

Razvoj primijenjene programske potpore

12. Web-servisi

Servis

- 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
- Pristup omogućen heterogenim klijentima
- Implementacija kao crna kutija
 - Promjena implementacije ne utječe na klijente
- Obično govorimo o web-servisima, ali servis predstavlja općenitiji pojam, ne nužno baziran na HTTP protokolu
 - Različiti protokoli, standardi, koncepti:
 - HTTP, XML, JSON, SOAP, OData, GraphQL, gRPC ...

Remote Procedure Call

- Temelji u RPC-u (Remote Procedure Call)
 - *Remote Method Invocation* u kontekstu objektno orijentiranih jezika
- Proces na jednom računalu (klijent) poziva proceduru/metodu koja se izvršava na drugom računalu (server)
 - Ulazni parametri i povratna vrijednost prenose se mrežom. Postupak pakiranja u poruku prikladnu za prijenos preko mreže naziva se *marshalling*.
 - Iz perspektive programera takav udaljeni poziv se ne razlikuje od poziva lokalne procedure
 - pakiranje, slanje i primanje odrađuju namjenske biblioteke – *stubs*
- Prve konkretne implementacije u Xeroxu početkom 1980.-tih
 - doktorat Brucea Jaya Nelsona te rad zajedno s Andrewom Birellom
<https://www.cs.cmu.edu/~dga/15-712/F07/papers/birrell842.pdf>
http://www.bitsaves.org/pdf/xerox/parc/techReports/CSL-81-9_Remote_Procedure_Call.pdf

- 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
 - kako izložiti servis, koristiti ponuđene servise, katalogizirati ih ...
- Distribuirani sustav u kojem sudjeluje više autonomnih servisa međusobno šaljući poruke preko granica
 - Granice mogu biti određene procesom, mrežom, ...
 - Otvoreni standardi i generičke poruke koje nisu specifične za pojedinu platformu ili pojedini jezik

Četiri stupa servisno orijentirane arhitekture

- Jasno određene granice
 - Jasno iskazana funkcionalnost i struktura podataka, što manja to bolja
 - Implementacija je crna kutija, a cilj što lakše korištenje
- 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 i ne utječu na shemu
- Semantika, a ne samo sintaksa
 - Logička kategorizacija servisa, smisleno imenovanje postupaka

Problemi prilikom izrade servisa

- Problem višenitnosti, skalabilnost, brzina obrade postupka
- Sigurnost komunikacije
- Pouzdanost i robusnost servisa
 - Klijent treba znati je li servis primio poruku.
 - Pogreške u servisu treba obraditi
- Konzistentnost stanja
 - Neuspjeh prilikom izvršavanja servisa ne smije ostaviti sustav u stanju pogreške
- **Interoperabilnost**
 - Tko sve može pozvati servis?

Različiti aspekti interoperabilnosti

- Komunikacijski protokol i vrsta komunikacije
 - HTTP, HTTP/2, TCP, ...
 - Sinkrona, asinkrona, streaming, ...
- Format poruke
 - XML, JSON, binarni format, ...
- Struktura poruke
 - zaglavlja, sadržaj, poredak podataka, ...
- Preslikavanje podataka među tipovima podataka u različitim jezicima
- *Interface Definition Language* (IDL) – generički jezik (neovisan o konkretnom programskom jeziku) kojim se opisuju sučelja i strukture podataka (i preslikavanja u konkretne jezike)
 - različite formati: CORBA, WSDL, protocol buffers, OpenAPI, ...

Standardi za web servise - SOAP

- 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)
 - **Unutar XML-a nalazi se informacija 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 i opise SOAP web-servisa

REST - Representational State Transfer (1)

- Arhitekturni obrazac za izradu mrežnih aplikacija koji je osmislio Roy Fielding u doktorskoj disertaciji 2000.
- Općeniti koncept formalno nezavisan za HTTP, ali isprepleten s nastankom HTTP/1.1 (npr. uvođenje metoda PUT, PATCH, DELETE)
 - Fielding je ujedno i koautor specifikacije za HTTP/1.1
- Postavlja nekoliko ograničenja koje sustav morao zadovoljiti, pri čemu treba istaknuti koncept jedinstvenog sučelja (engl. uniform interface), odnosno jedinstvenog identifikatora resursa
- Detaljnije na
 - <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
 - <https://codewords.recurse.com/issues/five/what-restful-actually-means>

REST - Representational State Transfer (2)

- Poslužio kao temelj za novi koncept izrade web-servisa orijentiran na resurse (engl. *resource driven*), a ne na akcije kao SOAP
 - adresa servisa jednoznačno određuje resurs
 - akcije nad resursom određene vrstom HTTP zahtjeva, npr.
 - GET – dohvat podataka
 - POST - stvaranje novog podatka
 - PUT i PATCH – izmjene cijelog ili dijela podatka
 - Možemo li dodati podatak s PUT? Ni HTML ni REST to ne određuju.
 - DELETE – brisanje podatka
 - SOA → ROA (Resource oriented architecture)?
- Istovremeno otvorio Pandorinu kutiju zloupotreba pojma REST
 - npr. usvajanje pojedinačnih elemenata REST-a, ali ne u cijelosti, uz zadržavanje pojma REST
 - Kako nazivati takve servise? WebAPI? Nije li i SOAP servis isto neka vrsta web-API-a? Više na slajdu Richardsonov model zrelosti.

REST* vs SOAP (OpenAPI vs SOAP)

- * = REST ili bazirano na odabranim elementima REST-a. (WebAPI?)
 - Slanjem SOAP poruke na pristupnu točku provjerava se sadržaj i na osnovu sadržaja se određuje postupak koji se treba izvršiti
 - Kod REST* web-servisa 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 i manje poruke
 - POX (Plain old XML) – XML poruke bez SOAP zaglavlja
 - JSON – (JavaScript Object Notation)
 - Dovoljna je podrška za HTTP - nije potrebno implementirati složene WS-* standarde
 - Lakše cacheiranje
 - SOAP koristi WSDL, REST* servisi koriste OpenAPI (Swagger)

WebAPI i HTTP metode

- GET: 2 metode za dohvat podataka koje vraćaju podatke ili 404
 - Dohvat svih ili podskupa podataka
 - Dohvat jednog elementa temeljem primarnog ključa (identifikatora)
- POST: Kreiranje novog podatka
 - Za uspješno stvoreni podatak vraća HTTP status 201 (te često lokaciju stvorenog resursa i sam resurs)
- PUT/PATCH: Ažuriranje cijelog (PUT) ili dijela (PATCH) podatka
 - Nakon uspješnog ažuriranja vraća HTTP status 200 ili 204
 - Nigdje nije propisano da je CREATE=POST. Može i PUT, ali nije često
- DELETE: Brisanje podatka
 - Ako je podatak uspješno obrisani vraća 200 ili 204
 - Ako je podatak već ranije bio obrisani, ili je identifikator neispravan vraća HTTP status 404
 - U kontradikciji s činjenicom da bi DELETE trebao biti idempotentan

Kostur Web API upravljača u .NET-u

```
[ApiController] [Route("[controller]")]
public class ValuesController : BaseController {
    // GET: api/values
    [HttpGet]
    public IEnumerable<string> Get() {
        return new string[] { ... };
    }
    // GET api/values/5
    [HttpGet("{id}")]
    public string Get(int id) { ... }
    // 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) { ... }
```

MVC – Web API : Razlike u *Program.cs*

■ Web API


```
builder.Services.AddControllers()  
...  
var app = builder.Build();  
...  
app.MapControllers();
```

■ MVC

```
builder.Services.AddControllersWithViews();  
...  
var app = builder.Build();  
...  
app.UseEndpoints(endpoints => {  
    endpoints.MapDefaultControllerRoute();  
});
```


- Napomena: MVC može sadržavati i WebAPI upravljače

Primjer Web API servisa – Web API za mjesta

- Definira se slično kao i upravljači u MVC-u, ali ima definirano vlastito usmjeravanje, ne vraća pogled već podatke i/ili statusni kod
 - Nasljeđuje ControllerBase
- Označen atributom *ApiController*
 - **Automatska provjera valjanost modela. Ako model nije valjan rezultat je status 400 - BadRequest**
- Ovisnosti o sučeljima i razredima u konstruktoru kao i kod MVC-a
 - Primjer:  WebServices \ ... WebAPI \ Controllers \ MjestoController.cs

```
[ApiController]
[Route("[controller]")]
public class MjestoController : ControllerBase {
    public MjestoController(FirmaContext ctx) {
        this.ctx = ctx;
    }
    ...
}
```

Web API – dohvat svih mjesta (1)

- Naziv postupka nebitan – postupak se određuje prema atributu `HttpGet` i usmjeravanju upravljača
 - Konkretno u ovom primjeru `/mjesto`
- Klijentu isporučiti prezentacijski model neovisno o sličnosti s modelom iz EF-a
 - Omogućava naknadno neovisnu izmjenu podatkovnog sloja
- Primjer:  ... \ WebApi \ Controllers \ MjestoController.cs
 - Napomena: EF model u zasebnom projektu, jer se koristi i u drugim primjerima


```
[HttpGet(Name = "DohvatiMjesta")]
public async Task<List<MjestoViewModel>> GetAll(...) {
    var query = ctx.Mjesto.AsQueryable(); //.AsNoTracking()?
    ...
    var list = await query.Select(m => new MjestoViewModel {
        ...
    })
    ... .ToListAsync();

    return list;
}
```


Web API – dohvat svih mjesta (2)

- U slučaju velikog broja mjesta poželjno definirati parametre za dohvat podskupa elemenata
 - Parametri nisu standardizirani, već često ovise o ciljanom klijentu

```
public ... GetAll([FromQuery] LoadParams loadParams)
```

- Primjer:  ...WebApi\ViewModels\LoadParams.cs
- U konkretnom primjeru parametri prilagođeni za jTable

```
public class LoadParams {  
    [FromQuery(Name = "jtStartIndex")]  
    public int StartIndex { get; set; }  
    [FromQuery(Name = "jtPageSize")]  
    public int Rows { get; set; }  
    [FromQuery(Name = "jtSorting")]  
    public string Sort { get; set; }  
    ...  
}
```

Izvori podataka za argumente Web API servisa

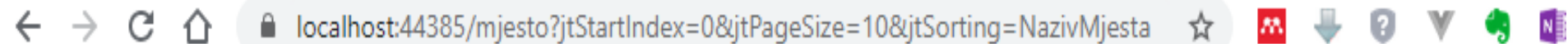
- Ispred tipa i naziva argumenta postupka Web API servisa može se navesti neki od atributa
 - [FromBody], [FromForm], [FromHeader], [FromQuery], [FromRoute], [FromServices]
- Za upravljače označene s [ApiController] ne vrijede pravila, tj. redoslijed kao kod upravljača u MVC-u. Ako nije naveden niti jedan, onda se koriste sljedeća pravila
 - jednostavni tipovi (int, string, ...) trebaju biti dio *query stringa*
 - složeni tipovi se nalaze u tijelu zahtjeva
 - Datoteke su dio podataka forme
- **Postupak može imati samo jedan složeni podatak čiji su podaci iz tijela zahtjeva.**
- Detaljnije na <https://docs.microsoft.com/en-us/aspnet/core/web-api/#binding-source-parameter-inference>

Web API – dohvat svih mjesta (3)


- Otvaranjem adrese

<https://.../mjesto?jtStartIndex=0&jtPageSize=10&jtSorting=NazivMjesta> dobit će se JSON sadržaj s popisom prvih 10 mjesta poredanih po nazivu mjesta

- Popis serijaliziran u JSON
- Konfiguracijskom datotekom može se postaviti i drugi format (npr. Xml)




```
{ "IdMjesta":1,"PostBrojMjesta":3121,"NazivMjesta":"Ada","PostNazivMjesta":"Laslovo","OznDrzave":"HR","NazivDrzave":"Croatia"
{ "IdMjesta":8569,"PostBrojMjesta":24430,"NazivMjesta":"ADA","PostNazivMjesta":null,"OznDrzave":"CS","NazivDrzave":"Serbia an
{ "IdMjesta":2,"PostBrojMjesta":10363,"NazivMjesta":"Adamovec","PostNazivMjesta":"Belovar","OznDrzave":"HR","NazivDrzave":"Cr
{ "IdMjesta":8404,"PostBrojMjesta":22244,"NazivMjesta":"ADAŠEVCI","PostNazivMjesta":null,"OznDrzave":"CS","NazivDrzave":"Serb
Montenegro"},{ "IdMjesta":8375,"PostBrojMjesta":21251,"NazivMjesta":"ADICE","PostNazivMjesta":null,"OznDrzave":"CS","NazivDrz
Montenegro"},{ "IdMjesta":7856,"PostBrojMjesta":8341,"NazivMjesta":"ADLEŠIĆ","PostNazivMjesta":null,"OznDrzave":"SI","NazivDr
```

- Ne odredi li se drugačije (serijalizacijskim atributima ili postavkama u Program.cs), koristit će se camelCase
 - Primjer:  ... WebApi \ Program.cs

```
builder.Services.AddControllers()
    .AddJsonOptions(configure =>
        configure.JsonSerializerOptions.PropertyNamingPolicy = null);
```

Web API – dohvat određenog mjesta

- Putanja oblika /mjesto/idmjesto (npr. /mjesto/123)
 - Usmjeravanje imenovano posebnim nazivom što će kasnije imati utjecaj na generirani klijent
- Povratna vrijednost može biti konkretno mjesto ili status 404
 - Rezultat je ActionResult<MjestoViewModel>
 - Primjer:  ... WebApi \ Controllers \ MjestoController.cs


```
[HttpGet("{id}", Name = "DohvatiMjesto")]
public async Task<ActionResult<MjestoViewModel>> Get(int id) {
    var mjesto = await ctx.Mjesto.Where(m => m.IdMjesta == id)
        .Select(m => new MjestoViewModel { ... })
        .FirstOrDefaultAsync();

    if (mjesto == null)
        return Problem(statusCode: StatusCodes.Status404NotFound,
            detail: $"No data for id = {id}");
    else
        return mjesto; ...
}
```

ActionResult i ProblemDetails

- U prethodnom primjeru postupak je mogao vratiti
 - konkretno mjesto (objekt tipa *MjestoViewModel*) – posljedično status 200
 - Status + poruku da mjesto ne postoji
- U takvim slučajevima koristi se povratni tip *ActionResult<T>*
 - Definira implicitnu konverziju iz T u *ActionResult<T>*
- Poruke o pogrešci bi trebale biti u skladu s RFC 7807
 - <https://tools.ietf.org/html/rfc7807>
- U prethodnom primjeru može se koristiti
 - *NotFound* - obratiti pažnju da treba koristiti varijantu bez argumenata)
 - *Problem* koji vraća *ProblemDetails* u skladu s RFC 7807

Web API – dodavanje mjesta

- Postupak POST proizvoljnog imena
 - U slučaju uspjeha vraća se status 201, podatak i njegova adresa
 - Adresa je dio zaglavlja odgovora, podatak je u tijelu
 - Neispravan model uzrokuje statusnu poruku 400 (BadRequest)
 - Primjer:  ... WebApi \ Controllers \ MjestoController.cs

```
[HttpPost(Name = "DodajMjesto")]
public async Task<IActionResult> Create(MjestoViewModel model) {
    Mjesto mjesto = new Mjesto {
        NazMjesto = model.NazivMjesto, OznDrzave = model.OznDrzave,
        PostBrMjesto = model.PostBrojMjesto,
        PostNazMjesto = model.PostNazivMjesto
    };
    ctx.Add(mjesto);
    await ctx.SaveChangesAsync();
    var addedItem = await Get(mjesto.IdMjesto);
    return CreatedAtAction(nameof(Get), new { id = mjesto.IdMjesto },
        addedItem.Value);
}
```

Kako isprobati POST i ostale zahtjeve? (1)


- Postman – API klijent/alat (<https://www.postman.com/downloads>)
 - Alternativa Fiddler ili slični alati, vlastita konzolna aplikacija, generirani klijent preko Swaggera (o tome malo kasnije)
 - Postaviti Content-Type na application/json i poslati odgovarajući JSON

The screenshot shows the Postman interface for a POST request to `https://localhost:44385/mjesto`. The 'Headers' tab is selected, showing a single header: `Content-Type: application/json`. Below this, the 'Body' tab is selected, showing the 'raw' radio button selected and the 'JSON (application/json)' format chosen. The JSON body is displayed as follows:

```
1 {  
2   "PostBrojMjesta": 123, "NazivMjesta" : "probno mjesto",  
3   "OznDrzave" : "HR", "PostNazivMjesta" : "probno mjesto"  
4 }
```

Kako isprobati POST i ostale zahtjeve? (2)

Body Cookies Headers (6) Test Results Status: 201 Created

Pretty Raw Preview JSON 

```
1 {  
2   "IdMjesta": 36460,  
3   "PostBrojMjesta": 123,  
4   "NazivMjesta": "probno mjesto",  
5   "PostNazivMjesta": "probno mjesto",  
6   "OznDrzave": "HR",  
7   "NazivDrzave": "Croatia"  
8 }
```

Body Cookies Headers (6) Test Results

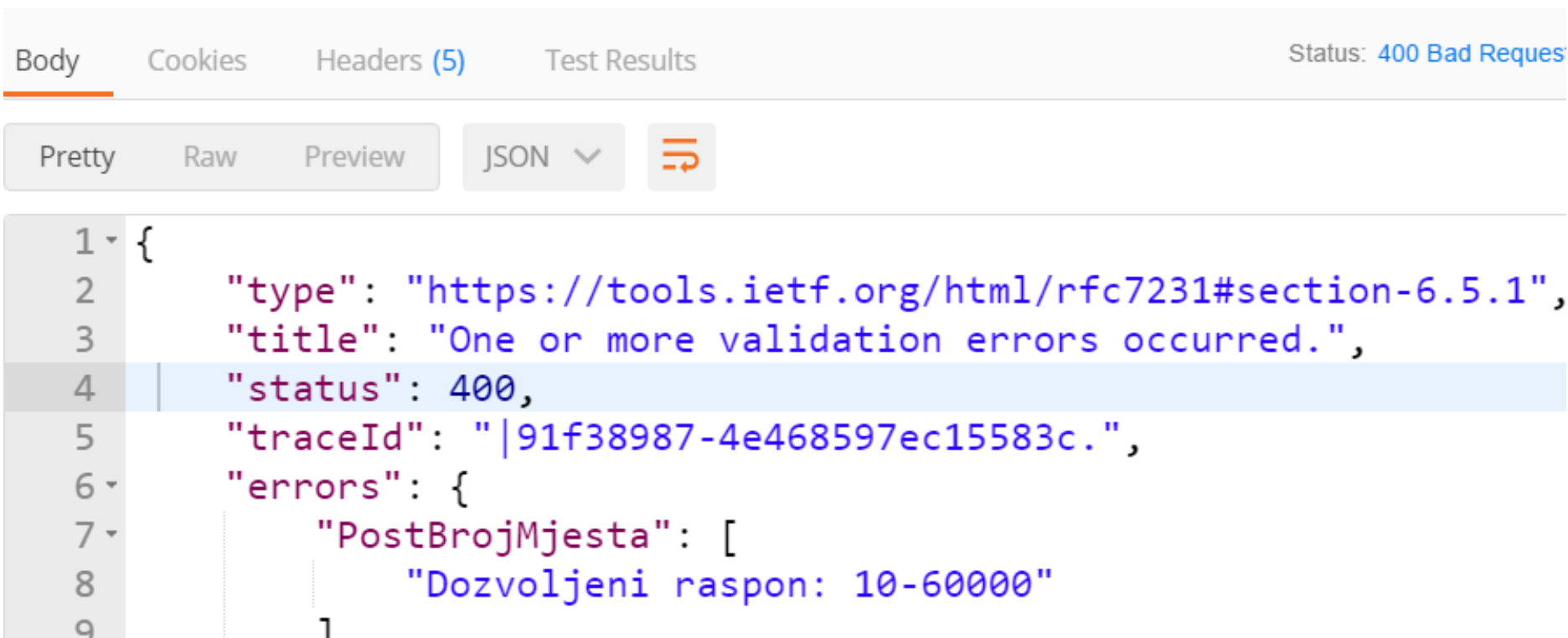
Content-Type → application/json; charset=utf-8

Date → Mon, 11 Jan 2021 10:26:36 GMT

Location → https://localhost:44385/Mjesto/36460

Kako isprobati POST i ostale zahtjeve? (3)


- U slučaju validacijske pogreške, odgovor sadrži poruku o pogrešci i statusni kod 400 (*Bad Request*)
 - Posljedica validacijskih atributa (ili *FluentValidationa*) te atributa [*ApiController*] na upravljaču



The screenshot shows the 'Body' tab of a web browser's developer tools. The status bar at the top right indicates 'Status: 400 Bad Request'. The response body is displayed in JSON format, showing a validation error. The JSON object contains a 'type' pointing to an RFC 7231 section, a 'title' describing the error, a 'status' of 400, a 'traceId', and an 'errors' object. The 'errors' object has a 'PostBrojMjesta' property with a message indicating a valid range of 10-60000.


```
1 {
2   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
3   "title": "One or more validation errors occurred.",
4   "status": 400,
5   "traceId": "|91f38987-4e468597ec15583c.",
6   "errors": {
7     "PostBrojMjesta": [
8       "Dozvoljeni raspon: 10-60000"
9     ]
10  }
```

Web API – ažuriranje mjesta

- Postupak PUT proizvoljnog imena
 - U primjeru se šalje cijeli model, a ne samo parcijalne promjene (PATCH)
 - Identifikator mjesta iz zahtjeva i onaj iz modela moraju biti jednaki
 - Uspješna promjena podatka vraća statusnu poruku 204
 - Neispravni podaci: 400 ili 404 (ako mjesto ne postoji)
 - Primjer:  ... WebApi \ Controllers \ MjestoController.cs


```
[HttpPut("{id}")]
public async Task<IActionResult> Update(int id,
                                         MjestoViewModel model) {
    var mjesto = await ctx.Mjesto.FindAsync(id);
    if (mjesto == null)
        return Problem(statusCode: StatusCodes.Status404NotFound,
                        detail: $"Invalid id = {id}");
    mjesto.NazMjesta = model.NazivMjesta;
    mjesto.OznDrzave = model.OznDrzave; ... //ostala pridruživanja
    await ctx.SaveChangesAsync();
    return NoContent();
}
```

Web API – brisanje mjesta

- Postupak DELETE proizvoljnog imena sa šifrom mjesta u adresi
- Uspješno brisanje vraća statusnu poruku 204, a za nepostojeće mjesto vraća se 404
- Primjer:  ... WebServices \ Controllers \ MjestoController.cs

```
[HttpDelete("{id}", Name = "ObrisiMjesto")]
public async Task<IActionResult> Delete(int id) {
    var mjesto = await ctx.Mjesto.FindAsync(id);
    if (mjesto == null)
        return Problem(statusCode: StatusCodes.Status404NotFound,
                        detail: $"Invalid id = {id}");
    else {
        ctx.Remove(mjesto);
        await ctx.SaveChangesAsync();
        return NoContent();
    }
}
```

Primjer poziva WebAPI-a iz konzolne aplikacije (1)

- Primjer:  ...WebApi \ ConsoleClientApps \ ConsoleClient \ Program.cs
 - Razred *HttpClient* za interakciju s web stranicama/servisima
 - Paket Microsoft.AspNet.WebApi.Client za proširenje ReadAsAsync<T>
 - Razred *Mjesto* definiran da po strukturi odgovara isporučenom Jsonu (nastalom iz razreda *MjestoViewModel*)


```
private static async Task DohvatiMjesto(int id) {  
    using (var client = new HttpClient()) {  
        var response = await client.GetAsync($"url/{id}");  
        if (response.IsSuccessStatusCode) {  
            var mjesto = await response.Content.ReadAsAsync<Mjesto>();  
            Console.WriteLine($"Mjesto s id-om {id} je : ");  
            Console.WriteLine(mjesto.ToString());  
        }  
        else  
            Console.WriteLine($"Neuspješan dohvat mjesta {id}");  
    }  
}
```



Microsoft.AspNet.WebApi.Client by Microsoft v5.2.7

This package adds support for formatting and content negotiation to System.Net.Http.

Primjer poziva WebAPI-a iz konzolne aplikacije (2)

- Primjer:  WebApi \ ... \ ConsoleClient \ Program.cs : DodajMjesto
 - Potrebno postaviti tip sadržaja na application/json
 - Poziva se postupak *PostAsJsonAsync*
 - Slično u primjerima koji slijede PutAsJsonAsync, DeleteAsync, ...

```
using (var client = new HttpClient()) {  
    var response = await client.PostAsJsonAsync<Mjesto>(url, mjesto);  
    if (response.IsSuccessStatusCode) {  
        Console.WriteLine($"Mjesto dodano i može se dohvatiti na adresi  
                           {response.Headers.Location}");  
        mjesto = await response.Content  
                        .ReadAsAsync<ContentResultValue<Mjesto>>();  
        Console.WriteLine(mjesto);  
        return mjesto.IdMjesta;  
    }  
    else {  
        string content = await response.Content.ReadAsStringAsync();  
        // response.StatusCode, response.ReasonPhrase, ...  
    }  
}
```

Što u slučaju pogreške unutar upravljača?

- 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?

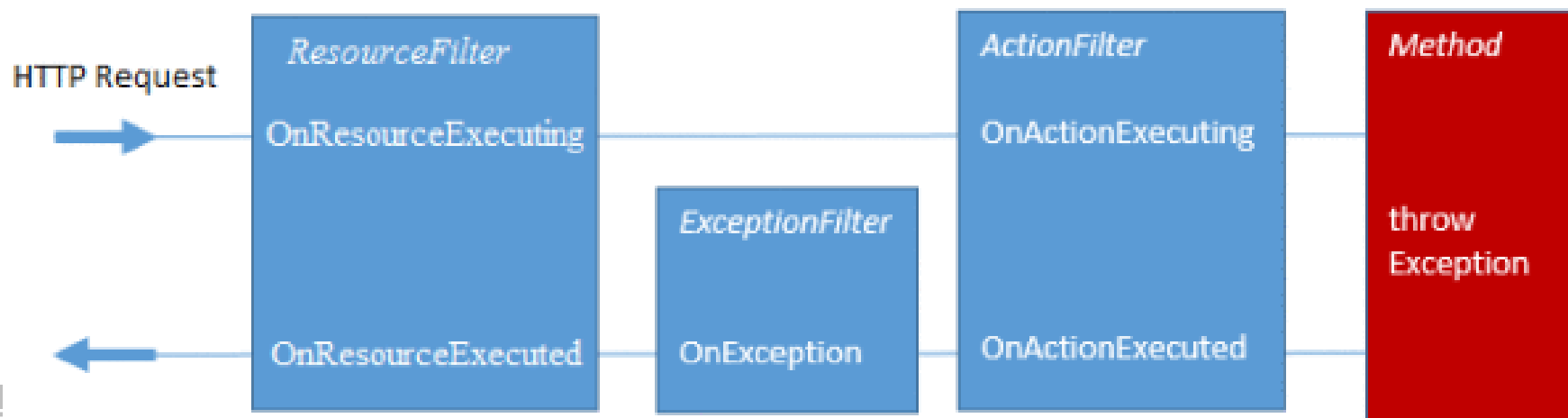
Primjer korištenja servisa koristeći *jTable*

- Proučiti sljedeće sadržaje
 -  WebServices \ ... \ WebApi \ wwwroot \ mjesta.html
<https://localhost:44385/mjesta.html>
 -  WebServices \ ... \ WebApi \ Controllers \ jTable \ jTableController.cs
 -  WebServices \ ... \ WebApi \ Controllers \ jTable \ MjestoJtableController.cs
 -  WebServices \ ... \ WebApi \ Controllers \ jTable \ LookupController.cs
 -  WebServices \ ... \ WebApi \ ViewModels \ jTable \ *
- jTable ne radi s WebAPI servisima direktno, već koristi POST, pa je potreban *wrapper*.
 - jTable koristi koncept da je poziv uvijek uspješan (status 200) uz poruku je li akcija uspjela ili ne
 - U slučaju validacijske pogreške *wrapper* ne vraća 400, već 200 uz (primjerice) sadržaj

```
{ "Result": "ERROR",      "Message":  
      "PostBrojMjesta: Dozvoljeni raspon: 10-60000; " }
```

Obrada iznimke korištenjem atributa

- Umjesto napornog pisanja try-catch blokova u svakom postupku (a i za nepredviđene iznimke), obrada iznimke se centralizira posebnim atributima
 - Vlastiti atribut izveden iz `ExceptionFilterAttribute` ili implementacijom `IExceptionFilter`
 - Zbog DI-a ne navodi se direktno, nego koristeći atribut `TypeFilter`, npr. `[TypeFilter(typeof(ErrorStatusTo200WithErrorMessage))]`
 - Primjeri:
 - ... WebApi \ Controllers \ MjestoController.cs
 - ... WebApi \ Controllers \ Jtable \ JTableController.cs
 - ... WebApi \ Util \ ExceptionFilters \ ProblemDetailsForSqlException.cs
 - ... ExceptionFilters \ ErrorStatusTo200WithErrorMessage.cs
- Ako se više atributa primijeni na upravljač, bitan je njihov poredak (ili redom, ili eksplicitno naveden). Za obradu iznimke poredak je obrnut.




Dokumentacija za web servise - OpenAPI

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

The screenshot displays the Swagger UI for an API named 'WebServices'. At the top, the Swagger logo is visible, along with the text 'Supported by SMARTBEAR'. A dropdown menu labeled 'Select a definition' shows 'RPPP Firma WebAPI' as the selected option. Below this, the API title 'WebServices' is shown with version '1.0' and 'OAS3' tags. The URL '/swagger/v1/swagger.json' is listed. The main section is titled 'Mjesto' and lists several endpoints:

- GET** `/Mjesto/count`: Vraća broj svih mjesta filtriran prema nazivu mjesta
- GET** `/Mjesto`: Dohvat mjesta (opcionalno filtrirano po nazivu mjesta). Broj mjesta, poredak, početna pozicija određeni s loadPa
- POST** `/Mjesto`: Stvara novo mjesto opisom poslanim modelom
- GET** `/Mjesto/{id}`: Vraća grad čiji je IdMjesta jednak vrijednosti parametra id
- DELETE** `/Mjesto/{id}`: Brisanje mjesta određenog s id
- PUT** `/Mjesto/{id}`: Ažurira mjesto

Aktivacija Swaggera (1)

- Dodati NuGet paket Swashbuckle.AspNetCore
- U Program.cs aktivirati Swagger
 - Swagger automatski pronalazi sve upravljače i attribute `Http[Get|Post|...]`
 - S `ApiExplorerSettings(IgnoreApi = true)` moguće izbaciti željene upravljače
 - U postavkama projekta uključiti kreiranje XML dokumentacije
 - Navesti putanje do svih xml datoteka koje treba uključiti
 - Primjer  ... WebApi \ Program.cs

```
builder.Services.AddSwaggerGen(c => {  
    c.SwaggerDoc("A URI-friendly name", new OpenApiInfo {  
        Title = "RPPP Web API", Version = "v1"  
    });  
    var xmlFile = $"{Assembly.GetExecutingAssembly().  
        .GetName().Name}.xml";  
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);  
    c.IncludeXmlComments(xmlPath);  
    ...  
});
```


Aktivacija Swaggera (2)

- Primjer  ... WebApi \ Program.cs

```
...
app.UseSwagger();
app.UseSwaggerUI(c => {
    c.SwaggerEndpoint("/swagger/v1/swagger.json",
                      "RPPP WebAPI");
    c.DocumentTitle = "RPPP Web Api";
    c.RoutePrefix = "docs";
});
...
```

- Nakon navedenih postavki automatski generirana dokumentacija za WebApi servise je dostupna na `http://.../route-prefix/`
 - Npr. <https://localhost:44377/docs>

Informacije o statusnim porukama

- Ako upravljači nisu označeni s [APIConventions] 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  ... \ WebApi \ Controller \ MjestoController.cs

```
[HttpPut("{id}")]  
[ProducesResponseType(StatusCodes.Status204NoContent)]  
[ProducesResponseType(StatusCodes.Status404NotFound)]  
[ProducesResponseType(StatusCodes.Status400BadRequest)]  
public async Task<IActionResult> Update(  
    int id, [FromBody] MjestoViewModel model)
```

- Napomena: Ovo ne znači da se ne može pojaviti neki drugi status (npr. 500 – Internal Server Error), već da se takav rezultat ne bi trebao pojaviti.

Informacije o povratnim porukama

- Swagger određuje povratne tipove na temelju potpisa metode, npr. ako je to `ActionResult<povratni_tip>`, kolekcija, ...
- *IAsyncResult* je općeniti rezultat, pa *Swagger* nema automatski tu informaciju. U tom slučaju, koriste se atributi *ProducesResponseType* i *SwaggerResponseAttribute* (izvedeni razred iz *ProducesResponseType* s mogućnošću dodavanja opisa)

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

Generiranje klijenta na osnovi Swagger dokumentacije (1)

- Na temelju Swaggerove json datoteke i alata NSwag moguće generirati klijente razrede (kroz Visual Studio ili samostalno)
- Desni klik na projekt -> Add Service Reference -> OpenAPI
 - Lokalno kopira datoteku *swagger.json*
 - Generira *swaggerClient.cs* u podmapi *obj* s kodom koji sadrži
 - metode za poziv servisa
 - parcijalne razrede za podatke koji služe kao ulazne i izlazne vrijednosti servisa (npr. *MjestoViewModel*)

Add new OpenAPI service reference

Select a file or URL

☐ File

☒ URL

`https://localhost:44377/swagger/v1/swagger.json`

Provide the namespace for the generated code

`OpenAPIClient`


Provide the class name for the generated code

`RPPPClnt`

Code generation language

`C#`

Generiranje klijenta na osnovi Swagger dokumentacije (2)

- Generirani klijent sadrži
 - razrede koji služe kao ulazno/izlazni podaci postupaka web-servisa
 - postupke čiji **naziv odgovara nazivu rute** i s parametrima koji određuju pojedini resurs
 - Smanjuje mogućnost pogrešnog poziva u odnosu na prethodni primjer s razredom *HttpClient*.
- Primjer  ... \ ConsoleClientApps \ StronglyTypedClient \ Program.cs

```
using(HttpClient client = new HttpClient()) {  
    var apiClient = new RPPPCClient(url, client);  
    await apiClient.DodajMjestoAsync(model);  
}
```

- Primjer  ... WebApi \ Controllers \ MjestoController.cs

```
[HttpPost(Name = "DodajMjesto")]  
public async Task<IActionResult> Create(MjestoViewModel model)
```

Richardsonov model zrelosti

- 4 nivoa zrelosti Web API-a prema ograničenjima REST-a
 - Leonard Richardson 2008.
 - <https://martinfowler.com/articles/richardsonMaturityModel.html>
- Nivo 0: RPC/RPI (Remote Procedure Call/Invocation) preko HTTP-a
- Nivo 1: Izlaganje resursa umjesto metoda
 - interakcija vezana za konkretni resurs (URL identificira resurs)
- Nivo 2: Pravilna upotreba vrsta HTTP zahtjeva, statusa i sadržaja odgovara (npr. lokacija dodanog resursa)
 - Napomena: primjer s mjestima (ali ne i wrapper za jTable) pripada ovom nivou
- Nivo 3: Odgovor sadrži hipermedijske elemente (npr. poveznice na ostale resurse i akcije koje se mogu napraviti s tim resursom)
 - Hypermedia as the Engine of Application State (HATEOAS)
 - omogućavaju samostalno otkrivanje ostalih resursa/mogućnost servisa
- Kreator REST-a, Roy Fielding smatra da je nivo 3 preduvjet da bi se nešto nazivalo REST servisom
 - <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

Generalni problemi REST servisa

- Prekomjerno preuzimanje (engl. *overfetching*)
 - skup povratnih vrijednosti određen je prezentacijskim modelom
 - rezultat možda sadrži i više podataka nego što klijentu treba
 - može uzrokovati nepotrebne join upite i povećati veličinu rezultata
- Nedovoljno preuzimanje (engl. *underfetching*)
 - FT1P („fali ti jedan podatak”)
 - npr. ako rezultat s mjestima ne sadrži podatak u nazivu države, to će možda uzrokovati još jedan poziv za popis država
 - generalno vodi ka n+1 problemu
 - Zamislimo upit u kojem trebamo države i sva njihova mjesta. Upit za popis država + n upita za mjesta u pojedinoj državi
 - Zaobilazno rješenje je raditi upit s mjestima uz uključenu državu
 - Što s upitima na još jednu razinu niže
- Alternative: OData, GraphQL, vlastite dinamičke projekcije, ...