

Web-aplikacije

ASP.NET Core MVC


2016/17.09

Sortiranje i straničenje na serverskoj strani.

Dodavanje, brisanje i ažuriranje jednostavnih podataka.

Straničenje na serverskoj strani

2

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

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

URL i model za prikaz svih država

3

➡ Adresa: /Drzava/Index

➡ Upravljač: Drzava(Controller), Akcija: Index

➡ Zbog postavki usmjeravanja može i bez navođenja akcije Index

➡ Opcionalno se predaje broj stranice i redoslijed sortiranja

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

➡ Primjer:  Firma.Mvc \ ViewModels \ DrzaveViewModel.cs

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

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

Parametri sortiranja i straničenja

4

➤ Primjer:  Firma.Mvc \ Views \ Drzava \ Index.cshtml

➤ Omogućeno sortiranje i straničenje

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

```
@model FirmaMvc.Models.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? false : true : true)">
          Naziv države
        </a>
      </th>...
```

Akcija dohvata podskupa država (1)

5

► Primjer:  Firma.MVC \ Controllers \ DrzavaController.cs

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

► O svojstvu *TempData* naknadno

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

Akcija dohvata podskupa država (2)

6

► Primjer:  Firma.MVC \ Controllers \ DrzavaController.cs

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

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

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

}

Akcija dohvata podskupa država (3)

7

► Primjer:  Firma.MVC \ Controllers \ DrzavaController.cs

► Upit se formira tako da se proširi željenim sortiranjem

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

Akcija dohvata podskupa država (4)

8

► Primjer:  Firma.MVC \ Controllers \ DrzavaController.cs

► Nakon pripreme upita, rezultat izvršavanja se pohrani u listu koja postane dio modela

```
public class DrzavaController : Controller {  
    ...  
    public IActionResult Index(int page = 1, int sort = 1,  
                               bool ascending = true) {  
        int pagesize = appData.PageSize;  
        ...  
        var drzave = query  
            .Skip((page - 1) * pagesize)  
            .Take(pagesize)  
            .ToList();  
  
        var model = new DrzaveViewModel {  
            Drzave = drzave,  
            PagingInfo = pagingInfo  
        };  
        return View(model);  
    }  
}
```


Postupci za dodavanje novog podatka

9

➤ Primjer:  Web \ Firma.Mvc \ Views \ Drzava \ Index.cshtml

➤ Poveznica na akciju Create na istoimenom upravljaču

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

➤ Dva postupka naziva Create (ali zajednički pogled Create.cshtml)

➤ inicijalno otvaranje forme (GET zahtjev) – trivijalni kod, samo prikaz pogleda

➤ prihvrat popunjenih podataka (POST zahtjev)


➤ određivanje akcije na osnovi atributa *HttpGet* i *HttpPost*

➤ Primjer  Web \ Firma.Mvc \ Controllers \ DrzavaController.cs

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

Kontrole za unos novog podatka

10

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

```
@model Drzava
...
<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-default">Odustani</a>
```

Smisao atributa oblika asp-nešto

11

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

- 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

12

- Za svako svojstvo ispisuje se prikladno ime svojstva

➤ Primjer:  Firma.Mvc \ Views \ Drzava \ Create.cshtml

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

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

➤ Dodan naknadno nakon generiranja modela

➤ Primjer:  Firma.Mvc \ Models \ Drzava.cs

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

- Postupak može primiti cijeli model i/ili bilo koji broj imenovanih parametara kojima se pokušavaju pridijeliti vrijednosti primljene prilikom zahtjeva
- Povezivanje se vrši na osnovu svojstva *name* pojedine html kontrole
- Redoslijed povezivanja (prvi koji bude pronađen):
 1. `Request.Form`
 2. `RouteData.Values`
 3. `Request.QueryString`
- Korištenjem dodatnih atributa moguće eksplicitno odrediti izvor povezivanja te zanemariti gore navedeni redoslijed.
 - Detaljnije na <https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding>

Provjera ispravnosti izvora zahtjeva


14

- 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:  `Firma.Mvc \ Controllers \ DrzavaController.cs`

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

Postupak za dodavanje novog podatka


15

- 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, ali s dotad upisanim podacima (sadržani su u modelu)
- Primjer:  Firma.Mvc \ Controllers \ DrzavaController.cs

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

Pogreške pri pohrani promjena u EF modelu

16

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

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


Prijenos podataka između zahtjeva


17

- Osim popunjavanjem modela pogledu se podaci mogu predati korištenjem svojstava upravljača *ViewBag* ili *ViewState* (primjeri naknadno)
- Uspješno dodavanje države ne prikazuje pogled, već preusmjerava na pregled država
 - Nije moguće iskoristiti *ViewBag* ili *ViewState* (nema ni modela)
- Poruke između zahtjeva stavljaju se u *TempData*
 - Podaci iz *TempData* se brišu nakon dovršetka http zahtjeva
 - Interno se koristi sjednica (session), tj. podaci pohranjeni unutar sjednice
- *TempData* staviti u pogled ili u glavnu stranicu
 - Primjer: `Firma.Mvc \ Views \ Shared \ _Layout.cshtml`

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

Validacijski atributi


18

- 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:  Firma.Mvc \ Models \ Drzava.cs

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

Prikaz validacijskih pogrešaka


19

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

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

Stil prikaza validacijskih pogrešaka


20

- 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:  Firma.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;  
}
```

Provjera ispravnosti modela prije snimanja

21

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

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

Poveznica za ažuriranje države

22

➤ Primjer:  Firma.Mvc \ Views \ Drzava \ Index.cshtml

- Ažuriranje vodi na akciju *Edit* u upravljaču *Drzava*
- Postupak *Edit* u *DrzavaController* očekuje *id* kao identifikator države
 - *id* se postavlja na konkretni *IdMjesta* 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-warning btn-xs" title="Ažuriraj">
      <span class="glyphicon glyphicon-pencil"></span>
    </a>...
```

Akcije ažuriranja podataka

23

➡ Primjer:  Firma.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

24

- 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

25

➤ Nalik pogledu za unos podatka

➤ Postojeći podaci se automatski stavljaju kao *value* kontrole `<input asp-for = "nazivsvojstva" ...`

➤ Identifikator podatka (obično primarni ključ) se ne mijenja i za njega se ne generira kontrola, već se koristi skriveno polje ili (u ovom primjeru) je dio rute (adrese zahtjeva)

➤ Primjer:  Firma.Mvc \ Views \ Drzava \ Edit.cshtml

```
<form 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

26

➡ U slučaju neispravnog identifikatora vratiti NotFound

➡ korisniku se prikazuje pogreška 404

➡ Primjer:  Firma.Mvc \ Controllers \ DrzavaController.cs

```
public class DrzavaController : Controller {  
    [HttpGet]  
    public IActionResult Edit(String id,  
                                int page = 1, int sort = 1, bool ascending = true) {  
        var drzava = ctx.Drzava.Find(id);  
        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

27

- 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)

28

- 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

29

► Primjer:  Firma.Mvc \ Controllers \ DrzavaController.cs

```
public class DrzavaController : Controller {


    [HttpPost, ActionName("Edit")]
    public async Task<IActionResult> Update(String id,
        int page = 1, int sort = 1, bool ascending = true) {

        ...
        Drzava drzava = await ctx.Drzava.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();
            ...
        }
    }
}
```

Forma za brisanje podatka

30

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

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

Akcija brisanja

31


➡ Primjer:  Firma.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

32

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

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

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

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