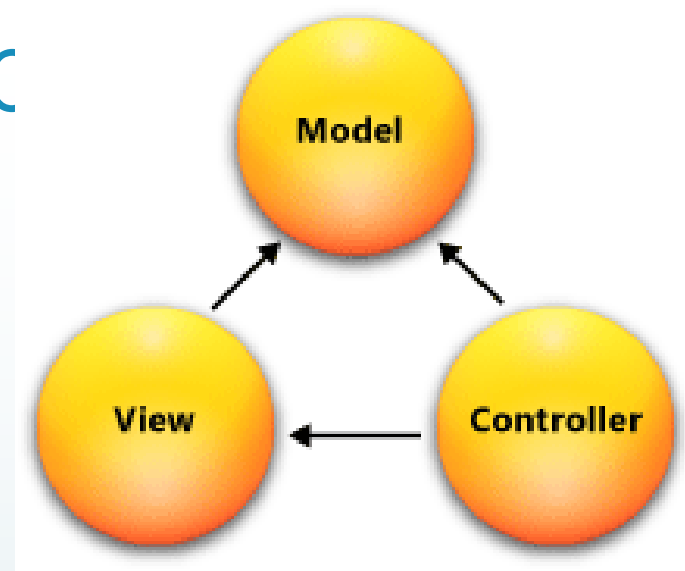
14

- ViewState proizvodi veliku količinu podataka pri svakom zahtjevu
  - Komplikiran životni ciklus stranice
  - Ograničena kontrola nad proizvedenim HTML-om
  - Prevelike \*.cs datoteke uz često miješanje prezentacijske i aplikacijske logike
  - Slaba mogućnost testiranja
- 
- ASP.NET MVC kao rješenje navedenih problema
    - Objedinjena iskustva MVC implementacija u drugim jezicima
    - MVC kao koncept nastao u kasnim sedamdesetim godinama prošlog stoljeća (Smalltalk projekt unutar Xeroxa)

# MVC

## ASP.NET Core MVC – osnovne postavke

- 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 niže slojeve i određuje redoslijed prikaza pogleda



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



# Prednost MVC-a nad web formama

17

- Smanjena složenost podjelom aplikacije u model, pogled i upravljač
- Ne koristi se *ViewState* ni serverske kontrole čime se omogućava potpuna kontrola ponašanja aplikacije.
- Zahtjevi centralizirani na jedan upravljač
  - Front Controller pattern nasuprot Page Controller pattern
    - Varijanta Page Controllera „se vratila” s tehnologijom RazorPages unutar ASP.NET Corea
  - bogata podrška za interpretiranje/usmjeravanje zahtjeva
- Podrška za test-driven development (TDD).
- Dobar okvir i za veće razvojne timove i za dizajnere

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

# Stvaranje nove web-aplikacije

19

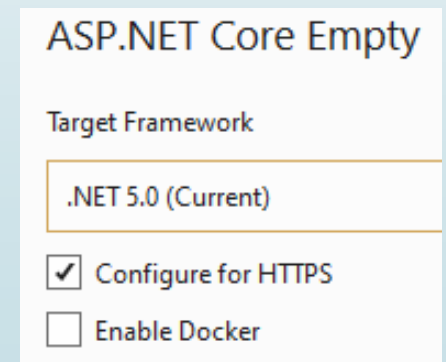
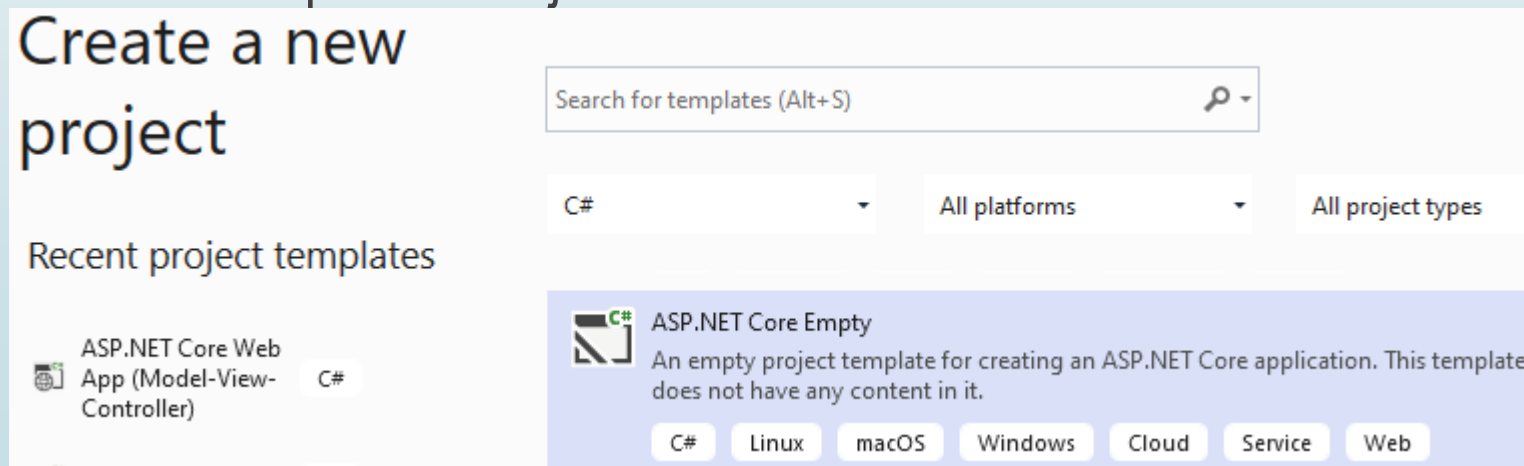
## ➤ Iz naredbenog retka

- `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

20


## ➤ Samo Program.cs i Startup.cs.

- Web-aplikacija se izvršava na privremenom web-serveru (localhost:port)
  - Može se spojiti s klasičnim web serverom (IIS, Apache, Nginx i slično)
- Ispisuje *HelloWorld* - definirano u postupku *Configure*
- Primjer:  BiloKojaPraznaAplikacija \ Startup.cs

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env) {  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    app.UseRouting();  
  
    app.UseEndpoints(endpoints => {  
        endpoints.MapGet("/", async context => {  
            await context.Response.WriteAsync("Hello World!");  
        });  
    });  
}
```

# Postavke pokretanja programa


21

- Program.cs je „glavni program” web-aplikacije
  - Pokreće web-server i određuje postavke aplikacije ili određuje razred u kojem se to obavlja (uobičajeno Startup)
    - Putanja, konfiguracijske datoteke, „tajne” datoteke, *pratitelji traga*, ...  
<https://docs.microsoft.com/en-us/dotnet/core/extensions/generic-host>
- Primjer:  MVC \ Program.cs

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

# Uključivanje podrške za MVC

22

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

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

# Način rada MVC aplikacije

23

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

## ➤ 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 pogleda i upravljače na istom nivou

- Praktično kod velikih aplikacija

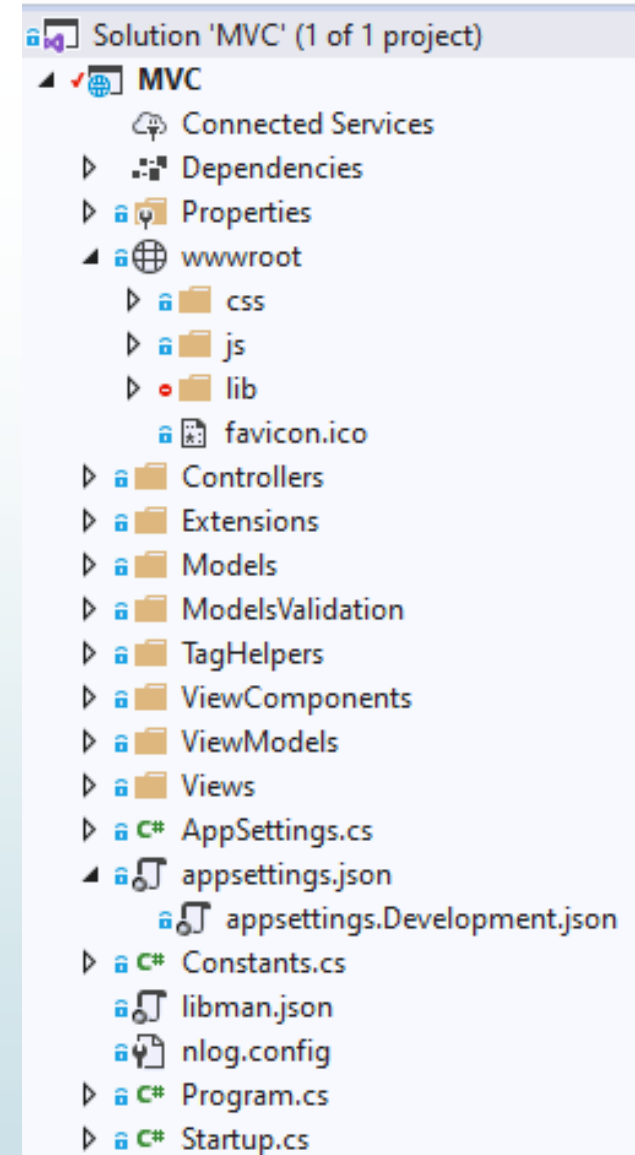


# Struktura primjera iz predavanja

25


## ➤ 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 prihvata podataka iz pogleda
  - prezentacijski modeli
- *TagHelpers*: vlastiti razredi s atributima kojim se proširuju uobičajene HTML oznake
- *ViewComponents*: komponente (dijelovi aplikacije) koje se koriste na više mjesta, dizajniraju se zasebno te se uključuju u pojedinim pogledima
- Konfiguracijske i projektne datoteke
- Ostali razredi po potrebi



# Uključivanje sadržaja mape wwwroot

26

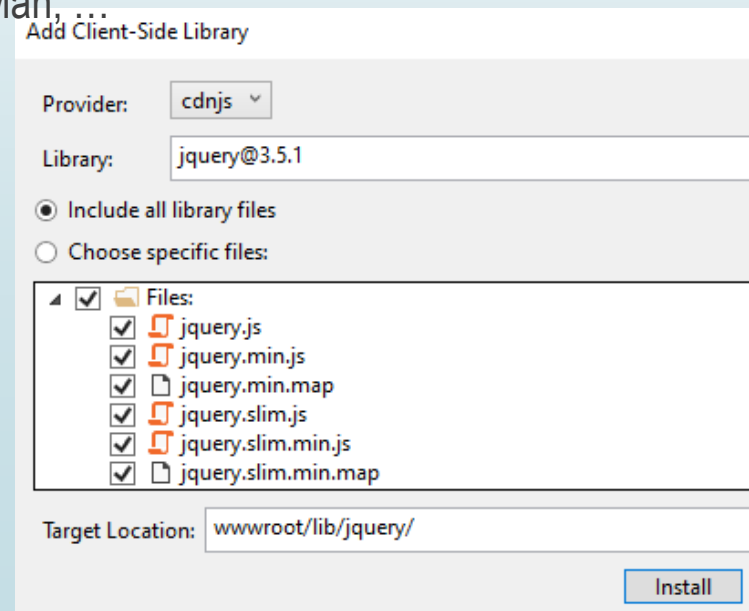
- Sadržaj u mapi *wwwroot* je namijenjen za statički sadržaj koji se koristi iz aplikacije i u konačnici će biti kopiran na produkcijski server
- Potrebno uključiti mapu sa statičkim sadržajem u aplikaciju
  - Ne mora biti nužno *wwwroot*
- Primjer:  MVC \ Startup.cs

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

# Paketi klijentskih biblioteka

27

- Bootstrap, jQuery, ...
- Način uključivanja
  - Samostalno skinuti distribuciju i staviti na odgovarajuće mjesto
    - uobičajeno `wwwroot\lib`
  - Uključiti direktno preko CDN-a (najjednostavnije!)
    - 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)

28

- 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)

29

```
{
  "version": "1.0", "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "twitter-bootstrap@4.5.3",
      "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

30











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

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

Manage Bower Packages: 'Firma.Mvc

Browse Installed Update Available

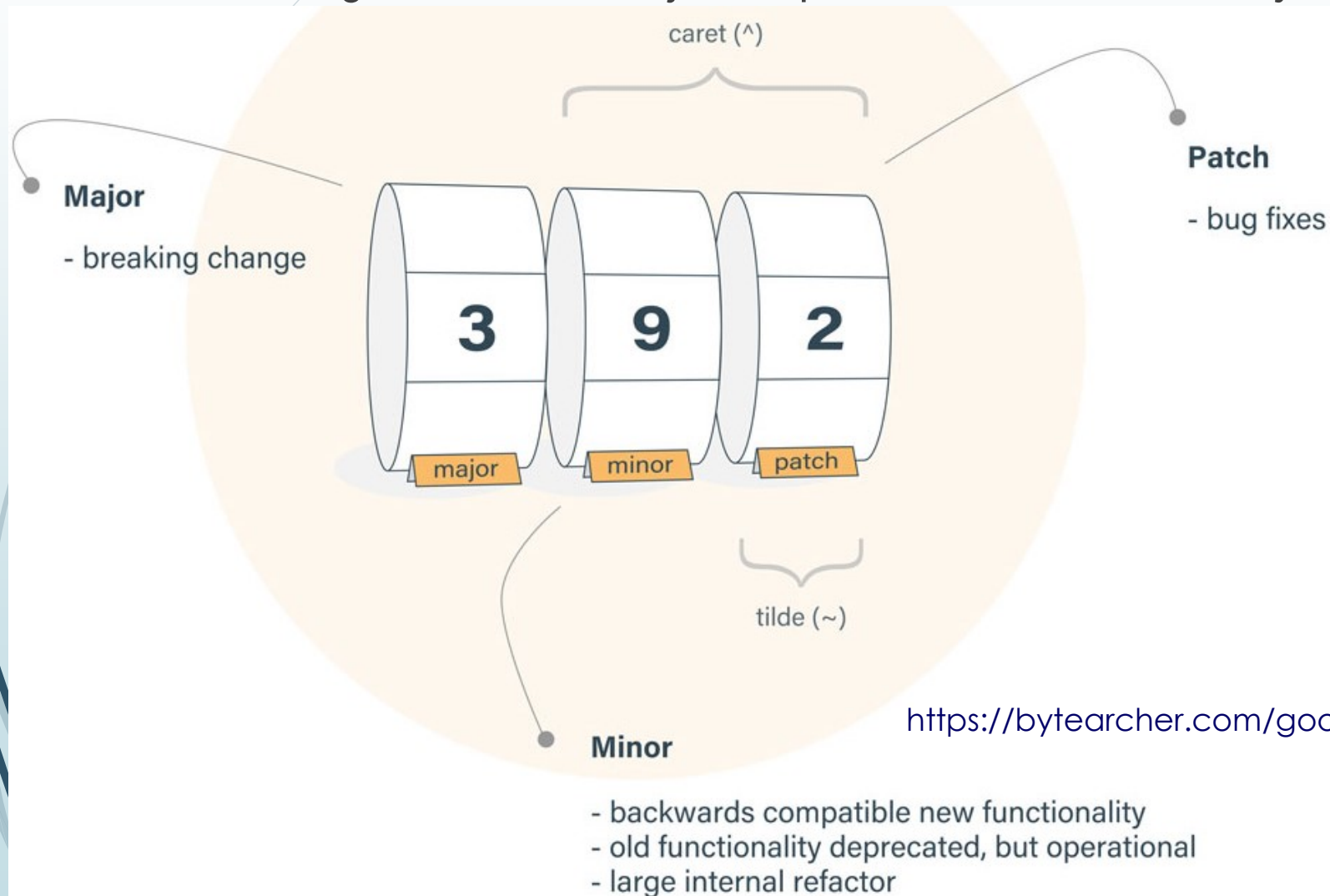
Search (Ctrl+E)

	bootstrap	
<div>Uninstall</div>		
	jQuery	
	jquery-ui	
	jquery-validation Form validation made easy	
	jquery-validation-unobtrusive	

# Značenje znakova ^ i ~ u konfiguracijskoj datoteci

31

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



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

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

# Početna stranica (1)

32

- Programski kod akcije Index na upravljaču Home kao rezultat vraća pogled ne navodeći ime pogleda

- Tip rezultata je *ActionResult*
  - 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 ActionResult Index() {  
        return View();  
    }  
}
```

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



# Uobičajeni rezultati akcije upravljača (izvedeni iz *ActionResult*)

33

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

# Sintaksa pogleda

34

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

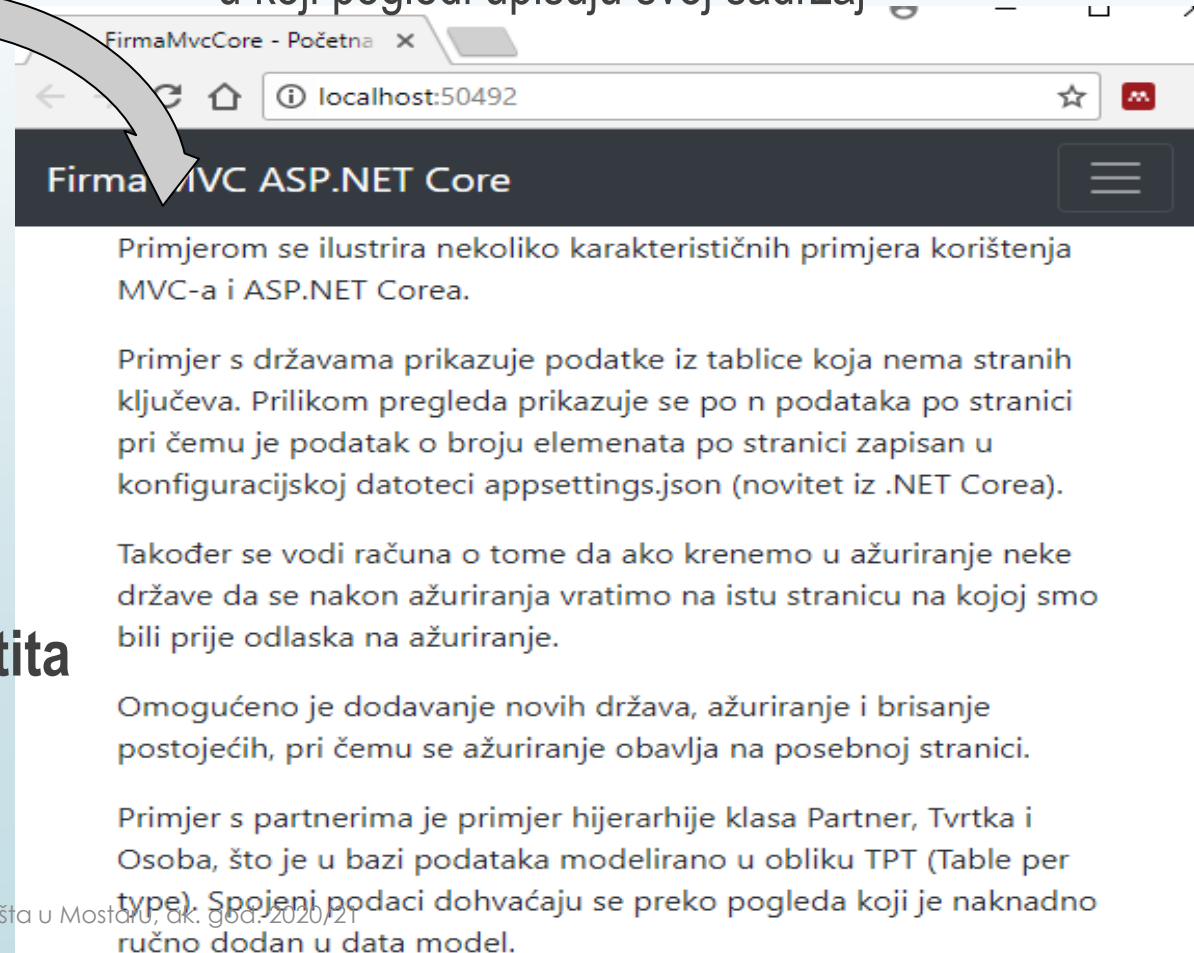
# Početna stranica (2)

35

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

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

- 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



- Pretpostavljena 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 pretpostavljena glavna stranica
- Glavna stranica uključuje stil i javascript datoteke, definira navigaciju, prostor za javascript pojedinog pogleda...
  - Konkretni sadržaj za neki zahtjev dobit će se pomoću RenderBody

# Primjer glavne stranice


37

► 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)

38

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

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

# Vlastite komponente

39

► 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-route-nazivparametra` i slično, na osnovi kojih nastane konkretna poveznica

# Konfiguracijske datoteke

40

- Inicijalna konfiguracija uključuje datoteku *appSettings.json*, datoteke s „tajnim ključevima” i nadjačavanja ovisno o varijabli okruženja
- Dohvat vrijednost vrši se kroz objekt tipa *IConfiguration*
  - Umetnut u konstruktor razreda Startup
- Primjer:  MVC \ Startup.cs

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



# Dodavanje dodatne konfiguracijske datoteke

41

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

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

# Preslikavanje konfiguracijske datoteke u vlastiti razred (1)

42

- 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 u vlastiti razred (2)

43

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

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

- Omogućava da se u konstruktor pojedinog upravljača umetne *IOptions<AppSettings>*, odnosno *IOptionsSnapshot<AppSettings>*

# Podsjetnik: Stvaranje EF modela na osnovu postojeće BP

44

## 1. Instalirati dotnet-ef na računalu

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

## 2. U mapi ciljanog projekta izvršiti sljedeće naredbe

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

➡ ili dodati koristeći opciju Manage NuGet Packages


## 3. U naredbenom retku izvršiti sljedeće dvije naredbe

```
dotnet restore
```

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

# Postavke za spajanje na bazu podataka

45

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


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

- Primjer:  MVC \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {  
    services.AddDbContext<Models.FirmaContext>(  
        options => options.UseSqlServer(  
            Configuration.GetConnectionString("Firma")  
                .Replace("sifra", Configuration["FirmaSqlPassword"])  
        ));
```

# Akcija dohvata svih država (1)

46

- 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 *Startup.ConfigureServices*, 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)

47

➡ 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

48

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


```
@using MVC.Models;
@model IEnumerable<Drzava>
...
<table class="table ... table-striped">
...
@foreach (var item 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)

49

➡ Javascript biblioteka <https://datatables.net>

- ➡ Dostupna i kao plugin za Libman i ostale paketne alatne
- ➡ Aktivira se kodom u jQueryu, nakon što se dokument učitava
- ➡ 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)

50

➤ Brzo, jednostavno i vizualno privlačno rješenje...

➤ ali nije prikladno za velike količine podataka (svi podaci se uvijek dohvaćaju)

➤ država ima malo, pa to nije toliko primjetno, ali u primjeru s mjestima se primijeti

FirmaMvcCore - Popis država

localhost:58403/Drzava

Firma MVC ASP.NET Core Početna stranica Države Mjesta Artikli Partneri Dokumenti Izveštaji Pregled log datoteka WebApi dokumenti

## Popis država

Prikaži 10 zapisa

Pretraga

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

Programsko inženjerstvo, Fakultet strojarstva, računarstva i elektrotehnike Sveučilišta u Mostaru, akad. god. 2020/21

41 - 50 od ukupno 222 zapisa

Prethodna 1 ... 4 5 6 ... 23 Sljedeća

➤ Prijevod se može postaviti prilikom inicijalizacije

➤ Primjer: MVC \ Views \ Drzava \ IndexSimple.cshtml

# Izdvajanje prijevoda za *DataTables* u zasebnu datoteku

51

➡ Prijevod prebačen u datoteku site.js koja se uključuje u glavnoj stranici

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

```
...  
<script src="~/js/site.js" asp-append-version="true"></script>  
@RenderSection("scripts", required: false)  
</body>  
</html>
```

➡ Primjer:  MVC \ wwwroot \ js \ site.js

```
tableLangSettings = {  
  search: "Pretraga",  
  info: "_START_ - _END_ od ukupno _TOTAL_ zapisa",  
  lengthMenu: "Prikaži _MENU_ zapisa",  
  paginate: {  
    first: "Prva", previous: "Prethodna",  
    next: "Sljedeća", last: "Zadnja"  
  },  
  ...  
}
```

# Uključivanje prijevoda za *DataTables*

52


➡ Prijevod prebačen u datoteku site.js koja se uključuje u glavnoj stranici

➡ Primjer:  MVC \ Views \ Drzava \ IndeksSimple.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({  
                language: tableLangSettings  
            });  
        });  
    </script>  
}
```

# Straničenje na serverskoj strani

53

- Javascript biblioteka za tablice koja koristi postupke sa servera ili vlastite metode?
  - U ovom primjeru vlastiti postupci, kao motivacija za neke druge principe
- Primjer:  MVC \ ViewModels \ PagingInfo.cs
  - Razred za informacije o trenutnoj stranici, broju stranica i aktivnom sortiranju

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

# URL i model za prikaz svih država

54

➤ Adresa: /Drzava/Index

➤ Upravljač: Drzava(Controller), Akcija: Index

➤ Zbog postavki usmjeravanja može i bez navođenja akcije Index

➤ Opcionalno se predaje broj stranice i redoslijed sortiranja

➤ Npr. /Drzava?page=27&sort=4&ascending=false

➤ Primjer:  MVC\ ViewModels \ DrzaveViewModel.cs

➤ Model za prikaz država sadrži popis država i podatke o trenutnoj stranici, ukupnom broju stranica i redoslijedu sortiranja (razred *PagingInfo*)

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

# Parametri sortiranja i straničenja

55

➡ Primjer:  MVC \ Views \ Drzava \ Index.cshtml

➡ Omogućeno sortiranje i straničenje

- ➡ Zaglavlje tablice s podacima sadrži poveznice na trenutnu akciju (Index) na trenutnom upravljaču s parametrima za broj stranice i način sortiranja
- ➡ Bit će dio adrese stranice
- ➡ Postavlja se atributima oblika `asp-route-nazivparametra`

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

# Akcija dohvata podskupa država (1)

56

► Primjer:  MVC \ Controllers \ DrzavaController.cs

► Upravljač provjerava postoji li uopće bilo koja država u BP te ako ne postoji preusmjerava rezultat na akciju *Create* uz odgovarajuću poruku

► O svojstvu *TempData* naknadno

```
public class DrzavaController : Controller {  
    ...  
    public IActionResult Index(int page = 1, int sort = 1,  
                                bool ascending = true) {  
        int pagesize = appData.PageSize;  
  
        var query = ctx.Drzava  
                    .AsNoTracking();  
  
        int count = query.Count();  
        if (count == 0) {  
            TempData[Constants.Message] = "Ne postoji niti jedna država."  
            TempData[Constants.ErrorOccurred] = false;  
            return RedirectToAction(nameof(Create));  
        }  
    }  
}
```



# Akcija dohvata podskupa država (2)

57

► Primjer:  MVC \ Controllers \ DrzavaController.cs

► Formira se podatak o stranicama i sortiranju, a zatim se provjerava je li unutar valjanog raspona te ako nije, preusmjerava se na zadnju stranicu

► Drugi parametar pri preusmjeravanju je anonimni objekt koji se sastoji od postavki usmjeravanja

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

# Akcija dohvata podskupa država (3)

58

➤ 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)

59

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

```
public static IQueryable<Drzava> ApplySort(
    this IQueryable<Drzava> query, int sort, bool ascending) {
    System.Linq.Expressions.Expression<Func<Drzava, object>>
        orderSelector = null;

    switch (sort) {
        case 1:
            orderSelector = d => d.OznDrzave;
            break;
        case 2:
            orderSelector = d => d.NazDrzave;
            break;
        ...
    }
    if (orderSelector != null)
        query = ascending ? query.OrderBy(orderSelector) :
            query.OrderByDescending(orderSelector);
    return query;
}
```

# Akcija dohvata podskupa država (5)

60

► 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;  
        ...  
        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

62

➤ 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, samo prikaz pogleda

➤ prihvrat popunjenih 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

63

- 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" ... >
```

# Smisao atributa oblika asp-nešto

64

- Prilikom generiranja stranice atributi oblika asp-nešto će uzrokovati stvaranje standardnih HTML atributa

- Navedena funkcionalnost kao posljedica tzv. *tag-helpera*

- Primjerice asp-for uzrokuje stvaranje atributa id i name

```
<input asp-for="NazDrzave" class="form-control" />
```

→ 

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

- Po atributu id se jednoznačno može odrediti HTML kontrola, a atribut name će se koristiti prilikom slanja popunjenih podataka na server



# Tekst pored kontrola za unos novog podatka

65

- 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" />
    ...
```

- Umjesto tvrdo-kodiranog teksta koristi se atribut *Display* u samom modelu

- Dodan 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; }
```

# Prihvat podataka

66

- 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?view=aspnetcore-5.0#sources>

# Provjera ispravnosti izvora zahtjeva


67

- Atributom *ValidateAntiForgeryToken* provjera se je li zahtjev stigao sa stranice za unos novog podatka ili netko pokušava direktno izvršiti unos s nekog drugog mjesta
  - skriptom ili kroz neki drugi alat (npr. Chrome Postman, Fiddler, ...)
  - 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


68

- Podatak se doda u kontekst i pozove snimanje promjena
  - U slučaju uspjeha, rezultat je preusmjerenje 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

69

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

70

- 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 preusmjerenja na akciju pregleda svih država
  - *ViewBag* ili *ViewState* nisu pogodni, jer se ne iscrtava pojedini pogled nego dolazi do preusmjerenja

# Prijenos podataka između zahtjeva (2)

71

➤ Poruke između zahtjeva stavljaju se u *TempData*

➤ Podaci iz *TempData* se brišu nakon konzumiranja

➤ Interno se koristi sjednica (session), tj. podaci unutar sjednice

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

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

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

➤ Parcijalni pogled koji oblikuje izgled poruke

➤ Primjer:  MVC \ Views \ Shared \ ShowMessage.cshtml


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

# Validacija podataka



# Validacijski atributi


73

- Neispravni ili nepotpuni podaci uzrokuju 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*
- Nekoliko postojećih: *Required*, *MaxLength*, *Range*, *RegularExpression* ...
  - Sadrže i odgovarajući javascript kôd koji prikazuje pogreške odmah prilikom unosa podatka i onemogućava slanje podataka na server
- Primjer:  MVC \ Models \ Drzava.cs

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

# Validacija korištenjem paketa *Fluent Validation*

74

- 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")  
            .MaxLength(3).WithMessage("Oznaka države ...");  
  
        RuleFor(d => d.NazDrzave)  
            .NotEmpty().WithMessage("Naziv države je obvezno polje");  
  
        ...  
    }  
}
```

# Uključivanje validacije napisane korištenjem paketa Fluent Validation

75


- Pojedini validacijski razred se može uključiti sa *services.AddTransient* ili se mogu automatski pronaći i uključiti svi razredi koji nasljeđuju *AbstractValidator*

- Primjer:  MVC \ Startup.cs

```
using FluentValidation.AspNetCore;
...
public void ConfigureServices(IServiceCollection services) {
    ...
    services.AddControllersWithViews()
        .AddFluentValidation(fv =>
            fv.RegisterValidatorsFromAssemblyContaining<Startup>());
}
```

# Prikaz validacijskih pogrešaka


76

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

# Stil prikaza validacijskih pogrešaka

77

- 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 \ Views \ Drzava \ Create.cshtml
  - U primjeru postavljen stil tako da 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


78

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

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

# Provjera ispravnosti modela prije snimanja

79

- 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);
    }
}
```

# Brisanje podatka



# Forma za brisanje podatka

81

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

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

# Akcija brisanja

82


➡ Primjer:  MVC \ Controllers \ DrzavaController.cs

- ➡ Brisanje nema vlastiti pogled - nakon brisanja, upravljač preusmjerava rezultat na drugu 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();
        ...
    }
    catch (Exception exc) {
        TempData[Constants.Message] = ...
    }
    return RedirectToAction(nameof(Index), ...)
```

# Potvrda brisanja podatka

83

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

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


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

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

# Ažuriranje jednostavnog podatka

# Poveznica za ažuriranje države

85

- 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

86

➡ Primjer:  MVC \ Controllers \ DrzavaController.cs

- ➡ Dva postupka Edit i Update koji se odazivaju na istu akciju
  - ➡ *Edit* prima predstavlja inicijalno otvaranje forme (GET zahtjev)
  - ➡ *Update* naknadno slanja popunjenih podataka (POST zahtjev)
- ➡ U ovom primjeru oba postupka imaju iste argumente, pa moraju imati različite nazive
- ➡ Postupci primaju i podatke o trenutnoj stranici i načinu sortiranja 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

87

- 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

88

## ➤ 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 method="post" asp-route-page="@ViewBag.Page"
      asp-route-sort="@ViewBag.Sort"
      asp-route-ascending="@ViewBag.Ascending">
  <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" />
    ...
  </div>
</form>
```



# Priprema stranice za ažuriranje

89

➡ 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);
        }
    }
}
```

# Prihvat vrijednosti za ažuriranje

90

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

# Asinkroni postupci (ukratko)

91

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

# Prihvat vrijednosti za ažuriranje

92

► 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.FindAsync(id);
        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();
            ...
        }
    }
}
```

# Modeli s agregiranim podacima


# Proširenje primjera na popis mjesta

94

- Rad s mjestima izveden 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
- Primjer:
  -  MVC \ Controllers \ MjestoController.cs
  -  MVC \ Views \ Mjesto \ \*.cshtml
  -  MVC \ ViewModels \ Mjest\*ViewModel.cs

# Model s agregiranim podacima

95

- 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; }  
}
```

# Stvaranje agregiranog modela

96

➡ Primjer:  MVC \ Controllers \ MjestoController.cs

➡ Prilikom upita u Selectu napraviti projekciju na željeni prezentacijski model

```
public class MjestoController : Controller {  
    ...  
    public IActionResult Index() {  
        var mjesta = ctx.Mjesto  
            .OrderBy(m => m.OznDrzaveNavigation.NazDrzave)  
            .ThenBy(m => m.PostBrMjesta)  
            .Select(m => new MjestoViewModel  
            {  
                IdMjesta = m.IdMjesta,  
                NazivMjesta = m.NazMjesta,  
                PostBrojMjesta = m.PostBrMjesta,  
                PostNazivMjesta = m.PostNazMjesta,  
                NazivDrzave = m.OznDrzaveNavigation.NazDrzave  
            })  
            .ToList();  
        return View(mjesta);  
    }  
}
```



# Primjer unosa objekta s ograničenim skupom vrijednosti za neko svojstvo

97

- 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 mjesta

Poštanski broj mjesta

Naziv mjesta

Poštanski naziv mjesta

Država

Odaberite državu

Odaberite državu

Croatia

Andora

Angola

Argentina

Armenia

Bahamas

Bangladesh

Barbados

Belarus


Belgium

Belize

# Priprema podataka za padajuću listu

98


## ➤ 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

99

- 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 inicijalno 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

100

- 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 usmjeravanja

101

➡ Moguće definirati i drugačija usmjeravanja (oprezno!)

➡ Primjer:  MVC \ Startup.cs (Configure)

```
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>