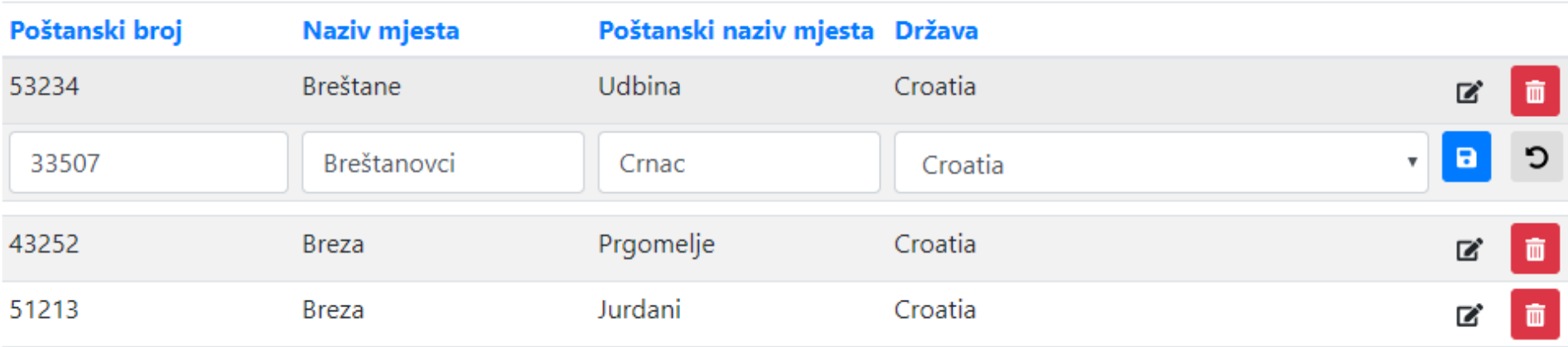
Razvoj primijenjene programske potpore

7. ASP.NET Core MVC

Parcijalni pogledi i dinamičko ažuriranje
Specijalizacija i generalizacija
Nadopunjavanje umjesto padajuće liste

Ideja primjera s dinamičkim ažuriranjem

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



- Složeniji načini interaktivnosti bi vjerojatno iziskivali korištenje radnih okvira ili biblioteka (DevExpress, Vue.Js, …), a možda i promjenu koncepta (npr. SPA + web-servisi)
  - Za jednostavne primjere bit će dovoljan https://htmx.org/

Parcijalni pogled za prikaz pojedinog mjesta (1)

- Svaki redak generira se parcijalnim pogledom uključenim u pogleda za prikaz svih mjesta.
  - Nije poziv akcije, već iscrtavanje pogleda!
  - Primjer: 🗁 MVC \ Views \ Mjesto \ Index.cshtml

```
@model MjestaViewModel
<table> ...
  <tbody>
    @foreach (var mjesto in Model.Mjesta)
    {
      <partial name="Get" model="mjesto" />
    }
  </tbody>
</table>
...
@section scripts {
  <script src="~/lib/htmx/htmx.min.js" ...
```

Parcijalni pogled za prikaz pojedinog mjesta (2)

- Svaki redak generira se parcijalnim pogledom uključenim u pogleda za prikaz svih mjesta
  - Primjer: 🗁 MVC \ Views \ Mjesto \ Get.cshtml

```
@model MjestoViewModel
<tr ...
  <td class="text-left">@Model.PostBrojMjesta</td>
  <td class="text-left">@Model.NazivMjesta</td>
  <td class="text-left">@Model.PostNazivMjesta</td>
  <td class="text-left">@Model.NazivDrzave</td>
  <td>
    ...
```

Akcija za prikaz pojedinačnog mjesta

- Primjer: 🗁 MVC \ Controllers \ MjestoController.cs

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

Izvedba dinamičkog brisanja na klijentu

- Nešto se dinamički mora dogoditi s retkom u kojem se nalazi podatak s mjestom, npr. mora se zamijeniti s kontrolama za unos ili u slučaju brisanja jednostavno ukloniti.
  - Primjer: 🗁 MVC \ Views \ Mjesto \ Get.cshtml

```
<tr hx-target="this" hx-swap="outerHTML">
  ...
  <button class="btn btn-sm btn-danger"
      hx-confirm="Obrisati mjesto?"
      hx-delete="@Url.Action(nameof(MjestoController.Delete),
                             new { id = Model.IdMjesta })"
                    title="Obriši">
              ...
```

  - Napomena: ovakva izvedba ne koristi *AntiForgeryTokena*, niti će raditi ako je isključen JavaScript, iako htmx omogućava i to, ali izlazi iz okvira ovih predavanja

Akcija brisanja mjesta (1)

- U slučaju uspjeha vraća se EmptyResult, pa će redak s mjestom biti uklonjen iz prikaza.
  - Primjer: 🗁 MVC \ Controllers \ MjestoController.cs

```
[HttpDelete]
public async Task<IActionResult> Delete (int id) {
      var mjesto = await ctx.Mjesto.FindAsync(id);
      if (mjesto != null) {
        try {
          string naziv = mjesto.NazMjesta;
          ctx.Remove(mjesto);
          await ctx.SaveChangesAsync();
          ...
          return new EmptyResult();
```

Akcija brisanja mjesta (2)

- Ako brisanje nije uspješno, onda bi rezultat trebao biti takav da redak ostane nepromijenjen
  - Primjer: 🗁 MVC \ Controllers \ MjestoController.cs

```
[HttpDelete]
public async Task<IActionResult> Delete (int id) {
      var mjesto = await ctx.Mjesto.FindAsync(id);
      if (mjesto != null) {
        try { ... }
        catch (Exception exc)  {
           return await Get(id); //može i return RedirectToAction...
        }
      }
      else {
         return await Get(id);
...
```

Biblioteka za prikaz poruka

- Što se dogodilo prilikom neke akcije prikazat će se u malom dinamičkom prozoru (toast), za što će biti iskorištena neka od dostupnih biblioteka, npr. *toastr*
  - Primjer: 🗁 MVC \ Views \ Shared \ _Layout.cshtml

```
<script src="~/lib/toastr.js/toastr.min.js"
        asp-append-version="true"></script>
<script>
    toastr.options = {
      positionClass: "toast-bottom-right"
    };
</script>
```

  - Primjer: 🗁 MVC \ libman.json

```
...
"library": "htmx@1.8.4", "destination": "wwwroot/lib/htmx/"
```

Razred za prikaz poruka

- *record* je specifičan oblik razreda
  - drugačije definiran operator Equals (usporedba svih svojstava), nadjačan *ToString*, …
  - https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/records
  - U ovom primjeru koristi se tzv. *positional records* (*init only*) te je odmah definirano kako će se pretvoriti u JSON.
  - Primjer: 🗁 MVC \ ViewModels \ ActionResponseMessage.cshtml

```
public class MessageType {
    public const string Success = "success";
    public const string Info = "success";
    public const string Warning = "warning";
    public const string Error = "error";
}
public record ActionResponseMessage(
    [property:JsonPropertyName("messageType")] string MessageType,
    [property: JsonPropertyName("message")]  string Message);
```

Akcija brisanja mjesta (3)

- Ovisno što se dogodilo stvorimo različite zapise
  - HX-Trigger u zaglavlju uzrokuje događaj na klijentu (vidi sljedeći slajd)
  - Primjer: 🗁 MVC \ Controllers \ MjestoController.cs

```
responseMessage = new ActionResponseMessage(MessageType.Success,
            $"Mjesto {naziv} sa šifrom {id} uspješno obrisano.");
...
responseMessage = new ActionResponseMessage(MessageType.Error,
                        $"Pogreška prilikom brisanja mjesta:
                              {exc.CompleteExceptionMessage()}");
...
Response.Headers["HX-Trigger"] = JsonSerializer.Serialize(
                                new { showMessage = responseMessage });

return responseMessage.MessageType == MessageType.Success ?
        new EmptyResult() : await Get(id);
```

Prikaz poruke rezultata akcije

- Definirano što će se dogoditi na vlastiti događaj *showMessage* (posljedica sadržaja zaglavlja HX-Trigger) ili na općenitu pogrešku prilikom dinamičke promjene uzrokovane htmx-om
  - Primjer: 🗁 MVC \ Views \ Mjesto \ Index.cshtml

```
@section scripts {
  <script>
    document.body.addEventListener("showMessage", function(evt){
       toastr[evt.detail.messageType](evt.detail.message);
    })
    document.body.addEventListener("htmx:responseError",
                                        function (evt) {
                                toastr["error"](evt.detail.error);
    })
  </script>
}
```

Ideja ažuriranja unutar retka

- Klikom na gumb za ažuriranje pozvati akciju na serveru koja vraća parcijalni pogled s formama za unos slično kao i kod normalnog ažuriranja, ali tako da se mijenja redak za redak.
  - Primjer: 🗁 MVC \ Views \ Mjesto \ Edit.cshtml

```
<tr hx-target="this" hx-swap="outerHTML">
  <td class="text-start">
    <input asp-for="PostBrMjesta" class="form-control" />
  </td>
  ...
  <td>
    <input type="hidden" asp-for="IdMjesta" />
    <button hx-include="closest tr"
            hx-post="@Url.Action(nameof(MjestoController.Edit))"
            class="btn btn-sm btn-primary" title="Spremi ...
    <button class="btn btn-sm cancel"
            hx-get="@Url.Action(nameof(MjestoController.Get), new {
id = Model.IdMjesta })" title="Odustani"> ...
```

Akcija za prikaz retka za ažuriranje

- Izvedba slična kao kod „normalnog” ažuriranja
  - pripremiti države za padajuću listu
  - Primjer: 🗁 MVC \ Controllers \ MjestoController.cs

```
[HttpGet]
public async Task<IActionResult> Edit (int id) {
     var mjesto = await ctx.Mjesto
                           .AsNoTracking()
                           .Where(m => m.IdMjesta == id)
                           .SingleOrDefaultAsync();
     if (mjesto != null) {
       await PrepareDropDownLists();
       return PartialView(mjesto);
     }
     else {
       return NotFound($"Neispravan id mjesta: {id}");
     }
}
```

Akcija za prihvat ažuriranih podataka

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

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

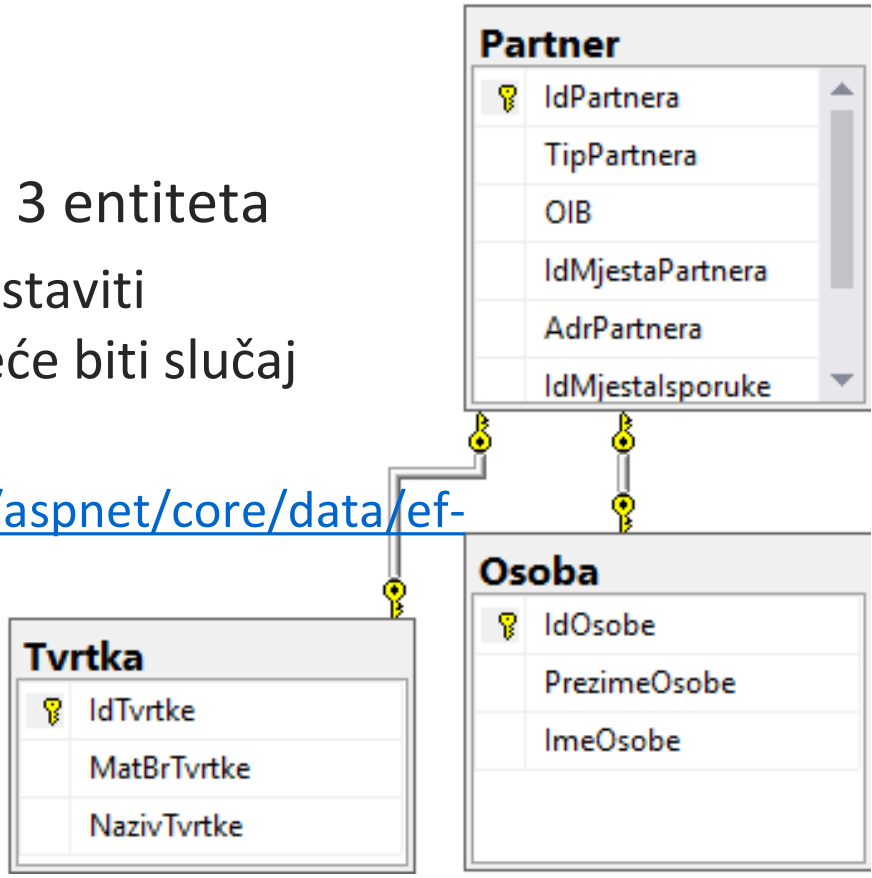
Izvedba dinamičkog ažuriranja na klijentu

- Navodi se što uzrokuje promjenu trenutnog retka.
  - Primjer: 🗁 MVC \ Views \ Mjesto \ Get.cshtml

```
<tr hx-target="this" hx-swap="outerHTML">
  <td class="text-left">@Model.PostBrojMjesta</td>
  <td class="text-left">@Model.NazivMjesta</td>
  <td class="text-left">@Model.PostNazivMjesta</td>
  <td class="text-left">@Model.NazivDrzave</td>
  <td>
    <a class="btn btn-sm"
       hx-get="@Url.Action(nameof(MjestoController.Edit),
                                  new { id = Model.IdMjesta })"
       title="Ažuriraj"><i class="fas fa-edit"></i></a>
  </td>
  ...
```

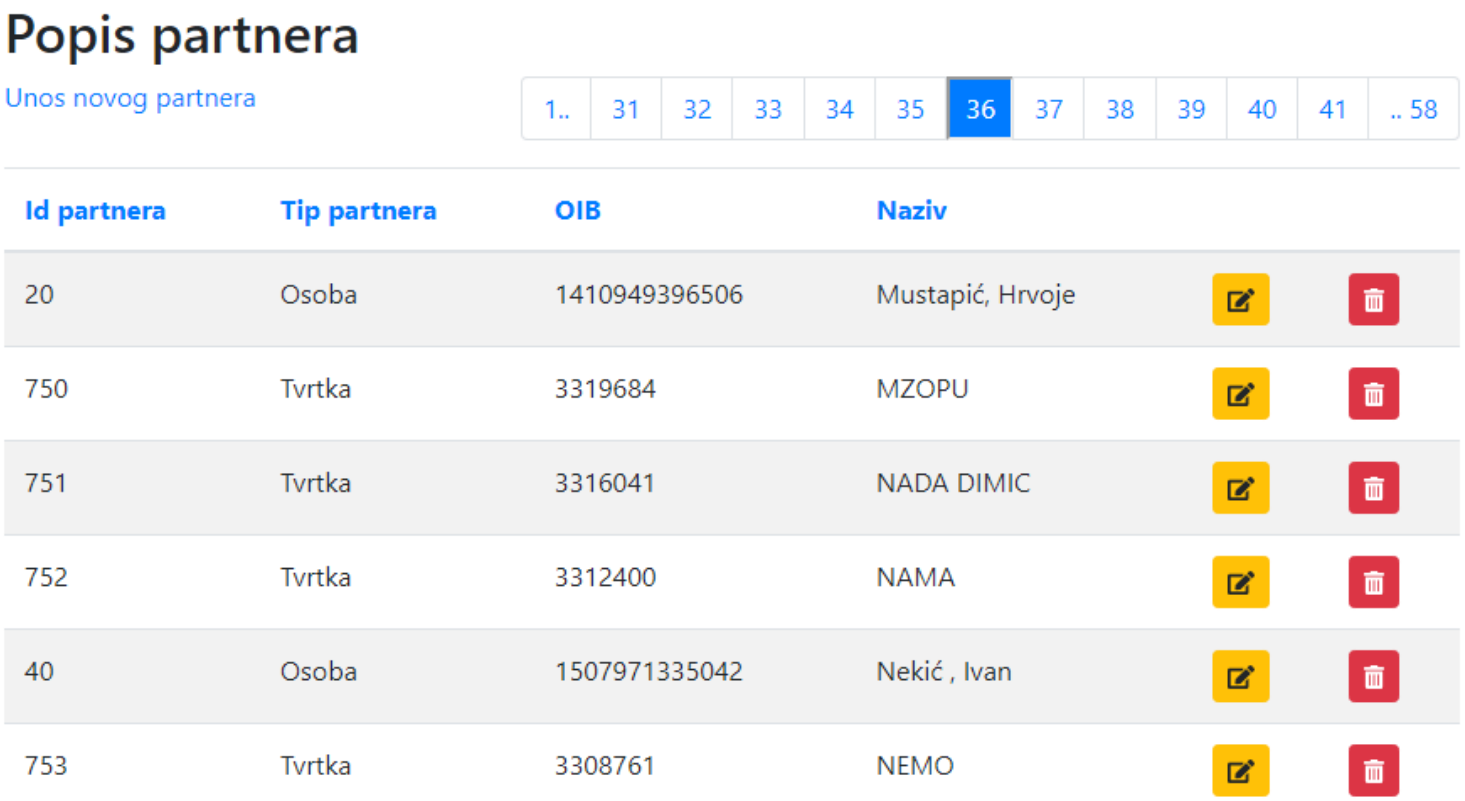
Pohrana generalizacija i specijalizacija

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



Problem prikaza specijalizacija nekog entiteta

- Što ako u prikazu svih partnera želimo ispisati id partnera, vrstu partnera, OIB i naziv partnera?
  - Upit korištenjem EF-a se komplicira i postaje neefikasan
    - potrebno dohvatiti sve osobe, pa tvrtke te napraviti uniju
  - Što ako treba sortirati podatke?



Pogled za dohvat podataka o partnerima

- Rješenje prethodnog problema je napisati pogled u bazi podataka te ga uključiti u model
  - Pogled ima sljedeću definiciju

```
CREATE VIEW [dbo].[vw_Partner]
AS
SELECT IdPartnera, TipPartnera, OIB,
       ISNULL(NazivTvrtke, NazivOsobe) AS Naziv
FROM
(
  SELECT IdPartnera, TipPartnera, OIB,
         PrezimeOsobe + ', ' + ImeOsobe AS NazivOsobe,
         NazivTvrtke
  FROM Partner
  LEFT OUTER JOIN Osoba ON Osoba.IdOsobe = Partner.IdPartnera
  LEFT OUTER JOIN Tvrtka ON Tvrtka.IdTvrtke = Partner.IdPartnera
) T
```

Uključivanje pogleda u EF-model (1)

- Kreirati razred proizvoljnog imena koji odgovara rezultatu pogleda
  - Primjer: 🗁 MVC \ ModelsPartial \ ViewPartner.cs
    - Dodatno napisano svojstvo koje opisno prikazuje tip partnera
    - Smještan u posebnu mapu, ali u isti *namespace* kao i ostatak modela

```
namespace MVC.Models;
public class ViewPartner {
   public int IdPartnera { get; set; }
   public string TipPartnera { get; set; }
   public string OIB { get; set; }
   public string Naziv { get; set; }
   public string TipPartneraText
                => TipPartnera == "O" ? "Osoba" : "Tvrtka";
}
```

Uključivanje pogleda u EF-model (2)

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

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

Model za prikaz svih partnera

- (Kao i u prethodnim primjerima) model se sastoji od enumeracije razreda kojim se opisuje pojedinačni podatak i informacija o straničenju i sortiranju
- Primjer: 🗁 MVC \ ViewModels \ PartneriViewModel.cs

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

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

Priprema modela za prikaz svih partnera

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

```
public async Task<IActionResult> Index(string filter, int page = 1,
                        int sort = 1, bool ascending = true) {
      int pagesize = appData.PageSize;
      var query = ctx.vw_Partner.AsQueryable();
      ...proširenje upita (sortiranje), provjera broja stranica
      var partneri = query
                     .Skip((page - 1) * pagesize)
                     .Take(pagesize)
                     .ToList();
      var model = new PartneriViewModel {
        Partneri = partneri,
        PagingInfo = pagingInfo
      };
      return View(model);
}
```

Proširenje upita redoslijedom sortiranja

- 🗁 MVC \ Extensions \ Selectors \ PartnerSort.cs

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

  Expression<Func<ViewPartner, object>> orderSelector = null;
  switch (sort) {
        case 1:
          orderSelector = p => p.IdPartnera;
          break;
        case 2:
          orderSelector = p => p.TipPartnera;
          break;
        ...
     }
  if (orderSelector != null)
        query = ascending ?
                query.OrderBy(orderSelector) :
                query.OrderByDescending(orderSelector);
  return query;
```

Prikaz svih partnera (1)

- Po uzoru na prethodne primjere koristi se parcijalni pogled za pojedini redak iz tablice, ali će ažuriranje biti na posebnoj stranici (jer tablica pokazuje samo dio podataka)
  - PagingInfo iz modela se kopira u ViewData/ViewBag
  - Primjer 🗁 MVC \Views \ Partner \ Index.cshml

```
@model PartneriViewModel
@{
  ViewBag.Title = "Partneri";
  var viewdata = new ViewDataDictionary(ViewData) {
          { "PagingInfo", Model.PagingInfo }
  };
}
...
 @foreach (var partner in Model.Partneri) {
     <partial name="Get" model="partner" view-data="viewdata" />
                 ...
```

Prikaz svih partnera (1)

- 🗁 MVC \Views \ Partner \ Get.cshml

```
<tr hx-target="this" hx-swap="outerHTML">
  <td class="text-left">@Model.IdPartnera</td>
  <td class="text-left">@Model.TipPartneraText</td>
  <td class="text-left">@Model.OIB</td>
  <td class="text-left">@Model.Naziv</td>
  <td>
    <a asp-action="Edit"
       asp-route-id="@Model.IdPartnera"
       asp-route-page="@ViewBag.PagingInfo.CurrentPage"
       asp-route-sort="@ViewBag.PagingInfo.Sort"
       asp-route-ascending="@ViewBag.PagingInfo.Ascending"
       ...
    <button class="btn btn-sm btn-danger"
            hx-confirm="Obrisati partnera?"
            hx-delete="@Url.Action(nameof(PartnerController.Delete),
                                   new { id = Model.IdPartnera })"
            title="Obriši"> ...
```

Stvaranje objekta kao jedne od specijalizacija

- Za prijenos podataka između pogleda i upravljača za stvaranje novog partnera definiran je novi prezentacijski model koji sadrži sve atribute osobe, ali i tvrtke
  - Alternativa: razviti dvije odvojene akcije i dva različita pogleda
  - Primjer: 🗁 MVC \ ViewModels \ PartnerViewModel.cs

```
public class PartnerViewModel {
   public int IdPartnera { get; set; }
   RegularExpression("[OT]")]
   public string TipPartnera { get; set; }
   public string PrezimeOsobe { get; set; }
   public string ImeOsobe { get; set; }
   public string MatBrTvrtke { get; set; }
   public string NazivTvrtke { get; set; }
   [Required]
   [RegularExpression("[0-9]{11}")]
   public string Oib { get; set; }
   public string AdrPartnera { get; set; }
   public int? IdMjestaPartnera { get; set; }
   public string NazMjestaPartnera { get; set; }
  ...
```

Priprema za unos novog partnera

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

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

Odabir tipa partnera (1)

- Odabir se vrši korištenjem *radiobuttona*
  - *radiobutton* ima naziv generiran na osnovi *tag-helpera asp-for* i naziva odgovarajućeg svojstva iz modela te pridruženu vrijednost
  - tekst se neovisno navodi ispred ili iza
  - Primjer: 🗁 MVC \ Views \ Partner \ Create.cshtml

```
@model PartnerViewModel
...
<form asp-action="Create" method="post">
  <input … type="radio" asp-for="TipPartnera" value="O">
  <label … asp-for="TipPartnera">Osoba</label>
  ...
  <input … type="radio" asp-for="TipPartnera" value="T">
  <label … asp-for="TipPartnera">Tvrtka</label>
```

Odabir tipa partnera (2)

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

```
@model PartnerViewModel
...
<form asp-action="Create" method="post">
  ...
  <div class="mb-3 row samotvrtka">
        <label asp-for="MatBrTvrtke" class="… ></label>
        <input asp-for="MatBrTvrtke" class="… />
  </div>
  <div class="mb-3 row samoosoba">
        <label asp-for="ImeOsobe" class="… ></label>
        <input asp-for="ImeOsobe" class="… />
  </div>
  ...
```

Odabir tipa partnera (3)

- Nakon učitavanja stranice te pri svakoj promjeni označenog radiobuttona skrivaju se odnosno prikazuje odgovarajuće kontrole
  - Primjer: 🗁 MVC\ Views \ Partner \ Create.cshtml

```
@section scripts{
    <script type="text/javascript">
        $(function () {
            $('input:radio').change(function () {
                OsobaIliTvrtka($(this).val());
            });
            OsobaIliTvrtka($('input:checked').val());
        });
        function OsobaIliTvrtka(tip) {
            if (tip == 'O') {
              $(".samotvrtka").hide(); $(".samoosoba").show();
            }
            else {
              $(".samoosoba").hide(); $(".samotvrtka").show();
            }
        }
```

Validacija prilikom unosa novog partnera (1)

- Model *PartnerViewPartner* sadrži podatke i za osobu i za tvrtku
- Atribut *Required* na imenu osobe (ili *RuleFor(p => p.ImeOsobe).NotEmpty()* koristeći FluentValidation) nema smisla ako je partner tvrtka
  - Štoviše morao bi se popuniti nekim besmislenim podatkom da bi se forma mogla poslati na server
- Potrebno napraviti dodatnu vlastitu provjeru
  - Može se napisati vlastiti atribut
  - Moguće koristiti *FluentValidation* i *When*
    - U oba slučaja moguće (iako ne baš jednostano) moguće napraviti i javascript za klijentsku validaciju
  - Implementirati sučelje IValidatableObject
    - Validacija samo na serveru, ali prihvatljivo za primjer

Validacija prilikom unosa novog partnera (2)

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

```
public class PartnerViewModel : IValidatableObject {
 public IEnumerable<ValidationResult>
            Validate(ValidationContext validationContext) {
...

   new ValidationResult("Potrebno je upisati ime osobe",
                                    new [] { nameof(ImeOsobe) } );
...
```

Validacija prilikom unosa novog partnera (3)

- Primjer: 🗁 MVC \ ViewModels \ PartnerViewModel.cs
- yield return vraća jedan po jedan rezultat
  - Može se koristiti ako je povratni tip *IEnumerable* ili *IEnumerator*

```
public IEnumerable<ValidationResult>
             Validate(ValidationContext validationContext) {
 if (TipPartnera == "O") {
    if (string.IsNullOrWhiteSpace(ImeOsobe))
        yield return new ValidationResult(
            "Potrebno je upisati ime osobe",
             new [] { nameof(ImeOsobe) } );

    if (string.IsNullOrWhiteSpace(PrezimeOsobe))
        yield return new ValidationResult(
            "Potrebno je upisati prezime osobe",
             new [] { nameof(PrezimeOsobe) } );
 }
 ...
```

Unos novog partnera (1)

- Potrebno stvoriti novi objekt tipa Partner te kopirati svojstva iz modela (uključuje i inicijalizaciju svojstva Osoba ili Tvrtka)
  - Vlastita metoda CopyValues prikazana na sljedećem slajdu
  - Primjer: 🗁 MVC \ Controllers \ PartnerController.cs

```
public async Task<IActionResult> Create(PartnerViewModel model) {
  if (ModelState.IsValid) {
       Partner p = new Partner();
       p.TipPartnera = model.TipPartnera;
       CopyValues(p, model); //kopiraj podatke iz modela u p
       try
       {
         ctx.Add(p);
         await ctx.SaveChangesAsync();
         ...
```

Unos novog partnera (2)

- Kopiraju se potrebna svojstva te stvara nova instanca Osobe ili Tvrtke
  - Primjer: 🗁 MVC \ Controllers \ PartnerController.cs

```
void CopyValues(Partner partner, PartnerViewModel model) {
      partner.AdrIsporuke = model.AdrIsporuke;
      partner.AdrPartnera = model.AdrPartnera;
      partner.IdMjestaIsporuke = model.IdMjestaIsporuke;
      partner.IdMjestaPartnera = model.IdMjestaPartnera;
      partner.Oib = model.Oib;
      if (partner.TipPartnera == "O") {
        partner.Osoba = new Osoba();
        partner.Osoba.ImeOsobe = model.ImeOsobe;
        partner.Osoba.PrezimeOsobe = model.PrezimeOsobe;
      }
      else {
        partner.Tvrtka = new Tvrtka();
        partner.Tvrtka.MatBrTvrtke = model.MatBrTvrtke;
        partner.Tvrtka.NazivTvrtke = model.NazivTvrtke;
      }
}
```

Strani ključ i *identity* tip podatka

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

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

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

Nadopunjavanje umjesto padajuće liste (1)

- Izbor mjesta partnera
  - Velik broj mogućih mjesta – nije prikladno za padajuću listu
- Koristi se nadopunjavanje (engl. *autocomplete*), odnosno dinamička padajuća lista
  - U pogledu se definira obično polje za unos kojem se pridružuje klijentski kod koji poziva određenu stranicu na serveru koja vraća tražene podatke osnovi trenutno upisanog teksta
    - Npr. za tekst `breg` stranica će vratiti sva mjesta koja u nazivu sadrže riječ `breg` (npr. Bregana, Lugarski Breg, Bregi, …)
    - Rezultat ovisi o postupku na serveru
  - Podaci koje stranica vraća bit će parovi oblika (identifikator, oznaka)
    - npr. `5489, "21000 Split"`
  - Upisani tekst predstavljat će naziv mjesta, a dohvaćeni identifikator mjesta će se pohraniti u skriveno polje: podaci su parovi oblika (identifikator, tekst)

Razred za pohranu rezultata servisa (1)

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

```
public class IdLabel {
    public string Label { get; set; }
    public int Id { get; set; }
    public IdLabel() { }
    public IdLabel(int id, string label) {
        Id = id;
        Label = label;
    }
}
```

- Pretvorbom u JSON nastat će rezultat nalik sljedećem tekstu

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

pri čemu nije sigurno hoće li nazivi svojstava biti zapisani velikim ili malim slovom

Razred za pohranu rezultata servisa (2)

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

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

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

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

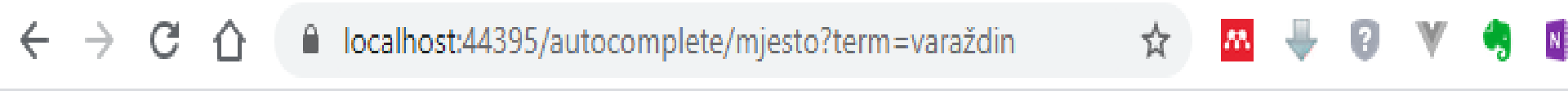
Upravljač za dohvat podataka za nadopunjavanje

- Postupak ne vraća pogled, već enumeraciju parova (id, oznaka)
  - ASP.NET Core automatski pretvara u JSON (ili xml ovisno o postavkama)
  - traži se podniz – ulazni argument se mora zvati *term*
    - u čemu tražimo podniz? uključujemo li i pretragu po poštanskog broja
  - projekcija iz skupa entiteta *Mjesto* u listu objekata tipa *IdLabel*
  - Primjer: 🗁 MVC \ Controllers \ AutoCompleteController.cs (Mjesto)

```
public async Task<IEnumerable<IdLabel>> Mjesto(string term) {
  var query = ctx.Mjesto
                 .Select(m => new IdLabel {
                   Id = m.IdMjesta,
                   Label = m.PostBrMjesta + " " + m.NazMjesta
                  })
                 .Where(l => l.Label.Contains(term));
  var list = await query.OrderBy(l => l.Label)
                        .Take(appData.AutoCompleteCount)
                        .ToListAsync();
  return list;
```

Nadopunjavanje umjesto padajuće liste (2)

- Upravljač i akciju možemo isprobati direktnim pozivom u pregledniku



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

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

Priprema i označavanje kontrola za unos

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

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

Javascript biblioteke za nadopunjavanje

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

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

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

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

Aktiviranje nadopunjavanja (1)

- Primjer: 🗁 MVC \ wwwroot \ js \ autocomplete.js
  - Za svaki element koji ima definiran vlastiti atribut data-autocomplete:
    - dohvati relativnu adresu izvora podataka (iz data-autocomplete)
    - dohvati naziv elementa kojim se prepoznaje kontrola za pohranu dohvaćene vrijednosti
      - Ako se ne navede (bit će praktično kasnije) koristi se naziv akcije
    - …

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

Aktiviranje nadopunjavanja (2)

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

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

Aktiviranje nadopunjavanja (3)

- Primjer: 🗁 MVC \ wwwroot \ js \ autocomplete.js
  - Za svaki element koji ima definiran vlastiti atribut data-autocomplete:
    - … Aktiviraj autocomplete (postavlja se adresa izvora podataka, minimalna potrebna duljina teksta za nadopunjavanje i slično)
      - akcija koja će se izvršiti odabirom nekog elementa iz liste – u primjeru tekst će se kopirati u polje za unos, a identifikator u skriveno polje (vidi sljedeći slajd)

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

Aktiviranje nadopunjavanja (4)

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

```
$("[data-autocomplete]").each(function (index, element) {
  ...
  $(element).autocomplete({
      ...
      select: function (event, ui) {
                $(element).val(ui.item.label);
                var dest = $(`[data-autocomplete-
                      placeholder='${resultplaceholder}']`);
                $(dest).val(ui.item.id);
                $(dest).data('selected-label’,
                              ui.item.label);
            }
  });
```

Pogreške prilikom dodavanja novog partnera

- Model može biti neispravan ili se može dogoditi pogreška prilikom snimanja
  - Prethodno povezani podaci su dio modela koji se vraćaju pogledu
  - Naziv mjesta je dio modela i koristi se asp-for na odgovarajućem polju, pa ga ne treba ponovo rekonstruirati
  - Primjer: 🗁 MVC \ Controllers \ PartnerController.cs

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

Dohvat podataka o partneru prilikom ažuriranja

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

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

Ažuriranje podataka o partneru (1)

- Podaci povezani kroz model se provjeravaju na validacijske pogreške (slično kao kod *Create*)
  - Ako je model ispravan, vrši se dohvat partnera iz BP te se (vlastitim postupkom) kopiraju vrijednosti iz primljenog modela u entitet iz EF
  - Primijetiti da se ne radi Include na Osoba ili Tvrtka
    - više na slajdovima koji slijede
  - Primjer: 🗁 Mvc \ Controllers \ PartnerController.cs

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

Ažuriranje podataka o partneru (2)

- Mijenjaju se sva svojstva osobe ili tvrtke
  - prilikom dohvata partnera nije uključen i dohvat podataka o osobi ili tvrki
    - stoga je svojstvo Osoba (Tvrtka) jednako null te se instancira novi objekt
  - Primjer: 🗁 Mvc \ Controllers \ PartnerController.cs

```
void CopyValues(Partner partner, PartnerViewModel model) {
      partner.AdrIsporuke = model.AdrIsporuke;
      ...
      partner.Oib = model.Oib;
      if (partner.TipPartnera == "O") {
        partner.Osoba = new Osoba();
        partner.Osoba.ImeOsobe = model.ImeOsobe;
        partner.Osoba.PrezimeOsobe = model.PrezimeOsobe;
      }
      else {
        partner.Tvrtka = new Tvrtka();
        partner.Tvrtka.MatBrTvrtke = model.MatBrTvrtke;
        partner.Tvrtka.NazivTvrtke = model.NazivTvrtke; ...
```

Snimanje promjena

- Povezani dio za osobu ili tvrtku nastao stvaranjem novog objekta, a ne ažuriranjem onog dohvaćenog iz BP
  - EF ga smatra novim objektom te bi za njega definirao insert upit
  - Eksplicitno mijenjamo stanje tog objekta iz *Added* u *Modified i postavljamo vrijednost PK* => uzrokuje *update* upit, a ne *insert*
  - Primjer: 🗁 Mvc \ Controllers \ PartnerController.cs

```
public async Task<IActionResult> Edit(PartnerViewModel model,
   ...
   var partner = ctx.Partner.Find(model.IdPartnera);
   ...
   CopyValues(partner, model);
   if (partner.Osoba != null) {
      partner.Osoba.IdOsobe = partner.IdPartnera;
      ctx.Entry(partner.Osoba).State = EntityState.Modified;
   }
   if (partner.Tvrtka != null) ...
   await ctx.SaveChangesAsync();
```

Brisanje partnera

- Definirano kaskadno brisanje u BP.
  - Prilikom generiranja EF modela ta je činjenica uzeta u obzir
- Dovoljno obrisati se entitet iz skupa Partner.
  - Odgovarajući zapis iz tablice Osoba ili Tvrtka se automatski briše
- Dohvat se može izvršiti s *Where* ili postupkom *Find* (vrijednost PK)
- Primjer: 🗁 MVC \ Controllers \ PartnerController.cs

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

  - Umjesto ctx.Remove moglo se napisati i

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

Komentar oko izvedbe upravljača za partnere

- Upravljač za partnere je primjer „debelog klijenta” i poslužit će kao motivacija za naknadnu diskusiju oko uslojavanja
- Umjesto prezentacijskom logikom, upravljač se bavi i podatkovnom logikom odnosno isprepleten je tehnikalijama oko EF-a
- Bolje rješenja bi bilo napraviti proceduru za spremanje partnera i/ili izdvojiti manipulaciju podacima u zasebnu komponentu