

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
- <https://docs.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-5.0#the-benefits-and-costs-of-server-and-client-rendered-ui>
- 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?

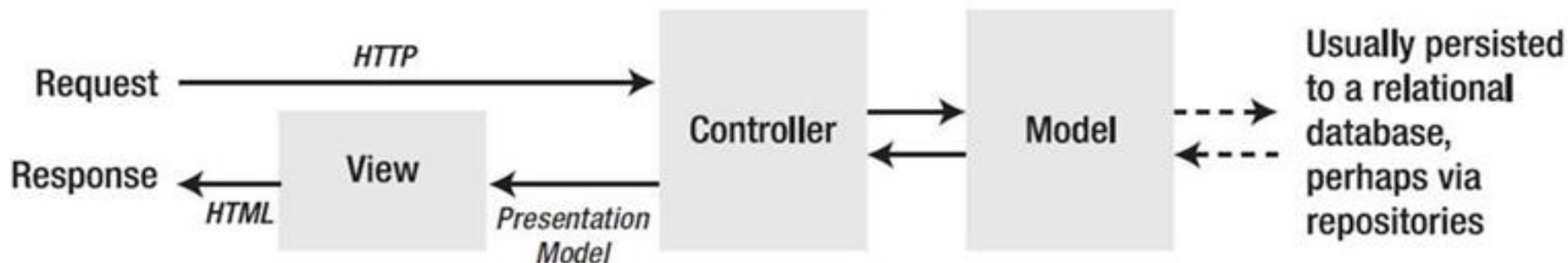
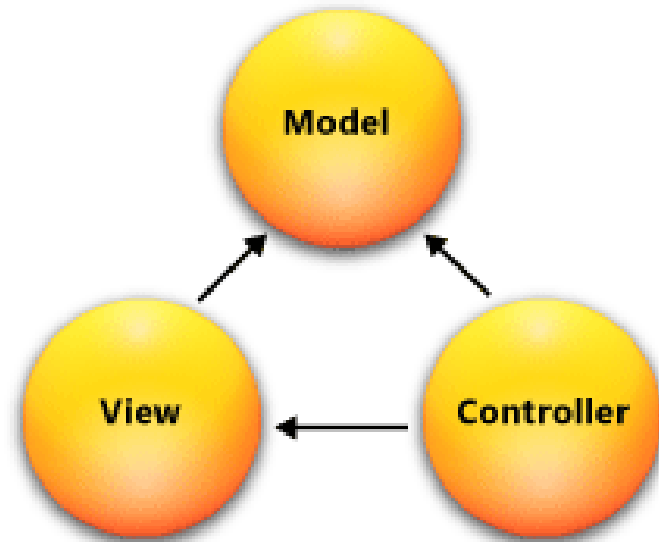
MVC kao primjer obrasca *Front controller*

- Zahtjevi centralizirani na jedan upravljač (*Front Controller pattern*)
 - *RazorPages* i stare *WebForms* 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

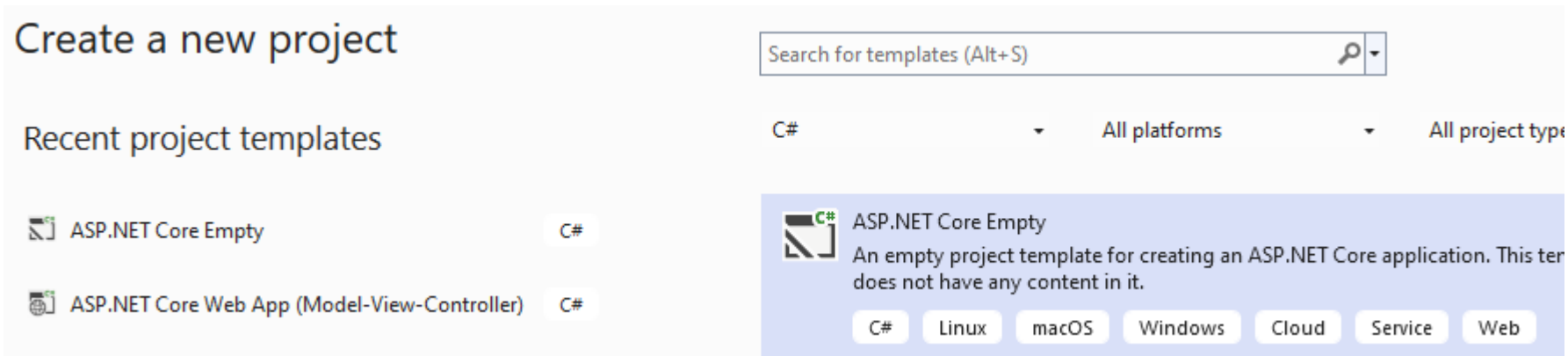
- Model - Pogled - Upravljač
- Detaljnija razrada **prezentacijskog sloja**
 - Pogled definira izgled korisničkog sučelja
 - Obično vezan za objekt iz modela
- **Upravljač predstavlja prezentacijsku logiku**
 - Prima ulaz iz pogleda, obrađuje ga, puni i dohvaća model, poziva postupke iz nižih slojeva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

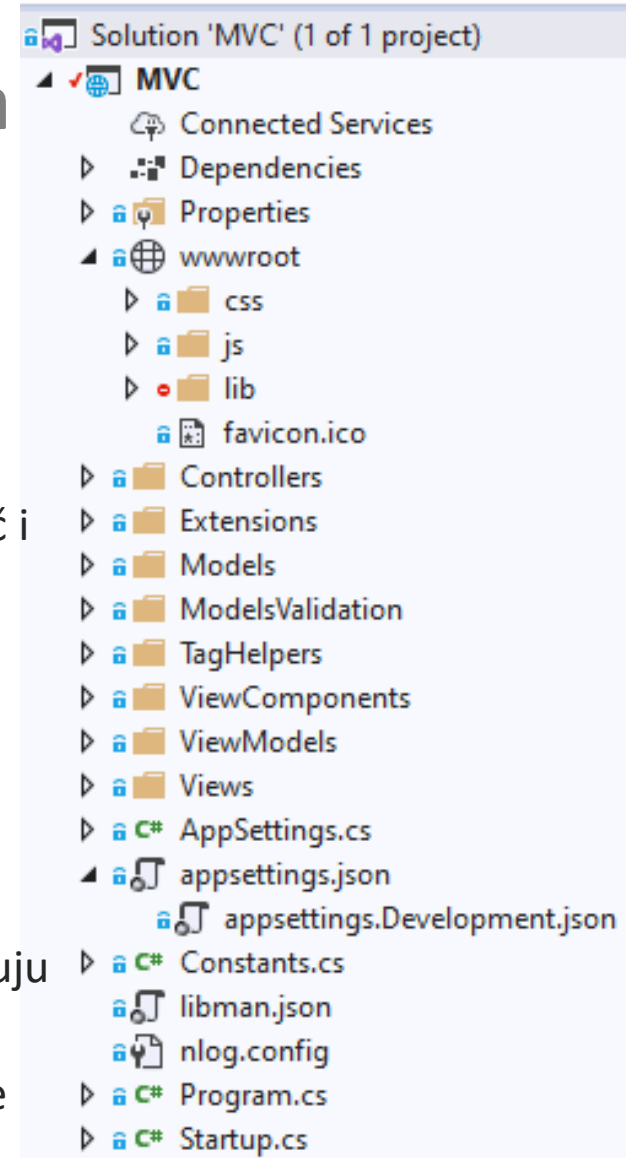
- Iz adrese zahtjeva i postavki usmjeravanja bit će odabran upravljač (razred izveden iz razreda *Controller*) na kojem će se izvesti određena akcija (postupak u odabranom razredu)
 - pozvani upravljač (najčešće) popunjava model i aktivira generiranje HTML-a temeljem pogleda u kojem su umetnute vrijednosti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

Add Client-Side Library

Provider: cdnjs

Library: jquery@3.5.1

☒ Include all library files
☐ Choose specific files:


Files:

- ☒ jquery.js
- ☒ jquery.min.js
- ☒ jquery.min.map
- ☒ jquery.slim.js
- ☒ jquery.slim.min.js
- ☒ jquery.slim.min.map

Target Location: wwwroot/lib/jquery/

Install

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

...

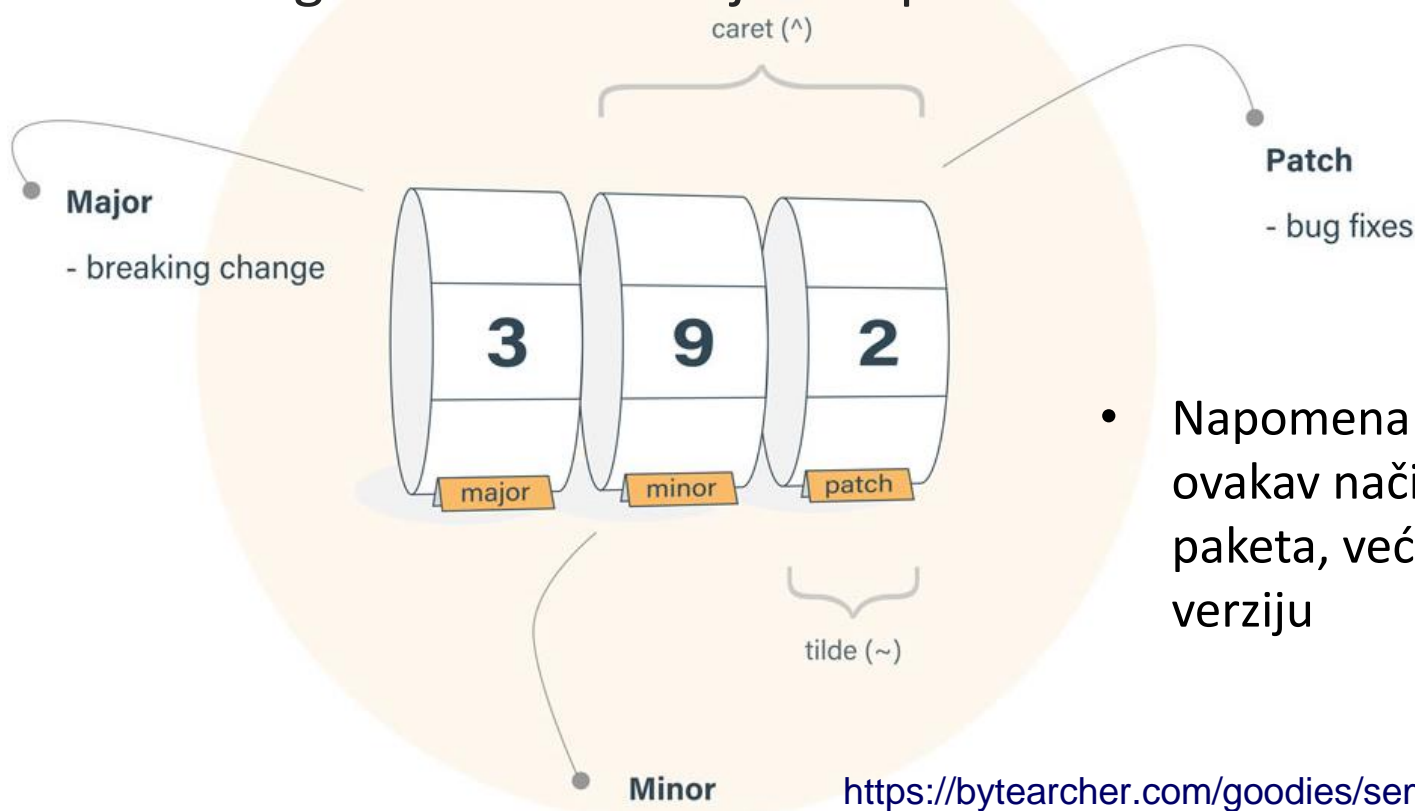
Primjer konfiguracijske datoteke za neke druge upravljače paketima

- 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*



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

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

- backwards compatible new functionality
- old functionality deprecated, but operational
- large internal refactor

Početna stranica (1)

- 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

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

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

Uobičajeni rezultati akcije upravljača (izvedeni iz IActionResult)

Tip	Opis rezultata/povratne vrijednosti	Postupak u upravljaču
ViewResult	Prikazuje pogled	View
PartialViewResult	Prikazuje parcijalni pogled	PartialView
RedirectToRouteResult	Privremeno ili trajno preusmjerava zahtjev (HTTP kod 301 ili 302), stvarajući URL na osnovu postavki usmjeravanja	RedirectToAction RedirectToActionPermanent RedirectToRoute RedirectToRoutePermanent
RedirectResult	Privremeno ili trajno preusmjerava rezultat na određeni URL	Redirect RedirectPermanent
ContentResult	Vraća tekstualni sadržaj	Content
FileResult	Vraća binarni sadržaj	File
JsonResult	Vraća objekt serijaliziran u JSON format	Json
JavaScriptResult	Vraća JavaScript odsječak	JavaScript
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
 - Konkretni vrijednosti predanog modela dobije se s *@Model*
 - Tekst izvan html oznake prefiksira se s @: ili se stavlja oznaka <text>
 - Prostori imena uključuju se s @using
 - Često korišteni mogu se dodati u datoteku Shared_ViewImports.cshtml
- Pogled može biti parcijalni - nema HTML zaglavlje niti koristi glavnu stranicu

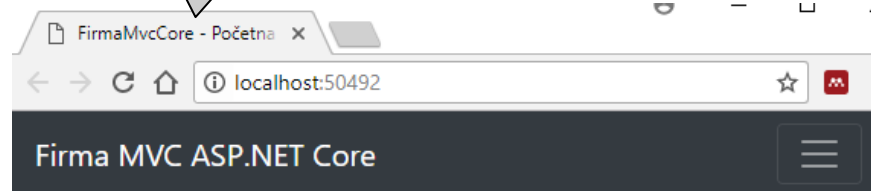
```

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

```

- 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 Core.

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 Core).


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 hijerarhijske 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

- 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

- 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 mjesta, 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
 - `referenca?.član` vraća član objekta na kojeg referenca pokazuje ili null u slučaju da je referenca null
 - `referenca.clan` bi mogao izazvati `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 Artikal -t Dokument -t Drzava -t Mjesto -t Osoba -  
t Partner -t Stavka -t Tvrtka
```

Postavke za spajanje na bazu podataka


- Automatski generirani razred za EF kontekst sadrži tvrdo kodirane postavke za spajanje na bazu podataka u postupku *OnConfiguring*
 - 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 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,  
        IOptionSnapshot<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 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)

- 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 dohvatiti svi podaci, pa se prikazuju samo određeni

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

41 - 50 od ukupno 222 zapisa


Prethodna 1 ... 4 5 6 ... 23 Sljedeća

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


```
@model 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 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 < 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
 - 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


- 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; }
```

Prihvat 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 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

- Podatak se dodaje 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

- 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 stavljaju 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")  
            .MaxLength(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 se mogu automatski pronaći i uključiti svi razredi koji nasljeđuju *AbstractValidator*
 - Primjer:  MVC \ Program.cs

```
using FluentValidation.AspNetCore;  
...  
builder.Services  
    .AddFluentValidationAutoValidation()  
    .AddFluentValidationClientsideAdapters();
```

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 pomoću 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 odrađivanja posla, preusmjerava rezultat na neku akciju
 - *TempData* sadrži tekst pogreške ili poruku o uspjehu

```
[HttpPost]
public IActionResult Delete(string OznDrzave, ...) {
    var drzava = ctx.Drzava.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 inicijalno otvaranje forme (GET zahtjev),
 - *Update* naknadno slanja popunjenih 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);  
        }  
    }  
}
```

Prihvat vrijednosti za ažuriranje

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

Asinkroni postupci (ukratko)

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

Prihvat 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 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
 - 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 mjesta

- 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 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

- 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 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

- 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

- Moguće definirati i drugačija usmjeravanja (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>