MYNDIGHETEN FÖR DIGITAL FÖRVALTNING
Agency for Digital Government

REST API-profil

REST API-profil

För att den offentliga sektorn ska kunna bidra till utvecklingen av en förvaltningsgemensam digital infrastruktur som främjar innovation och bred samhällsnytta genom bättre förutsättningar för en effektiv datadelning och dataanvändning, är utvecklingen av API:er en central fråga. Här presenteras hur man gör detta utifrån en rekommenderad REST API-profil för offentliga aktörer. Profilen är en förvaltningsgemensam specifikation för REST API:er.

REST API-profilen är framtagen i samverkan mellan Myndigheten för digital förvaltning, Bolagsverket, Arbetsförmedlingen, Lantmäteriet och E-hälsomyndigheten som en rekommendation till offentliga organisationer. Profilen gör det möjligt att bygga REST API:er på ett effektivare sätt utifrån bästa praxis och en uppsättning identifierade SKALL/BÖR/KAN kravformuleringar. Denna gemensamma profil förenklar processen för både konsumenter och producenter av REST API:er, vilket påskyndar acceptans för och användning av API.

> Om man väljer att följa den rekommenderade REST API-profilen är det viktigt att följa de SKALL/BÖR/KAN krav som specificeras för att det utvecklade API:et ska kunna anses följa profilen.

Nedan beskrivs den senaste versionen av rekommendationerna för den nationella REST API-profilen. Rekommendationerna kommer att uppdateras över tid och andra profiler kan komma att tas fram för fler typer av maskinläsbara gränssnitt.

Innehåll

Om REST API-profilen

Profilen riktar sig främst till API-designers och API-utvecklare som arbetar med framtagande av nya REST API:er eller ändrar i befintliga för att t.ex. förbättra funktionalitet, användbarhet, läsbarhet eller säkerhet.

Profilen ska användas i frågeställningar som uppstår vid framtagandet av nya/utveckling av befintliga API:er och ger en bild över vilka förväntningar som kan finnas på API:er som ska verka nationellt mellan organisationer och myndigheter.

Vidare så kan nyttjande av denna profil leda till en generellt högre standard och kvalitet på API:er, både på dokumentationen och själva API:et. Detta underlättar för både producenter och konsumenter.

REST API-profilen uppdateras kontinuerligt och den senaste versionen finns tillgänglig online. Aktuell version av profilen samt tidigare versioner finns tillgängliga som nedladdningsbara filer nederst på denna sida.

Som bilaga till denna profil finns från och med version 1.1.0 ett Excel-dokument som avser att underlätta avstämning mot respektive krav och för att ge en översikt till hur ett API uppfyller denna REST API-profil. Bilagan finns tillgänglig som nedladdningsbar fil nederst på denna sida.

Nyckelord i profilen

I denna profil använder vi endast följande nyckelord gällande krav: **SKALL**, **BÖR** och **KAN** samt **SKALL INTE** och **BÖR INTE**. Nyckelorden ska tolkas enligt RFC 2119 enligt nedanstående tabell.

Nyckelord i profil	Nyckelord enligt RFC 2119	Betydelse
SKALL, **SKALL INTE**	MUST, MUST NOT SHALL, SHALL NOT REQUIRED	Detta är ett absolut krav för att uppfylla profilen.
BÖR, BÖR INTE	SHOULD, SHOULD NOT RECOMMENDED	Då detta nyckelord används är det rekommenderat att uppfylla detta krav men det är inte obligatoriskt.
KAN	MAY OPTIONAL	Detta krav kan användas.

Om man väljer att följa profilen, så är det viktigt förhålla sig till skrivelserna i tabellen ovan eftersom det finns indirekta beroenden mellan de olika sektionerna i profilen.

Nyckelorden används för att uttrycka ett krav, och efterföljs av en identifikation av kravet på formen *(XXX.YY)* där *XXX* är en prefix till avsnittet där kravet ingår och *YY* avser en stigande numrering av kraven inom avsnittet.

Medvetna val

- Profilen fokuserar på att använda HTTP REST API:er som grund för design.
    - REST API är ett sätt att bygga ett API baserat på den funktionalitet som finns i HTTP (HyperText Transfer Protocol). Vi har valt REST som grund för profilen på grund av en rad olika orsaker.
    - REST, står för Representational State Transfer och är i grund och botten en standardiserad mjukvaruarkitekturstil, som består av en speciell typ av API:er, som för branschen är känd och använd.
    - REST förlitar sig på ett tillståndslöst klient-serverprotokoll och i nästan alla fall kommer det att vara HTTP. Tidigare förlitade sig utvecklare främst på SOAP för att implementera API:et i webbtjänster, men på senare år har REST blivit utvecklares val på grund av dess enkelhet och skalbarhet. REST skapades för att behandla objekt på serversidan som resurser som i sin tur kan skapas, uppdateras och raderas. REST kan användas av praktiskt taget alla programmeringsspråk.
    - För mer information se: En introduktion till REST
- REST profilen är främst fokuserad till extern åtkomst, där graden av interoperabilitet i den offentliga sektorn behöver stärkas, men det går lika bra att applicera profilen för enbart intern åtkomst. Här är det lämpligt att låta verksamhetsbehovet styra.
- I de första versionerna av profilen, så kommer området språkbruk inte att behandlas, utan det kommer att ske i en senare version.
- De exempel som förekommer i profilen kan vara både existerande och fiktiva. Syftet är att förtydliga texten som omger exemplet, och översyn kommer att ske senare för att göra dem mer enhetliga och funktionella.

Versionshantering

Nedan specificeras de förändringar sedan profilen publicerades första gången.

Version	Datum	Kommentar
1.0.0	2022-07-11	Första publicering efter remissgenomgång
1.1.0	2023-06-29	Uppdaterat följande: – avsnittet Säkerhet införd. – förtydligat i formatering gällande kod, viktiga benämningar och annat. – tagit bort blanksteg som giltig indikation att tidselementet startar i avsnittet Datum- och tidsformat.

		– förtydligat att det är camelCase eller snacke_case som ska användas som namnsättningskonvention för söksträng i message och för body i request/response, och att det är viktigt att det är konsekvent. – för API request header Authorization ska inte 'Användarnamn + Lösenord' eller 'Basic Auth' användas. – HTTP response koder förhåller sig till RFC 7231, inte RFC 7807. – Cachning refererar till nyare RFC 9111, istället för den gamla RFC 7234. – 'Användbara responseheaders' är borttagna från Cache-Control avsnittet, och RFC får agera källa. – Infört en formatmallsbeskrivning som visar betydelsen av de olika formateringarna av text som återfinns i profilen. - Alla krav har erhållit ett identifikationsnummer. Detta identifikationsnummer återfinns även i den avstämningsfil (Excel) som följer som komplement till version 1.1.0 av REST API profilen.

Formatmall

Följande formateringar finns i profilen för att underlätta läsandet och förtydliga innehållet.

Format	Betydelse
Fet	Nyckelord från RFC 2119, rubriker, ingress
Kursiv	Viktiga ord, definitioner
Understrykning	Länkar
grå bakgrund	Attribut från kod och exempelvärden
`kodblock`	Komplett kodexempel
citatblock	Text som behöver lyftas/belysas i texten
infoblock	Beskrivning av format

Resurser

Datum	Namn	Kommentar
2023-06-29	REST API-profil v.1.1.0	Version 1.1.0 av REST AP-profilen
2023-06-29	Avstämning REST API profil v1.1.0	Avstämningsfil (Excel) som följer som komplement till version 1.1.0 av REST API-profilen.
2022-07-11	REST API-profil v.1.0.0	Första utgåvan av REST API-profilen

Begrepp

Ett API (Application Programming Interface) är en uppsättning definitioner och protokoll för att bygga och integrera applikationsprogramvara. REST (Representational State Transfer) i sin tur är en arkitekturstil som bygger på vanliga internet standarder och möjliggör interaktion med andra webbtjänster. Nedan beskrivs de mest central begreppen som används i samband med REST API:er.

Resurs

För att utforma ett användbart och rent API bör systemet delas upp i logiska grupperingar (ofta kallade modeller eller resurser). I de flesta fall är resurserna "substantiv" i ett system.

I grunddatadomänen Företag kan resurserna till exempelvis vara *organisationer*, *personer* och *firmateckningsalternativ*.

Genom att dela upp system i dessa logiska områden möjliggör det en ren separation av problem (t.ex. fungerar funktioner kopplade till organisationer endast på *organisationer*). Det säkerställer att varje del av data som returneras från ditt API är det minsta som det behöver vara för att uppfylla klientkravet (t.ex. när du ber om *organisationer* får du inte också tillbaka årsredovisningar).

Identifierare

Varje resurs som är tillgänglig i ditt system (t.ex. varje *organisation* eller varje *person*) måste kunna identifieras unikt i systemet. Detta är ett viktigt element i API:ets anammande av REST, möjligheten att individuellt adressera alla objekt i systemet och lagra dessa identifierare för senare användning.

Identifierare till en resurs kan vara något av följande:

Namn	**Exempel**
numerisk	/organisationer/5569010111
sträng	/landskoder/sverige
UUID (RFC 4122)	/medarbetare/0d047d80-eb69-4665-9395-6df5a5e569a4
datum (kort form)	/datum/2018-09-17

Så länge identifieraren är unik i programmet kan det vara vilken teckensträng eller nummer som helst.

Representation

Ett nyckelkoncept i REST API design är idén om representation av en resurs vid en viss tidpunkt.

När du ber systemet om företagsinformation får du en representation av organisationen, t.ex.

```
HTTP 1.1 GET /organisationer/5568108988

Accept: application/json

200 OK

Content-Type: application/json

organisationslista: [

{

  "namn" : "Volvo Cars AB",

  "organisationsnummer" : "5568108988",

  "bolagsform" : "Aktiebolag",

  "registreringsar" : "2010"

}]
```

Avsikten är att den här representationen kan ändras med tiden när systemet och data i ändras. Ett framtida anrop till samma endpoint kan ge en annan representation, t.ex. om organisationen bytt organisationsform.

Det är också möjligt att begära en helt annan representation av samma resurs om systemet stöder den. Det kan till exempel finnas ett fall där du behöver en PDF-version av den här företagsinformationen och detta kan understödjas från producentsidan genom att tillåta en begäran om en annan representation via `Accept` header:

```
HTTP 1.1 GET /organisationer/5568108988

Accept: application/pdf

200 OK

Content-Type: application/pdf

...<BINARY CODE>...
```

Mer information om representationer och hur du stöder dem finns i avsnittet API Message.

Operationer

För att utföra en operation på en resurs behöver man ange en HTTP-metod följt av sökvägen till resursen.

Operationer avgör hur API kan användas och det finns inget annat sätt att nyttja API än genom de operationer den stödjer.

En operation definieras av:

- en HTTP-metod och
- en resurssökväg

Exempel:

`GET /organisationer`

`GET /organisationer/5568108988`

`DELETE /organisationer/5568108988`

Dokumentation

Väldokumenterade API:er är centrala. Med hjälp av gemensamt strukturerad dokumentation skapas förutsättningar för interoperabla tjänster. Genom standardiserade strukturer, metoder och namngivning säkerställs en gemensam upplevelse för utvecklare som använder tjänster från många olika API-leverantörer.

Ett API:s dokumentation kan delas in i två kategorier:

- API dokumentation – API dokumentation kan ses som manualen för ett API. Den talar om för API-konsumenter hur man använder API:et. API dokumentation är avsedd för människor, vanligtvis utvecklare, att läsa och förstå. Att tillhandahålla dokumentation som är väldesignad, heltäckande och lätt att följa är avgörande när det gäller att säkerställa att utvecklare har en bra upplevelse av API:et. API dokumentationen brukar inkludera en snabbstartsguide, tutorials och interaktiv dokumentation så att utvecklare kan prova att anropa API:et.
- API specifikation – API specifikationen ger en bred förståelse för hur ett API beter sig och hur API:er länkar till andra API:er. Den förklarar hur API:et fungerar och vilka resultat man kan förvänta sig när man använder API:et. En API specifikation är avsedd för både människor och maskiner. Ett exempel på en standard för API specifikationen är OpenAPI Specification.

I vissa fall finns ingen tydlig gräns mellan vad som återfinns i dokumentationen och vad som återfinns i specifikationen. Därför ska uppdelningen nedan i API dokumentation och API specifikation inte ses som ett måste. Dock gäller kraven oberoende av vart de finns dokumenterade. För mer information om dokumentation för API:er se Understanding the Differences Between API Documentation, Specifications, and Definitions.

I regel **BÖR** (DOK.01) dokumentationen och specifikationen för ett API finnas allmänt tillgänglig online. Undantaget är om det finns juridiska, säkerhetsmässiga eller affärsmässiga skäl att inte tillgängliggöra dokumentationen. Vidare **BÖR** (DOK.02) ett API:s dokumentation och specifikation finnas sökbara via Sveriges dataportal.

API dokumentation

Dokumentationen för ett API **SKALL** (DOK.03) innehålla följande

- Om API
- Användarvillkor
- Datamodell för representation av resurser
- Krav på autentisering
- Livscykelhantering och versionshantering

- Kontaktuppgifter

Följande är generella riktlinjer på dokumentation av REST API:er

- Dokumentationen och i synnerhet specifikationen **SKALL** (DOK.04) ses som ett kontrakt mellan designer och utvecklare samt mellan producent och konsument av API:et.
- Dokumentationen **SKALL** (DOK.05) versionshanteras tillsammans med API:et.
- Dokumentationen **BÖR** (DOK.06) finnas på både svenska och engelska.
- Dokumentationen av ett API **BÖR** (DOK.07) innehålla övergripande information om API:et.
- Ett API:s servicenivå **SKALL** (DOK.08) finnas tydligt beskriven i dokumentationen.
- Kända problem och begränsningar **SKALL** (DOK.09) finnas tydligt beskrivna i dokumentationen.
- Om det är känt **SKALL** (DOK.10) tidpunkt för när API:et tas ur bruk anges i dokumentationen.

Följande gäller för API:ets specifika resurser, operationer och förväntade resultat.

- Avsikten och beteendet hos API:et **SKALL** (DOK.11) beskrivas så utförligt och tydligt som möjligt.
- Ett API:s resurser och de möjliga operationer som kan utföras på resursen **SKALL** (DOK.12) beskrivas så utförligt och tydligt som möjligt
- Förväntade returkoder och felkoder **SKALL** (DOK.13) vara fullständigt dokumenterade.
- Krav på autentisering **SKALL** (DOK.14) anges i dokumentationen
- I dokumentationen av API:et **SKALL** (DOK.15) exempel på API:ets fråga (en:request) och svar (en:reply) finnas i sin helhet.

Notera att detta nödvändigtvis inte finns i API dokumentationen utan kan återfinnas i API specifikationen, se nästa kapitel.

API specifikation

REST API:er **SKALL** (DOK.16) ha en API specifikation som bör kunna generera en datamodell för representation av API:ets resurser. API specifikation **BÖR** (DOK.17) dokumenteras med den senaste versionen av OpenAPI Specification. I sällsynta fall kan man behöva göra ett undantag att inte använda OpenAPI för att specificera ett API. Exempelvis i de fall det finns en erkänd och spridd branschstandard för att specificera API:er.

API specifikationen **BÖR** (DOK.18) beskrivas med antingen JSON eller YAML.

En API specifikation **SKALL** innehålla

- Ett API:s resurser och de möjliga operationer som kan utföras på resursen **SKALL** (DOK.19) beskrivas så utförligt och tydligt som möjligt
- Förväntade returkoder och felkoder **SKALL** (DOK.20) vara fullständigt dokumenterade.
- Krav på autentisering **SKALL** (DOK.21) anges i specifikationen

Varje ny major-version av ett API, enligt versionshanteringsavsnittet, **SKALL** (DOK.22) följas av en ny API specifikation. Till exempel om ett API tillgängliggör och hanterar tre major version så måste det finnas tre API specifikationer, en för varje version.

API specifikationen **SKALL** (DOK.23) återfinnas under API-roten, det vill säga

> {protokoll}://{domännamn}/{API}/{version}/

Om API specifikationen specificeras med OpenAPI **SKALL** (DOK.24) roten till specifikationen namnges med *openapi.yaml* eller *openapi.json*, se exemplet nedan

> https://gw.api.bolagsverket.se/foretagsinformation/v2/openapi.yaml
> https://gw.api.bolagsverket.se/foretagsinformation/v2/openapi.json

Datum- och tidsformat

Ett standardiserat sätt att hantera datum och tid underlättar informationsutbyte och förståelse av informationen mellan parter.

Datum och tid **SKALL** (DOT.01) hanteras enligt följande

- Använd alltid RFC 3339 för datum och tid
- Acceptera alla tidszoner i API:er
- Returnera datum och tid i UTC
- Använd inte tidsdelen om du inte behöver den

Datum och tid

Datum och tid **SKALL** (DOT.02) anges enligt RFC 3339 som bygger på ISO-8601.

I RFC 3339 sorterar man datum och tid genom att ange den största enheten först och den minsta enheten sist: år, månad, dag, timme, minut, sekund och delar av en sekund.

Formatet att representera datumobjekt är

> YYYY-MM-DD

Formatet att representera tidsobjekt är

> hh:mm:ss.ff

Nedan beskrivs varje del för sig.

Element	Beskrivning	Exempel
YYYY	Årtal med fyra siffror	1907 eller 2013
MM	Månad med två siffror. I de fall månaden endast är en siffra inleds detta med 0.	04 eller 11
DD	Dag med två siffror. I de fall dagen endast är en siffra inleds detta med 0.	01 eller 23
T	Tecknet 'T' indikerar att tidselementet startar här.	T
hh	Timme med två siffror med 24-timmar klocka. I de fall timmen endast är en siffra inleds detta med 0.	04 eller 18
mm	Minut med två siffror. I de fall minuten endast är en siffra inleds detta med 0.	00 eller 45

ss	Sekund med två siffror. I de fall sekunden endast är en siffra inleds detta med 0.	`09` eller `17`
ff	Delar av en sekund, två siffor (valfri).	`12` eller `08`

Då dessa element kombineras representeras datum och tid enligt följande exempel

`1907-04-01T04:00:09.12`

Tidszoner

När man använder RFC 3339 format **BÖR** (DOT.03) tidszonen anges. Detta för att ge tydlighet vilken tidpunkt som avses. T.ex. så kommer tidszoner vara bra att ha vid användande av internationellt distribuerade system, men det förekommer såklart lösningar som inte kräver tidszon.

Tidszoner kan anges på ett antal olika sätt. Tidzonen **BÖR** (DOT.04) representeras med UTC formatet, där tid anges som offset från UTC (Coordinated Universal Time). Detta sker med följande sträng som adderas på datum och tidsangivelsen.

> ±hh:mm

Element	Beskrivning	Exempel
+ eller -	+ representerar ett positivt offset från UTC - representerar ett negativt offset från UTC	`+`
hh	Anger offset i antal timmar. I de fall timmen endast är en siffra inleds detta med 0. Giltiga värden är från -12:00 till +14:00.	`04` eller `10`
mm	Anger offset i antal minuter. I de fall minuten endast är en siffra inleds detta med 0. Giltiga värden är från 00 till 59. Ofta anges 00, 15, 30 eller 45.	`00` eller `45`

Lokal tid i Sverige när klockan var 9 sekunder efter 04:00 den 1:a april 1997

`1997-04-01T04:00:09+01:00`

Sverige har idag, i och med användande av sommartid, följande tidszoner:

- +01:00 (normaltid)
- +02:00 (sommartid)

Även formatet Z för Zulu Time för att ange UTC+0 förekommer. Z har ursprung från militär- och navigationsområdet, som en generell term för Universal Coordinated Time.

Tillämpas cachning enligt RFC 9111 kan GMT (Greenwich Mean Time) tidzonen användas i dessa svar.

Resurser

Resurser är centrala för den webbaserade arkitekturen. Det är resursen man utgår ifrån och det är den som man sedan manipulerar med hjälp av operationer (t.ex. GET och POST).

All information som kan namnges kan vara en resurs och resurser är ett centralt begrepp för REST API:er.

Samlingar

En samling (eng. *collection*) är en speciell typ av resurs och kan ses som en lista med resurser av samma typ. Resurser beskrivs nästan alltid i en struktur med både samlingar och resurser, men resurser kan existera utanför en samling också.

Exempel:

`/organisationer`

En GET request på denna samling ger en lista på alla organisationer.

Ofta delas en samling in i ytterligare samlingar, så kallade nästlade samlingar.

Exempel på undersamling *hemvist* under resursen *2021005489* i samlingen *organisationer*:

`/organisationer/2021005489/hemvist`

Identifierare

Följande regler används gällande identifierare för en resurs

- Identifieraren **BÖR** (RES.01) vara beständig, d.v.s. att vara globalt unik över tid. Med global unik identifierare avses en identifierare, som består över tid och refererar till samma resurs.
- Primärnycklar eller personligt identifierbar information (personnummer, etc) **BÖR INTE** (RES.02) exponeras. Om detta är svårt att uppnå är det troligt att API:et behöver abstraheras ytterligare från den underliggande datakällan. Läs mer om Amundsens Maturity Model
- När numeriska ID:n används **BÖR** de **INTE** (RES.03) vara sekventiella. Om informationen är av känslig karaktär ska den skyddas med behörighetskontroll.
- En identifierare för en nästlad resurs **SKALL** (RES.04) vara unik inom sin förälder resurs om den är beroende av sin förälder.

`organisationer/2021005489/hemvist/2345`

Identifieraren för hemvist 2345 är unik inom organisationer 2021005489 men 2345 är nödvändigtvis inte globalt unikt. Dock är den globalt unik tillsammans med organisationer 2021005489, dvs '2021005489/hemvist/2345'.

När det kommer till att identifiera resurser så **BÖR** (RES.05) designen av detta ta höjd för en del olika aspekter:

- Säkerhet – beakta den identifikation på resurs som förmedlas via API gällande om den kan anses vara känslig eller onödig att exponera. T.ex. består en identifierare av en räknare så är det enkelt att lista ut nästa resurs. Avstå också från att använda primärnycklar i databasen utåt.
- Användandet av logiska nycklar – den logiska nyckeln används i exponering utåt medan relationer döljs i backend. Passar bra när resurserna tillhör en domän, såsom SEK tillhör domänen valutor.
- Överväg användande av UUID (eller motsvarande) – likt de två ovanstående punkterna så används databasgenererade id:n för att hantera data i backend, medan den logiska nyckeln döljs genom att skapa en UUID (t.ex. genom MD5 hashning av logisk nyckel). UUID kan såklart också genereras trots att det inte finns en logisk nyckel att använda.

Genom att tänka på ovanstående aspekter så kan både säkerhet och unikhet hanteras för identifierare av resurser. En identifierare som använder UUID eller liknande kan anses vara globalt unik.

Namnsättningskonvention för resurser

All information som kan namnges kan vara en resurs. En informationsmängd är en entitet och ska därför namnges med hjälp av substantiv. Verb ska inte användas för att namnge resurser. Låt istället HTTP verben (t.ex. POST, GET, PUT och DELETE) utgöra de metoder eller operationer som används för att manipulera resurserna.

Följande regler för resurser **SKALL** (RES.06) följas

- Resurser namnges som substantiv.
- Resurser namnges i pluralform, även om den nu kända träffbilden bara ger ett resultat
- Resurser följer den namnsättningskonvention som beskrivs för URL:er, det vill säga att resurser anges med gemener, använder endast alfanumeriska tecken och bindestreck för att separera eventuella ord.

Bra exempel
`/anstallda`
`/tjanster`

Följande ska inte användas:
`/get-anstallda` - namnge inte resurser med verb

/bokning - namnge inte resurser i singularis

/TJANSTER - namnge inte resurser i versaler

URL format och namngivning

URL:ers syntax och semantik beskrivs i specifikationen RFC 3986. Specifikationen förenklar utveckling och användning av REST API:er genom att standardisera strukturen och betydelsen av URL:er. Beskrivningen i detta kapitel följer denna standard.

Läs mer om specifikationen RFC 3986.

URL komponenter

Strukturen på ett API:s URL ska vara meningsfullt för en konsument. För att öka förståelsen och användbarheten underlättar det om en URL följer en förutbestämd struktur.

En URL för ett API **BÖR** (UFN.01) följa namnstandarden nedan:

> {protokoll}://{domännamn}/{api}/{version}/{resurs}/{identifierare}?{parametrar}

Exempel:

```
GET

https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/2021005489?fields=postadress,firmateckning
```

I vissa fall kan domännamnet vara det samma som API:et och då behövs inte API-delen i URL:en anges. I andra fall, i synnerhet för länkade data, brukar man inte ange versions-delen. I dessa fall krävs att versionshanteringen sköts på annat sätt.

Nedan visas en nedbrytning av hur man konstruera en URL enligt ovan.

URL element	API element	Beskrivning	Exempel
Scheme	protokoll	Alla API:er **SKALL** (UFN.02) exponeras via HTTPS på port 443	https
Authority	domännamn	Domännamnet där API:ets endpoint exponeras	gw.api.bolagsverket.se
Path	api	Sökväg inom domänen till API:et. Ofta är sökvägen logiskt indelad i ett affärsområde, verksamhetsområde eller tjänsteområde.	/foretagsinformation

		(Tänk på att namnsättning ej bör återspegla en föränderlig organisationsstruktur.) Ibland består domännamnet av själva API:et då faller denna del bort.	
Path	version	Alla API:er **BÖR** (VER.05) exponera versionen enligt versionshanteringsbeskrivningen i denna standard, se vidare under Versionshantering.	/v2
Path	resurs	Resurser SKALL anges som ett substantiv i pluralform, se vidare under Resurser.	/organisationer
Path	identifierare	En identifierare till en resurs.	/2021005489
Query	parameter	Parametrar i sökfrågor **SKALL INTE** (UFN.03) användas för att transportera payload eller faktiska data. Ofta används sökfrågefältet för att filtrera, sortera, avgränsa vilka fält som ska visas eller paginera resultatet. Om API:et stödjer begränsning av returnerade fält, komplicerad filtrering, sortering eller paginering **BÖR** (UFN.04) API:et stödja: • fields - specificera eller begränsa vilka fält som ska returneras • filter - villkor som begränsar/filtrerar resultatets träffbild vid mer komplicerad filtrering (t.ex. större än). Vid enkel filtrering använd	?fields=postadress,firmateckning Denna sökfråga kommer att returnera fälten postadress och firmateckning. ?filter=bildat%20gt%202020-01-01 (alternativt filtrering utan filter: ?bildat %3E2020-01-01) Sökträff kommer bara att innehålla objekt som har bildats efter datum 2020-01-01 ?sort=hemvist%20desc Träffbilden kommer att sorteras i fallande ordning på fältet hemvist ?page=2 Returnerar sida 2 i träffbilden

		exempelvis ”=” direkt • sort - hur resultatet ska sorteras • page - specificerar vilken sida som ska returneras i en träffbild Se vidare under Filtrering, paginering och sökparametrar.	

URL maxlängd

En URL **BÖR INTE** (UFN.05) vara längre än 2048 tecken. Trots att RFC 7231 inte sätter någon gräns tillåter många webbläsare max 2048 tecken därför är det säkrast att längden är max 2048 tecken lång.

URL namnsättningskonvention

En URL skall följa nedan beskrivna namnkonvention

- Bokstäver i URL:n **SKALL** (UFN.06) bestå av enbart gemener
- URL:n **SKALL** (UFN.07) använda tecken som är URL-säkra (tecknen A-Z, a-z, 0-9, "-", ".", "_" samt "~", se vidare i RFC 3986)
- Endast bindestreck '-' **SKALL** (UFN.08) användas för att separera ord för att öka läsbarheten samt förenkla för sökmotorer att indexera varje ord för sig. Blanksteg ' ' och understreck '_' **SKALL INTE** (UFN.09) användas i URL:er med undantag av parameter-delen.
- Understreck '_' **SKALL** (UFN.10) endast användas för att separera ord i parameternamn.
- Understreck '_' **SKALL INTE** (UFN.11) vara del av bas URL:en.

Se vidare under Filtrering, paginering och sökparametrar rörande namnsättning av sökfrågors parameternamn.

Goda exempel

Lista organisationer

```
GET
https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer
```

Filtreringsfråga

```
GET
https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer?filter=bildat%20gt%2020
20-01-01
```

Hämta information om en enskild organisation

```
GET https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/2021005489
```

Hämta endast enstaka fält för en organisation

```
GET

https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/2021005489?fields=posta
dress,firmateckning
```

Uppdatera en enstaka organisation

```
PUT https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/2021005489
```

Undvik följande exempel

Singularis

```
POST https://gw.api.bolagsverket.se/foretagsinformation/v2/person
```

Verb i URL

```
POST https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/2021005489/skapa
```

Filtrering i sökväg istället för i query strängen

```
POST https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/2021005489/desc
```

Mognad

När man designar ett nytt API så är en av de viktigaste delarna att ta hänsyn till konsumenternas upplevelse av API:et. Utvecklaren kommer att ha en bättre upplevelse av API:et om det redan följer vedertagna och grundläggande koncept.

Ett API som överensstämmer med de principerna och begränsningar som definierar arkitekturstilen REST sägs vara ett RESTful API. För att underlätta för utvecklare och säkerställa att ett API är RESTful finns en modell, Richardson Maturity Model. Denna modell består av fyra nivåer beroende på användning av URL:er, HTTP metoder och principen HATEOAS (Hypermedia As The Engine Of Application State).

- Nivå 0 – På denna nivå använder man endast en resurs och en operation. API:et använder den enda resursen som en tunnel där meddelanden skickas fram och tillbaka mellan klient och server. Resursen och operationen, typiskt POST, saknar semantisk betydelse.
- Nivå 1 – Till skillnad från nivå 0 använder API:er på nivå 1 flera resurser men använder fortfarande endast en HTTP-operation som en tunnel. På denna nivå har resurserna en semantisk betydelse men operationen saknar det.
- Nivå 2 – På nivå 2 använder man flera resurser och använder olika operationer. HTTP operationerna har nu en så korrekt semantisk betydelse som möjligt, t.ex. betyder GET för att hämta, DELETE för att ta bort osv.
- Nivå 3 – Ett API på nivå 3 bygger på nivå 2 men servern ger i svaret på en förfrågan, information om nästa möjliga åtgärder förutom den begärda resursen. Nivå 3 följer alltså principen HATEOAS.

Notera att ett API måste uppfylla nivå 3 för att kallas RESTful.

Alla API:er **SKALL** (MOG.01) designas för att uppnå nivå 2 enligt Richardson Maturity Model. Alla API:er **BÖR** (MOG.02) samtidigt designas för att uppnå nivå 3 enligt Richardson Maturity Model.

I vissa undantagsfall kan man behöva göra avsteg från den korrekta semantiska betydelsen vid val av HTTP-operation. Exempelvis då man inte vill exponera potentiellt känsliga personuppgifter (personnummer) i header-delen i en GET operation eller då söksträngen i URL:n är väldigt lång (över 2000-tecken). Även om RFC 7231 inte förbjuder att man skickar med information i body-delen i en GET saknar body-delen semantisk betydelse och ignoreras av många ramverk och verktyg och bör därför inte användas. I dessa fall brukar man i stället använda POST operationen. Om man använder POST operationerna där man vanligen använder GET operationer **SKALL** (MOG.03) det dokumenteras tydligt varför. I denna profil betraktas dock detta som att man fortfarande uppnår nivå 2 då man inte använder HTTP-operationen som en tunnel.

Säkerhet

Att använda "rätt" nivå på säkerhet i dina APIer ger dig möjlighet att användandet av dessa kan ske utan att säkerheten äventyras och att informationen som hanteras skyddas på ett korrekt sätt. Ytterligare säkerhetsöverväganden kan gälla beroende på API-design och krav från informationens informationsklassning m.m.

Transportsäkerhet

- All transport **SKALL** (SÄK.01) ske över HTTPS med minst TLS 1.2.
- Alla certifikat **SKALL** (SÄK.02) vara från SHA-2 (Secure Hash Algorithm 2) kryptografiska hashfunktioner med minsta nyckellängd på 2048.
- Alla offentligt tillgängliga endpoints **SKALL** (SÄK.03) använda ett digitalt certifikat som har undertecknats av en godkänd certifikatutfärdare.
- Endpoints avsedd för internt bruk **KAN** (SÄK.04) använda självsignerade digitala certifikat.
- Omdirigera inte HTTP-trafik till HTTPS – avvisa dessa förfrågningar.
- Oanvända HTTP-metoder **BÖR** (SÄK.05) inaktiveras och returnera HTTP-status 405.
- Alla requests **SKALL** (SÄK.06) valideras.

Beroende på säkerhetsklassificeringen av informationen i API:et **KAN** (SÄK.07) du behöva upprätta följande kontroller utöver standardkontrollerna:

- Ömsesidig autentisering mellan konsumenten och API-gateway.
- Ömsesidig autentisering mellan API Gateway och back-end API.
- PKI Mutual TLS OAuth-klientautentiseringsmetod.
- Vitlistning av IP-nummer för API-konsumenter som använder antingen API Gateway Policy eller brandväggskonfigurationer.
- Vitlistning av IP-nummer för API-producenter som använder antingen API Gateway Policy eller brandväggskonfigurationer.
- Kryptering av payload under överföring.
- Signering av payload för integritet och verifiering.

Autentisering och auktorisation

Om man i informationsklassningen har restriktioner till vilka som får konsumera informationen i API:et så **SKALL** (SÄK.08) autentisering och auktorisation användas.

- Basic- eller Digest-autentisering **SKALL INTE** (SÄK.09) användas.
- Authorization: Bearer header **SKALL** (SÄK.10) användas för autentisering/auktorisation.

- En uppdateringstoken **SKALL** (SÄK.11) tillhandahållas för att förlänga giltighetstiden för befintliga token utan att behöva tillhandahålla referenserna igen.
- Ställ alltid in en rimlig giltighetstid för tokens. En OIDC-åtkomsttokens livslängd **BÖR INTE** (SÄK.12) överstiga 5 minuter.
- Klientapplikationsidentitet **BÖR** (SÄK.13) etableras via en konsekvent mekanism. Detta kan vara via en API-nyckel eller via en mer robust mekanism som en OAuth-serveridentitet.
- Rotationspolicy för API-nycklar **BÖR** (SÄK.14) implementeras där så är tillämpligt.
- API-nycklar **SKALL INTE** (SÄK.15) inkluderas i URL eller querysträngen.
- API-nycklar **SKALL** (SÄK.16) inkluderas i HTTP-headern eftersom querysträngar kan sparas av klienten eller servern i okrypterat format av webbläsaren eller serverapplikationen.
- CORS-headers (Cross-origin resource sharing) **BÖR** (SÄK.17) endast användas när det är nödvändigt eftersom det kringgår de övergripande säkerhetsmekanismerna som är inbyggda i webbläsare genom att selektivt lätta på restriktioner för cross-origins, RFC 6454.
    - Använd aldrig URL-adresser med jokertecken (*) i response headers (dvs. *Access-Control-Allow-Origin*) såvida inte REST-resursen verkligen är offentlig och kräver liten eller ingen auktorisation för användning.
    - En request från domän A anses vara Cross-Origin när den försöker göra en request till ett API som finns på domän B.
        - Av säkerhetsskäl begränsar webbläsares HTTP-förfrågningar med flera ursprung.
        - Cross-Origin Resource Sharing-standarden fungerar genom att lägga till nya HTTP-headers (d.v.s. Access-Control-Allow-Origin) som tillåter servrar att beskriva vilka domäner som har tillåtelse att komma åt API:et

OAuth

OAuth version 2.0 eller senare **BÖR** (SÄK.18) användas för auktorisation. OAuth är ett auktorisationsprotokoll som säkert delegerar behörighet till en annan resurs. Det tillåter användare att godkänna applikationer att agera för deras räkning utan att dela med sig av sitt lösenord.

Abstraktion

Det finns flera abstraktionsnivåer, benämningar av abstraktionsnivån baseras till stor del på antalet konsumenter och kostnaden för att ändra systemet på grund av en API-ändring.

Som en allmän princip **BÖR** (SÄK.19) det finnas en abstraktionsnivå av källsystemets data- och affärslogik i din API-design för att frikoppla konsumenterna och API-tjänsteleverantörerna. Det rekommenderas att tillämpa grundläggande abstraktion på ett SaaS API (som ett Azure API). Till exempel; om det finns ändringar i SaaS-autentiseringsschemat kan det hanteras inom API:t.

En högre abstraktionsnivå skulle krävas för API:er som har flera konsumenter och kostnaden för förändring för API-konsumenterna är större än kostnaden för att byta backend-system/tjänsteleverantörer.

De olika abstraktionsnivåerna är följande:

Abstraktionsnivå	**Beskrivning**
Grundläggande	Representerar den minsta abstraktion som krävs för alla API:er (inklusive punkt-till-punkt) • Använd JSON som din standardrepresentation. • Ange alltid ett versionsnummer i URL:en för att underlätta ändringen. • Använd alltid ett API Key ID • Lite eller ingen abstraktion av datamodellen och exponera data vid behov
Mellan	API:er som syftar till återanvändning **BÖR** (SÄK.20) överväga mellanliggande abstraktionsnivån • Payload representerar affärsresurser som en abstraktion av domänmodellen, oberoende av implementationen, med fokus på klientanvändbarhet och självbetjäning. • Interna databastabellstrukturer **SKALL INTE** (SÄK.21) exponeras direkt i ditt API. • Hantera fel i källsystem på ett konsekvent och informativt sätt. • Exponering av interna systemidentifierare **BÖR INTE** (SÄK.22) (exempelvis ett databas-ID) med konsumenter. Använd en kandidatnyckel eller en sekundär identifierare.
Avancerad	Den högsta abstraktionsnivån omfattar alla andra nivåer. • Använd API Gateway för att ta hand om övergripande problem som säkerhet, trafikhantering och analys/övervakning. • Använd hypermedia med länkad data för att främja "upptäckbarheten" av dina API-resurser och relationer. • HATEOAS **BÖR** (SÄK.23) användas för att abstrahera tillåtna åtgärder.

Begränsa användandet av API:et

Tillämpa policyer för att begränsa antal requests eller systematiskt missbruk för att förhindra felaktig användning av ditt API. Se till att lämpliga varningar implementeras och svara med informativa felmeddelanden när tröskelvärdena närmar sig eller har överskridits.

Följande headers returneras vid användning av "rate limits":

Header	Beskrivning
X-RateLimit-Limit	Max antal meddelanden för en specifik tidsperiod (exempelvis, 100 meddelanden)
X-RateLimit-Remaining	Antal återstående meddelanden återstår innan max begränsningen är nådd
X-RateLimit-Reset	Återstående tid innan antal nyttjade meddelanden återställs

Felhantering

Felmeddelanden **SKALL INTE** (SÄK.24) avslöja information som kan användas för att attackera ditt system. För att undvika detta upprätta följande kontroller när du tillhandahåller felmeddelanden:

- Ett API **SKALL** (SÄK.25) maskera alla systemrelaterade fel bakom vanliga HTTP-statuskoder och felmeddelanden, t.ex. exponera inte information på systemnivå i ditt felmeddelande
- Ett API **SKALL INTE** (SÄK.26) skicka tekniska detaljer (t.ex. anropsstackar eller andra interna tips) till klienten

Loggning

En viktig aspekt av säkerheten är att bli meddelad när något fel inträffar och att kunna undersöka det. Man **BÖR** (SÄK.27) definiera loggningsnivåer som kan användas för att trigga lämpliga varningar och larm.

- Skriv audit-loggar före och efter säkerhetsrelaterade händelser som kan utlösa varningarna
- Sanering av loggdata för att förhindra logginjektionsattacker

Validering av ingående parametrar

Validering av ingående parametrar **BÖR** (SÄK.28) utföras för att säkerställa att endast korrekt formaterad data tas emot av ditt system, detta hjälper till att förhindra skadliga attacker.

- Validering av indata bör ske så tidigt som möjligt, helst så snart informationen har tagits emot från den externa parten
- Definiera exempelvis en lämplig storleksgräns för request och avvisa requests som överskrider gränsen
- Validera exempelvis: längd/intervall/format och typ
- Överväg att logga valideringsfel på indata. Anta att någon som utför hundratals misslyckade indatavalideringar per sekund har en skadlig avsikt
- Begränsa stränginmatningar med reguljärt uttryck där så är lämpligt

Validering av innehållstyp

Man **BÖR** (SÄK.29) respektera angiven `content-type` i header. Avvisa förfrågningar som innehåller oväntade eller saknade content-type headers med HTTP-status "*415 Unsupported Media Type*"

Gateway säkerhetsfunktioner

Det är att föredra att använda säkerhetspolicyfunktionerna som finns i API Gateway-plattformar än att implementera policyerna i ditt back-end API.

Säkerhet	**Kommentar**	**Implementationskrav**
HTTP verb	HTTP verb bör användas för att beskriva sökvägen för ett API	**BÖR** (SÄK.30)
Validering av parametrar	Olika typer av filter bör användas bekräfta giltigheten på input till ett API. Exempelvis meddelandestorlek, kontroll av scheman, kontroll av headers eller kontroll av sökparametrar	**BÖR** (SÄK.31)
SSL	SSL protokoll (TLS 1.2)	**BÖR** (SÄK.32)
Digitala certifikat	Olika filter och funktioner kan användas för att säkerställa ett certifikat.	**KAN** (SÄK.33), beroende på affärskrav
JWT	Ett meddelande kan signeras med hjälp av JWT:er	**KAN**, (SÄK.34) beroende på affärskrav

API Keys	API Keys bör användas för identifiera en klient	**BÖR** (SÄK.35)
OAUTH	OAUTH bör användas för identifiera en klient	**BÖR** (SÄK.36), beroende på affärskrav
CORS	CORS bör tillåta resursdelning över domängränser när det är nödvändigt	**BÖR** (SÄK.37)

API Message

Datamodell för representation

Datamodellen för en representation **BÖR** (AME.01) beskrivas med JSON enligt senaste versionen, RFC 8259

Det **BÖR** (AME.02) förutsättas att alla request headers som standard använder `Accept` med värde `application/json`. Även andra datamodeller **KAN** (AME.03) användas, exempelvis XML.

Namnsättningskonvention för message

För fältnamn i request och response body **BÖR** (AME.04) *camelCase* eller *snake_case* notation användas. camelCase innebär att första ordet skrivs med gemener och samtliga andra efterföljande ord börjar med en versal. Inom ett API **SKALL** (AME.05) namnsättningen vara konsekvent, dvs blanda inte *camelCase* och *snake_case*.

Exempel

```
// dettaArCamelCase

{
 "individId" : "1234"
}
```

Fältnamn som representerar listor **SKALL** (AME.06) namnges i pluralform.

Fältnamn **BÖR** (AME.07) använda tecken som är alfanumeriska.

API Request

Ett request är ett meddelande som skickas av klienten för att initiera en åtgärd på servern.

Request headers

Ett request **BÖR** (ARQ.01) skickas i UTF-8.

Alla API:er **SKALL** (ARQ.02) supportera följande request headers

Header	Värde
Authorization	Om APIet kräver autentisering ska en av följande användas: • API nyckel • Bearer (token)
Content-Type	Använd ett av följande tre format, men om ytterligare format stödjs bör man ange dessa också: • application/json • application/xml • multipart/form-data (för filer)

Alla API:er **BÖR** (ARQ.03) supportera följande request headers

Header	Värde
Accept	Ett av följande **BÖR** (ARQ.04) användas: • application/json • application/xml
Accept-Charset	utf-8
Date	Datum och tid då meddelandet skickades, i format definierat i RFC 3339
Cache-Control	Används för att specificera direktiv som måste följas av alla cachemekanismer, t.ex. ingen cache.

Header	Värde
ETag	Används för att identifiera den specifika versionen av en resurs som uppdateras för att förhindra flera användaruppdateringar. Detta borde matcha det som för närvarande är lagrat på servern.
Connection	För den aktuella anslutningen ska keep-alive användas.
Cookie	En HTTP cookie som tidigare skickats av servern.

Payload data **SKALL INTE** (ARQ.05) användas i HTTP-headers.

Supporterade HTTP Request metoder

REST API-operationer baseras på standarderna för HTTP-request som definieras i RFC 7231 och RFC 5789. Ytterligare information finns att läsa i artikeln En introduktion till REST.

HTTP metod	Beskrivning
GET	Hämta en resurs
POST	Skapa en ny resurs, eller att utföra en operation på en resurs som ändrar systemets tillstånd, t.ex. skicka ett meddelande. Används även då ingen HTTP-metod naturligt passar.
PUT	Ersätter en resurs med en annan som anges i begäran
PATCH	Delvis uppdatera en resurs
DELETE	Helt ta bort en resurs

En begäran om att hämta resurser kan göras för en resurs eller för en samling.

Exempel:
`https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/{id}`

För att hämta en samling organisationer skickas en begäran till namnet på resursen `/organisationer`.

För att hämta en enda organisation skickas en begäran till identifikationen på resursen `/organisationer/{id}`.

Enkel resurs

HTTP metod	Resurs sökväg	Operation
GET	/resurs/{id}	Hämta förekomsten som motsvarar resurs-ID
PUT	/resurs/{id}	För att uppdatera en resursinstans genom att ersätta den - "Ta den här nya saken och lägg den där"
DELETE	/resurs/{id}	För att ta bort resursinstansen baserat på resursen, t.ex. id
PATCH	/resurs/{id}	Utför ändringar som att lägga till, uppdatera och radera till de angivna attributen. Används ofta för att utföra partiella uppdateringar av en resurs

Samling av resurser

Följande åtgärder gäller för en samling resurser:

HTTP metod	Resurs sökväg	Operation	Exempel
GET	/resurs	Få en samling av resursen	GET /organisationer eller GET /organisationer?filter=organisationsform eg AB
POST	/resurs	Skapa en ny instans av den här resursen	

Notering:
Skapa eller uppdatera flera resursinstanser i samma begäran är för närvarande inte standardiserat. Det finns faktorer som kvitto och hur man hanterar partiell framgång i en uppsättning satser som måste övervägas från fall till fall.

Idempotens

En idempotent HTTP-metod kan anropas många gånger med samma resultat. I vissa fall kommer sekundära anrop att resultera i en annan svarskod, men resursens tillstånd ändras inte. Som ett exempel, när du anropar DELETE flera gånger på samma resurs, kommer den första begäran att ta bort resursen och svaret blir 200 (OK) eller 204 (Inget innehåll). Ytterligare förfrågningar returnerar 404 (hittades inte). Det är

uppenbart att svaret skiljer sig från den första begäran, men det finns ingen ändring av tillstånd för någon resurs på serversidan eftersom originalresursen redan har raderats.

HTTP metod	**Skall vara idempotent**
GET	Ja
POST	Nej
PUT	Ja
PATCH	Nej
DELETE	Ja
HEAD	Ja
OPTIONS	Ja

REST API-metoder **SKALL** (ARQ.06) följa den angivna idempotensen i tabellen ovan.

API Response

En response skickas som ett svar av servern på request-meddelandet klienten skickade.

Response Headers

Följande responsheaders **KAN** (ARP.01) inkluderas i svaret:

Header	Värde
Access-Control-Allow-Origin	Tillåter webbadresser som får åtkomst till tjänsten. **Obs**! Använd aldrig URL-adresser med wildcard(*) såvida inte REST-resursen verkligen är offentlig
Access-Control-Allow-Methods	Tillåtna metoder som får åtkomst.
Access-Control-Allow-Headers	Tillåtna HTTP headers som får åtkomst.
Content-Type	Exempelvis: • application/json • application/xml • multipart/form-data • text/html
Cache-Control	Informerar cachemekanismerna.
Date	Datum och tid då meddelandet skickades, i format definierat i RFC 3339
Expires	Datum och tid efter vilket svaret anses vara inaktuellt, i format definierat i RFC 3339
ETag	Används för att identifiera den specifika versionen av en resurs. Klienten bör inkludera detta i alla uppdateringsförfrågningar för att säkerställa att den är oförändrad

HTTP Response koder

I följande tabell definieras svar som är vanligt förekommande som **BÖR** (ARP.02) stödjas av ett API när det är tillämpbart enligt RFC 7231.

Kod	Orsak	POST	GET	PUT	DELETE	PATCH

200	OK		X	X		
201	Created	X				
202	Accepted	X		X	X	X
204	No content			X	X	X
400	Bad Request	X	X	X	X	X
401	Unauthorized	X	X	X	X	X
403	Forbidden	X	X	X	X	X
404	Not Found	X	X	X	X	X
405	Not Allowed	X	X	X	X	X
408	Request Timeout	X	X	X	X	X
415	Unsupported Media Type	X	X	X		X
422	Unprocessable Entity	X		X		X
500	Internal Service Error	X	X	X	X	X
501	Method Not implemented	X		X	X	X
502	Bad Gateway	X	X	X	X	X
503	Service Unavailable	X	X	X	X	X
504	Gateway Timeout	X	X	X	X	X

Statuskoderna är tillämpliga för HTTP requests på både samlingar (`collections`) och enskilda resurser (`resources{id}`). Statuskoderna och dess definitioner finns definierade i RFC 2616.

HTTP Response koder - beskrivningar

Kod	Status	När ska man använda den?
200	OK	Begäran har behandlats. Används av GET- och PUT-metoden
201	Created	Resursen skapades med hjälp av POST-metoden. Svarets HTTP header `Location` **BÖR** (ARP.03) returneras för att ange var den nyskapade resursen är tillgänglig

202	Accepted	Används för asynkron bearbetning för att indikera att servern har accepterat begäran men resultatet är inte tillgängligt ännu. Svarets HTTP header Location **BÖR** (ARP.04) returneras för att ange var den skapade resursen kommer att vara tillgänglig. Används av metoderna POST, PUT, DELETE eller PACTH.
204	No content	Servern behandlade framgångsrikt begäran och returnerar inget innehåll. Används av metoderna PUT, DELETE och PATCH.
400	Bad Request	Servern kan inte behandla förfrågan (såsom felaktig formningssyntax, storlek för stor, ogiltig eller vilseledande förfrågan, ogiltiga värden i begäran)
401	Unauthorised	Begäran kunde inte autentiseras
403	Forbidden	Begäran autentiserades men har inte behörighet att komma åt resursen
404	Not found	Resursen hittades inte
405	Not Allowed	Metoden implementeras inte för den här resursen. Svaret kan innehålla en Allow i headern som innehåller en lista över giltiga metoder för resursen
408	Request Timeout	Förfrågan avbröts innan svaret mottogs
415	Unsupported Media Type	Denna statuskod anger att servern vägrar att acceptera begäran eftersom innehållstypen som anges i begäran inte stöds av servern
422	Unprocessable Entity	Denna statuskod indikerar att servern tog emot begäran men att den inte uppfyllde kraven i backend. Ett exempel är att ett obligatoriskt fält inte tillhandahölls i payload.
500	Internal Server error	Ett internt serverfel. Response body kan innehålla felmeddelanden
501	Not Implemented	Det indikerar att förfrågningsmetoden inte stöds av servern och inte kan hanteras för någon resurs. Till exempel stöder servern GET, POST, PUT, DELETE och PATCH men inte OPTIONS-metoden
502	Unauthorised	Bad gateway
503	Forbidden	Servicen är inte tillgänglig

504	Not found	Gateway timeout

Felhantering

Servern ska vid varje HTTP anrop returnera en HTTP svarskod. Om ytterligare information behöver delges konsumenten skall schemat som specificeras i RFC 7807 användas.

Om HTTP svarskoderna inte räcker **SKALL** (FEL.01) API:et beskriva feldetaljer enligt schemat nedan.

Att beskriva feldetaljer (eng: problem details) är baserat på RFC 7807.

Schema för att beskriva feldetaljer

Schemat enligt RFC 7807 kan innehålla nedan attribut och kräver att mediatypen `application/problem+json` eller `application/problem+xml` används i svaret.

Attribut	Beskrivning
type	En URI som identifierar problemtypen som skall vara mer specialiserad än statuskoden. URI:n skall peka på dokumentation av feltypen om den används.
title	En kort beskrivning av problemtypen. Dvs att beskrivningen alltid ska vara identisk för samma typ av fel. Om inte type är angivet skall title reflektera beskrivningen av statuskoden.
status	Detsamma som statuskoden för HTTP-svaret om det används. Kan vara lämpligt att använda om svaret behöver vara självförklarande.
detail	Kort beskrivning av det faktiska felet, avsedd för läsning av människa.
instance	En URI som identifierar den faktiska resursen.

Feldetaljer kan utökas med nya attribut.

Exempel:

```
curl -X POST "https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/1234567-922/kontaktuppgifter" \

   -H  "Accept: */*" -H "Content-Type: application/json" \

   -d '{"email":"lisa.testperson@example.se"}' -i



HTTP/1.1 400
```

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

{

   "type" : "https://example.com/probs/req-parameters",

   "title" : "Felaktiga anropsparametrar",

   "status" : 400,

   "detail" : "Felaktigt organisationsnummer",

   "instance" : "/foretagsinformation/v2/organisationer/1234567-922",

   "invalidParameters" : [

      {

         "reason" : "Organisationsnummer ej giltigt",

         "value" : "1234567-922",

         "property" : "organisationsnummer"

      }

   ]

}
```

Utökning med nya attribut

I exemplet ovan är `invalid-parameters` en utökning. Konsumenter av ett API skall ignorera utökningar som de inte känner igen.

> Notera!
>
> Det är inte möjligt att utöka med nya ”standardattribut” utan att ändra mediatypen med denna design.

Utökning med identifierare för ett specifikt fel

För att underlätta felsökning och få en ökad spårbarhet kan en identifierare för det specifika felet förmedlas till konsumenten. Om det mönstret tillämpas skall identifieraren förmedlas på följande sätt.

```
HTTP/1.1 400

Content-Type: application/problem+json

{
```

```
  "type" : "https://example.com/probs/req-parameters",
  "title" : "Felaktiga anropsparametrar",
  "status" : 400,
  "detail" : "Felaktigt organisationsnummer",
  "instance" : "/foretagsinformation/v2/organisationer/1234567-922",
  "invalidParameters" : [
    {
      "reason" : "Organisationsnummer ej giltigt",
      "value" : "1234567-922",
      "property" : "organisationsnummer",
    }
  ]
  "transactionId" : "86032cbe-a804-4c3b-86ce-ec3041e3effc"
}
```

Versionshantering

Ett API kommer att förändras över tid och kommer därför att behöva versionshanteras.

Bakåtkompabilitet

Ett API **BÖR** (VER.01) så långt som möjligt ha en lös koppling mellan producent och konsument. Därför är bakåtkompabilitet en viktig aspekt för ett API. Med ett bakåtkompatibelt API kan förändringar i ett API genomföras utan att påverka konsumenten av API:et.

Följande är exempel på typer av ändringar anses bakåtkompatibla:

- Tillägg av ett nytt valfritt fält till en representation
- Tillägg av en ny länk till `_links` listan för en representation
- Tillägg för en ny resurs för ett API
- Tillägg av ny valfri sökparameter
- Ytterligare support för mediatyper
- Borttag av icke obligatoriska fält från representationer

Följande är exempel på typer av ändringar anses vara icke bakåtkompatibla:

- Borttag av obligatoriska fält från representationer
- Ändring av datatyp på fält
- Borttag av resurser eller operationer
- Borttag av support för mediatyp

Även beteendeändringar hos ett API kan vara icke bakåtkompatibla även om gränssnittet är intakt. Se även upp med exempelvis scheman och dess validering, där även bakåtkompatibla ändringar som tillägg av fält i en XML-struktur kan ge valideringsfel om ANY-elementet saknas i schemat.

Att införa icke bakåtkompatibla förändringar, dvs införa en ny MAJOR-version, är komplext både för konsumenter och producenter. Varje ny MAJOR version kräver att äldre versioner stöds tills alla konsumenter bytt till senaste version. Ju fler versioner som stöds, desto större belastning på API:ets förvaltning och support.

Som producent av ett API **BÖR** (VER.02) man undvika att införa icke bakåtkompatibla ändringar i sitt API och ibland är det oundvikligt. Som konsument **BÖR** (VER.03) man alltid vara tolerant, förvänta sig och hantera oväntade svar i ett meddelande. Detta skapar robusthet.

Versionsnumrering

Samtliga API:er **SKALL** (VER.04) använda semantisk versionshantering:

> {MAJOR}.{MINOR}.{PATCH}

Enligt semantisk versionshantering, hanteras kommande versioner enligt följande:

- MAJOR version ökas med 1 när du gör ändringar som påverkar bakåtkompatibiliteten för API:et.
- MINOR version ökas med 1 när du lägger till ny funktionalitet men 100 % bakåtkompatibilitet bibehålls.
- PATCH version ökas med 1 när du fixar buggar förutsatt att 100 % bakåtkompatibilitet bibehålls.

Versionsnumrering i URL:er

Alla API:er **BÖR** (VER.05) inkludera MAJOR versionen i den URL som används för ett specifikt API.

Exempelvis
`https://gw.api.bolagsverket.se/foretagsinformation/v2/`

MINOR och PATCH versionerna används inte i URL:n då bakåtkompabilitet råder.

Information om ett API och dess version

Att informera konsumenter om kommande ändringar är viktigt och det kan ske på olika sätt. Exempelvis kan man ta kontakt med samtliga konsumenter och informera om förändringar. Ibland är detta inte möjligt och då kan man informera via webbsidan där dokumentationen finns eller om man har nyhetsbrev kan detta vara en kanal. Det går även att tekniskt informera förändringar via en specifik resurs eller som en del av svarsmeddelandet på ett API-anrop, se nedan.

API nycklar eller liknande kan användas för att identifiera användare och därmed veta vilka som använder gamla versioner och stödja dem i sin uppgradering, för att då kunna avveckla de gamla versionerna.

Resursen api-info

Information om ett API **SKALL** (VER.06) tillgängliggöras via resursen *api-info* under roten till API:et. Genom att utföra GET på resursen *api-info* **SKALL** (VER.07) följande information returneras (exemplet nedan med camelCase)

- `apiName`: Namn på API
- `apiVersion`: API-version med MAJOR, MINOR och PATCH version
- `apiReleased`: Datum som denna API-version publicerades

- apiDocumentation: En länk till aktuell API dokumentation. Om API:et består av flera delar (t.ex. flera webbsidor) bör en samlingsplats skapas för alla delar skapas där alla delar är åtkomliga via länkar eller liknande.
- apiStatus: Visar API:ets tillstånd eller status enligt livscykelhantering längre ned på denna sida. T.ex. *beta*, *active* eller *deprecated*.

Ovan information ska anges, men ytterligare attribut och värden kan läggas till om önskas.

Exempel:

```
GET https://gw.api.bolagsverket.se/foretagsinformation/v2/api-info

//HTTP 200 OK

{

  "apiName": "foretagsinformation",

  "apiVersion": "2.0.1"

  "apiReleased": "2022-11-03"

  "apiDocumentation": https://gw.api.bolagsverket.se/foretagsinformation/v2/dokumentation

  "apiStatus": "active"

}
```

API utfasning

Utfasning (eng. deprecation) av ett API sker när ett nyare API finns tillgängligt. API ägare **BÖR** (VER.08) informera konsumenten i svaret på ett anrop, när en gammal version används. Denna information förser konsumenten med en påminnelse att API:et kommer att ersättas och att det troligen redan finns en nyare version redo att migrera till.

För att ange information om utfasning i svaret på ett anrop används bland annat förslaget till standard The Deprecation HTTP Header Field (Dalal Deprecation Header) samt standarderna RFC 8594 (Sunset HTTP Header Field) samt RFC 8288 (Web Linking).

Fältet Deprecation följt av ett datum i svaret indikerar för konsumenten att en nyare version av API:t finns. Fältet Sunset följt av ett datum i svaret indikerar att denna version av API:t slutar att fungera vid det aktuella datumet. Datumet anges enligt standarden RFC 3339. Till dessa två fält kan man också lägga till länkar till den nyaste versionen eller ett alternativt API som man kan använda. Något av följande relationsvärden **BÖR** (VER.09) användas

- successor-version: Pekar på den resurs som efterträdde den resurs som anropades (RFC 5829)
- latest-version: Pekar på den senaste versionen av resursen (RFC 5829)
- alternate: Pekar på en alternativ resurs (HTML 4.01 Specification)

Följande exempel visar hur ett svar på ett API som är under utfasning kan se ut.

```
GET `https://gw.api.bolagsverket.se/foretagsinformation/v1/organisationer/5568108988`


// 200 OK

Deprecation: 01 Jan 2022 00:00:00 UTC+1

Link: https://gw.api.bolagsverket.se/foretagsinformation/v2/; rel="latest-version"

Sunset: 30 Jun 2023 23:59:59 UTC+1

Content-Type: application/json

{

  "namn" : "Volvo Cars AB",

  "organisationsnummer" : "5568108988",

  "bolagsform" : "Aktiebolag",

  "registreringsar" : "2010"

}
```

Livscykelhantering

Ett API genomgår, precis som vilken produkt som helst, en livscykel, från utveckling tills dess att den tas ur bruk. Med ett tydligt och gemensamt regelverk för vilka tillstånd ett API kan befinna sig i blir det enklare för konsumenter av ett API att förstå vad som kan förväntas av API:et gällande exempelvis dokumentation och support.

Tillstånd för ett API

Följande tillstånd eller status **BÖR** (VER.10) användas för att beskriva vart i livscykeln ett API befinner sig i.

- alpha
- beta
- active
- deprecated
- retired
- decommisioned

En livscykel kan indelas i tre faser; utvecklingsfasen, aktiv fas samt avvecklingsfas.

Utvecklingsfas

Under utvecklingsfasen är det lämpligt att skilja på de två tillstånden *alpha* och *beta*. *alpha* används främst internt inom organisationen, medan *beta* kan förekomma i t.ex. integrationstester med andra organisationer.

alpha

Ett API i status *alpha* genomgår snabb iteration med en känd uppsättning användare, där användarna måste vara toleranta mot förändring. Antalet användare är relativt få i antal, så att det är möjligt att kommunicera med dem alla individuellt.

Att införa icke bakåtkompatibla ändringar är både tillåtet och förväntat för API:er i status *alpha*, och användarna kan inte förvänta sig några krav på stabilitet.

beta

Ett API i status *beta* anses vara fullständigt och redo att förklaras *active* i närtid. API:er i status *beta* kan exponeras för en okänd och potentiellt stor uppsättning användare och anses tillgänglig för allmänheten.

Eftersom användare av *beta* API:er tenderar att ha en lägre tolerans för förändring bör dessa vara så stabila som möjligt men tillåtas ändras med tiden som omfattar även icke bakåtkompatibla ändringar.

Aktiv fas

I denna fas har API:et lämnat utvecklingsfasen och är allmänt tillgängligt.

active

Ett API i status *active* är den senaste versionen och är fullt supporterad. Det är den rekommenderade versionen att använda. Ibland kallas denna status också för *general availability*, GA.

Avvecklingsfas

API:er är inte avsedda att finnas för evigt, utan producenten har ett ansvar att tydligt förmedla när API:et kommer att dras tillbaka. Vissa API:er syftar bara till att stödja ett begränsat användningsfall under en kortare tid, för att sedan förvinna och kanske ersättas med ett nyare bättre anpassat API.

En *End-of-Life* (EOL) policy definierar en process som API:er går igenom i dess livscykel, från *active* till *decommissioned*.

Policyn avser att säkra en tydlig och konsistent tidsplan som API konsumenter kan ta del av för att planera sin migration från de äldre API:erna till de nya. Konsumenten kan därmed planera hur de ska hantera sin tekniska skuld som förflyttningen ger.

deprecated

En API version i status *deprecated* har efterträtts av en nyare version. Den supporteras ofta i flera månader efter den hamnar i status *deprecated*.

retired

En API version i status *retired* supporteras inte längre. Applikationer som använder en API som är *retired* ska migrera till en API version i status *active* så fort som möjligt.

decommissioned

En API version i status *decommissioned* är inte längre tillgänglig i produktion. Denna fas infaller efter status *retired*. Denna status kallas ibland för *sunset*, se vidare avsnitt API utfasning ovan.

Versionshanteringsregler för ett API:s livscykel

Versioner som befinner sig i status *alpha* eller *beta* **SKALL** (VER.11) börja med en MAJOR version med siffran ”0”. I övrigt gäller samma regler som för versionshantering i den aktiva fasen.

API:ets första publika version i status *active* **SKALL** (VER.12) alltid börja med en MAJOR version med siffran ”1”.

Eftersom MINOR versioner är bakåtkompatibla med föregående MINOR versioner inom samma MAJOR version **SKALL** (VER.13) MINOR versioner anses *retired* direkt när en ny MINOR version av samma MAJOR version blir *active*. Förändringar av denna sort bör inte ha någon påverkan på befintliga konsumenter, så API:et behöver inte gå igenom status *deprecated* och därmed en migrering för kund.

När en ny MAJOR version publiceras så **SKALL** (VER.14) de äldre versionerna fasas ut. En MAJOR version **SKALL INTE** (VER.15) sättas till *deprecated* förrän en ersättare är *active* och det finns en tydlig migrationsväg för all funktionalitet som upprätthålls framåt. Detta **BÖR** (VER.16) inkludera dokumentation och migrationsverktyg/exempelkod som beskriver vad konsumenterna behöver göra för en ren migration. Enda undantaget är i fall API:et ska fasas ut helt och hållet.

Om ett API ska sättas till *deprecated* och samtidigt har externa konsumenter **BÖR** (VER.17) tid för utfasning och begränsningar övervägas för att minimera kundpåverkan. API i statusen *deprecated* **SKALL** (VER.18) vara i status *deprecated* under en tillräckligt lång tid för att ge konsumenterna möjlighet att kunna migrera.

Om ett API i status *active* eller *deprecated* inte har några användare så **BÖR** (VER.19) det direkt sättas i status *retired*.

Webhooks

Webhooks möjliggör för API konsumenter att bli notifierade när en specifik händelse inträffar i ett API gränssnitt. För att tillgängliggöra detta så ska API konsumenten sätta upp en URL endpoint som kan acceptera och bearbeta en HTTP POST request. Denna endpoint ska också registreras hos API producenten tillsammans med en specifikation på vilka händelser som är intressanta att lyssna på.

Webhooks används i flera olika sammanhang som t.ex. händelser gällande uppdateringar av data hos producenten, eller att meddela att kod är incheckad till ett repository - som i sin tur kanske triggar ett bygge enligt Continuous Integration.

I en API Webhook implementation skickas vanligen payload som ett JSON dokument som förser mottagaren med begränsad information om den inträffade händelsen, t.ex. så kan innehållet bara vara en referens till resursen som har uppdaterats. Det innebär att konsumenten måste göra en request mot ett API för att få denna information i sin helhet.

Om man implementerar en API Webhook som skickar med detaljerna direkt i dess payload, så underlättar man för konsumenten och minskar nättrafiken, men kan bygga komplexitet i design av API:et. Känsligheten i det data som skickas i payload bör beaktas här.

Om Webhooks väljs att implementeras så bör följande designriktlinjer övervägas

- Om känslig information skickas i Webhook, **SKALL** (WEB.01) kryptering/signering övervägas i POST request för att erhålla integritet och verifikation. TLS certifikat är också ett alternativ.
- Ett unikt händelseid/transaktionsid **BÖR** (WEB.02) inkluderas i POST request, för att underlätta felsökning och loggning.
- Konsumentens endpoint **SKALL** (WEB.03) alltid returnera ett svar på inkommen POST, så producenten vet att svaret är levererat.
- API **SKALL** (WEB.04) stödja en återsändningsfunktion i de fall konsumentens URL inte är tillgänglig, eller möjliggöra att konsument gör en request som använder tidpunkt eller räknare. På detta sätt kan konsumenten antingen få begärd information från producent, eller begära den själv med senast sparat tidpunkt eller räknare.
- En tydlig onboarding/offboarding process **SKALL** (WEB.05) finnas, som beskriver den process som konsumenten genomgår för att börja använda webhooks mot producenten samt hur avslut av samma sker.
- Eftersom användandet av Webhooks kräver ytterligare infrastruktur hos API konsumenterna, **BÖR** (WEB.06) detta tas hänsyn till när lösningen designas.

Hypermedia

Hypermedialänkar kan användas i API:er. Dessa informerar API konsumenten vart ytterligare innehåll kan hämtas. Dessa API länkar tillåter konsumenten att lokalisera resurser utan att behöva förstå denna resurs och dess relationer fullt ut. Hypermedia är högsta nivån av mognad (nivå 3) enligt Richardsons Maturity Model, där denna profil har satt implementation som ett BÖR krav och är en förutsättning för att ett API ska kallas RESTful.

För att vara ett hypermedia kompatibelt API ska följande vara uppfyllt:

- Det finns ett väldefinierat index eller startpunkt för navigering för varje API som klienten navigerar till för att få tillgång till samtliga resurser.
- Klienten behöver inte bygga logiken för att sätta samman URL som används för att utföra olika förfrågningar eller att koda någon slags affärslogik till svaret.
- Klienten förstår att det är serversidan som äger processen att sätta giltiga URL:er.
- Klienter hanterar URL:er som ogenomskinliga (eng: opaque).
- API:er som använder hypermedia i sin representation kan sömlöst bli utökade. När nya metoder introduceras, kan svaren inkluderas med relevanta HATEOAS länkar. På detta sätta kan klienter ta tillvara på de inkrementella förbättringar som API:et får över tid. Om ett API till exempelvis börjar stödja PATCH operationen, kan klienten använda detta för att delvis uppdatera resursen.

Genom att skicka med dessa länkar underlättar man för klienterna att förstå API:erna. Dock tar det inte bort ansvaret för klienten att förstå vilken data som kan användas för att använda operationerna POST/PUT/PATCH. Ett API **SKALL** (HYP.01) ge god dokumentation som tydligt beskriver alla länkar, dess relationstyper och de svarsformat som varje URL ger.

Länkar

Hypermedia liknar navigering på en webbsida, där användarna inte förväntas veta strukturen av hemsidan innan de besöker den. De kan enkelt hitta runt på hemsidan med den inbyggda navigeringen genom länkar.

API:er som inte bidrar med länkar är svårare att använda och förväntar sig att användaren refererar till dokumentationen för att förstå dem eller vid exempelvis felsökning.

Exempel:

```
GET /foretagsinformation/v2/organisationer

HTTP/1.1 200 OK
```

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

...

{

    "_meta": [
    {

        "totalRecords": 1,

        "page": 1,

        "limit": 20,

        "count": 1

    }],

    "_links": [

    {

        "href": "/foretagsinformation/v2/organisationer",

        "rel": "self"

    }],

    "organisationslista": [

    {

        "namn": "Aktiebolaget Volvo",

        "_links": [

        {

            "href" : "/foretagsinformation/v2/organisationer/5560125790",

            "rel": "self",

            "method": "GET"

        }]

    }]

}
```

HATEOAS

Uttal: HAT-E-OAS

HATEOAS (Hypermedia As The Engine Of Application State) syftar till att exponera de metoder som kan utövas för varje resurs genom hyperlänkar. Varje länk i svarsdata representerar övergångar till de möjliga tillstånd konsumenten kan ta utifrån det aktuella tillståndet.

Bankexempel:

```
{
    "kontonummer":"12345",
    "balans": 100.00,
    "_links":[
        {"rel": " **insättning**", "href":"/konton/12345/insattning"},
        {"rel": " **uttag**", "href":"/konton/12345/uttag"},
        {"rel": " **överföring**", "href":"/konton/12345/overforing"}
    ]
}
```

Om kontot i exemplet har övertrasserats med 25 så är det enda tillgängliga alternativet insättning:

```
{
 "kontonummer":"12345",
 "balans": -25.00,
 "_links":[
 {
   "rel": "insättning",
   "href":"/konton/12345/insattning"
 }]
}
```

Valet att implementera HATEOAS design in API:er beror på hur redo konsumenterna är att konsumera HATEOAS och hur mycket energi och tid man är redo att lägga på design och implementation.

Hypermedia kompatibla API:er

Ett hypermediakompatibelt API erbjuder konsumenterna en uppsättning transaktionstillstånd att använda.

API metoderna DELETE, PATCH, POST och PUT initierar alla en tillståndsförändring för en resurs. En GET-förfrågan **SKALL INTE** (HYP.02) ändra tillstånd på den resurs som hämtas.

För att kunna erbjuda en bättre upplevelse för API konsumenterna, **BÖR** (HYP.03) API:er erbjuda en lista på tillståndsförändringar som finns tillgängliga för varje resurs i `_links` listan.

Ett svar **BÖR** (HYP.04) returnera relaterade och giltiga länkelement. Ett svar **SKALL** (HYP.05) också returnera ett länkelement till sig själv, `self` .

Hypermedia response

Existerande standarder, såsom JSON, ATOM, Collection-JSON, HAL m.fl. **BÖR** (HYP.06) användas istället för egna format.

Exemplen nedan använder sig av formatet HAL (`Content-Type: application/hal+json`) som har stöd för länkade data genom attributet `_links`.

Nedan följer ett exempel på ett API som exponerar dess operationer och använder HATEOAS gränssnittet:

En klient startar kommunikationen med tjänsten genom att skicka en POST på URL `/organisationer` . Denna URL stödjer både GET och POST operationer.

Request

```
POST https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer

organisationslista: [

{

    "namn": "Volvo Bikes AB",

    "organisationsnummer": "5568101234",

    ...

} ]
```

Response

API:et skapar en ny organisation och returnerar följande länkar till klienten som response

- En länk till den skapade resurser i headern Location (för att efterleva specifikationen för ett 201 svar)
- En länk för att hämta den kompletta representationen av organisationen (self länken för GET)
- En länk för att ta bort organisationen (DELETE)
- En länk för att uppdatera organisationen (PUT)
- En länk för att delvis uppdatera organisationen (PATCH)

```
HTTP/1.1 201 CREATED

Content-Type: application/hal+json

Location: https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234

...

{

    "_links": [

    {

       "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",

        "rel": "self",

       "method": "GET"

     },

     {

       "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",

         "rel": "delete",

         "method": "DELETE"

     },

     {

         "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",

         "rel": "replace",

         "method": "PUT"

     },

     {
```

```
        "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",
        "rel": "edit",
        "method": "PATCH"
      }
    ]
}
```

En klient som får detta svar kan spara dessa länkar för senare användning.

För att visa samtliga organisationer gör klienten en GET på URL `/organisationer`.

Request

```
GET https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer
```

API:et svarar med samtliga organisationer i systemet med respektive `self` länk.

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/hal+json
Location: https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234
...
{
    "_links": [
    {
        "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",
        "rel": "self",
        "method": "GET"
    },
    {
        "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",
        "rel": "delete",
        "method": "DELETE"
```

```
    },
    {
        "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",
        "rel": "replace",
        "method": "PUT"
    },
    {
        "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5568101234",
        "rel": "edit",
        "method": "PATCH"
    }
   ]
}
```

Klienten kan använda self länken för varje organisation och får där operationerna som kan genomföras på resursen *organisationer*.

Request

```
GET https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5560125790
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
...
organisationslista: [
{
    "namn": "Aktiebolaget Volvo",
    "organisationsnummer": "5560125790",
    ...
    "_links": [
    {
```

```
        "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5560125790",
          "rel": "self",
          "method": "GET"
      },
      {
          "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5560125790",
          "rel": "delete",
          "method": "DELETE"
      },
      {
          "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5560125790",
          "rel": "replace",
          "method": "PUT"
      },
      {
          "href":
"https://gw.api.bolagsverket.se/foretagsinformation/v2/organisationer/5560125790",
          "rel": "edit",
          "method": "PATCH"
       }
     ]
}]
```


För att ta bort en organisation, använder klienten URL från länkrelationen ”rel”: ”delete” och genomför operationen med följande URL

```
DELETE https://gw.api.bolagsverket.se/foretagsinformation/v1/organisationer/5560125790
```

Relativa kontra absoluta URL:er

Webbadresserna som genereras av API:et **BÖR** (HYP.07) vara absoluta webbadresser.

Anledningen är att det leder till enkel användning för klienten, så att den aldrig behöver ta reda på rätt bas-URL för en resurs, som behövs för att tolka en relativ URL. RFC 3986 för URL:er anger algoritmen för att bestämma en bas-URL, vilket är ganska komplext. Ett av alternativen för att hitta basadressen är att använda URL:en för den begärda resursen. Eftersom en resurs kan visas under flera webbadresser (till exempel som en del av en samling eller fristående) skulle det vara en betydande overhead för en klient att komma ihåg var han hämtade en representation från. Genom att använda absoluta webbadresser uppstår inte det här problemet.

API Gateways

I ett hypermediasvar, eftersträvas användning av absoluta URL:er (enligt ovan). Om API:er befinner sig bakom en API Management lösning i form av en API Gateway kommer den faktiska URL:en för (backend) API:et att vara annorlunda än URL som är anropas externt, det vill säga den i API gateway.

Som regel **SKALL** (HYP.08) en API Management gateway skicka ytterligare en header med i anropet till backend API:et som innehåller API Gateway bas URL:en för det aktuella API:et. Headern **SKALL** (HYP.09) namnges som `forwarded`, enligt beskrivningen i RFC 7239. De två parametrarna `host` och `proto` är fördefinierade i RFC, men ytterligare parametrar behövs:

`host`: Måste innehålla värdnamnet för API-gateway. Detta är värden som en API-klient använder för att faktiskt prata med API:et.

`proto`: Det protokoll/schema som ska användas.

Tillägg:

- `port`: Porten för API-gatewayen. Vanligtvis kan detta antas vara om lika med port 443; använd `proto` om du avviker från https.
- `prefix`: Bas-URL för API på API-gateway, t.ex. prefix/api/v1

Exempel:

En begäran går till *https://api.exempel.se/ingest/v1/bulk/63874638746834*, som vidarebefordras till tjänsten i backend. API-gateway måste vidarebefordra den här informationen på följande sätt:

```
Forwarded: proto=https; host=api.exempel.se; port=443; prefix=/ingest/v1
```

Om hypermedia används måste informationen i forwarded headern användas för att skapa korrekta hypermedialänkar i svaret, så att en API konsument dirigeras till rätt plats.

Om denna header inte finns **BÖR** (HYP.10) API backend använda sin egen absoluta bas URL som fallback.

Link Description Object

Länkar **SKALL** (HYP.11) beskrivas med ett *Link Description Object* schema. Ett Link Description Object schema beskriver varje länks relation. Nedan följer en sammanfattning över de delar som ingår i en Link Description Object.

href

- Ett värde för href **SKALL** (HYP.12) ges.
- Värdet för href **SKALL** (HYP.13) vara en URL som används för att bestämma målet för den relaterade resursen.
- Används endast absoluta URL värden för href. Klienter kan spara URL för en länk för att använda senare. Utvecklare **SKALL** (HYP.14) använda värden från inkommande host headern (t.ex. gw.api.bolagsverket.se) i den absoluta URL:n.

rel

- rel står för relation som definieras nedan i Link Relation Type.
- Värdet av rel indikerar relationens namn i resursen.
- Ett värde för rel **SKALL** (HYP.15) anges.

method

- method identifierar HTTP verbet som **SKALL** (HYP.16) användas i förfrågan i länken.
- Om ingen method anges är GET defaultmetod, men method **SKALL** (HYP.17) anges.

title

- title används för att rubricera länken för att på så sätt ge värdefull dokumentation för att enklare förstå länkens syfte. title är inte obligatorisk.

Link Relation Type

En *Link Relation Type* agerar identifierare för en länk. Ett API **SKALL** (HYP.18) tilldela en meningsfull Link Relation Type som entydigt beskriver semantiken för länken. Klienter använder Link Relation Type för att hitta länken att använda från en representation.

När semantiken av en Link Relation Type som önskas användas också återfinns i IANA's lista över standardiserade länkrelationer **SKALL** (HYP.19) IANA’s typ användas.

Tabellen nedan beskriver de vanligaste typerna:

Link Relation Type	Beskrivning
self	Förmedlar en identifierare för länkens sammanhang. Vanligen en länk som pekar på resursen själv
create	Refererar till en länk som kan användas för att skapa en ny resurs
edit	Refererar till ett ändra (helt eller delvis) representationen identifierad av länken. Använd denna för att representera en PATCH operation
delete	Använd Extended link relation type för att representera en DELETE operation
replace	Refererar till att helt uppdatera (eller ersätta) representationen identifierad av länken. Använd Extended link relation type för att representera en PUT operation
first	Refererar till den första sidan i en resultatlista
last	Refererar till sista sidan i en resultatlista
next	Refererar till nästa sida i en resultatlista
prev	Refererar till förra sidan i en resultatlista
collection	Refererar till en kollektions resurs (t.ex. */v2/organisationer*)
latest-version	Pekar på en resurs som har senaste (nuvarande) versionen
search	Refererar till en resurs som kan användas för att söka genom länken kontext och relaterade resurser
up	Refererar till föräldraresursen i en hierarki av resurser

Vid användande av olika typer av händelser, ska motsvarande händelsenamn användas som Link Relation Type (t.ex. `activate`, `cancel`, `send`, ...).

Filtrering, paginering och sökparametrar

Möjligheten till att söka och sortera informationen skapar en flexibilitet för konsumenten att enbart konsumera den information som är av intresse. Det handlar om att på ett enkelt sätt öka användbarheten av API:et.

Rekommendationen är att utföra filtrering med hjälp av frågeparametrar. En tumregel vid design är att först utesluta att parametern som potentiellt ska användas för filtrering inte egentligen borde vara en header parameter samt att informationen inte är känslig (exempelvis lösenord). I slutändan är det viktigaste att de vanligaste användningsfallen ska vara enkla att realisera för användaren och det ska vara svårt att göra fel.

> **Notis!**
>
> **Filtrerings och sökparametrar är inte en del utav definitionen av en webbresurs. Felaktig URL: /annonser/from/20210101/to/20210131**

Filtrering

Filtrering ger användaren möjligheter att välja den information som den är intresserad av genom API:et.

Den enklaste modellen är att ange ett attribut som används som filter för en samling av resurser.

Tänk en samling som exempelvis innehåller alla kommuner i Sverige:
`/kommuner`

Ett sätt att filtrera så att svar enbart innehåller kommuner baserat från ett län:
`/kommuner?lan=17`

Då det kan vara svårt att veta hur ett län kan anges ska det tydligt framgå vilka potentiella värden som kan anges, i schema, dokumentation eller via en hyperlänk. Ex. https://www.scb.se/hitta-statistik/regional-statistik-och-kartor/regionala-indelningar/lan-och-kommuner/

Exempel
`/kommuner?lan=17`

`/annonser?from=2021-01-01&to=2021-01-31`

`/bolag?SNI_kod=10`

Operator

Likhetstecken (=) är den enda operator som används vid enklare filtrering och (&) används för att separera flera parametrar. Det finns andra sätt att filtrera som erbjuder fler operatorer.

Namnsättningskonvention för sökparametrar

Att filtrera kan realiseras genom att uttrycka sig med sökfrågor (eng. query). Nedan följer några rekommendationer kring detta:

- Parameternamn **SKALL** (FNS.01) anges med en konsekvent namnkonvention inom ett API, exempelvis antingen snake_case eller camelCase.
- Sökparametrars värden **SKALL** (FNS.02) vara percent-endcoded (RFC3986) på UTF-8 format.
- Exempelvis skall ‘ ', dvs blanksteg, kodas till '%20’.
- Sökparametrar **SKALL** (FNS.03) starta med en bokstav.
- Sökparametrar **BÖR** (FNS.04) använda enbart gemener.
- Sökparametrar **BÖR** (FNS.05) vara frivilliga.
- Sökparametrar **BÖR** (FNS.06) använda tecken som är URL-säkra (tecknen A-Z, a-z, 0-9, "-", ".", "_" samt "~", se vidare i RFC 3986).

Exempel
`https://taxonomy.api.jobtechdev.se/v1/taxonomy/specific/concepts/country?preferred_label=South%20america&version=1`

Paginering

Avser metoden att dela upp större datamängder till mindre mer hanterbara delar som levereras i varje request.

Paginering rekommenderas användas för att hantera icke-funktionella krav såsom prestanda och upptider, ofta uttrycka i SLA krav.

Parametrar

Vid användande av paginering, **SKALL** (FNS.07) följande parametrar ingå i request.

Parameter	Beskrivning	Exempel
`page` eller `offset`	Page är sida i träfflista som konsument vill erhålla Offset är startposition i träfflistan som konsument vill börja hämta ifrån	`page=1` (default: 1) `offset=0` (default:0)

limit	Antal sökträffar per sida som konsumenten vill erhålla	limit=20 (default:20)

Strukturen blir då följande:
/organisationer?page=3&limit=20

/organisationer?offset=60&limit=20

Denna struktur är lätt att förstå och kan användas direkt i ett gränssnitt.

page **SKALL** (FNS.08) alltid starta med värde 1. Defaultvärde för limit **BÖR** (FNS.09) vara 20. offset börjar ofta med 0 som startposition, och beräknas med hjälp av limit till efterföljande sökning.

Response

När paginering implementeras, **BÖR** (FNS.10) man inkludera fält för _meta och _links för att förse konsumenten med information om resultatet, och erbjuda enkel navigation i träffbilden. Dessa fält tar bort klientens behov hantera pagineringen på konsumentsidan, alternativen serveras direkt i svaret.

- _links fältet **BÖR** (FNS.11) inkludera länkar for self, first, last, next och prev.
- Varje länk **SKALL** (FNS.12) inkludera alla eventuellt övriga frågeparametrar som skickades i första request.

Ett exempel på svar:

```
HTTP/1.1 200 OK

Content-Type: application/hal+json

...

{

    "_meta": [

    {

        "totalRecords": 98,

        "page": 3,

        "limit": 20,

        "count": 20

    }] ,

    "_links": [
```

```
    {
        "href": "/organisationer?page=3&limit=20",
        "rel": "self"
    },
    {
        "href": "/organisationer?page=1&limit=20",
        "rel": "first"
    },
    {
        "href": "/organisationer?page=5&limit=20",
        "rel": "last"
     },
     {
        "href": "/organisationer?page=2&limit=20",
        "rel": "prev"
     },
     {
        "href": "/organisationer?page=4&limit=20",
        "rel": "next"
     }
    ],
    "organisationslista": []
}
```


Följande struktur **BÖR** (FNS.13) appliceras på `meta` objektet

Attribut	Typ	Beskrivning
totalRecords	Heltal	Totalt antal träffar på aktuella sökparametrar, oaktat sida och antal
page	Heltal	Aktuell sida för träffbilden
offset	Heltal	Startposition i träfflistan
limit	Heltal	Begärt antal resultat per sida
count	Heltal	Antal resultat på aktuell sida

Utanför träffbilden

Om konsumenten begär en sida som är utanför det giltiga spannet för träffbilden, t.ex. om sidspannet går från 1 till 15 men konsumenten begär `?page=0` eller `?page=99`, **BÖR** (FNS.14) svaret innehålla en tom träffbild med HTTP statuskod 200.

Ett exempel följer:

```
HTTP/1.1 200 OK

Content-Type: application/hal+json

...

{

    "_meta": [

    {

        "totalRecords": 98,

        "page": 99,

        "limit": 20,

        "count": 0

    }],

    "_links": [

    {

        "href": "/organisationer?page=99&limit=20",

        "rel": "self"

    },

    {

        "href": "/organisationer?page=1&limit=20",

        "rel": "first"

    },

    {

        "href": "/organisationer?page=15&limit=20",

        "rel": "last"
```

```
    }

    ],

    "organisationslista": []

}
```

Cachning

Cachelagring ger möjlighet att lagra kopior av data för att öka tillgänglighet och prestanda. Skall appliceras när det är tillämpbart enligt RFC 9111.

Cachning är ett av de sex kraven enligt restprinciperna (se vidare En introduktion till REST), ett API behöver överväga för att benämnas som RESTful. Kravet innebär inte att all data faktiskt behöver cachas, utan att resurser som tjänar på att mellanlagras identifieras och märks upp enligt nedan förslag.

> **Alla API:er behöver alltså inte cachelagring, för exempelvis ett API som servar med realtidsdata är det troligtvis inte brukligt. Det är viktigt att förstå hur HTTP-cachelagring påverkar en implementation innan det införs.**

Begäran

När en konsument begär en resursrepresentation, går begäran igenom en cache eller en serie cacheminnen (lokal cache, proxycache eller omvänd proxy) mot den tjänst som är värd för resursen.

Genom att använda HTTP-headers anger en serverkälla om ett svar kan cachelagras och i så fall av vem och hur länge. Att optimera nätverket med cachelagring förbättras den övergripande servicekvaliteten på följande sätt:

- Minskar bandbredden
- Minskar latensen
- Minskar belastningen på servrar
- Kan dölja nätverksfel

Cachelagring i REST-API:er

Att cachelagra är en av de arkitektoniska begränsningarna för REST, och beskrivs i RFC 9111.

1. GET-request **BÖR** (CAC.01) kunna cachelagras som standard. Vanligtvis behandlar webbläsare alla GET-request cachelagrade.
2. POST-request kan inte cachelagras som standard men kan cachelagras som antingen en `Cache-Control` header eller en `Expires` header med en instruktion (för att uttryckligen tillåta cachelagring) läggs till i svaret.
3. Svar på PUT och DELETE förfrågningar kan inte cachelagras alls.

Det finns två huvudsakliga HTTP-cache headers: `Cache-Control` och `Expires`.

Cache-Control

Cache-Control meddelar om klienten ska cachelagra.

Exempel:

Alla delar av nätverket ska cachelagra innehållet i en timme:
Cache-Control: public, max-age=3600, must-revalidate

Expires

Expires instruktionen, när det används tillsammans med Cache-Control, anger ett datum då innehållet anses inaktuellt. Om både Expires och Cache-Control anges har Cache-Control företräde.

Exempel:
Cache-Control: public

Expires: Mon, 25 Jun 2019 21:31:12 GMT

Tidpunkt i Expires skrivs alltid i GMT.

Övriga HTTP-cacherubriker

Andra förekommande HTTP-cacherubriker beskrivs nedan översiktligt.

Age

Age headern berättar hur lång tid som förflutit sedan svaret genererades eller validerades senast av servern.

Pragma

Pragma är en HTTP/1.0-header. Pragma: no-cache är som Cache-Control: no-cache genom att det tvingar cachen att skicka begäran till serverkällan för validering innan en cachekopia lämnas. Pragma är dock inte specificerad för HTTP-svar och är därför inte en tillförlitlig ersättning för det allmänna HTTP/1.1 Cache-Control headern.

Pragma bör endast användas för bakåtkompatibilitet med HTTP/1.0-cache där Cache-Control HTTP/1.1-headern ännu inte finns.

Warning

Den generella HTTP headern Warning innehåller information om eventuella problem med meddelandets status. Mer än en Warning header kan visas i ett svar.

Warning headers attribut kan i allmänhet tillämpas på alla meddelanden, men vissa varningskoder är specifika för cache och kan endast tillämpas på svarsmeddelanden.

ETag

En ETag (entitetstagg) kan användas i response header för entitetsdata. En ETag är en sträng som anger en resursversion – varje gång en resurs ändras, ändras även ETag. ETag ska cachelagras som en del av data hos klienten. ETag är främst användbart för att spara bandbredd.

```
GET /organisationer/5560125790


HTTP/1.1 200 OK
...
Cache-Control: max-age=600, private
Content-Type: text/json;
ETag: "2147483648"
 Content-Length: ...
{ "organisationsnummer": "5560125790", "namn": "Aktiebolaget Volvo", ... }
```

Klienten konstruerar en GET-begäran som innehåller den ETag för närvarande cachelagrade versionen av resursen som refereras i ett If-None-Match HTTP-header:

```
GET /foretagsinformation/v1/organisationer/5560125790
If-None-Match: "2147483648"
```

ETag matchar

Returnera ett HTTP-response med en tom meddelandetext och en statuskod på 304 Not Modified.

ETag matchar inte

Data har ändrats och webb-API:et bör returnera ett HTTP-response med nya data i meddelande body och en statuskod för 200 OK.

Referenser

Följande källreferenser har använts i framtagande av Utvecklarportalens REST API-profil.

Referenserna kan delas in i följande områden:

- API guider
- Internet Engineering Task Force ISO (RFC)
- Övriga

API guider

Källa	URL	Kommentar
Microsoft REST API Guidelines	https://github.com/microsoft/apiguidelines/blob/vNext/Guidelines.md	
Paypal API Design Guidelines	https://github.com/paypal/apistandards/blob/master/api-styleguide.md	
Australian Government API	https://api.gov.au/	
UK Government API Design Guidance	https://www.gov.uk/government/collections/api-design-guidance	
Google Cloud API Design Guide	https://cloud.google.com/apis/design/	
Haufe API style guide	https://haufe-lexware.gitbooks.io/haufeapi-styleguide/content/	
Architectural Styles and the Design of Network-based Software Architectures	https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm	Roy Thomas Fieldings doktorsavhandling rörande REST
REST APIs must be hypertext-driven	https://roy.gbiv.com/untangled/2008/restapis-must-be-hypertext-driven	
Richardson Maturity Model steps toward the glory of REST	https://martinfowler.com/articles/richards onMaturityModel.html	
Architecture of the World Wide Web, Volume One	https://www.w3.org/TR/webarch/	

Förutom de angivna källorna gällande design guidelines så ligger många olika RFC specifikationer bakom de rekommendationer som anges under Utvecklarportalens profil.

Internet Engineering Task Force (RFC)

Källa	URL
RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels	https://datatracker.ietf.org/doc/html/rfc2119
RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1	https://datatracker.ietf.org/doc/html/rfc2616
RFC 3339 - Date and Time on the Internet: Timestamps	https://datatracker.ietf.org/doc/html/rfc3339
RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax	https://datatracker.ietf.org/doc/html/rfc3986
RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace	https://datatracker.ietf.org/doc/html/rfc4122
RFC 5789 - PATCH Method for HTTP	https://datatracker.ietf.org/doc/html/rfc5789
RFC 5829 - Link Relation Types for Simple Version Navigation between Web Resources	https://datatracker.ietf.org/doc/html/rfc5829
RFC 6454 - The Web Origin Concept	https://datatracker.ietf.org/doc/html/rfc6454
RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content	https://datatracker.ietf.org/doc/html/rfc7231
RFC 7239 - Forwarded HTTP Extension	https://datatracker.ietf.org/doc/html/rfc7239
RFC 7807 - Problem Details for HTTP APIs	https://datatracker.ietf.org/doc/html/rfc7807
RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format	https://datatracker.ietf.org/doc/html/rfc8259
RFC 8288 - Web Linking	https://datatracker.ietf.org/doc/html/rfc8288
RFC 8594 - The Sunset HTTP Header Field	https://datatracker.ietf.org/doc/html/rfc8594
RFC 9111 - HTTP Caching	https://datatracker.ietf.org/doc/html/rfc9111

Övriga

Källa	URL
Open API specification	https://swagger.io/specification/
Semantisk versionshantering	https://semver.org/lang/sv/
The Deprecation HTTP Header Field	https://tools.ietf.org/id/draft-dalal-deprecation-header-01.html
W3C HTML 4.01 Specification	https://www.w3.org/TR/html401/
HAL - JSON Hypertext Application Language	https://datatracker.ietf.org/doc/html/draft-kelly-json-hal-06