





---

OIOREST – webservice design.  
Guideline til design af REST-baserede  
webservices.  
Udgivet af:  
IT- & Telestyrelsen

IT- & Telestyrelsen  
Holsteinsgade 63  
2100 København Ø

Telefon: 3545 0000  
Fax: 3545 0010

Publikationen kan også hentes  
på IT- & Telestyrelsens  
Hjemmeside: <http://www.itst.dk>  
ISBN (internet):

---

>

---

---

# **OIOREST – webservice design**

Guideline til design af REST-baserede webservices.

Version 1.00



---

# 1. Indhold

>

---

1.	Indhold	5
2.	Indledning	6
3.	Formål	7
4.	Designovervejelser	8
4.1	Udvælgelse af data	8
4.2	Repræsentation	10
4.3	Operationer	12
4.4	Fejlscenarier	12
4.5	Caching	13
4.6	Queries	14
4.7	Servicespecifikationen	14
5.	Opsummering	15

---

## 2. Indledning

>

---

Der opleves en stadig større udbredelse i antallet af REST-baserede webservices, og det er oplagt, at meget offentligt data i Danmark (og for så vidt også på europæisk og globalt plan) med fordel kan udstilles vha. denne metode.

Traditionelt er REST-baserede services rettet mod mennesker og web-baserede GUI'er, men også indenfor system-system-kommunikation vil REST åbne op for en række nye muligheder.

For at lette arbejdet med at udstille offentlig data vha. REST-baserede webservices, har IT- og Telestyrelsen udarbejdet en række dokumenter, som dels diskuterer mulighederne og perspektiverne, dels giver en række retningslinier til støtte for frembringelsen og brugen af REST-baserede webservices.

---

### 3. Formål

>

---

Formålet med nærværende dokument er at yde støtte til frembringelsen af en REST-baseret webservice. Dokumentet vil centrere sig omkring et levende og realistisk eksempel og benytte dette til diskussion af centrale problemstillinger.

Undervejs i dokumentet vil der være henvisninger til bogen ”RESTful Web Services”, skrevet af Leonard Richardson og Sam Ruby. Denne bog er i høj grad et reference-opslagsværk indenfor REST-baserede webservices, og i henvisningerne hertil vil vi omtale den som ”referencebogen”.

---

## 4. Designovervejelser

>

---

Når man skal udarbejde en REST-baseret webservice, melder der sig i sagens natur en række spørgsmål - eller rettere en række muligheder. Lad os i det følgende kigge nærmere på, hvilke overvejelser vi gør os - båret af det konkrete eksempel.

Udgangspunktet er, at vi har en database, der indeholder samtlige adresser i Danmark og herunder, knyttet til hver adresse, information om region, kommune, valg- og skoledistrikt, vej og ikke mindst adressens koordinater.

Opgaven består i at udstille data, så man kan udsøge adresserne og evt. plotte dem ind på et kort.

Det komplette eksempel kan ses på: <http://oiorest.dk/danmark>

### 4.1 Udvalgelse af data

Erfaringerne viser, at jo mere data, der udstilles, jo flere anvendelsesmuligheder opstår der, jo mere kreativitet ser man på aftagersiden, og jo flere "uforudsete" klienter opstår der.

REST er grundlæggende en model til udstilling af data, og en veludvalgt REST-service udstiller ikke kun data til de usecases, som er kendte på design-tidspunktet, men "åbner" i stedet datasættet og udstiller også data, som eventuelt kunne blive interessant for andre aftagere eller for andre use cases. Når man designer REST-tjenesten, kan det være en god hjælp at se det som udstilling af *punkter* i datasættet (eller objektgrafene som afspejles af datasættet). Typisk vil data eller dele af data udspænde en træstruktur, og man vil være tilbøjelig til at fokusere på bladknuderne, fordi de ofte udgør de mest oplagte entiteter i datasættet, men hvis man i stedet udstiller både bladknuderne og forældrene hertil, giver det mulighed for at tilgå både entiteter og collections heraf - og derudover ligger det lige for at introducere søgefaciliteter. Forældrenes forældre eller relationer til andre punkter i datasættet åbner nye muligheder for at navigere i data, og dette skal sammenholdes med, at det formentlig ikke koster væsentligt mere at udstille praktisk talt alt, når man først har introduceret den underliggende motor.

I vores eksempel råder vi over begreber som regioner, kommuner, sogne, byer, adresser osv., og vi beslutter os for at udstille data som følger (her lidt forenklet):

URI	Hvad skal returneres?	Hvad skal være indeholdt i svaret?
/regioner	En liste af	Links til de

	regioner	enkelte regioner
/regioner/<regionsnummer>	Den enkelte region	Regionens navn og nummer Links til regionens kommuner og adresser
/kommuner	En liste af kommuner	Links til de enkelte kommuner
/kommuner/<kommunenummer>	Den enkelte kommune	Kommunens navn og nummer Links til kommunens veje og adresser
/postdistrikter	En liste af postdistrikter	Links til de enkelte postdistrikter
/postdistrikter/<postdistriktnummer>	Det enkelte postdistrikt	Postdistriktets navn Links til postdistriktets veje og adresser
/kommuner/<kommunenummer>/veje	En liste af kommunens veje	Links til de enkelte veje i kommunen
/kommuner/<kommunenummer>/vej/<vejnavn>	Den enkelte vej	Links til vejens kommune
/kommuner/<kommunenummer>/adresser	En liste af kommunens adresser	Links til kommunens adresser
/kommuner/<kommunenummer>/adresse	Den enkelte adresse	Adressens koordinater Links til vejen, postdistriktet,

---

>

---

		kommunen
OSV...	...	...

Læg mærke til, at URL'ere (egentlig URI'erne) afspejler hierakiet i data, på en læselig og entydig måde.

For en nærmere diskussion af processen omkring udvælgelsen af data, se kapitel 5 i referencebogen, specielt afsnittet "Figure Out the Data Set".

## 4.2 Repræsentation

Nu hvor vi har besluttet os for, hvad vi vil udstille, på hvilke URL'er og ikke mindst med ord har beskrevet, hvordan de enkelte data skal hænge sammen, er vi klar til at konkretisere det, som skal returneres.

Der er ikke nogen facitliste, der dikterer formen, men ofte vil et simpelt XML-skema give os lige præcis, hvad vi har brug for. XML har flere fordele, men vigtigst er det, at det er læsbart af både maskiner og mennesker, at det er godt til udtrykke en hierarkisk sammenhæng, og at det er entydigt (der er faktisk eksempler på, at XML ikke er entydigt, hvis namespaces og imports bruges tilpas uheldigt). Hertil kommer, at det er muligt, at der allerede findes et OIOXML-skema for den nødvendige entitet (det kunne f.eks. være, at man i det Digitale Danmark allerede havde standardiseret begrebet "postdistrikt"). Hvis det ikke er tilfældet, at der allerede findes et passende skema, og man selv vil definere et, er det vigtigt at holde for øje, at det skal være simpelt og dermed umiddelbart forståeligt og indlysende, hvad skemaet indeholder. XML er ikke eneste mulighed, og i sagens natur er der mere oplagte kandidater til returnering af f.eks. billeder eller MS-Office-dokumenter - hvis resourcen har en veldefineret MIME-type, er det formentlig oplagt at anvende denne. Da vores Adresse-resource bl.a. indeholder koordinaterne for en given adresse, er det oplagt, at vi skal kunne returnere forskellige standarder for koordinatsæt, så man udover at få punktet præsenteret med UTM-koordinater også kan få returneret punktet i Google Earth format eller som json predefineret til KMS's visstedet-service. Til det formål anvender vi muligheden for at postfix'e URL'en med en extension, så hvis vi eksempelvis angiver .kml som extension, får vi Google Earth repræsentationen. Denne fremgangsmåde er yderst velegnet til eksempelvis at returnere en given resource på flere sprog - her kan .en eller .dk extensions eksempelvis angive, at man ønsker den engelske hhv. den danske repræsentation af resourcen.

Det står således frit for at anvende andre formater end XML, men hvis man har brug for at udtrykke værdier og samhørighed (attributter og referencer), vil det være oplagt at anvende det, der allerede er konsensus om frem for at opfinde noget nyt og derved introducere fejlmuligheder i fortolkningen heraf - og her er

>

XML et godt valg. Og som sagt så er det vigtigt at holde det simpelt og umiddelbart overskueligt.

En sidste regel, vi skal have med er, at sammenhængen/hierarkiet i data afspejles i repræsentationen, eller med andre ord, at repræsentationerne for de enkelte ressourcer indeholder links til andre ressourcer. Helt konkret i vores eksempel kommer det f.eks. til udtryk i, at repræsentationen for ”kommune” indeholder links til ressourcerne for ”veje” og ”adresser”.

Vi har således valgt at modellere ressourcerne ”kommuner” og ”kommune” således:

```
<schema targetNamespace="http://itst.dk/schemas/danmarkservice"
elementFormDefault="qualified">
  <include schemaLocation="kommune.xsd"/>
  <element name="kommuner" type="dk:kommunertype"/>
  <complexType name="kommunertype">
    <sequence>
      <element name="kommune" type="dk:kommunetype" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="timestamp" type="time"/>
    <attribute name="ref" type="anyURI"/>
  </complexType>
</schema>
```

```
<schema targetNamespace="http://itst.dk/schemas/danmarkservice"
elementFormDefault="qualified">
  <include schemaLocation="veje.xsd"/>
  <include schemaLocation="adresser.xsd"/>
  <import namespace="http://rep.oio.dk/cpr.dk/xml/schemas/core/2006/05/23/"
schemaLocation="http://rep.oio.dk/cpr.dk/xml/schemas/core/2006/05/23/CPR_Municipi
palityName.xsd"/>
  <element name="kommune" type="dk:kommunetype"/>
  <simpleType name="KommuneNummerType">
    <restriction base="string">
      <pattern value="[0-9]{3}"/>
    </restriction>
  </simpleType>
  <complexType name="kommunetype">
    <sequence>
      <element name="nr" type="dk:KommuneNummerType" minOccurs="0"/>
      <element name="navn" type="cpr:MunicipalityNameType" minOccurs="0"/>
      <element name="veje" type="dk:vejetype" minOccurs="0"/>
      <element name="adresser" type="dk:adressertype" minOccurs="0"/>
    </sequence>
    <attribute name="ref" type="anyURI"/>
  </complexType>
</schema>
```

Hvis vi kigger på det konkrete XML, som en ressource returnerer i henhold til ovenstående skemaer, får vi eksempelvis:

```
<kommuner timestamp="20:00:17">
  <kommune ref="http://oioest.dk/danmark/kommuner/101">
    <nr>101</nr>
    <navn>København</navn>
  </kommune>
  <kommune ref="http://oioest.dk/danmark/kommuner/147">
    <nr>147</nr>
    <navn>Frederiksberg</navn>
  </kommune>
  <kommune ref="http://oioest.dk/danmark/kommuner/151">
```

---

>

---

```
<nr>151</nr>
<navn>Ballerup</navn>
</kommune>
```

på URL'en

<http://oiorest.dk/danmark/kommuner>

og:

```
<kommune ref="http://oiorest.dk/danmark/kommuner/101">
  <nr>101</nr>
  <navn>København</navn>
  <veje ref="http://oiorest.dk/danmark/kommuner/101/veje"/>
  <adresser ref="http://oiorest.dk/danmark/kommuner/101/adresser"/>
</kommune>
```

på URL'en

<http://oiorest.dk/danmark/kommuner/101>

Læg mærke til, at ressourcens repræsentation i begge tilfælde indeholder de direkte links til det refererede data.

For mere information vedr. design af repræsentationer, se kapitel 5 i referencebogen, afsnittene "Design Your Representations" og "Link the Resources to Each Other" samt kapitel 8, afsnittet om "Why Connectedness Matters".

### 4.3 Operationer

Indtil videre har vi fokuseret entydigt på læsning af data, dvs. GET af data. Måske skal servicen tillade både læsning og skrivning, og måske dækker skrivning over både opdatering af eksisterende data og oprettelse af ny data. Man kunne eksempelvis forestille sig, at en administrativ myndighed havde mulighed for at redigere eller oprette adresser. Her gælder præcis de samme overvejelser mht. datarepræsentation, og i de tilfælde hvor en given datatype både kan læses og skrives, er der igen grund til at opfinde forskellige repræsentationer for de to. Tværtimod kan der være en fordel i, at det data, der PUT'es, er præcis det samme som det, der sidenhen kan GET'es igen. I vores eksempel holder vi os til read only, altså kun GET-metoder.

### 4.4 Fejlscenarier

Vi mangler stadig at tage stilling til fejlscenarier. Der skal tages stilling til alle tænkelige fejlscenarier: Illegale dataværdier, ugyldig XML (hvis der anvendes XML), "resource not found", "resource already exists" (hvis man forsøger at

---

oprette noget, der allerede eksisterer), osv. Det er vigtigt, at den returnerede fejlstruktur er informativ - og vel at mærke uanset klient-typen.

Da vi ikke kan forudse brugen af vores webservice på længere sigt (læs: hvilke use cases og klienttyper der vil opstå over tid), vil det ofte være ønskværdigt, at fejlstrukturen indeholder information til både mennesker og maskiner - eller med andre ord både en statuskode og en sigende tekst (+ eventuelt et tilhørende stacktrace). En XML-struktur i stil med følgende vil oftest være passende:

```
<fault>
  <code>404</code>
  <parameters>
    <parameter key="region">23</parameter>
  </parameters>
  <description>Ukendt region</description>
</fault>
```

Det er her vigtigt at fremhæve, at ovenstående ikke skal ses som en mulighed for at erstatte de eksisterende HTTP-statuskoder - det skal udelukkende ses som en mulighed til at knytte ekstra information på koderne, f.eks. - som i ovenstående - hvilke parametre der forårsagede fejlen.

HTTP-koderne skal anvendes, og hver enkelt kode skal anvendes til det, den nu en gang er tiltænkt, hvilket igen betyder, at man i mange tilfælde vil kunne klare sig uden at tilføje ekstra information.

Kapitel 5, afsnittet om ”The HTTP Response” i referencebogen diskuterer i øvrigt brugen af HTTP-statuskoderne.

## 4.5 Caching

Offentligt data vil som hovedregel være konstant over længere tidsrum. Naturligvis er der undtagelser herfra, men eksempelvis store dele af vores adresse-data må formodes at være konstante over ret lange perioder. Omvendt kan man ikke forudsige, hvornår der sker eventuelle ændringer, eksempelvis at en given vej flyttes til et andet valgdistrikt, så en eventuel caching vil skulle afspejle sådanne afvejninger. Man kunne fristes til at hævde, at adresseforespørgsler er relativt sjældne, så caching er ikke nødvendigt, men da vi ikke kan forudsige, hvem der vil anvende vores webservice og til hvad, og da det underliggende datagrundlag er meget stort (der er 3.2 mill adresser i Danmark), vil det være fornuftigt at tillade caching i eksempelvis 24 timer. Hermed har vi taget stilling til, at data med fordel kan cache's, men hvis der skulle ske en ændring, vil den trods alt slå igenem efter senest 1 døgn. Til det formål anvender vi HTTP headeren `Cache-control: max-age=86400`, som angiver, at klienten *må* cache data i op til 24 timer, men ikke at den nødvendigvis *skal* gøre det.

Kapitel 8 i referencebogen beskriver caching-principperne i lidt flere detaljer.

---

>

---

## 4.6 Queries

Vores udstilling af data er ret grovkornet; når man forespørger på en collection, får man den komplette liste tilbage. Ofte vil der være brug for at kunne raffinere forespørgslen lidt mere. Til det formål beslutter vi, at man skal kunne give parametre med og eksempelvis søge efter en specifik kommune:

```
/kommuner?q=<del af kommunenavn>
```

Det er her vigtigt at holde for øje, at man ofte vil mappe søge-parametrene direkte til søgestrengene i den underliggende sql, og dermed har man åbnet for eventuel sql-injection. Man bør i disse tilfælde altid indskyde et validerings-lag, som kan bortfiltrere uønskede søgekriterier.

## 4.7 Servicespecifikationen

Vi er nu klar til at lave den endelige service-specifikation.

Da servicen er så "rig" og omfattende, og da den først og fremmest skal kunne forstås og fortolkes af andre udviklere (dvs. mennesker i modsætning til maskiner), er det vigtigt, at vi tilpasser servicespecifikationen hertil. En tekstuel beskrivelse på skemaform er meget velegnet - men sørg for at være specifik, så misforståelser undgås.

I vores tilfælde ser servicespecifikationen således ud:

(den komplette specifikation kan findes på

<http://oiorest.dk/danmark/Documentation/apidoc.aspx>)

Resource	URI	Method	Repræsentation	Caching	Status
Regioner	/regioner	GET	regioner.xsd	Max-age=86400	200,405,500
Region	/regioner/<regionsnr>	GET	region.xsd	Max-age=86400	200,404,405,500
Kommuner i region	/regioner/<regionsnr>/kommuner	GET	kommuner.xsd	Max-age=86400	200,405,500
Adresser i region	/regioner/<regionsnr>/adresser?<søgekriterie>	GET	adresser.xsd	Max-age=86400	200,405,500
...	...	...	...	...	...

---

## 5. Opsummering

>

---

1. Find ud af, hvilke data der skal eksponeres af webservicen.
2. Opdel datasættet i resourcer. En resource vil typisk være en konkret entitet eller en collection af entiteter.
3. For hver resource:
  - Tildel en URI
  - Fastlæg operationerne (GET, PUT, ...)
  - Bind resourcen sammen med andre relevante resourcer.
  - Fastlæg datarepræsentationen
  - Fastlæg fejlscenarier
  - Fastlæg caching-strategier
4. Lav servicespecifikationen

Som støtte til ovennævnte punkter og ikke mindst for at sikre at webservicen bliver fornuftigt anvendt, kan det være relevant at gøre sig følgende overvejelser:

- Udstil alt. Hvis der med minimal indsats kan udstilles mere data, så gør det. Nye anvendelsesmuligheder vil med sikkerhed opstå.
- Anvend simple og entydige repræsentationer. Hvis resourcen er en kendt type, så anvend den gængse repræsentation (f.eks. OIOXML-skemaer eller kendte MIME types).
- Anvend samme repræsentation for både læse- og skrive-data.
- Lav informative fejlbeskeder.
- Antag, at servicen på et eller andet tidspunkt bliver anvendt væsentligt anderledes end først antaget. Dette har eksempelvis betydning for caching strategier.
- Antag, at servicen bliver udsat for forsøg på sql-injection.



-----

<

-----

-----