REST

Matjaž B. Jurič

Java Champions | ORACLE ACE Director | IBM CHAMPION | ORACLE Certified Expert | Certified for IBM SOA technology | Certified for IBM CLOUD COMPUTING technology

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Laboratorij za integracijo informacijskih sistemov

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

STORITVE REST IN IMPLEMENTACIJA Z JAX-RS

Vsebina

- Uvod
- REST arhitekturni koncepti
- Oblikovanje RESTful storitev
- Storitve REST v Javi (JAX-RS)

Representational **S**tate **T**ransfer

- Arhitekturni stil
- Abstrakcija arhitekturnih elementov znotraj distribuiranih hipermedijskih sistemov
- Odjemalci – strežniki
- Pomenljivi, preprosti in naslovljivi viri
- Brez stanja
- Močna uporaba medpomnjenja

Ključne arhitekturne lastnosti REST

Skalabilnost

Splošnost

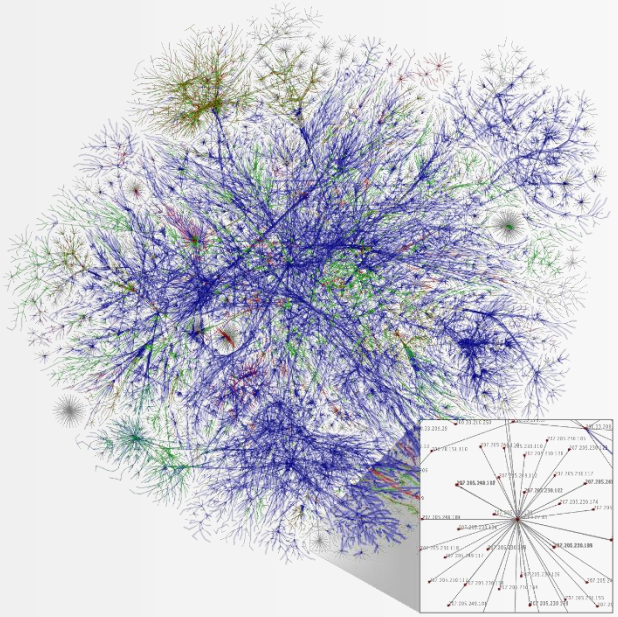
Neodvisnost

Odzivnost na spremembe

Latenca

Varnost

Enkapsulacija



Vir: Roy Fielding

HATEOAS

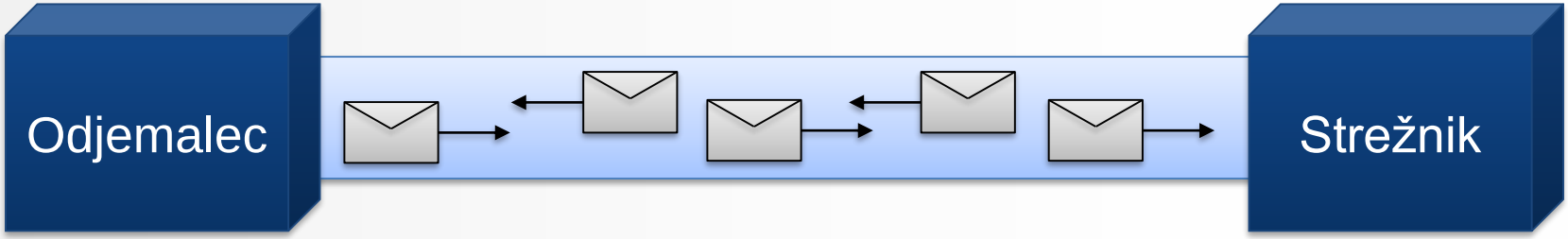
Hypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate

- Dodatna omejitev REST arhitekture (čisti pristop)
- Ideja: V principu bi naj odjemalec komuniciral z aplikacijo preko omrežja zgolj preko hipermedija, ki ga dinamično streže strežnik
- Povzeto po odjemalcu, ki s kliki brska po spletni strani
- REST odjemalec vstopa preko znane vstopne točke v sistem

Realnost?

REST viri

- Odjemalec in strežnik si izmenjujeta **reprezentacijo virov**
- **Oblika** reprezentacije je pomembna
- Vir je v postopku **enolično naslovljen**
- Odjemalcu reprezentacija **zadošča za potrebno manipulacijo** vira
- Reprezentacije sporočil so **samoopisne**
- Sporočila vsebujejo vsa potrebna **navodila za procesiranje**



OBLIKOVANJE RESTFUL STORITEV

RESTFUL API

- RESTful spletni API ali RESTful spletna storitev
  - Na osnovi REST arhitekture in HTTP protokola
  - Osnovni URI storitve
  - Internet media type
  - HTTP akcije
  - Hipertekst

- Poljubno število nivojev lahko izvaja mediacije na zahtevah

REST storitve

- REST storitve gradijo spletni API
- Razumljiv API razvijalcu izven okolja razvoja storitve
- Uporabljiv preko naslovne vrstice brskalnika
- Pragmatičnost pred ideologijo
- Preprost, intiutiven in konsistenten
- Razširljiv
- Efektiven, vendar ohranja ravnovesje z drugimi zahtevami

Viri

- Kaj če bi imeli vire glede na obnašanje/funkcionalnost?

/registrirajUporabnika

/posodobiRacun

/preveriRacun

/vrniRacun

/ustvariRacun

/ustvariSkupino

/aktivirajSkupino

/dodajUporabnika

/posodobiUporabnika

/preveriEmailNaslovRacuna

Z dodajanjem funkcionalnosti se število končnih točk nenadzorovano poveča!

Viri

Vire je potrebno oblikovati enostavno in učinkovito

(grobo zrnati, samostalniki v množini)

Vir zbirke

/razmerja

Vir instance

/razmerja/22312

Akcije

- Za določanje akcij na teh virih se uporabljajo standardne HTTP metode:
  - GET
  - PUT
  - POST
  - DELETE
  - HEAD
- Ni dodatnih poimenovanj
- URL struktura je enostavna, čista in jasna

Akcije

- **GET /razmerja** – Pridobi seznam razmerij
- **GET /razmerja/12** – Pridobi razmerje z ID 12
- **POST /razmerja** – Ustvari novo razmerje
- **PUT /razmerja/12** – Posodobi razmerje z ID 12
- **PATCH /razmerja/12** – Delno posodobi razmerje z ID 12
- **DELETE /razmerja/12** – Izbriše razmerje z ID 12

Akcije

- Akcije, ki ne spadajo v standardne CRUD operacije lahko vključimo na več načinov:
  - Restrukturiranje akcije, da jo predstavimo kot polje na viru
  - Obravnavamo kot pod-vir

```
POST /razmerja/31241/skleni
```

```
POST /razmerja/31241/prekini
```

  - V primeru, da se ne da predstaviti z enim virom, se ustvari novi navidezni vir

```
POST /iskanje
```

Akcije

- GET pomeni branje
- HEAD pomeni branje HTTP zaglavja brez vsebine
- POST pomeni ustvarjanje novega vira
- PUT pomeni posodabljanje celotnega obstoječega vira
- PATCH pomeni delno posodabljanje obstoječega vira
- DELETE pomeni brisanje / deaktiviranje

Akcije

- Relacije virov obravnavamo kot podvire

- **GET /razmerja/12/storitve** – Pridobi seznam storitev za razmerje z ID 10
- **POST /razmerja/12/storitve** – Ustvari novo storitev, ki pripada razmerju z ID 12

- Posamezen element podvira se pridobi na osnovnem URI-ju tega vira

- Podvire gnezdimo samo do prvega nivoja

Akcije

- GET za branje vira
  - Odjemalec pridobi seznam vseh elementov vira

```
GET /razmerja
```

  - Uporabi se HTTP koda 200 za odgovor

```
200 OK

[
    {
      "id": 12351
      "naziv": "Razmerje1",
      ...
    },{
      "id": 12352,
      "naziv": "Razmerje2",
      ...
    },
    ...
]
```

Akcije

- GET za branje točno določenega vira
  - Odjemalcu je znan ID vira (aplikacijska zahteva)

```
GET /razmerja/12351
```

  - Uporabi se HTTP koda 200 za odgovor

```
200 OK

{
   "id": 12351
   "naziv": "Razmerje1"
   ...
}
```

Akcije

- POST za ustvarjanje

```
POST /razmerja

{
  "naziv": "Razmerje5"
  ...
}
```

- Informacija o rezultatu se vrne z odgovorom
- Z odgovorom se vrne ustvarjen vir
- Priporočljivo se uporabi HTTP koda 201 za odgovor

```
201 Created
Location: https://api.ts.si/razmerja/422123

{
  "id": 422123
  "naziv": "Razmerje5"
  ...
}
```

Akcije

- PUT za posodabljanje
  - Polno posodabljanje vira (vse vrednosti morajo biti podane)

```
PUT /razmerja/12352

{
   "id": 12352,
   "naziv": "Razmerje2",
   "tveganje": 4,
   ...
}
```

  - Z odgovorom se vrne posodobljen vir
  - Uporabi se HTTP koda 200 za odgovor

```
200 OK

{
  "id": 12352,
  ...
}
```

Akcije

- DELETE za brisanje elementa vira
  - Izvede se na instanci vira
  - Telo zahtevka mora biti prazno (ID v URI-ju je edini identifikator)

```
DELETE /skladi/AABB22
```

  - Z odgovorom se vrne prazno telo
  - Uporabi se HTTP koda 204 (No Content) za odgovor

```
204 No Content
```

Pot virov

- Osnovni URL do storitve tehnološko ni pomemben
- Enostaven URL omogoča lažjo interpretacijo razvijalcem

https://www.ts.si/razvoj/storitve/api/rest/
ali
https://api.ts.si/

- Prilagojena vsebina glede na tip odjemalca (API odjemalec ali brskalnik)

Verzioniranje

- URL pot storitve lahko odraža verzijo storitve
  - Verzioniranje s potjo storitve je trenutno najbolj uporabljeno

https://api.ts.si/v1/razmerja

- Verzioniranje s pomočjo MIME formata sporočila

```
Content-Type: application/si.ts.tipi.v1.razmerje+json
```

```
Content-Type: application/si.ts.tipi.razmerje+json;v=1
```

  - "Pravilnejši" rest pristop, saj ohranja URL naslove virov
  - Zahteva razumevanje tipov formatov in formuliranje zahtev
  - Slabša podpora odjemalcev in strežnikov

Verzioniranje

- Verzioniranje s pomočjo dodatnega poljubnega polja v HTTP glavi

```
X-API-Version: v1
```

  - V primeru, da se polje ne poda se uporabi privzeta verzija
  - Nestandardno polje – slabša podpora

- Dobre prakse:
  - Verzionira se samo "major" verzija
  - Druge (manjše) spremembe morajo obdržati kompatibilnost za nazaj
  - Stare verzije naj delujejo še določen čas za lažji prehod odjemalcev

Iskanje med zbirkami

- Zbirke rezultatov omejujemo z URL parametri
- Filtriranje rezultatov
  - Vsako polje v elementu je svoj parameter
  - Kot vrednost parametra podamo želeni filter glede na polje

```
GET https://api.ts.si/razmerja?prioriteta=5
```

  - Če želimo bolj fleksibilno in močno iskanje lahko definiramo dodatno sintakso z parametrom "where"

```
GET https://api.ts.si/razmerja?where=vrednost:gte:521
```

- Sortiranje rezultatov
  - Uporabimo generičen parameter "sort" in podamo seznam polj z smerjo

```
GET https://api.ts.si/razmerja?order=naziv ASC,prioriteta DESC
```

Iskanje med zbirkami

- Za namene splošnega iskanja lahko dodamo iskalni parameter
  - Uporabimo generični parameter "q"
  - Uporabimo lahko zunanjo storitev za iskanje (npr. ElasticSearch)

```
GET https://api.ts.si/razmerja?q=alta
```

- Za pogoste poizvedbe lahko naredimo aliase
  - Povečamo preglednost API-ja
  - Omogočamo boljšo izkušnjo razvijalcem odjemalcev
  - Lažje medpomnenje

```
GET https://api.ts.si/novice/danes
```

Iskanje med zbirkami

- Če kombiniramo zgornje filtre lahko sestavljamo slednje poizvedbe:

```
GET /razmerja?q=tilen&order=naziv DESC&status=aktiven
```

```
GET /razmerja?status=zakljucen&order=ustvarjeno DESC
```

```
GET /razmerja?order=posodobljeno ASC,naziv DESC&q=dan
```

```
GET /razmerja?where=vrednost:gte:1223&q=telefon
```

```
GET /razmerja?where=naziv:in:[naziv1,naziv2]&order=status
```

Delne predstavitve virov

- Uporabnik ne potrebuje vedno celotne predstavitve vira
- API naj omogoči delne predstavitve virov
- Določanje polj z naštevanjem parametrov vira v URL parameter „fields"

```
GET https://api.ts.si/razmerja/1223?
                              fields=naziv,aktiven,ustvarjeno

{
   "id": 1223,
   "naziv": "Razmerje2",
   "aktiven": true,
   "ustvarjeno": "2015-02-10T13:00:33.000Z"
}
```

Format sporočil

- Specifikacija formata sporočil
- Pravila razčlenjevanja (parsing)
- Uporabljajo se uveljavljeni MIME formati definicije vsebine sporočil
  - application/json
  - application/xml
  - application/pdf
  - image/jpeg
  - image/gif
  - video/mp4
  - ...
- V zaglavju HTTP sporočila podamo format zapisa

```
Content-Type: application/json
```

Format sporočil

- V zaglavju zahteve podamo sprejemljiv tip formata zapisa odgovora

```
Accept: application/json
```

- Več sprejemljivih tipov ločimo z vejico

```
Accept: application/json,application/xml
```

- V nekaterih primerih so tipi podani z URL končnico
  - Ima večjo prioriteto kot zahteva v zaglavju

```
https://api.ts.si/sektorji/abc123.json
https://api.ts.si/sektorji/abc123.csv
```

Format sporočil

- Kakšen format sporočil?
- JSON (JavaScript Object Notation)
  - Uporabljen MIME tip **application/json**
  - Najpogosteje uporabljen v REST storitvah
  - Enostaven za razčlenjevanje
  - Dobra podpora v vseh okoljih
  - Uporaba v porastu
  - Lahko berljiv
  - Učinkovit z porabo prostora

```
{
   "id": 3,
   "naziv": "Razmerje3",
   "oznaka": "S3",
   "sektor":    {
      "id": 19,
      "naziv": "Razviti trgi"
   },
   "tveganje": 2,
   "zadnjaVrednost":    {
      "datum": "2013-10-
11T00:00:00.000",
      "id": 0,
      "valuta": "EUR",
      "vrednost": 54.24,
   }
}
```

Format sporočil

- Kakšen format sporočil?
- XML (Extensible Markup Language)
  - Uporabljen MIME tip **application/xml**
  - Najpogosteje uporabljen v trenutnih integracijskih rešitvah
  - Dobro integriran v obstoječih integracijskih okoljih
  - Podpora validaciji sporočila glede na shemo

```
<razmerje
xmlns="http://api.ts.si/v1/razmerja">
   <id>3</id>
   <naziv>Razmerje3</naziv>
   <oznaka>S3</oznaka>
   <sektor>
      <id>19</id>
      <naziv>Razviti trgi</naziv>
   </sektor>
   <tveganje>2</tveganje>
   <zadnjaVrednost>
      <datum>2013-10-11T00:00:00.000</datum>
      <id>0</id>
      <valuta>EUR</valuta>
      <vrednost>54.24</vrednost>
   </zadnjaVrednost>
</razmerje>
```

Format sporočil

- Kakšen format sporočil?
- Lastni formati sporočil

  - Nadgradnja osnovnih JSON in XML tipov

```
Content-Type: application/si.ts.tipi.registri+json
```

  - Adaptacija lastnih formatov sporočil po potrebi

Format sporočil

- Oblika zapisa imen polj?
  - snake_case

```
{
   ...
   "zadnja_vrednost": 54.24,
   ...
}
```

  - camelCase

```
{
   ...
   "zadnjaVrednost": 54.24,
   ...
}
```

  - CamelCase zapis priporoča Google*
  - Najpomembnejša je konsistenca

*http://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml

Format sporočil

- Oblika zapisa datuma/časa/časa in datuma
  - Brez lastnih oblik tekstovnega zapisa
  - Uporaba standardnega tekstovnega zapisa ISO 8601
  - Zapisi časa v UTC
  - Omogočena lažja integracija in neodvisnost od programskih okolji
  - Omogočena neodvisnost glede na časovni pas uporabnika/storitve

```
{
   ...
   "ustvarjeno": "2013-10-11T10:12:44.000Z",
   ...
}
```

Ostranjevanje zbirk

- API lahko omogoči ostranjevanje zbirk virov
- Način podajanja parametrov v URL

```
GET https://api.ts.si/razmerja?offset=50&limit=25
```

- Strežnik vrne podatke o nastavljenem ostranjevanju v "Link" polju v glavi odgovora

```
Link: <https://api.ts.si/razmerja?offset=50&limit=0>;rel=prev,
 <https://api.ts.si/razmerja?offset=50&limit=75>;rel=next,
```

Ostranjevanje zbirk

- Lahko vrne tudi kot del telesa z ovito vsebino

```
{
   "metadata": {
       "offset": 50,
       "limit": 25,
       "total": 252
   },
   "data": [
       ...
   ]
}
```

- Uporabljanje ovojnic ni priporočljivo, saj doprinese k težjem razumevanju in bolj kompleksnem API-ju
- V primeru, da se potrebuje število vseh elementov v viru:
  - Uporaba dodatnega polja v HTTP glavi – **X-Total-Count**
  - Uporaba metode count na viru – **GET /razmerja/count**

Povezovanje virov

- Po HATEOS je za skalabilnost najpomembnejši vidik enostavno vključevanje distribuiranih virov
- Vsak dostopen vir ima unikaten URL
- Povezovanje virov zamenja identifikatorje z URL povezavami

```
GET https://api.ts.si/razmerja/1223

{
   "id": 1223,
   "naziv": "Razmerje2",
   "oznaka": "S2",
   "sektor": {
      "id": 123123
      "link": "https://api.skladiapp.si/sektorji/123123"
   },
   "tveganje": 2
}
```

- Slabša integracija v primeru verzioniranja z URL
- Odjemalec potrebuje več klicov na strežnik, da pridobi vse kar želi o elementu vira

Povezovanje virov

- Povezovanje virov v XML se vrši z XLink
- JSON nima posebne podpore za povezovanje virov
- Odprtokodne knjižnice
  - Npr. HAL uporablja objekt _links, ki vsebuje reference nase in morebitne sorodne elemente

- Enostavna rešitev je objekt z parametrom "link" in "id" vira (če gre za element)

```
GET https://api.ts.si/razmerja/1223

{
   "id": 1223,
   "naziv": "Razmerje3",
   "oznaka": "S2",
   "sektor":    {
      "id": 123123
      "link": "https://api.skladiapp.si/sektorji/123123"
   },
   "tveganje": 2
}
```

- Povezava lahko kaže na vir ali zbirko virov

Razširjanje virov

- API lahko omogoči razširjanje določenih delov povezanih virov
- Razširjanje z naštevanjem parametrov vira v URL parameter „expand"
- Z uporabo „link" povezav se struktura ohranja

```
GET https://api.ts.si/razmerja/1223?expand=sektor

{
   "id": 1223,
   "naziv": "Razmerje2",
   "oznaka": "S2",
   "sektor":    {
      "id": 123123
      "link": "https://api.ts.si/sektorji/123123",
      "naziv": "Razviti trgi",
      "tveganje": 2,
      ...
   },
   "tveganje": 4
}
```

Povezave več-proti-več

- Povezave več-proti-več (Many-to-Many) se realizirajo z dodatnimi viri
- Povezava lahko vsebuje dodatne parametre

```
GET https://api.ts.si/skladVlagatelji/445566
{
        "id": 445566,
        "uporabnik": {
                "id": 11223344
                "link": "https://api.ts.si/uporabnik/11223344"
        },
        "sklad": {
                "id": "AA2233",
                "link": "https://api.ts.si/razmerja/2233"
        },
        "ustvarjeno": "2013-10-11T10:12:44.000Z"
}
```

- Alternativno so povezave lahko predstavljene z zbirko znotraj vira

Omejevanje dostopa

- Preprečevanje zlorab
- Omejevanje posameznih odjemalcev
- Omejitev število zahtevkov na časovno enoto
- V primeru prekoračitve omejitve se vrne HTTP status 429 (Too Many Requests)
- Odjemalca se ob vsakem zahtevku obvesti o omejitvah s pomočjo polj v HTTP glavi:
  - **X-Rate-Limit-Limit** – Št. dovoljenih zahtevkov v trenutni časovni enoti
  - **X-Rate-Limit-Remaining** – Št. preostalih zahtevkov v trenutni časovni enoti
  - **X-Rate-Limit-Reset** – Št. preostalih sekund v trenutni časovni enoti

Napake

- Informacije o napakah naj bodo čim bolj informativne
- Podajanje dovolj informacij, da odjemalec reagira
- Sporočilo je lahko pripravljeno na izpis uporabniku
- Vključevanje URL dokumentacije napak v odgovor
- Opcijsko vključevanje informacij za razvijalca

Napake

```
POST https://api.ts.si/razmerja

422 Unprocessable Entity
{
   "status": 422,
   "code": 422008,
   "message": "Neveljavni parametri zahteve",
   "moreInfo": "https://api.ts.si/doc/napake/422008"
   "errors": [
        {
           "code": 422015,
           "field": "oznaka",
           "message": "Zahtevano polje ni podano",
           "moreInfo": "https://api.ts.si/doc/napake/422015",
        },
        ...
   ]
}
```

Varnost

- REST je stateless, izogibanje sejam, če je to le mogoče
- Avtentikacija preko obstoječega protokola
  - Vedno SSL
  - HTTP Basic avtentikacija
  - OAuth2
  - Zunanji ali notranji ponudnik
- Lastna avtentikacijska shema v posebnih primerih
  - Obvezna distribucija SDK
- Avtorizacija glede na vsebino, ne glede na URL

HTTP kode

- Seznam „uporabnih" HTTP kod za uspešno procesiranje:
  - 200 OK – Odgovor na uspešno akcijo
  - 201 Created – Odgovor na uspešno akcijo, ki rezultira v kreiranje vira.
  - 204 No Content – Odgovor na uspešno akcijo brez vsebine odgovora
  - 304 Not Modified – Informacija odjemalcu, da drži aktualno medpomnjeno instanco

HTTP kode

- Seznam "uporabnih" HTTP kod za napake:
  - 400 Bad Request – Zahteva je neustrezno oblikovana, vsebine ni mogoče razčleniti, podatki manjkajo
  - 401 Unauthorized – Avtentikacija uporabnika ni uspešna
  - 403 Forbidden – Avtorizacija uporabnika do vira ni uspešna
  - 404 Not Found – Zahteva po viru, ki ne obstaja
  - 405 Method Not Allowed – Zahteva po HTTP metodi, ki uporabniku ni dovoljena
  - 410 Gone – Vir na URL ne obstaja več (uporabno za verzioniranje)
  - 415 Unsupported Media Type – Tip vsebine zahteve ni veljaven
  - 422 Unprocessable Entity – Validacijska napaka
  - 429 Too Many Requests – Zahteva zavrnjena zaradi preobremenitve strežnika ali preveč zahtev (uporaba X-Rate-Limit-... značk)
  - 500 Internal Server Error – Generalna napaka na strežniku

HTTP medpomnenje

- Uporablja naj se standardno HTTP medpomnjenje
- Odgovor z virom vsebuje informacijo o verziji ali časovno značko (trije standardni načini zapisa)

```
ETag: 123231abca6c5a4d77ff
```

```
Last-Modified: Mon, 14 Oct 2013 06:12:31 GMT
```

- Nadaljnja zahteva po viru

```
If-None-Match: 123231abca6c5a4d77ff
```

```
If-Modified-Since: Mon, 14 Oct 2013 06:12:31 GMT
```

- Strežnik vrne novo instanco vira ali vrne HTTP kodo „304 Not Modified"

Kompresiranje vsebine

- Za zmanjšanje uporabe pasovne širine omrežja
- Opcijsko, ne obvezno
- Kompresiranje z gzip ali base64 (deflate)
- Definicija kompresije sporočila v HTTP zaglavju

```
Content-Encoding: gzip
```

- Sprejemljivi tipi kompresiranja odgovora

```
Accept-Encoding: gzip, deflate
```

- Kompresiranje lahko zmanjša velikost sporočil za do 90%

Dokumentiran vmesnik

- Je sploh potreben?
- Web Service Description Language (WSDL) 2.0
- Web Application Description Language (WADL) – ni standardiziran in se v večini ne uporablja
- Najpogosteje se uporablja opisna dokumentacija s primeri uporabe

STORITVE REST V JAVI

API

- REST storitve v Javi implementiramo s pomočjo JAX-RS (Java API for RESTful Web Services)
- JAX-RS 2.0 (JSR 339) je del uradne specifikacije Java EE 7
- Trenutne implementacije:
  - Jersey (RI, Oracle, je del GlassFish 4.1)
  - RESTeasy (JBoss Community, je del WildFly 8.2)
- JAX-RS API uporablja anotacije za dekoriranje programske kode z navodili za vzpostavitev REST storitev

REST aplikacija

- REST aplikacijo definiramo z aplikacijskim razredom
- **@ApplicationPath** definira relativno pot namestitve rest aplikacije
- Primer poti:

https://api.skladi.si/v1

```
@ApplicationPath("/v1")
public class RestStoritve extends javax.ws.rs.core.Application
{

	@Override
	public Set<Class<?>> getClasses() {

		Set<Class<?>> resources = new
			java.util.HashSet<Class<?>>();
		resources.add(RazmerjaStoritev.class);
		return resources;
	}
}
```

Poti virov

- REST storitev in njene vire umestimo z anotacijo **@Path**

```
@Path("razmerja")
public class RazmerjaStoritev {

        public Response vrniRazmerja(...) {... }

        @Path("{id}")
        public Response vrniRazmerje(...) {... }

        @Path("{id}/vrednosti")
        public Response vrniVrednosti(...) {... }

        @Path("{id}/vrednosti/{idVrednosti}")
        public Response vrniVrednost(...) {... }
}
```

- Primer poti:

https://api.ts.si/v1/razmerja/1212/vrednosti/123441

Navezovanje na HTTP metode

- Z anotacijami **@GET, @POST, @PUT, @DELETE in @HEAD** definiramo na katere HTTP akcije implementirajo katere javne metode storitve
- **@HEAD, @OPTIONS** vračata informacije o viru brez vsebine vira,
  (nekatere implementacije pripravijo metodi samodejno)

```
@GET
@Path("{id}")
public Response vrniRazmerje(...) {... }

@POST
public Response dodajRazmerje(...) {... }

@PUT
@Path("{id}")
public Response posodobiRazmerje(...) {... }

@DELETE
@Path("{id}")
public Response odstraniRazmerje (...) {... }
```

Parametri poti virov

- Parametre poti lahko definiramo z regularnim izrazom
- Na parametre poti se sklicujemo z **@PathParam**

```
@Path("{id}/vrednosti/{idVrednosti: \d{4}-\d{2}-\d{2}}")
public Response vrniVrednost(
        @PathParam("id")
        String skladId,
        @PathParam("idVrednosti")
        String vrednostId
) {... }
```

Parametri poti virov

- Preostale anotacije za vstavljanje vrednosti parametrov:
  - **@QueryParam** za parametre povpraševanja
    (../skladi?filter=naziv)
  - **@HeaderParam** za vrednosti HTTP zaglavja
  - **@MatrixParam** za matrične vrednosti
    (../skladi;filter=aktivni/vrednosti;items=6)
  - **@CookieParam** za vrednosti prejetih piškotov
  - **@FormParam** za vrednosti spletnih obrazcev

- **@DefaultValue** se uporabi za privzete vrednosti

```
public Response vrniRazmerja (
        @DefaultValue("*")
        @QueryParam("filter")
        String filter
) {... }
```

Parametri poti virov

- Javanski tipi v katere lahko preslikamo parametre
  - Primitivni javanski tipi
  - Tipi s konstruktorjem, ki sprejme samo en String atribut
  - Tipi z registriranim konverterjem nadtipa **ParamConverter**
  - Tipi s statično metodo **valueOf** ali **fromString**, ki sprejme samo en String atribut

```
public Response vrniRazmerja(
        @QueryParam("filter")
        FilterParam filter
) {... }
```

- Anotacija **@Encoded** na metodi ali razredu onemogoči preslikavo parametrov

Atributi metod

- Atribut metode vira, ki ne uporablja nobene anotacije parametrov je **entitetni parameter**
- Preslika se iz vsebine zahteve

```
public Response zapisiRazmerje(Razmerje razmerje) {
        return razmerje;  }
```

- Metode virov vračajo enega izmed tipov
  - Brez tipa (**void**) – odgovor vrne HTTP status 204 No Content
  - **Response** – vrne entiteto in poljubni HTTP status.
  - **GenericEntity** – wrapper za entitete zavija entiteto v atribut **entity**
  - **Poljubni razred** – vrne vsebino entitete

- Metode vračajo status HTTP 200 kadar je vsebina entitete definirana in status HTTP 204 kadar vsebina ni definirana

Formati sporočil

- Z **@Consumes** določimo sprejemljive formate sporočila zahteve
- Z **@Produces** določimo možne formate sporočila odgovora

```
@Path("skladi")
@Consumes({ MediaType.APPLICATION_JSON, "application/xml" })
@Produces({ "application/json"})
public class RazmerjaStoritev {

        @GET
        @Path("{id}")
        public Response vrniRazmerje(...) {... }

        @GET
        @Path("{id}")
        @Produces({ "application/xml"})
        public Response vrniRazmerje (...) {... }

        @POST
        @Consumes({"application/si.ts.v1.razmerje+json"})
        public void dodajRazmerje(...) {... }
}
```

Ponudniki formatov entitet

- Z **@Provider** označimo programsko kodo, ki služi kot razširitev JAX-RS aplikacije

- **MessageBodyReader** za razčlenjevanje (InputStream v entiteto)
  - Metodi isReadable in readFrom

- **MessageBodyWriter** za produciranje sporočil (entiteta v OutputStream)
  - Metode getSize (zastarela), isWriteable, writeTo

- **ContextResolver<T>** za zagotavljanje konteksta drugim virom in ponudnikom

- **ExceptionMapper<T>** za preslikavo proženih izjem v Response objekt, ki se vrne odjemalcu

Ponudniki formatov entitet

- Primer

```
@Provider
@Consumes("text/xml")
public class RazmerjeReader implements MessageBodyReader<Razmerje> {

        public boolean isReadable(java.lang.Class<Sklad> type,
                java.lang.reflect.Type genericType,
                java.lang.annotation.Annotation[]
                annotations, MediaType mediaType) {
                ...
        }

        public Sklad readFrom(java.lang.Class<Sklad> type,
                java.lang.reflect.Type genericType,
                java.lang.annotation.Annotation[]
                annotations, MediaType mediaType, MultivaluedMap<
                java.lang.String,java.lang.String> httpHeaders,
        java.io.InputStream
                entityStream) {
                ...
        }
}
```

Ponudniki formatov entitet

- Primer

```
@Provider
@Consumes("text/xml")
public class RazmerjeWriter implements MessageBodyWriter<Razmerje> {

        public boolean isWriteable(Class<?> aClass, Type type,
                Annotation[] annotations, MediaType mediaType) {
                ...
        }

        public long getSize(Razmerje r, Class<?> aClass, Type type,
                Annotation[] annotations, MediaType mediaType) {
                ...
        }

        public void writeTo(Movie movie, Class<?> aClass, Type type,
                Annotation[] annotations, MediaType mediaType,
                MultivaluedMap<String, Object> multivaluedMap,
                OutputStream outputStream) throws IOException,
                WebApplicationException {
                ...
        }
}
```

Ponudniki formatov entitet

- Implementacija JAX-RS 2.0 mora vsebovati ponudnike pretvornikov v obe smeri:
  - **byte[]** – Vsi formati (*/*)
  - **java.lang.String** – Vsi tekstovni formati (*/*)
  - **java.io.InputStream** – Vsi formati (*/*)
  - **java.io.File** – Vsi formati (*/*)
  - **javax.activation.DataSource** – Vsi formati (*/*)
  - **javax.xml.transform.Source** – Vsi XML formati (text/xml, application/xml, application/*+xml)
  - **javax.xml.bind.JAXBElement** – Vsi XML formati (text/xml, application/xml, application/*+xml)
  - **MultivaluedMap<String,String>** – Vsebina obrazca (application/x-www-form-urlencoded)

Ponudniki konteksta

- Konfiguracijo komponent JAX-RS podamo z uporabo anotacije @Provider
  - npr. nastavitve formata datuma JSON serializacije

```
@Provider
public class JacksonProvider implements ContextResolver<ObjectMapper> {

    final ObjectMapper defaultObjectMapper;

    public JacksonProvider () {
        defaultObjectMapper = new ObjectMapper();
    }

    @Override
    public ObjectMapper getContext(Class<?> type) {
        SimpleDateFormat dt = new SimpleDateFormat("yyyy-MM-dd HH:mm a z");
        defaultObjectMapper.setDateFormat(df);
        return defaultObjectMapper;
    }
}
```

Izjeme

- Vse aplikacijske in sistemske izjeme metode vira naj bodo ulovljene in ustrezno procesirane:
  - Proži izjemo **WebApplicationException**, ki nosi objekt **Response** (HTTP status in entiteto napake)
  - Proži izjemo, ki ima definiran **ExceptionMapper** ponudnika

```
@Provider
public class SkladiIzjema implements
ExceptionMapper<SkladiIzjema> {

    @Override
    public Response toResponse(SkladiIzjema izjema) {
        return Response.status(
                        Response.Status.BAD_REQUEST).build();
    }
}
```

Filtri in prestrezniki

- JAX-RS 2.0 definira dva nova tipa ponudnikov

- Vmesniki filtrov (metoda filter):
  - **ClientRequestFilter**
  - **ClientResponseFilter**
  - **ContainerRequestFilter**
  - **ContainerResponseFilter**

- Vmesniki interceptorjev entitet
  - **ReaderInterceptor** (metoda aroundReadFrom)
  - **WriterInterceptor** (metoda aroundWriteTo)

- Vsi morajo biti anotirani z @Provider

Filtri

- Primer filtra za logiranje
- Filtri najpogosteje uporabljajo za manipulacijo glave sporočil

```
@Provider
class LoggingFilter implements ContainerRequestFilter,
        ContainerResponseFilter {

        public void filter(ContainerRequestContext
                                requestContext)
                throws IOException {
                        ...
        }

        public void filter(ContainerRequestContext
requestContext,
                ContainerResponseContext responseContext)
                        throws IOException {
                ...
        }
 }
```

Filtri

- Filtre lahko registriramo na posamezne metode
- Definiramo lastno anotacijo, katero anotiramo z @NameBinding

```
@NameBinding
public @interface PoljubenFilter {}
```

- Nato to anotacijo uporabimo na filtru in metodah, na katere želimo ta filter aplicirati

```
@PoljubenFilter
public class MojFilter implements ContainerRequestFilter {...}


@Path
public class MyResource {
   @GET
   @PoljubenFilter
   public String get() {...}
```

Prestrezniki

- Primer prestreznika za kompresiranje odgovora
- Prestrezniki se uporabljajo za manipuliranje teles sporočil

```
@Provider
public class GZIPEncoder implements WriterInterceptor {

   public void aroundWriteTo(WriterInterceptorContext ctx)
               throws IOException, WebApplicationException {
      GZIPOutputStream os = new
       GZIPOutputStream(ctx.getOutputStream());

      try {
         ctx.setOutputStream(os);
         return ctx.proceed();
      } finally {
         os.finish();
      }
   }
}
```

Kontekst storitve

- Z anotacijo @Context lahko vstavimo odvisnost v objekte tipa:
  - **UriInfo** – informacije o komponentah zahteve (URI, ipd.)
  - **HttpHeaders** – informacije o HTTP zaglavju zahteve
  - **SecurityContext** – informacije o varnostnem kontekstu (poverilnica, vloge, uporabljen način avtentikacije)
  - **Request** – celotno zahtevo (če želimo ročno odločati o vrsti procesiranja)
  - **Providers** – vsi registrirani ponudniki funkcionalnosti
  - **ResourceContext** – Kontekst priprave vira
  - **Configuration** – konfiguracija strežnika / odjemalca

Asinhrono izvajanje

- Ogrodje omogoča asinhrono procesiranje REST metod na osnovi asinhronega izvajanja podprtega v Servlet 3.0 specifikaciji
- Sprošča niti, ki upravljajo s HTTP povezavami
- Odjemalec ne bo opazil razlik času zahtevka
- Povečana efektivnost strežnika pri veliki količini povezav

```
@Path("/skladi")
public class RazmerjaStoritev {
    @GET
    public void asyncRazmerja(
                @Suspended final AsyncResponse asinhroniOdgovor) {

        new Thread(() -> {
            String rezultat = dragoPridobivanjeRazmerij();
            asinhroniOdgovor.resume(rezultat);
        }).start();
    }
}
```

Asinhrono izvajanje

- Določanje maksimalnega časa izvajanja dolgotrajne operacije
- V primeru prekoračitve naj se vrne napaka 503 (SERVICE_UNAVAILABLE)

```
@Path("/razmerja")
public class RazmerjaStoritev {
    @GET
    public void asyncRazmerja(
                @Suspended final AsyncResponse asinhroniOdgovor) {
        asyncResponse.setTimeoutHandler(asyncResponse ->
                asyncResponse.resume(Response.status(503)
                        .entity("Operation time out.").build());
        );

        asyncResponse.setTimeout(20, TimeUnit.SECONDS);

        new Thread(() -> {
            String rezultat = dragoPridobivanjeRazmerij();
            asinhroniOdgovor.resume(rezultat);
        }).start();
    }
}
```

Asinhrono izvajanje

- Možna tudi uporaba objekta "CompletableFuture"
- Omogoča nam uporabo "promise" vzorca
- Podpora "reactive" stila programiranja
- Lahko podamo poljuben bazen niti, ki nam ga zagotavlja strežnik

```
@Path("/razmerja")
public class RazmerjaStoritev {
    @GET
    public void asyncRazmerja(
                @Suspended final AsyncResponse asinhroniOdgovor) {
          CompletableFuture
                .runAsync(() -> dragoPridobivanjeRazmerij())
                .thenApply(result -> asyncResponse.resume(result));
    }
}
```

Integracija v Java EE

- JAX-RS lahko sodeluje z Enterprise Java Beans (EJB) ali Context and Dependency Injection (CDI) zrni
  - Primer EJB

```
@Stateless
@Path("skladi")
public class SkladiStoritev {...}
```

  - Primer CDI

```
@RequestScoped
@Path("skladi")
public class SkladiStoritev {...}
```

Podatkovne preslikave

- JAX-RS podpira podatkovne preslikave
  - Med XML zapisom in Java objekti – Java API for XML Binding
  - Med JSON zapisom in Java objekti – Java API for JSON Processing

```
public class Sklad {

            private long id;
            private String naziv;
            private String oznaka;
            private Integer tveganje;
            private ZadnjaVrednost zadnjaVrednost;
            private Druzba druzba;
            private Sekt
            private Tip
            private Set<
            private Stat
}
```

```
{
   "id": 3,
   "naziv": "Sklad2",
   "oznaka": "S2",
   "sektor":    {
      "id": 19,
      "naziv": "Razviti trgi"
   },
   "tveganje": 2,
   "zadnjaVrednost":    {
      "datum": "2013-10-11T00:00:00.000",
      "id": 0,
      "valuta": "EUR",
      "vrednost": 54.24,
   }
}
```

```
<sklad xmlns="http://api.skladiapp.si/v1/skladi">
   <id>3</id>
   <naziv>Sklad2</naziv>
   <oznaka>S2</oznaka>
   <sektor>
      <id>19</id>
      <naziv>Razviti trgi</naziv>
   </sektor>
   <tveganje>2</tveganje>
   <zadnjaVrednost>
      <datum>2013-10-11T00:00:00.000</datum>
      <id>0</id>
      <valuta>EUR</valuta>
      <vrednost>54.24</vrednost>
   </zadnjaVrednost>
</sklad>
```

API za gradnjo odjemalcev

- JAX-RS 2.0 specifikacija vključuje odjemalca REST storitev
- Za izvajanje klicev se uporablja **ClientBuilder**
- Z **WebTarget** definiramo končno točko vira
- Vključimo tudi spremenljivke poti

```
Client client = ClientBuilder.newClient();
WebTarget base = client.target("https://api.ts.si/");
WebTarget razmerja = base.path("razmerja");
WebTarget razmerje = razmerja.path("{id}").queryParam("id",
       "1211");
```

- Klic storitve

```
Response odgovor = sklad.request().get();
Object objekt = odgovor.getEntity();
```

API za gradnjo odjemalcev

- Klienta lahko izvajamo tudi asinhrono s pomočjo "Future" objekta

```
Client client = ClientBuilder.newClient();
Future<Stranka>
future = client.target("http://api.ts.si/stranke")
               .queryParam("ime", "Tilen Faganel")
               .request()
               .async()
               .get(Stranka.class);

Stranka str = future.get();
```

- Še vedno blokiramo nit, ko čakamo na razrešitev Future objekta
- Za popolnoma asinhron klic uporabimo objekt ComparableFuture, ki implementira "promise" vzorec

```
CompletableFuture.supplyAsync(() -> future.get())
                 .thenApply(System.out::println)
```

API za gradnjo odjemalcev

- Alternativno se lahko uporablja tudi "callback" vzorec

```
InvocationCallback<Response> callback =
new InvocationCallback() {

   public void completed(Response res) {
      System.out.println("Zahteva uspela!");
   }

   public void failed(ClientException e) {
      System.out.println("Zahteva ni uspela!");
   }
};

client.target("http://api.ts.si/stranke")
      .queryParam("ime", "Tilen Faganel")
      .request()
      .async()
      .get(callback);
```

API za gradnjo odjemalcev

- Podobno kot na strežniku, lahko klientu registriramo filter

```
public class PreveriZahtevoFilter
                        implements ClientRequestFilter {

    @Override
    public void filter(ClientRequestContext rc)
                            throws IOException {
        if (rc.getHeaders().get("Client-Name") == null) {
            requestContext.abortWith(Response.status(400)
                .entity("Client-Name mora biti podan.")
                        .build());
         }
    }
}
```

```
client.target("http://api.ts.si/stranke")
      .register(PreveriZahtevoFilter.class)
      ...
```

API za gradnjo odjemalcev

- Enako z interceptorji

```
public class GZIPWriterInterceptor implements WriterInterceptor
{
    @Override
    public void aroundWriteTo(WriterInterceptorContext context)
                throws IOException, WebApplicationException {
        OutputStream os = context.getOutputStream();
        context.setOutputStream(new GZIPOutputStream(os));
        context.proceed();
    }
}
```

```
client.target("http://api.ts.si/stranke")
      .register(GZIPWriterInterceptor.class)
      ...
```

PROCESIRANJE JSON FORMATA V JAVI Z JSON-P

API

- Zaradi popularnosti JSON formata se je pojavilo veliko št. knjižnic
- Pojavila se je potreba po standardiziranju API-ja za manipulacijo JSON formata
- JSON-P predstavlja prvi del standardiziranega vmesnika
- JSON-P 1.0 (JSR-353) je del uradne specifikacije Java EE 7
- Prinaša standardiziran nizkonivojski vmesnik za manipulacijo JSON sporočil
- Vključuje dva API-ja in sicer:
  - Streaming API – podobno kot StAX pri JAXP
  - Objektni model API – podobno kot DOM

Struktura

- JSON struktura vsebuje preproste konvencije, podobno kot Javanski razredi

- Struktura lahko predstavlja:
  - Kolekcija ključ/vrednost parov
  - Urejen seznam vrednosti

- JSON vsebuje tako primitivne kot strukturirane podatkovne tipe:
  - Število: Zapisano brez narekovajev
  - String: Zapisan znotraj dvojnih narekovajev
  - Seznam: Urejen seznam vrednosti znotraj [], ločeni z vejico
  - Objekt: Urejen seznam ključ/vrednost parov definirani znotraj {}
  - Vrednost: Katerikoli od zgornjih tipov

Struktura

- Primer JSON dokumenta

```
{
     "ime": "Sara",
     "priimek": "Jakopič",
     "naslov": {
         "ulica": "Pod lipami 01",
         "mesto": "Ljubljana",
         "postnaStevilka": 1000,
         "drzava": "Slovenija"
     },
     "telefonskeStevilke": [
         "031 344 112",
         "031 133 312"
     ]
 }
```

Streaming API

- Streaming API je nizkonivojski in učinkoviti način generiranja in parsanja JSON
- Za dostop do podatkov se uporabljajo tokovi
- Beremo oz. pišemo lahko vedno samo naprej
- Vsebuje 2 glavni abstrakciji
  - **JsonParser** – omogoča branje toka (forward access) JSON datoteke
  - **JsonGenerator** – prinaša metode za zapis JSON-a v tok z podporo veriženja metod
- Za ustvarjanje objektov uporabimo JsonParserFactory in JsonGeneratorFactory

Streaming API

- Primer:
  - Imamo sledečo JSON datoteko – seznam telefonskih številk stranke

```
[
  {
    "tip" : "domaci",
    "stevilka" : "(01) 11 11 111"
  },
  {
    "tip" : "mobilni",
    "stevilka" : "(02) 22 22 222"
  },
  {
    "tip" : "sluzbeni",
    "stevilka" : "(03) 33 33 333"
  }
]
```

Streaming API

- Za parsanje se uporablja model "pull-parsing programming"
  - Metoda next() vrne naslednji dogodek, ki je lahko:
    - START_ARRAY
    - END_ARRAY
    - START_OBJECT
    - KEY_NAME
    - VALUE_STRING
    - VALUE_NUMBER
    - VALUE_TRUE
    - VALUE_FALSE
    - VALUE_NULL
  - Za vsak dogodek nato pridobimo vrednost (če obstaja) z metodo getString()

Streaming API

- Primer:
  - Dogodki na primeru JSON datoteke

```
[START_ARRAY
  {START_OBJECT
    "tip"KEY_NAME : "domaci"VALUE_STRING,
    "stevilka"KEY_NAME : "(01) 11 11 111"VALUE_STRING
  }END_OBJECT,
  {START_OBJECT
    "tip"KEY_NAME : "mobilni"VALUE_STRING,
    "stevilka"KEY_NAME : "(02) 22 22 222"VALUE_STRING
  }END_OBJECT,
  {START_OBJECT
    "tip"KEY_NAME : "sluzbeni"VALUE_STRING,
    "stevilka"KEY_NAME : "(03) 33 33 333"VALUE_STRING
  }END_OBJECT
]END_ARRAY
```

Streaming API

- Primer:
  - Sprehodimo se čez vse številke in vsako izpišemo

```
JsonParserFactory factory = Json.createParserFactory(null);
JsonParser parser = factory.createParser(new FileReader(json));

while (parser.hasNext()) {
  Event event = parser.next();

  switch (event) {
    case KEY_NAME: {
      System.out.print(parser.getString() + "="); break;
    }
    case VALUE_STRING: {
      System.out.println(parser.getString()); break;
    }
  }
}
```

Streaming API

- Primer:
  - Zgeneriramo prejšnji JSON

```
JsonParserFactory factory = Json.createParserFactory(null);
JsonGeneratorFactory factory =
Json.createGeneratorFactory(null);

JsonGenerator generator = factory.createGenerator(System.out);

generator.writeStartArray().
  writeStartObject().
    write("tip", "domaci").
    write("stevilka", "(01) 11 11 111").writeEnd().
  writeStartObject().
    write("tip", "mobilni").
    write("stevilka", "(02) 22 22 222").writeEnd().
  writeStartObject().
    write("tip", "sluzbeni").
    write("stevilka", "(03) 33 33 333").writeEnd().
  writeEnd().close();
```

Objekti model API

- Objektni model API višjenivojski vmesnik za obdelovanje JSON-a
  - Enostaven, preprost za uporabo
  - Osnovan na streaming API-ju

- Zgradi drevesno strukturo JSON-a v spominu, katero lahko preprosto navigiramo in poizvedujemo

- Vsebuje 2 glavni abstrakciji, ki nista spremenljivi (immutable)
  - **JsonObject** – ponudi Map vmesnik za dostop do kolekcije ključ/vrednost parov
  - **JsonArray** – ponudi List vmesnik za dostop do seznama vrednosti

- Za ustvarjanje objektov se uporabi JsonObjectBuilder in JsonArrayBuilder

Objekti model API

- Primer:
  - Ponovno zgenerirajmo prejšnji JSON, tokrat z OM Api-jem

```
JsonBuilderFactory factory = Json.createBuilderFactory(null);

JsonArray jsonArray = factory.createArrayBuilder()

    .add(factory.createObjectBuilder().
        add("tip", "domaci").
        add("stevilka", "(01) 11 11 111"))

    .add(factory.createObjectBuilder().
        add("tip", "mobilni").
        add("stevilka", "(02) 22 22 222"))

    .add(factory.createObjectBuilder().
        add("tip", "sluzbeni").
        add("stevilka", "(03) 33 33 333")).build();
```

Objekti model API

- Primer:
  - Branje JSON-a

```
JsonReader jr = Json.createReader(new FileReader(json));

JsonArray jsonArray = jsonReader.readArray();

value.getValuesAs(JsonObject.class).stream()
        .map(v -> v.get("stevilka"))
        .forEach(System.out::println);
```

Objekti model API

- Objekt lahko zapišemo v datoteko s pomočjo razreda **JsonWriter**

```
JsonWriter jsonWriter = Json.createWriter(System.out))

jsonWriter.writeArray(jsonArray);
```

- Podobno lahko preberemo iz datoteke s pomočjo razreda **JsonReader**

```
JsonReader jsonReader = Json.createReader(new FileReader(json))

JsonArray array = jsonReader.readArray();
System.out.println(array);
```

Objekti model API

- Objekt lahko zapišemo v datoteko s pomočjo razreda **JsonWriter**

```
JsonWriter jsonWriter = Json.createWriter(System.out))

jsonWriter.writeArray(jsonArray);
```

- Podobno lahko preberemo iz datoteke s pomočjo razreda **JsonReader**

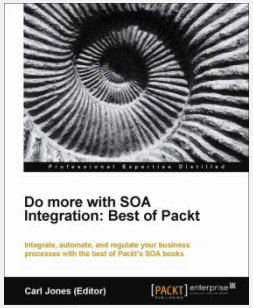
```
JsonReader jsonReader = Json.createReader(new FileReader(json))

JsonArray array = jsonReader.readArray();
System.out.println(array);
```

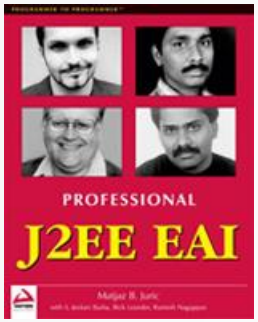
Prihodnost

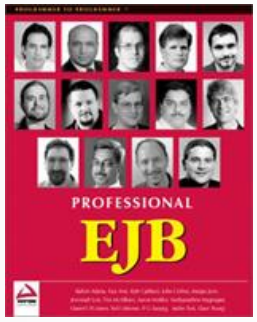
- JSON-P 1.1 (JSR-374) bo prinesel podporo JSON Pointer in JSON Patch specifikacijam

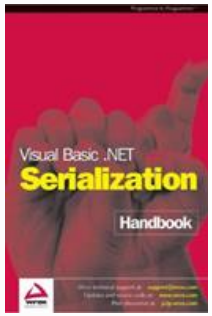
- Omogočalo bo lažje identificiranje vsebine v JSON datoteki in opravljanje operacij (transformacij) nad njo

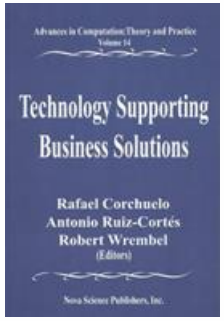
- Java EE 8 bo vključevala JSON-B 1.0 (JSR-367)
  - Razširitev JSON-P specifikacije
  - Omogočal bo standardiziran način vezave Java objektov na JSON dokumente (podobno kot JAXB)

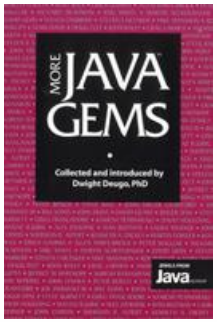












HVALA!

Java Champions · ORACLE ACE Director · IBM CHAMPION · ORACLE Certified Expert · Certified for IBM SOA technology · Certified for IBM CLOUD COMPUTING technology

e-naslov: **http://www.cloud.si**
e-naslov: **http://www.soa.si**
e-pošta: **info@cloud.si**