

1

Ponovna upotreba i uslojavanje aplikacije

2018/19.14b

Ponovna upotreba softvera

2

- engl. *software reuse*
- Kako već razvijeni softver upotrijebiti za neku drugu svrhu ili neke druge korisnike?
 - potprogrami, komponente
 - oblikovni obrasci
 - aplikacijski okviri
 - standardne programske biblioteke
 - generatori programa
 - web-servisi
 - „gotovi” proizvodi spremni za prilagodbu (npr. SAP)
 - prilagodbe naslijeđenih (baštinjenih, engl. *legacy*) sustava
 - ...

Podjela ponovne upotrebe po veličini dijelova

3

➤ Ponovna upotreba cijelog sustava

- postojećem sustavu se mijenja konfiguracija te se prilagođava novom korisniku uz eventualne manje promjene

➤ Ponovna upotreba dijela sustava

- dio postojećeg sustava srednje veličine (komponente razvijene za ponovnu upotrebu, web-servisi opće namjene) se ugrađuje u novi sustav

➤ Ponovna upotreba funkcije ili klase

- manji dio postojećeg sustava (određene funkcije i klase iz biblioteka općih namjena) se ugrađuje u novi sustav

Podjela ponovne upotrebe po načinu ponovne upotrebe

4

- Ponovna upotreba programskog koda ili izvršivog programa
 - već napisani softver uključuje se u novi sustav
 - biblioteke, komponente, web-servisi, nove klase izvedene iz postojećih
- Ponovna upotreba modela ili idejnog rješenja
 - koriste se poznati oblikovni obrasci, iz postojećih UML dijagrama generira se programski kod

- Jedna ili više funkcionalnih komponenti (ili cijeli sustav) s kojim se komunicira putem javno objavljenih i precizno definiranih sučelja
 - mrežno dostupan podatkovni i/ili računalni resurs
 - osigurava mehanizam za pozivanje udaljenih postupaka
 - prima jedan ili više zahtjeva i vraća jedan ili više odgovora
- Princip crne kutije
- Heterogeni klijenti
- Koristi otvorene web standarde:
 - HTTP, XML, JSON, SOAP, OData ...

- SOA = Servisno orijentirana arhitektura
 - engl. Service Oriented Architecture
 - omogućava izradu distribuiranih aplikacija između različitih platformi koje komuniciraju razmjenom podataka među servisima
- Skup precizno definiranih, međusobno neovisnih servisa povezanih u logički jedinstvenu aplikaciju
 - objektno orijentirana aplikacija povezuje objekte
 - servisno orijentirana aplikacija povezuje servise
- Distribuirani sustav u kojem sudjeluje više autonomnih servisa međusobno šaljući poruke preko granica
 - granice mogu biti određene procesom, mrežom, ...

Problemi i preporuke prilikom izradu servisa

7

- Sigurnost komunikacije
- Konzistentnost stanja
 - neuspjeh prilikom izvršavanja servisa ne smije ostaviti sustav u stanju pogreške
- Problem višenitnosti
- Pouzdanost i robusnost servisa
 - klijent treba znati je li servis primio poruku.
 - pogreške u servisu treba obraditi
- Interoperabilnost
 - Tko sve može pozvati servis?
- Skalabilnost
- Brzina obrade postupka

Osnovna pravila servisno orijentirane arhitekture

8

- Jasno određene granice
 - jasno iskazana funkcionalnost i struktura podataka
 - implementacija = crna kutija
- Neovisnost servisa
 - servis ne ovisi o klijentu, nekom drugom servisu, lokaciji i vrsti instalacije
 - verzije se razvijaju neovisno o klijentu. Objavljene verzije se ne mijenjaju.
- Ugovor, a ne implementacija
 - korisnik servisa i implementator servisa dijele samo listu javnih postupaka i definiciju struktura podataka
 - dijeljeni podaci trebaju biti tipovno neutralni
 - tipovi specifični za pojedini jezik moraju se moći pretvoriti u neutralni oblik i obrnuto
 - implementacijski postupci ostaju tajna
- Semantika, a ne samo sintaksa
 - logička kategorizacija servisa, smisleno imenovanje postupaka

Standardi za web servise - SOAP

9

➤ SOAP – Simple Object Access Protocol

- donedavno *de facto* standard za izradu web servisa i razmjenu informacija u distribuiranim, heterogenim okruženjima \approx HTTP POST + XML
- orijentiran na akcije (engl. action driven)
- sadržaj zahtjeva koji postupak web-servisa treba pozvati

➤ WSDL - Web Services Description Language

- XML shema za opis SOAP web-servisa
- definira format postupaka koje pruža Web servis

➤ UDDI - Universal Description, Discovery, and Integration

- propisuje način dokumentiranja servisa Discovery (URI i WSDL opisi) koji bi se objavljivali u registracijskim bazama (npr. <http://uddi.xml.org>)
- ideja UDDI napuštena

➤ WS-*

- skup standarda za sigurnost, podatke, opise i koordiniranje SOAP web-servisa

Standardi za web servise - REST

10

- REST = Representational State Transfer
 - alternativa SOAP protokolu
 - nije protokol, već način izrade servisa (konceptualno ne mora biti baziran nad HTTP protokolom)
 - orijentiran na resurse (engl. *resource driven*)
- REST web-servisi izlažu „resurse”.
 - adresa servisa jednoznačno određuje resurs
- Akcije određene vrstom HTTP zahtjeva
 - GET – dohvat podataka
 - POST - stvaranje novog podatka
 - PUT i PATCH – izmjene cijelog ili dijela podatka
 - DELETE – brisanje podatka
- Standard za dokumentiranje OpenAPI (Swagger)

REST vs SOAP

11

- Slanjem SOAP poruke na pristupnu točku provjerava se sadržaj i na osnovu sadržaja se određuje postupak koji se treba izvršiti
- Korištenjem REST-a postupak se određuje na osnovu url-a i http metode (GET, POST, DELETE, PUT)
- Prednosti REST-a u odnosu na SOAP
 - veći broj potencijalnih klijenata
 - dovoljna je podrška za Http i XML ili JSON
 - nije potrebno implementirati složene WS-* standarde
 - lakše cacheiranje
 - manje poruke
 - POX (Plain old XML) – XML poruke bez SOAP zaglavlja
 - JSON – (JavaScript Object Notation)
- Nedostatak : nema wsdl-a, veća mogućnost nepravilne poruke
 - problem se praktično rješava alatima za dokumentiranje servisa (Swagger i OpenAPI) i generatorima klijentskih razreda

REST i vrste HTTP poziva

12

➤ GET

- služi za dohvat podataka
 - dohvat svih podataka
 - dohvat jednog podatka po primarnom ključu
 - (opcionalno) dohvat na osnovu kriterija pretrage
- vraća rezultat ili HTTP status 404

➤ POST

- služi za kreiranje novog podatka
- za uspješno stvoreni podatak vraća HTTP status 201

➤ PUT/PATCH

- služi za ažuriranje cijelog podatka (PUT) ili dijela podatka (PATCH)
- nakon uspješnog ažuriranja vraća HTTP status 200 ili 204

➤ DELETE

- služi za brisanje podatka
- ako je podatak uspješno obrisani ili je već ranije bio obrisani vraća HTTP status 200 ili 204

➤ U kompletu čini serverski Web API


Kostur Web API upravljača

13

```
[Route("api/[controller]")]
public class ValuesController : BaseController {
    // GET: api/values
    [HttpGet]
    public IEnumerable<string> Get() {
        return new string[] { "value1", "value2" };
    }
    // GET api/values/5
    [HttpGet("{id}")]
    public string Get(int id) {
        return "value";
    }
    // POST api/values
    [HttpPost]
    public void Post([FromBody]string value) { }
    // PUT api/values/5
    [HttpPut("{id}")]
    public void Put(int id, [FromBody]string value) { }
    // DELETE api/values/5
    [HttpDelete("{id}")]
    public void Delete(int id) { }
```

Uslojavanje programskog koda

14

- Primjer:  Web\Firma.Mvc\...\ArtiklController.cs
- U navedenom primjeru upravljač vrši prihvata ulaznih argumenata, dohvat podataka i pripremu modela
 - problem je u načinu dohvat podataka
 - umjesto na „što”, upravljač u prethodnom primjeru fokusiran na „kako” (slaganje EF upita)
 - „prepametn” upravljač → debeli klijent
 - Što ako umjesto EF-a treba koristiti neku drugu tehniku pristupa podacima?
 - Hoće li entiteti i dalje biti isti?
 - ...i hoće li ih biti ako se ne koristi ORM?
 - Što ako su podaci agregirani iz više izvora?
 - ...

```
public IActionResult Index(int page = 1, int sort = 1,
{
    int pagesize = appData.PageSize;
    var query = ctx.Artikl.AsNoTracking();
    int count = query.Count();

    var pagingInfo = new PagingInfo
    {
        CurrentPage = page,
        Sort = sort,
        Ascending = ascending,
        ItemsPerPage = pagesize,
        TotalItems = count
    };
    if (page < 1)
    {
        page = 1;
    }
    else if (page > pagingInfo.TotalPages)
    {
        return RedirectToAction(nameof(Index), new { page
    }

    System.Linq.Expressions.Expression<Func<Artikl, obje
    switch (sort)
    {
        case 1:
            orderSelector = a => a.SlikaArtikla; //ima smisl
            break;
        case 2:
            orderSelector = a => a.SifArtikla;
            break;
        case 3:
            orderSelector = a => a.NazArtikla;
            break;
        case 4:
            orderSelector = a => a.JedMjere;
            break;
        case 5:
            orderSelector = a => a.CijArtikla;
            break;
        case 6:
            orderSelector = a => a.ZastUsluga;
            break;
    }
    if (orderSelector != null)
    {
        query = ascending ?
            query.OrderBy(orderSelector) :
            query.OrderByDescending(orderSelector);
    }

    var artikli = query
        .Select(a => new ArtiklViewModel
        {
            SifraArtikla = a.SifArtikla,
            NazivArtikla = a.NazArtikla,
            JedinicaMjere = a.JedMjere,
```

Repozitoriji umjesto konkretnog ORM-a? (1)

15

- Zamjenom ORM alata može se pretpostaviti da će i dalje postojati isti koncepti i entiteti u neznatno izmijenjenom obliku
 - entiteti su posljedica modela baze podataka
- Konkretni objekti tipa `DbSet<T>` iz EF-a mogu se zamijeniti sučeljima s postupcima za CRUD operacije - *repozitoriji*
 - kontekst iz EF-a se tako mijenja sa sučeljem koje koordinira više repozitorija i predstavlja jedinstveni kontekst pristupa podacima - *Unit of Work*
 - ako se i ne koristi ORM alat, mogu postojati repozitoriji i Unit of Work
 - u tom slučaju entiteti su zamijenjeni razredima kojima se opisuju izlazni podaci iz postupaka
- Upravljači tada ovise o sučeljima repozitorija, ali konkretna implementacija se umetne tehnikom *Dependency Injection*
 - u ovom slučaju rješava samo dio problema
 - skriva način perzistencije objektnog modela u relacijsku bazu i djelomično olakšava testiranje programa
 - moguće napisati testnu implementaciju repozitorija

Repozitoriji umjesto konkretnog ORM-a? (2)

16

- Nije promijenjen način rukovanja repozitorijima iz upravljača
 - prethodno prikazani upravljač bi se neznatno promijenio – i dalje bi slagao upit, ali ovaj put nad nekim nepoznatim repozitorijem
 - Koji je tip povratne vrijednosti kod operacija čitanja podataka?
 - `IQueryable<T>` bi omogućio daljnje upite (filtriranje, sortiranje, ...)
 - Iz čega je nastao taj `IQueryable`? Što sve podržava?
 - `IEnumerable<T>` - nije li možda upit već evaluiran, pa su svi podaci morali biti dovučeni u memoriju?
- Što ako podaci nisu u bazi podataka ili su agregirani iz više izvora?
 - kako dodatno oblikovati upit po želji?
- Bilo bi dobro kad bi upravljač pozivao jedan postupak koji bi mu vratio upravo one podatke koje treba
 - krivi smjer rješenja: proširiti repozitorije s nekoliko metoda koje primaju različite parametre
 - traži po nekoj vrijednosti, po kombinaciji vrijednosti, vrati samo dio složene po nekom kriteriju, ...
 - stvara prevelike repozitorije koje nije lako implementirati i otežava održavanje i testiranje

Odvajanje upita od naredbi

17

➤ Command-query separation

- odvojena sučelja za čitanje podataka (upiti, engl. queries) od onih koji mijenjaju podatke (naredbe, engl. commands)

- ovisno o rješenju mogu koristiti različita spremišta

- u implementaciji i naredbe i upiti mogu koristiti repozitorije

➤ Više upita u istom sučelju narušava SOLID principe

- *Single Responsibility, Open/Closed i Interface Segregation Principle*

→ Svaki upit predstavljen jednim sučeljem

➤ „Upit” specificira tip rezultata tog upita za neke ulazne podatke

➤ ne mora nužno imati argumente (svojstva)

➤ može se opisati generičkim sučeljem

➤ Primjer:  CommandQueryCore \ IQuery.cs

```
public interface IQuery<TResult> {}
```

➤ Primjeri opisa upita  Firma.DataContact \ Queries \ ...

➤ opis upita koji treba vratiti broj država ovisno o traženom tekstu za pretragu

```
public class DrzavaCountQuery : IQuery<int> {  
    public string SearchText { get; set; }  
}
```

➤ opis upita koji vraća podatke o državi sa zadanom oznakom

```
public class DrzavaQuery : IQuery<DrzavaDto> {  
    public DrzavaQuery(string oznDrzave) {  
        OznDrzave = oznDrzave;  
    }  
    public string OznDrzave { get; set; }  
}
```

Primjer složenijeg upita

19

➡ Primjeri opisa upita  Firma.DataContact \ Queries \ ...


➡ opis upita koji treba vratiti podskup država poredanih po određenim atributima i u ovisnosti o postavljenom tekstu pretrage

```
public class DrzaveQuery : IQuery<IEnumerable<DrzavaDto>>
{
    public string SearchText { get; set; }
    public int? From { get; set; }
    public int? Count { get; set; }
    public SortInfo Sort { get; set; }
}
```

```
public class SortInfo
{
    public enum Order {
        ASCENDING, DESCENDING
    }
    public List<KeyValuePair<string, Order>> ColumnOrder
    { get; set; } = new List<KeyValuePair<string, Order>>();
}
```

Rukovatelj upitom

20

- Upit (engl. *query*) opisan razredom `IQuery<TResult>` bit će izvršen u nekom rukovatelju upita (engl. *query handler*)
- Sučeljem se standardizira kako rukovatelji upita izgledaju
 - može se definirati „obična” i asinkrona varijanta
 - Primjer:  `CommandQueryCore \ IQueryHandler.cs`

```
public interface IQueryHandler<TQuery, TResult> where TQuery : IQuery<TResult> {  
    TResult Handle(TQuery query);  
    Task<TResult> HandleAsync(TQuery query);  
}
```

- Upit predstavlja opis upita (ulazne podatke i povratnu vrijednost), a rukovatelj izvršava tako opisani upit
 - preciznije, sučelje propisuje implementaciju rukovatelja određenim upitom

Primjeri rukovatelja upitom

21

➤ Primjer sučelja Firma.DataContact \ QueryHandlers \ ...

➤ Opis rukovatelja upitom za dohvat broja država

- obrađuje upit postavljen razredom DrzavaCountQuery i mora isporučiti cijeli broj kao rezultat

```
public interface IDrzavaCountQueryHandler : IQueryHandler<DrzavaCountQuery, int> { }
```

➤ Opis rukovatelja upitom za dohvat države s određenom oznakom

- obrađuje upit postavljen razredom DrzavaQuery i mora isporučiti podatkovni objekt s podacima o traženoj državi

```
public interface IDrzavaQueryHandler : IQueryHandler<DrzavaQuery, DrzavaDto> { }
```

Objekti koji „putuju” kroz slojeve

22

- DTO – Data Transfer Objects
- Čisti, podatkovni razredi koji služe za prijenos podataka između slojeva
- U jednostavnim primjerima dolazi do dupliciranja istih razreda, ali potrebno zbog potencijalnih promjena u budućnosti
 - promjene naziva u bazi podataka ili oblika spremišta ne bi smjela utjecati na opise podataka koji su dogovoreni s konzumentima web-servisa
 - može se koristiti AutoMapper ili neki drugi alat za automatsko kopiranje vrijednosti iz objekta jednog tipa u objekt drugog tipa
 - posebno praktično ako su nazivi svojstava isti

Primjeri implementacije rukovatelja upitom

23

➡ Primjer sučelja  Firma.DAL \ QueryHandlers \ ...

➡ dohvaća konkretnu državu i vraća odgovarajući DTO

```
public class DrzavaQueryHandler : IDrzavaQueryHandler {  
    private readonly FirmaContext ctx; private readonly IMapper mapper;  
    public DrzavaQueryHandler(FirmaContext ctx, IMapper mapper) {  
        this.ctx = ctx;  
        this.mapper = mapper;  
    }  
    public DrzavaDto Handle(DrzavaQuery query) {  
        var drzava = ctx.Drzava.Find(query.OznDrzave);  
        if (drzava != null)  
        {  
            var dto = mapper.Map<Drzava, DrzavaDto>(drzava);  
            return dto;  
        }  
        else  
            return null;  
    }  
}
```

Korištenje rukovatelja upitom iz upravljača

24

- Upravljač ovisi o sučelju, a konkretnu implementaciju rukovatelja upitom prima u konstruktoru preko DI-a

➤ Primjer  Firma.WebApi \ Controllers \ DrzavaController.cs

```
public class DrzavaController...
    public DrzavaController(IDrzaveQueryHandler drzaveHandler,
                           IDrzavaQueryHandler drzavaHandler,
                           IDrzavaCountQueryHandler drzavaCountQueryHandler, ...
    {
        this.drzavaHandler = drzavaHandler;
        ...
    }
    public async Task<IActionResult> Get(string oznDrzave) {
        var drzava = await drzavaHandler.HandleAsync(new DrzavaQuery(oznDrzave));
        if (drzava == null)
            return NotFound("Tražena država ne postoji");
        else {
            var model = mapper.Map<DrzavaDto, Drzava>(drzava);
            return Ok(model);
        }
    }
}
```


Postavljanje ovisnosti

25

- Povezivanje za DI postavljano u razredu Startup web-aplikacije

- Primjer  Firma.WebApi \ Startup.cs

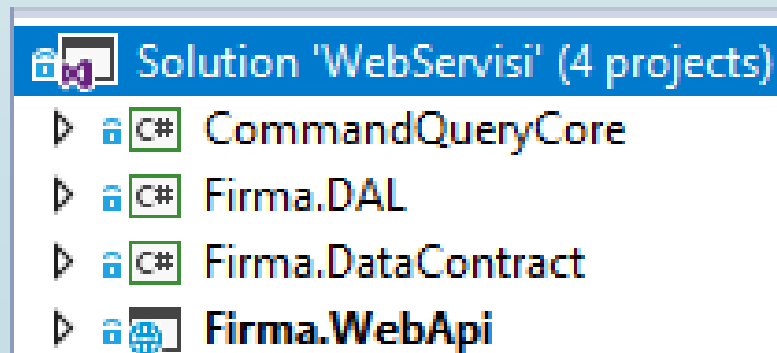
```
public class Startup {  
    public void ConfigureServices(IServiceCollection services) {  
        services.AddMvc();  
        string connectionString = Configuration.GetConnectionString("Firma");  
        connectionString = connectionString.Replace("sifra",  
                                                    Configuration["FirmaSqlPassword"]);  
  
        services.AddDbContext<FirmaContext>(options =>  
                                            options.UseSqlServer(connectionString));  
  
        services.AddTransient<IDrzaveQueryHandler, DrzaveQueryHandler>();  
        services.AddTransient<IDrzavaQueryHandler, DrzavaQueryHandler>();  
        ...  
    }  
}
```

- Umjesto repetitivnog pisanja može se riješiti refleksijom tražeći i registrirajući sve klase koje implementiraju neki *IQueryHandler<,>* iz *typeof(ArtiklQueryHandler).Assembly*

Međusobne ovisnosti projekata

26

- Smisao pojedinog projekta
 - *CommandQueryCore* – generička sučelja za upit, naredbu i njihove rukovatelje
 - *Firma.DataContract* – opis svih upita koji se koriste u rješenju i pomoćni razreda koji služe za razmjenu podataka s pozivateljem
 - Data transfer objekti (Value object) – samo sadrže vrijednosti
 - *Firma.Dal* – implementacija postupaka iz *Firma.DataContract*
- Nijedan element web-aplikacije (upravljači, pogledi, ...) osim razreda *Startup* ne zna za projekt *Firma.DAL* i ovise samo o *Firma.DataContract*
 - *Startup.cs* postavlja Dependency Injection pa stoga mora postojati referenca iz projekta *Firma.WebApi* na *Firma.DAL*



➡ „Naredba” predstavlja podatke koje neki rukovatelj akcijom treba zaprimiti i odraditi

➡ rukovatelja se može se opisati generičkim sučeljem

➡ Primjer:  CommandQueryCore \ ICommandHandler.cs

```
public interface ICommandHandler<TCommand> {  
    Task HandleAsync(TCommand command);  
}
```

➡ Sama naredba je podatkovni objekt

➡ Primjer:  Firma.DataContact \ Commands \ AddDrzava.cs

```
public class AddDrzava {  
    public string OznDrzave { get; set; }  
    public string NazDrzave { get; set; }  
    public string Iso3drzave { get; set; }  
    public int? SifDrzave { get; set; }  
}
```

Implementacija rukovatelja naredbom

28


► Implementacija naredbe u podatkovnom sloju

► primjeri  Firma.DAL \ CommandHandlers \ DrzavaCommandHandler.cs

```
public class DrzavaCommandHandler : ICommandHandler<DeleteDrzava>,
                                   ICommandHandler<AddDrzava>, ICommandHandler<UpdateDrzava> {
    private readonly FirmaContext ctx; private readonly IMapper mapper;
    public DrzavaCommandHandler(FirmaContext ctx, IMapper mapper) {
        this.ctx = ctx;
        this.mapper = mapper;
    }
    public async Task HandleAsync(AddDrzava command) {
        var drzava = new Drzava {
            Iso3drzave = command.Iso3drzave,
            NazDrzave = command.NazDrzave,
            OznDrzave = command.OznDrzave,
            SifDrzave = command.SifDrzave
        };
        ctx.Add(drzava);
        await ctx.SaveChangesAsync();
    }
}
```

Primjer Web API servisa – Web API za države


29

- Definira se slično kao i ostali MVC upravljači, ali ima definirano vlastito usmjeravanje te nasljeđuje ControllerBase
 - Ovisnosti o sučeljima i razredima definirane u konstruktoru
 - Primjer:  Firma.WebApi \ Controllers \ DrzavaController.cs

```
[Route("api/[controller]")]
public class DrzavaController : ControllerBase{
    ...
    public DrzavaController(IDrzaveQueryHandler drzaveHandler,
                           IDrzavaQueryHandler drzavaHandler,
                           IDrzavaCountQueryHandler drzavaCountQueryHandler,
                           ICommandHandler<AddDrzava> addHandler,
                           ICommandHandler<UpdateDrzava> updateHandler,
                           ICommandHandler<DeleteDrzava> deleteHandler,
                           IMapper mapper) {
        ...
    }
}
```

Web API – dohvat svih država (1)

30

- Naziv postupka nebitan – postupak se određuje prema atributu `HttpGet` i usmjeravanju upravljača
 - Konkretno u ovom primjeru `/api/drzava/sve`
 - Poželjno koristiti zasebni model neovisno o sličnosti s modelom iz EF-a
 - Omogućava naknadno neovisnu izmjenu podatkovnog sloja
 - Primjer:  `Firma.WebApi \ Controllers \ DrzavaController.cs`

```
[HttpGet("sve")]
public async Task<List<Drzava>> GetAll() {
    var query = new DrzaveQuery();
    query.Sort = new SortInfo();
    query.Sort.ColumnOrder.Add(new KeyValuePair<string, SortInfo.Order>(
        nameof(DrzavaDto.NazDrzave), SortInfo.Order.ASCENDING));
    List<Drzava> list = new List<Drzava>();
    var data = await drzaveHandler.HandleAsync(query);
    foreach (var drzava in data)
        list.Add(mapper.Map<DrzavaDto, Drzava>(drzava));
    return list;
}
```

Web API – dohvat svih država (2)

31

- ➡ `http://.../api/drzava/sve` vraća JSON sadržaj s popisom svih država
- ➡ Popis država serijaliziran u JSON (konfiguriranje se može postaviti i drugi format, npr. XML)



```
< > ↻ 🏠 🔒 https://localhost:44347/api/drzava/sve
[{"OznDrzave": "AD", "NazDrzave": "Andora", "Iso3drzave": "AND", "SifDrzave": 2013},
{"OznDrzave": "AO", "NazDrzave": "Angola", "Iso3drzave": "AGO", "SifDrzave": 24},
{"OznDrzave": "AR", "NazDrzave": "Argentina", "Iso3drzave": "ARG", "SifDrzave": 32},
{"OznDrzave": "AM", "NazDrzave": "Armenia", "Iso3drzave": "ARM", "SifDrzave": 51}, {"OznDrzave": "Ohz", "NazDrzave": "aTN", "Iso3d
{"OznDrzave": "BS", "NazDrzave": "Bahamas", "Iso3drzave": "BHS", "SifDrzave": 44},
{"OznDrzave": "BD", "NazDrzave": "Bangladesh", "Iso3drzave": "BGD", "SifDrzave": 50},
{"OznDrzave": "BB", "NazDrzave": "Barbados", "Iso3drzave": "BRB", "SifDrzave": 52},
{"OznDrzave": "BY", "NazDrzave": "Belarus", "Iso3drzave": "BLR", "SifDrzave": 112},
{"OznDrzave": "BE", "NazDrzave": "Belgium", "Iso3drzave": "BEL", "SifDrzave": 56},
{"OznDrzave": "BZ", "NazDrzave": "Belize", "Iso3drzave": "BLZ", "SifDrzave": 84}, {"OznDrzave": "BJ", "NazDrzave": "Benin", "Iso3d
{"OznDrzave": "BM", "NazDrzave": "Bermuda", "Iso3drzave": "BMU", "SifDrzave": 60},
{"OznDrzave": "BT", "NazDrzave": "Bhutan123", "Iso3drzave": "BTN", "SifDrzave": 64},
{"OznDrzave": "BO", "NazDrzave": "Bolivia", "Iso3drzave": "BOL", "SifDrzave": 68}, {"OznDrzave": "BA", "NazDrzave": "Bosnia and
Herzegovina", "Iso3drzave": "BIH", "SifDrzave": 71}, {"OznDrzave": "BW", "NazDrzave": "Botswana", "Iso3drzave": "BWA", "SifDrzave"
}
```


- ➡ U primjeru postavljeno da odredišni JSON zadrži PascalCase za svojstva (inače camelCase)
- ➡ Primjer 📁 Firma.WebApi \ Startup.cs

```
public void ConfigureServices(IServiceCollection services) {
    services.AddMvc()
        .AddJsonOptions(options =>
            options.SerializerSettings.ContractResolver =
                new DefaultContractResolver());
}
```

Web API – dohvat države s određenom oznakom

32


➡ Putanja oblika /api/drzava/oznakadrzave (npr. /api/drzava/BA)

- ➡ Usmjeravanje imenovano posebnim nazivom kao bi se moglo naknadno referencirati (vidi primjer za stvaranje nove države)
- ➡ Vraća jednu državu, ali povratna vrijednost tipa IActionResult
 - ➡ Ako država ne postoji, potrebno vratiti status 404 (Not found)
 - ➡ Od ASP.NET Core 2.1 moguće i ActionResult<T>
- ➡ Primjer:  Firma.WebApi \ Controllers \ DrzavaController.cs

```
[HttpGet("{oznDrzave}", Name = "DohvatiDrzavu")]
public async Task<IActionResult> Get(string oznDrzave) {
//ili public async Task<ActionResult<Drzava>> Get(string oznDrzave) {
    var drzava = await drzavaHandler.HandleAsync(new DrzavaQuery(oznDrzave));
    if (drzava == null)
        return NotFound("Tražena država ne postoji");
    else {
        var model = mapper.Map<DrzavaDto, Drzava>(drzava);
        return Ok(model); //status 200 + podatak
    }
}
```


Web API – dodavanje države

33

- Postupak POST proizvoljnog imena u kojem se model rekonstruira iz tijela zahtjeva
 - Ako je model ispravan posprema podatak u bazu podataka i vraća 201
 - Uz status vraća pohranjeni podatak i adresu podatka
 - Adresa je dio zaglavlja odgovora, podatak je u tijelu
 - Neispravni model uzrokuje statusnu poruku 400 (BadRequest)
 - Primjer:  Firma.WebApi \ Controllers \ DrzavaController.cs

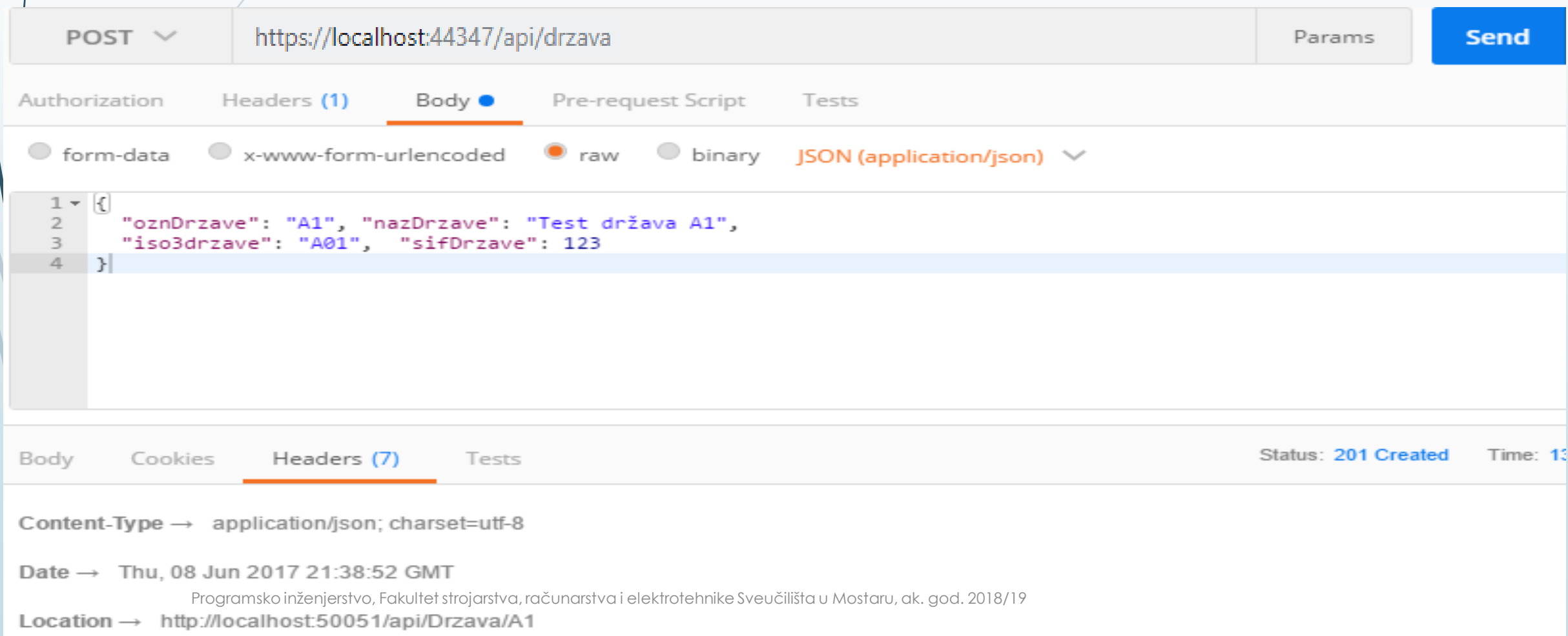
[HttpPost]

```
public async Task<IActionResult> Create([FromBody] Drzava model) {  
    if (model != null && ModelState.IsValid) {  
        AddDrzava addCommand = mapper.Map<Drzava, AddDrzava>(model);  
        await addHandler.HandleAsync(addCommand);  
        return CreatedAtRoute("DohvatiDrzavu",  
                               new { oznDrzave = model.OznDrzave }, model);  
    }  
    else  
        return BadRequest(ModelState);  
}
```


Kako isprobati POST i ostale zahtjeve?

34

- ➡ Postman – REST Client (<https://www.getpostman.com/>)
 - ➡ Alternativa Fiddler ili slični alati, vlastita konzolna aplikacija
 - ➡ Postaviti Content-Type na application/json i poslati odgovarajući JSON



➤ Postupak PUT proizvoljnog imena

- Potrebno poslati cijeli model, a ne samo parcijalne promjene (PATCH)
 - Oznaka države iz zahtjeva i ona iz modela moraju biti jednake
- Uspješna promjena podatka vraća statusnu poruku 204
 - Neispravni podaci: 400 ili 404 (ako država te oznake ne postoji)
- Primjer:  Firma.WebApi \ Controllers \ DrzavaController.cs

```
[HttpPut("{oznDrzave}")]
public async Task<IActionResult> Update(string oznDrzave, [FromBody] Drzava model) {
    if (model == null || model.OznDrzave != oznDrzave || !ModelState.IsValid)
        return BadRequest(ModelState);
    else {
        var drzava = await drzavaHandler.HandleAsync(new DrzavaQuery(oznDrzave));
        if (drzava == null)
            return NotFound($"Država s oznakom {oznDrzave} ne postoji");
        else {
            var updateCommand = mapper.Map<Drzava, UpdateDrzava>(model);
            await updateHandler.HandleAsync(updateCommand);
            return NoContent(); // status 204
        }
    }
}
```


Web API – brisanje države

36

➡ Postupak DELETE proizvoljnog imena sa šifrom države u adresi

➡ Uspješno brisanje vraća statusnu poruku 204

➡ Nepostojeća država 404

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

```
[HttpDelete("{oznDrzave}")]
public async Task<IActionResult> Delete(string oznDrzave) {
    var drzava = await drzavaHandler.HandleAsync(new DrzavaQuery(oznDrzave));
    if (drzava == null)
    {
        return NotFound($"Država s oznakom {oznDrzave} ne postoji");
    }
    else
    {
        await deleteHandler.HandleAsync(new DeleteDrzava(oznDrzave));
        return NoContent();
    }
}
```

Što u slučaju pogreške unutar upravljača?


37

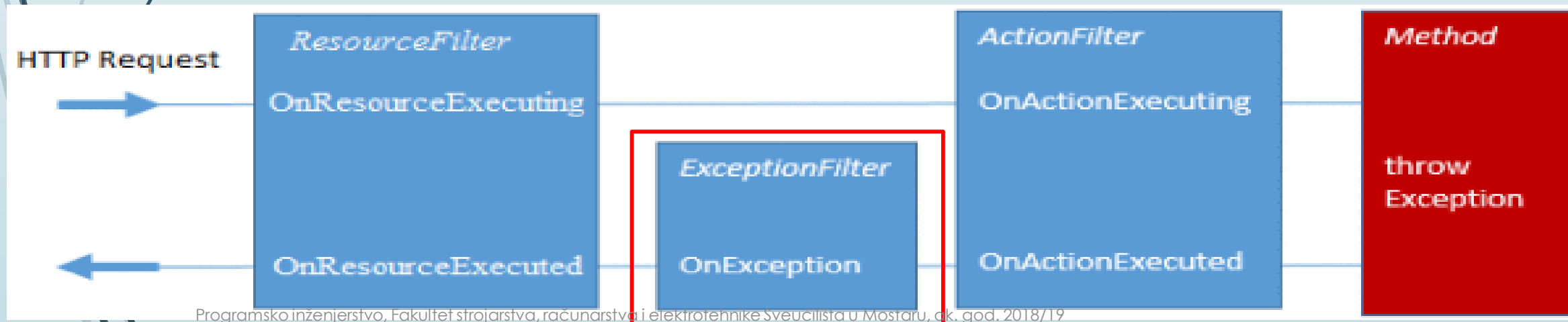
- Pogreška prilikom spremanja podatka?
- Neuhvaćena iznimka?
- ...
- Različiti pristupi
 - „Zabijanje glave u pijesak”
 - pravimo se da se iznimka neće dogoditi
 - Ako se dogodi izaziva statusnu poruku broj 500 (Interval Server Error) koja korisniku ne znači previše, a može otkriti neželjene interne podatke
 - „Sve OK”
 - Čitavi programski kôd servisa omotan u try-catch block, a rezultat je razred koji sadrži omotane podatke koje je trebao vratiti i informaciju o uspješnosti postupka i eventualnoj pogrešci
 - Korisnik uvijek dobiva statusnu poruku 200
 - Nešto treće?

Obrada iznimke korištenjem atributa

38

➤ Obrada iznimke unutar postupka

- Napisati vlastiti atribut implementiranjem sučelja `IFilterAttribute` ili izvođenjem iz razreda `ExceptionFilterAttribute`
- Zamjenjuje naporno pisanje try-catch blokova u svakom postupku i centralizira ponašanje u slučaju nepredviđene iznimke
- Koristi se kao tip unutar atributa `ServiceFilter`, npr. `[ServiceFilter(typeof(BadRequestOnException))]`
 - Potrebno registrirati u razredu `Startup`
 - `services.AddScoped<BadRequestOnException>();`
- Primjer  `Firma.WebApi \ Util \ ServiceFilters \ BadRequestOnException.cs`



- Za dokumentiranje Web API servisa koristi se alat Swagger
- Osim informacija o tipovima podataka i rezultata omogućava i pozivanje servisa

Swagger UI

https://localhost:44347/index.html

swagger

Select a spec Firma.Mvc API V1

Firma.Mvc API ^{v1}

/swagger/v1/swagger.json

Jednostan primjer Web API-a nad državama

Drzava

GET	/api/Drzava/sve	Postupak za dohvat svih država.
GET	/api/Drzava	Postupak za dohvat država prilagođenim parametrima za DataTables.net
POST	/api/Drzava	Postupak kojim se unosi nova država
GET	/api/Drzava/{oznDrzave}	Postupak za dohvat države čija je oznaka jednaka poslanom parametru
PUT	/api/Drzava/{oznDrzave}	Ažurira državu s oznakom države oznDrzave temeljem parametara iz zahtjeva
DELETE	/api/Drzava/{oznDrzave}	bríše državu s oznakom predanom u adresi zahtjeva

Aktivacija Swaggera (1)

40

- Dodati NuGet paket *Swashbuckle.AspNetCore*
- U *Startup.cs* dodati aktivaciju *Swaggera*
 - *Swagger* automatski pronalazi sve upravljače koji imaju definiranu rutu i atribut `Http[Get|Post|...]`
 - U postavkama projekta uključiti kreiranje XML dokumentacije
 - Navesti putanje do svih xml datoteka koje treba uključiti
- Primjer  *Firma.WebApi \ Startup.cs*

```
public void ConfigureServices(...  
    ...  
    services.AddSwaggerGen(c => {  
        c.SwaggerDoc("v1", new Info { Title = "Firma.WebAPI",  
            ...  
        });  
        var xmlFile =  
            $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";  
        var xmlPath = Path.Combine(  
            AppContext.BaseDirectory, xmlFile);  
        c.IncludeXmlComments(xmlPath);  
    });  
}
```


Aktivacija Swaggera (2)

41

➡ Primjer  Firma.WebApi \ Startup.cs


```
public void Configure(...  
    ...  
    app.UseSwagger();  
    app.UseSwaggerUI(c =>  
        {  
            c.SwaggerEndpoint("/swagger/v1/swagger.json",  
                "Firma.WebApi V1");  
            c.RoutePrefix = string.Empty;  
        });
```

➡ Nakon navedenih postavki automatski generirana dokumentacija za WebApi servise je dostupna na http://.../postavljeni_route_prefix/

➡ Npr. <https://localhost:44347>

Informacije o statusnim porukama


42

- Swagger pretpostavlja da svi postupci vraćaju status 200
 - U slučaju da nije tako, potrebno eksplicitno navesti moguće vrijednosti iznad postupka
 - Koristi se atribut *ProducesResponseType*
 - Primjer  Firma.WebApi \ Controllers \ DrzavaController.cs

```
[HttpPut("{oznDrzave}")]  
[ProducesResponseType((int)HttpStatusCode.NotFound)]  
[ProducesResponseType((int)HttpStatusCode.NoContent)]  
[ProducesResponseType((int)HttpStatusCode.BadRequest)]  
public async Task<IActionResult> Update(  
    string oznDrzave, [FromBody] DrzavaApiModel model)
```

Informacije o povratnim porukama

43

- **Swagger** provjerava tip povratne vrijednosti iz servisa
 - `ActionResult` je općeniti rezultat, pa Swagger nema automatski tu informaciju
 - Alternativa: koristiti `ActionResult<povratni_tip>`
 - Koristiti atribut *`ProducesResponseType`* ili izvedeni iz njega *`SwaggerResponseAttribute`*
 - Izvedeni razred sadrži mogućnosti dodavanja opisa
 - Primjer  `Firma.WebApi \ Controllers \ WebApi \ DrzavaController.cs`

```
[[HttpGet("{oznDrzave}", Name = "DohvatiDrzavu")]  
[ProducesResponseType(typeof(Drzava), (int)HttpStatusCode.OK)]  
[ProducesResponseType((int)HttpStatusCode.NotFound)]  
public async Task<ActionResult> Get(string oznDrzave)  
...  
[HttpGet]  
[SwaggerResponseAttribute((int)HttpStatusCode.OK,  
    typeof(List<Drzava>), Description = "Vraća listu država")]  
public async Task<List<Drzava>> GetAll()
```

Podrška za straničenje

44

- Vraćanje svih podataka je nepraktično
- Implementaciju straničenja i sortiranja potrebno prilagoditi klijentskom alatu koji će se koristiti i obrnuto
 - Primjer za DataTables
Firma.WebApi \ wwwroot \ drzava.html

Pregled svih država

https://localhost:44347/drzave.html

Podaci u tablici dobiveni su pozivom web servisa na adresi </api/drzava>

Pregled država

Prikaži 10 zapisa

Pretraga b|

Oznaka	Naziv	ISO3 oznaka	Šifra države
BA	Bosnia and Herzegovina	71	BIH
BB	Barbados	52	BRB
BD	Bangladesh	50	BGD
BE	Belgium	56	BEL
BF	Burkina Faso	854	BFA
BI	Burundi	108	BDI
BJ	Benin	204	BEN
BM	Bermuda	60	BMU
BN	Brunei Darussalam	96	BRN
BO	Bolivia	68	BOL

Prethodna 1 2 3 4 5 ... 8 Sljedeća

1 - 10 od ukupno 75 zapisa (filtrirano od ukupno 316 zapisa)

Standardi za web servise - OData

45

- OData – Open Data Protocol
 - Standard za izgradnju *RESTful API-a*
- Standardizira oblik adrese pojedinog REST servisa
- Definira sintaksu za oblikovanje/filtriranje rezultata
 - Npr. *http://.../odata/Dokument?\$filter=contains(NazPartnera,'Stipanović') and DatDokumenta lt 2016-01-01 and IznosDokumenta gt 10000&\$top=3&\$orderby=IznosDokumenta desc*
- Za detalje o standardu pogledati na <http://www.odata.org/>

➤ Command-query separation

- <https://cuttingedge.it/blogs/steven/pivot/entry.php?id=92>
- <https://www.future-processing.pl/blog/cqrs-simple-architecture/>
- <https://martinfowler.com/bliki/CQRS.html>

➤ JQuery, Datatables, WebApi

- <http://ezzylearning.com/tutorial/jquery-datatables-paging-sorting-and-searching-with-asp-net-web-api>
- <https://www.codemag.com/Article/1601031/CRUD-in-HTML-JavaScript-and-jQuery-Using-the-Web-API>
- <https://datatables.net/manual/server-side>