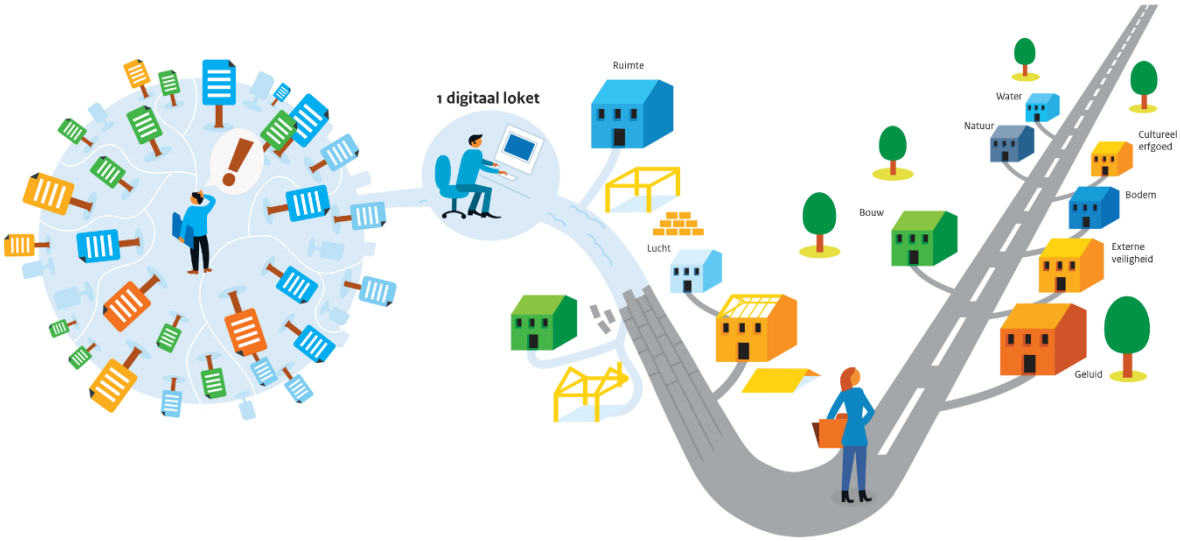
Deelprogramma
Digitaal Stelsel Omgevingswet

API-strategie

Versie 2.0 Vastgesteld 26-03-2020



Dit document is vastgesteld door het Stelsel Architectuur Board (SAB). Hiermee is de richting op hoofdlijnen goedgekeurd. Omdat de Tactische beheerorganisatie (TBO) een agile ontwikkelaanpak volgt, zullen nieuwe inzichten doorlopend op basis van wijzigingsvoorstellen worden voorgelegd aan het SAB. Wijzigingsvoorstellen die zijn vastgesteld zullen periodiek in een nieuwe versie van dit document worden verwerkt.

***Een API is als een UI voor ontwikkelaars**:*
"Zoals een goede User eXperience (UX) essentieel is voor een UI, is een goede Developer eXperience (DX) essentieel voor een API."

Colofon

Titel	:	API-strategie
Versie	:	2.0 Vastgesteld
Datum	:	26-03-2020
Opdrachtgever	:	Programma Implementatie Omgevingswet
Opdrachtnemer	:	Deelprogramma DSO
Auteur	:	Tony Sloos
Contactpersonen	:	A.J. (Tony) Sloos *Domeinarchitect Kernfuncties & Informatie* +31 6 1125 2597 tony.sloos@minbzk.nl

Versiehistorie

Versie	Status	Datum	Auteur(s)	Toelichting
1.0	Vastgesteld	20-07-2017	A.J. Sloos Stephen Oostenbrink Dimitri van Hees	Vastgesteld met wijzigingen.
1.1	Vastgesteld	12-03-2018	A.J. Sloos Dimitri van Hees	Vastgesteld met wijzigingen. Bronversie voor 2.0 major-update.
1.9	Concept	25-02-2020	A.J. Sloos	1e conceptversie major-update.
1.95	Concept	27-02-2020	A.J. Sloos	2e conceptversie major-update.
1.99	Concept	28-02-2020	A.J. Sloos	3e conceptversie major-update.
1.99a	Concept	13-03-2020	A.J. Sloos	Review Joost Farla verwerkt.
2.0	Vastgesteld	26-03-2020	A.J. Sloos	Vastgesteld met wijzigingen.

Goedkeuring

Functie	Naam	Versie	Datum	Handtekening
Stelselarchitect namens het Opdrachtgevend Beraad	René Kint	2.0		
Programma Directeur Implementatie Omgevingswet (namens de Programma Raad)	Bert Uffen	2.0		
Lead-architect programma	Anton van Weel	2.0		

Distributie

Functie/Orgaan	Versie	Opmerkingen
Opdrachtgevend Beraad Omgevingswet	2.0	
Programmaraad Implementatie Omgevingswet	2.0	
Stelsel Architectuur Board (SAB)	2.0	
Stelsel Architectuur Team (SAT)	1.95, 1.99, 2.0	
Programma/Project Architectuur Team (PAT)	1.99, 2.0	
Projecten	1.99, 1.99a, 2.0	

Review

Naam	Versies
Andre Batenburg (AB), BLA provincies	1.95, 1.99, 2.0
Jan van Langeveld (JvL), BLA gemeenten	1.95, 1.99, 2.0
Paul de Frankrijker (PdF), BLA rijk en BLA waterschappen	1.95, 1.99, 2.0

Inhoudsopgave

1. Inleiding

Dit document beschrijft de API-strategie van het DSO en is kaderstellend voor alle aanbieders van API's. Dit zijn alle API's die door stelselcomponenten worden aangeboden en API's van leveranciers van omgevingsinformatie (LvO's) die moeten voldoen aan de aansluitvoorwaarden voor informatieproducten. Met de voorliggende kaders wordt invulling gegeven aan een gemeenschappelijke strategie voor de realisatie van een ecosysteem waarin (open) API's centraal staan. Hiermee wordt op technisch koppelvlakniveau invulling gegeven aan het leidende DSO-principe 'Alles is een service'. Aangezien de API-strategie voor het DSO als geheel van belang is, is de API-strategie een bijlage bij de Overall Globale Architectuur Schets [4]. Bovenliggende kaders komen voort uit de Visie [1], het Globaal Programma van Eisen [2] en de Doelarchitectuur [3].

1.1 Doelgroep

Dit document richt zich op de Tactische beheersorganisatie (TBO), de Operationele beheerorganisaties (OBO's) en andere geïnteresseerden.

1.2 Doel en toepasbaarheid

Het primaire doel van dit document is het meegeven van ontwikkelkaders voor API's en het inrichten van een robuust ecosysteem.

☑	**Toepasbaarheid** Niet alle eisen zijn altijd van toepassing. Dit hangt namelijk sterk samen met de beoogde functionaliteit, ofwel met het "wat" van een API. De eisen zijn daarom primair gericht en van toepassing op het "hoe" van datgene een API aan middelen inzet en/of aanbiedt.

Indien de eisen van toepassing zijn geldt "pas toe of leg uit". Afwijken van de eisen kan alleen in overleg met en akkoord van de Stelsel Architectuur Board (SAB) van het DSO.

1.3 Resultaat

Het beoogde resultaat van een gemeenschappelijke strategie voor API's is een stelselbrede standaardisatie en uniformering van de manier waarop API's worden aangeboden. Met deze standaardisatie wordt de zogenaamde "Time To First Successful Call" (TTFSC)[1] verkort. Dit is van belang voor de landelijke voorzieningen, maar ook voor het succes van het Open Stelsel voor Derden (OSvD).

Een API is als een UI voor ontwikkelaars:
"Zoals een goede User eXperience (UX) essentieel is voor een UI, is een goede Developer eXperience (DX) essentieel voor een API."

[1] De tijd die een ontwikkelaar nodig heeft om de eerste keer met een aangeboden dienst of dataset van het DSO aan de slag te kunnen ofwel Time To First Successful Call (TTFSC) is hierbij cruciaal. Uitdaging voor het DSO is om API's aan te bieden met een lage TTFSC. Hiervoor wordt aansluiting gezocht bij defacto internet standaarden, aangevuld met in de geo-wereld gangbare standaarden.

1.4 Afkortingen en begrippen

Relevante afkortingen en begrippen zijn beschreven in Bijlage B: Afkortingen en begrippen.

1.5 Leeswijzer

Dit document dient in combinatie met de DSO URI-strategie [5] en de DSO Stelselafspraken [7] gelezen te worden, omdat deze documenten elkaar aanvullen. Om de API-strategie te begrijpen en op waarde te schatten is het belangrijk om te begrijpen vanuit welk paradigma deze is opgesteld. De basis wordt gevormd door een architectuurstijl die door Roy Thomas Fielding [8] in 2000 onder de naam REpresentational State Transfer (REST) is beschreven. Een korte introductievideo [9] is te vinden in Bijlage A: Bronnen.

Alle eisen die worden gesteld aan API's en het ecosysteem worden in dit document gedefinieerd en zijn verdeeld in een reeks modules:

Module	Omschrijving	Identificatie
Basis	Ondersteuning van alle basisaspecten voor RESTful API's inclusief beveiliging	API-Bxx
Ecosysteem	Het ecosysteem en de ondersteuning van "ecosystem awareness" voor API's	API-Exx
Identiteit	Ondersteuning van identiteitspropagatie t.b.v. autorisatie	API-Ixx
Hypermedia	Ondersteuning van hypermedia	API-Hxx
Query (projectie)	Ondersteuning van simpele projecties en/of query-taal-extensies	API-Qxx
Geo	Ondersteuning van geometrische data	API-Gxx
Tijdreizen	Ondersteuning van tijdreizen via parameters	API-Txx
API-profielen	Ondersteunen van en werken met API-profielen	API-Axx
Ontwerppatronen	Gebruik van standaard patronen die zijn vormgegeven op basis van RESTful API's	API-Oxx

Tabel 1 - Opdeling van eisen in modules

Door deze modulaire opzet is eenvoudiger te herkennen bij welke module of onderwerp een eis hoort. In Bijlage E: Overzicht eisen is snel te vinden welke eisen per module zijn gedefinieerd. Daarnaast is het mogelijk om bepaalde modules in z'n geheel in te wisselen voor een 'andere standaard', mocht daar op termijn een noodzaak voor bestaan.

Naast de bovengenoemde eisen zijn er ook basisprincipes, ontwerppatronen en "best practices" gedefinieerd. De zijn te vinden in Bijlage D: Overzicht principes, ontwerppatronen en "best practices".

Iedereen die wil weten wat het verschil is tussen V1.1 en V2.0 en/of welke eisen zijn komen te vervallen, wordt geadviseerd om Bijlage F: Migratie eisen V1.1 → V2.0 en Bijlage G: Uitfasering eisen (deprecated) te raadplegen.

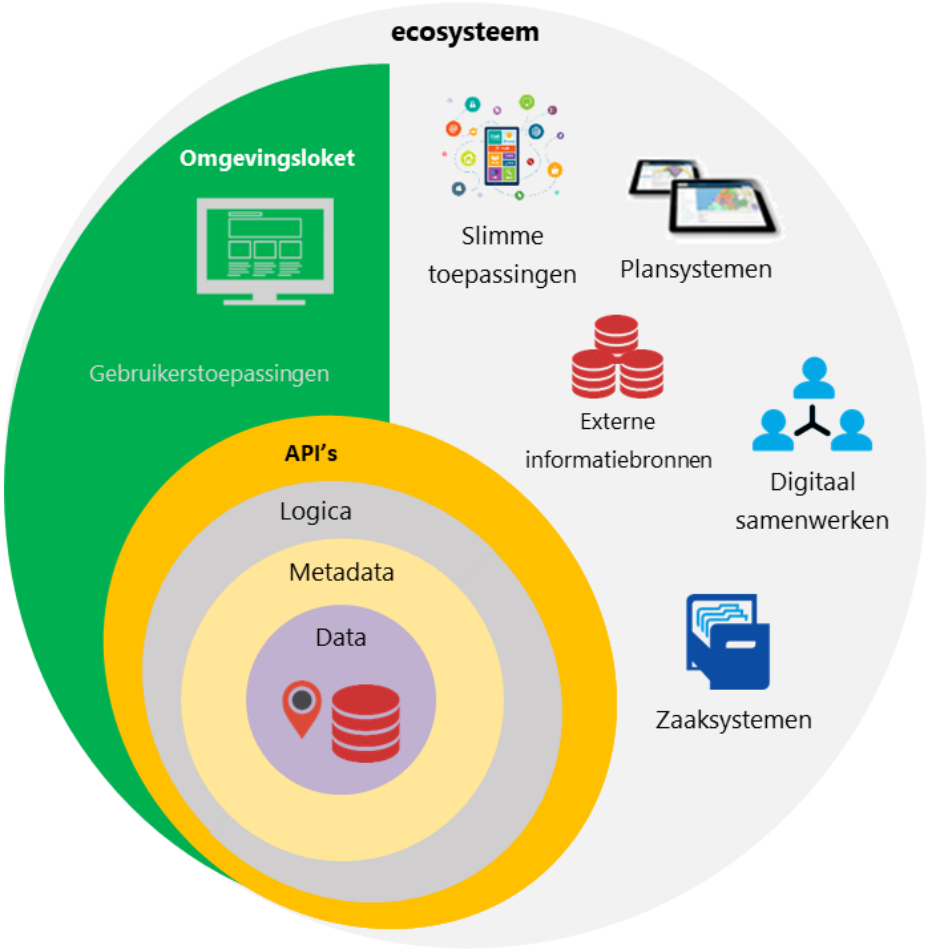
Tot slot, ook alle afbeeldingen, lijsten en voorbeelden in de document zijn in overzichtelijke lijsten opgenomen. Deze zijn te vinden in Bijlage C: Figuren, tabellen en voorbeelden.

2. API-strategie

In de Visie [1] van het Digitaal Stelsel Omgevingswet 2024 wordt het volgende gesteld:

> *Het DSO is een samenhangend stelsel van digitale voorzieningen. Het stelsel is gericht op het leveren van gebruikerstoepassingen en kwalitatief hoogwaardige gegevens en informatie die de processen van de Omgevingswet ondersteunen. Het staat in verbinding met haar omgeving; via gebruikerstoepassingen met eindgebruikers, met bronhouders (vaak bevoegd gezagen), met de Generieke Digitale Infrastructuur (GDI) en met andere afnemers (open koppelvlakken). Het stelsel is robuust wat betreft veilig en permanent gebruik. Het is open in het kunnen benutten van gegevens (open data) en koppelvlakken (web-services en API's) door eenieder.*

Dit document beschrijft de strategie waarmee de API's op een open en robuuste manier, onderdeel zijn van een ecosysteem en op een consistente en uniforme manier, binnen het stelsel, aan de keten en aan eenieder worden aangeboden.



Figuur 1 - Ecosysteem rondom API's

Het is goed om te benadrukken dat het niet primair om de ontwikkeling van individuele API's gaat, maar vooral ook om de inrichting van een ecosysteem dat is gebaseerd op de REST[2] architectuurstijl. Voor een goed werkend ecosysteem zijn ook de URI-strategie [5], de afspraken over tijdreizen [6] en talloze andere stelselafspraken [7] van belang.

[2] Een architectuurstijl die door Roy Thomas Fielding in 2000 onder de naam REpresentational State Transfer (REST) is geïntroduceerd.

2.1 Pas toe of leg uit

De API-strategie in dit document definieert een reeks "pas-toe-of-leg-uit" eisen. Dit betekent dat stelselcomponenten bij het publiceren van API's geacht worden zich te houden aan de API-strategie in dit document. Uiteraard is hierop ook het pas-toe-of-leg-uit principe van toepassing. Een voorbeeld van zo'n leg-uit is een API in een COTS[3]-pakket dat niet aangepast kan worden. Zie voor de toepasbaarheid ook paragraaf 1.2.

2.2 Basisprincipes

Er zijn verschillende ontwikkelpartners die samen het digitaal stelsel ontwikkelen. Het stelsel bestaat daarnaast uit een groot aantal stelselcomponenten. De functionaliteiten van deze stelselcomponenten worden beschikbaar gesteld middels services volgens de volgende basisprincipes:

DSO hanteert een 'API-first'-benadering

Alle services worden primair in de vorm van een REST-API beschikbaar gesteld (API-first benadering).

Alles is een service (API-first)

Alle functionaliteit in het stelsel is een service. Dit is een leidend principe om te zorgen dat functionaliteit slechts één keer wordt gerealiseerd en flexibel inzetbaar is. Om op technisch koppelvlakniveau invulling te geven aan het leidende DSO-principe 'Alles is een Service' worden er gedetailleerde standaarden voor REST-API's voorgeschreven.

De ontwikkelaars van gebruikerstoepassingen binnen het centrale Omgevingsloket, integreren functionaliteit van een groot aantal stelselcomponenten met behulp van API's. Om de integratie-inspanning zo laag mogelijk te houden dient de leercurve van de API's zo vlak mogelijk te zijn. Dit wordt bereikt door een goed API-ontwerp, herkenbaarheid over API's heen, het toepassen van internetstandaarden en het beschikbaar stellen van goede documentatie.

DSO kent geen aparte services voor derden

Er wordt geen onderscheid gemaakt tussen intern en extern gebruik van API's. Alle API's zijn in principe, conform het open stelsel gedachte, voor derden beschikbaar.

De onderstaande uitgangspunten zijn afgeleid van het principe in de Doelarchitectuur [3] dat zegt: "Nieuwe standaarden sluiten aan op bestaande standaarden" (APDSO13)", of ze komen uit de Overall Globale Architectuur Schets [4] en zijn leidend voor de API-strategie.

Services worden "as-is" beschikbaar gesteld

Hergebruik vereist ook dat er geen extra services worden ontwikkeld voor derden. Bestaande services worden "as-is" beschikbaar gesteld. Hierbij kunnen twee kanttekeningen worden gemaakt:

1. De ontsluiting van services via API's kan per beveiligingscontext anders zijn.
2. Niet alle interne services zijn ook als externe API ontsloten.

DISCLAIMER
Vanuit het oogpunt van beveiliging en privacy, is een service intern tenzij expliciet is aangegeven dat de service ook extern beschikbaar moet zijn.

[3] Commercial Off-The-Shelf Software

DSO hanteert 'intern = extern' (eat your own dogfood)

Het DSO hanteert het principe van "eat your own dogfood". Dit betekent dat alles in het stelsel een API is en dat deze API's door alle stelselonderdelen worden gebruikt. Er is geen andere manier van integratie toegestaan. Ook dit vereist dat alle API 's op een uniforme manier zijn opgezet, eenduidig werken en goed zijn gedocumenteerd.

Intern = extern (Eat your own dogfood)

Services werken voor interne en externe ontwikkelaars hetzelfde. Intern wordt gekozen voor REST-API's. De REST-API's zijn op een uniforme manier opgezet en goed gedocumenteerd. Deze documentatie is centraal online toegankelijk.

DISCLAIMER
Voor externe API's zullen eventueel aanvullende maatregelen worden getroffen om te zorgen dat alle eisen t.a.v. beveiliging en privacy adequaat kunnen worden afgedwongen.

DSO maakt gerichte uitzonderingen voor gegevensuitwisseling met rechtsgevolgen

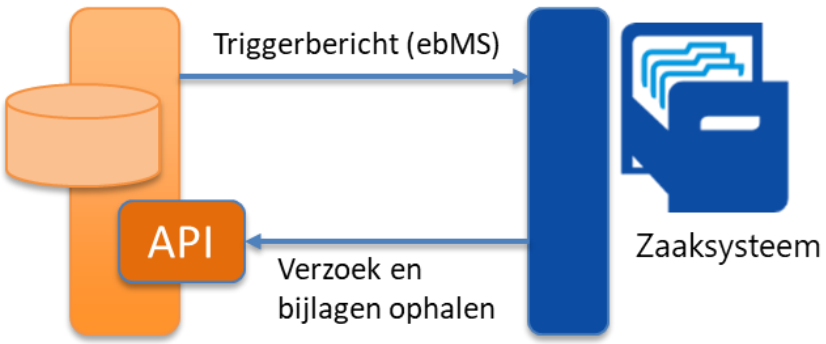
Voor gegevensuitwisseling met rechtsgevolgen worden zogenaamde "formele koppelvlakken" ingericht. Ieder van deze koppelvlakken maakt gebruik van een apart Digikoppeling contract.

Formele koppelvlakken werken met Digikoppeling

Extern gerichte formele koppelvlakken voldoen aan Digikoppeling. Deze koppelvlakken worden op de perimeter (grens) van het Stelselknooppunt vertaald van API's naar Digikoppeling. Stelselcomponenten hebben zelf geen Digikoppeling kennis en expertise.

DEFINITIE
Formele koppelvlakken worden gebruikt voor het overdragen van gegevens of aanroepen van functionaliteit (meestal tussen overheden) die rechtsgevolgen hebben. Bijvoorbeeld het indienen van een vergunningaanvraag of melding vanuit DSO-LV naar een bevoegd gezag.

Omdat ook in de context van formele koppelvlakken API's kunnen bijdragen aan meer flexibiliteit, wordt hier met API's gericht op ingezet. Bijvoorbeeld door een gerichte scheiding te maken tussen de overdracht van metadata en data. Een goed voorbeeld van deze 'hybride' aanpak is een asynchroon triggerbericht voor de overdracht van een formeel verzoek (metadata van een aanvraag of melding) in combinatie met een API voor het ophalen van de volledige inhoud van een verzoek (data van een aanvraag of melding).



Figuur 2 - Hybride opzet formeel koppelvlak

DSO fungeert als een ecosysteem met data, metadata en API's

Het DSO wordt ontwikkeld als een "open stelsel". Een ecosysteem waarin de overheid verantwoordelijk is voor de wettelijk bepaalde functionaliteit van het stelsel. De volledige functionaliteit en data van het stelsel wordt binnen een ecosysteem met data, metadata en API's aan derden beschikbaar gesteld. Dit noemen we het "open stelsel" principe. Hierdoor kunnen derden aanvullende toepassingen ontwikkelen binnen één en hetzelfde ecosysteem. De data, metadata en API's moeten dus toegankelijk genoeg zijn om een brede ontwikkel-community aan te spreken (ontwikkelvriendelijk zijn). Zo'n community is nodig om de functionaliteit van het DSO te verbreden en de ontwikkellast breder te delen. Welke ontwikkelaars met de API's en data zullen werken is onbekend. Deze groep 'onbekende ontwikkelaars' moet toch door het DSO op een goede manier worden bediend. Dit wordt gerealiseerd door zoveel mogelijk aan te sluiten op de internetstandaarden waarmee deze ontwikkelaars gewend zijn te werken.

DSO API's werken conform de stelselafspraken

Technisch gezien wordt het DSO vormgegeven met een stelsel van afspraken rondom lokale en landelijke voorzieningen. Zo'n samenspel vereist een hoge mate van interoperabiliteit, ofwel afspraken op het niveau van transport, logistiek en inhoud. Daarom zijn de API- en URI-strategie onderdeel van een groot aantal stelselbrede afspraken. Kortom, DSO API's werken conform de Stelselafspraken [7].

DSO ontwerpt API's voor een optimale Developer eXperience (DX)

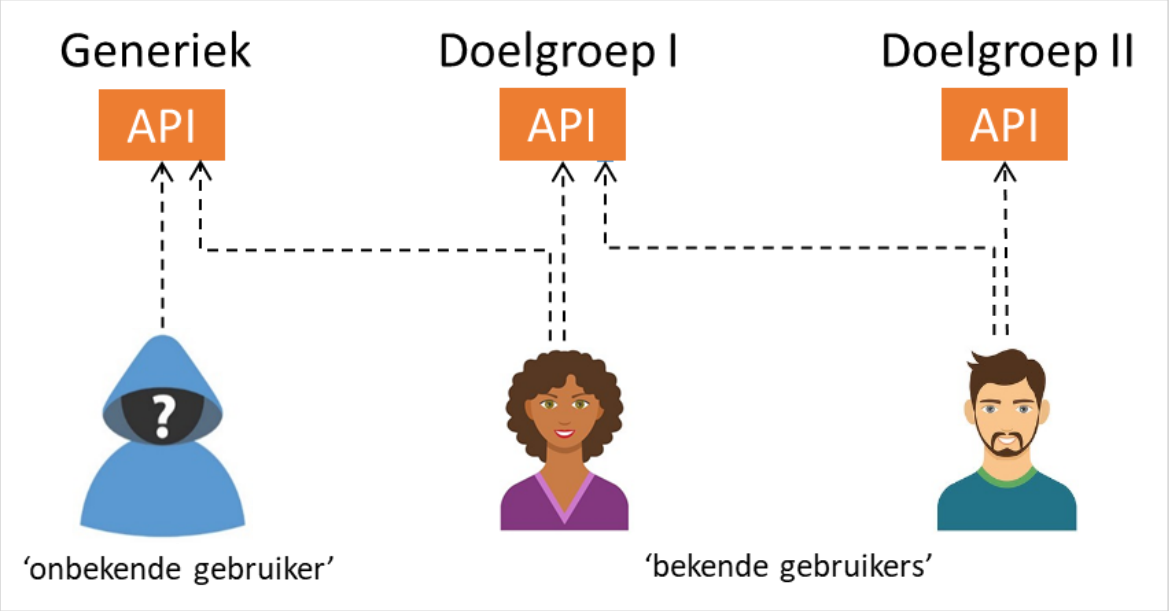
Om ontwikkelaars goed te faciliteren is het cruciaal dat zij geen onnodige blokkades ondervinden om op basis van de API's van het stelsel te ontwikkelen. Een zogenaamde optimale "Developer eXperience" (DX) is dus vereist. Dit wordt onder andere bereikt door goede API-ontwerpen, het toepassen van semantische standaarden, herkenbaarheid over API's heen en werkende voorbeelden in de documentatie. De volgende ontwerpaspecten spelen ook een belangrijke rol:

1) Een API maakt gebruik van internetstandaarden waar dit mogelijk is;
2) Een API is gebruiksvriendelijk voor ontwikkelaars en kan via de browser worden verkend;
3) Een API is eenvoudig, intuïtief en consistent in gebruik waardoor het gebruik niet alleen gemakkelijk is maar ook aangenaam;
4) Een API is kanaalonafhankelijk en voldoende flexibel om verschillende gebruiksscenario's te ondersteunen;
5) Een API is efficiënt, dit is in evenwicht met alle voorgaande aspecten;
6) Een API heeft een duidelijke positionering die samen met voldoende functionele documentatie te vinden is in een centraal API-register.

Dit alles geldt overigens, zoals eerder in dit document benoemd, ook voor de ontwikkelaars van de landelijke voorzieningen van het digitaal stelsel.

2.3 Ontwerpstrategie

In het algemeen kan men ervan uitgaan dat API's onafhankelijk van de toepassing kunnen worden ontworpen en gebouwd. Een API is een product op zichzelf. Een API moet in principe meerdere kanalen kunnen bedienen en niet één specifiek kanaal. Dit wil niet zeggen dat API's los van de werkelijkheid kunnen worden ontwikkeld. Er moet altijd goed worden afgestemd met 'bekende afnemers' om te komen tot bruikbare diensten. Wanneer er sprake is van verschillende doelgroepen, is het een prima strategie om een generieke API aan te bieden voor 'onbekende afnemers' en één of meer specifieke API's voor andere doelgroepen met 'bekende afnemers'.



Figuur 3 - Visualisatie van API's die zijn afgestemd op verschillende doelgroepen

API-B01 ***API's zijn altijd afgestemd op één of meer doelgroepen***

Wanneer er sprake is van verschillende doelgroepen, is het een prima strategie om een generieke API aan te bieden voor 'onbekende afnemers' en één of meer specifieke API's voor andere doelgroepen met 'bekende afnemers'. Indien er een hiërarchie ontstaat, valt deze ook onder het generieke beleid voor versionering (zie pararaaf 2.5.14) voor API's en is er geen sprake van uitzonderingen voor doelgroepen.

Een koppelvlak is altijd een combinatie van één of meer API's, up-to-date documentatie met werkende voorbeelden en andere ondersteunende hulpmiddelen.

2.4 Legacy en COTS API's

Bijzondere categorieën in relatie tot de API-strategie zijn bestaande API's, maar ook leverancier- en/of productspecifieke API's in "Commercial Off The Shelf" (COTS) Software. Het doel van de API-strategie is om stelselbreed een uniforme API-ervaring te bieden. Wat betekent dit dan voor bestaande API's en productspecifieke API's?

Bij bestaande API's dient altijd een kosten-baten afweging gemaakt te worden voor de inspanning die nodig is om aan de API-strategie te voldoen. Hierbij zijn in ieder geval drie strategieën mogelijk: 1) onderzoeken of de bestaande API (en wellicht zelfs de applicatie) nog wel nodig is; 2) opnieuw bouwen; 3) of een façade-API ontwikkelen.

API-B02 ***Bestaande API's voldoen waar mogelijk ook aan de API-strategie***

Er wordt een kosten baten afweging gemaakt of bestaande API's noodzakelijk zijn en eventueel dienen te worden omgezet om te voldoen aan de API-strategie.

Product- en/of leverancierspecifieke API's zijn een categorie op zichzelf. Dit zijn meestal API's die onderdeel zijn van een product en daarmee niet implementatieneutraal zijn. Het koppelvlak is dus product- of leverancierspecifiek. Dit soort API's dient binnen het stelsel nooit direct aan afnemers aangeboden te worden, maar altijd via een façade-API. Alleen dan is het mogelijk om een onderliggend systeem en/of bijbehorende technische API's te vervangen met geen of minimale impact voor de afnemers.

API-B03 ***Product- en leverancierspecifieke API's worden nooit direct aangeboden***

Product- en leverancierspecifieke API's worden nooit direct aangeboden, want alleen dan is het mogelijk om een onderliggend systeem en/of bijbehorende technische API's te vervangen met geen of minimale impact voor de afnemers.

ⓘ *Dit geldt niet voor producten die een op open standaarden gebaseerde API aanbieden.*

2.5 API-standaardisatie

Om stelselbreed op een effectieve manier te komen tot consistente en uniforme API's, is het noodzakelijk om minimaal de volgende aspecten te standaardiseren:

1. Gebruik van JSON;
2. Beveiliging (versleutelingen en toegang);
3. Identiteitspropagatie (authenticatie en autorisatie);
4. RESTful principes (inclusief HATEOAS-constraint[4] en HAL[5]);
5. Filteren, sorteren en zoeken;
6. Paginering;
7. Query-projectie;
8. GEO-ondersteuning;
9. Tijdreizen;
10. API-profielen;
11. Caching;
12. Ecosysteem en ecosysteem-bewustzijn;
13. Documentatie;
14. Versionering;
15. Ontwerppatronen;
16. Gecontroleerde degradatie en
17. Foutafhandeling (inclusief statuscodes en standaard foutmeldingen).

[4] Hypermedia As The Engine Of Application State (HATEOAS), is een principe binnen de REST applicatiearchitectuur. Het principe beoogt dat een client met een netwerkapplicatie interacteert op basis van hypermedia die dynamisch wordt geleverd door de server.
[5] Hypermedia Application Language

2.5.1 Gebruik van JSON

JavaScript Object Notation (JSON) is een formaat, net zoals XML, om gegevens te serialiseren, op te slaan en te versturen. JSON is het primaire representatieformaat voor API's. In tegenstelling tot XML kent JSON een compacte notatie, bijvoorbeeld:

```
{
  "persoon": {
    "naam": "Jan",
    "geboortejaar": 1983
  }
}
```

Voorbeeld 1 - Een object "persoon" in de JavaScript Object Notation (JSON)

API-B04 ***JSON-first: API's ontvangen en versturen JSON-objecten***

API's ontvangen en versturen JSON. Bovendien worden alleen JSON-objecten toegepast, dus geen (naamloze) arrays of primitieve datatypes als "top-level" element.

ⓘ *Het gebruik van alleen JSON-objecten als "top-level" element vergroot de uitbreidbaarheid.*

JUIST	ONJUIST
`{ "status": "OK" }`	`"OK"`
`{ "foutcode": 1024 }`	`1024`
`{ "succesvol": false }`	`false`
`{ "berichten": [ { "bericht": "Ontvangen" }, { "bericht": "Verwerkt" } ] }`	`[ "Ontvangen", "Verwerkt" ]`

Voorbeeld 2 - Een verzoek of antwoord met en zonder "top-level" object

API-B05 ***API's zijn optioneel voorzien van een JSON Schema***

API's ondersteunen JSON Schema (http://json-schema.org), zodat validatie (optioneel) mogelijk is en vereenvoudigd wordt. In schema's wordt alleen met gestandaardiseerde datatypen gewerkt (zie best practice - 01). Schema's worden beschikbaar gesteld via de Open API Specification door gebruik te maken de mogelijkheid om te verwijzen naar alternate schema's.

ⓘ *Dit is zinvol voor statische validatie t.b.v. acceptatie van data. Bijvoorbeeld in een keten waarin pre-validatie noodzakelijk is.*

API-B06 ***Content negotiation wordt volledig ondersteund***

Andere representaties zoals XML en RDF worden naast JSON via het standaard HTTP content negotiation mechanisme ondersteund. Als het gewenste formaat niet geleverd kan worden zal er een 406 Not Acceptable worden teruggegeven. De server moet in dit geval ook een fout-respons genereren met een lijst van beschikbare representaties en de bijbehorende resource-ID's waaruit de gebruiker of user-agent kan kiezen:

```
Content-Type: application/problem+json
{ // standaard fout-respons zoals gedefinieerd in [RFC7807] -> Zie 2.5.17
  "status": 406, → , → ,
  "acceptable": [ "application/json", "application/pdf", "application/rdf+xml" ]
}
```

API-B07 ***API's controleren voor verzoeken met een "payload" dat de Content-Type header is ingesteld***

Er wordt voor verzoeken met een "payload" gecontroleerd of de Content-Type header is ingesteld op `application/json` of andere ondersteunde content types, anders wordt HTTP-status code `415 Unsupported Media Type` geretourneerd.

Welke datum- en tijdnotitie wordt gehanteerd?

JSON schrijft geen standaardformaat of notatie voor. In de API- en URI-strategie [5] is echter afgesproken dat voor tijdreizen in API's met query-parameters [RFC3339] wordt gevolgd. Dezelfde lijn wordt daarom ook gevolgd voor de uitwisseling van datum en tijd in JSON. De IETF-standaard RFC3339 is het internetprofiel van de ISO 8601[6] standaard en wordt ook door de Open API Specification voorgeschreven.

Voorbeelden van datum- en tijdnotaties (conform RFC3339) in JSON:

```
{
  "verzoek": {
    "tijdstempel": "2019-04-12T23:20:50.52Z",
    "datumGrondslag": "2021-01-01"
  }
}
```

Voorbeeld 3 - JSON-verzoek of respons met datum en tijdstempels

De JSON-properties in Voorbeeld 3 representeren het volgende:

- **`tijdstempel`**: 20 minuten en 50.52 seconden na het 23e uur van 12 april 2019 in UTC (in het Nederlands ook aangeduid als gecoördineerde wereldtijd);
- **`datumGrondslag`**: 1 januari 2021.

BEST PRACTICE - 01 ***Gebruik gestandaardiseerde datatypen in JSON-objecten***

Naast het gebruik van standaard uitwisselformaat voor datum- en tijdvelden is het zeer aan te bevelen om voor andere type velden en formaten gebruik te maken van de volgende gestandaardiseerde datatypen:

Type	Formaat	Standaard	Voorbeeld
integer	int32		7721071004
integer	int64		772107100456824
integer	bigint		77210710045682438959
number	float	IEEE754-2008	3.1415927
number	double	IEEE754-2008	3.141592653589793
number	decimal		3.14159265358979323846264338327 9
string	bcp47	BCP47 (taal)	"nl-NL" (taal en taalvariant)
string	byte	RFC7493	"dGVzdA=="
string	date	RFC3339	"2019-07-30"
string	date-time	RFC3339	"2019-07-30T06:43:40.252Z"
string	email	RFC5322	"dso-lv@omgevingswet.overheid.nl"
string	gtin-13	GTIN (barcode)	"5710798389878"
string	hostname	RFC1034	"pre.omgevingswet.overheid.nl"
string	ipv4	RFC2673	"104.75.173.179"
string	ipv6	RFC2673	"2600:1401:2::8a"
string	iso-3166	ISO3166-1 alpha-2	"NL" (landcode)
string	iso-4217	ISO4217	"EUR" (munteenheid)
string	iso-639	ISO639-1	"nl" (taalcode)
string	json-pointer	RFC6901	"/verzoeken/0/id"
string	password		"geheim"
string	regex	ECMA262	"^[a-z0-9]+$"
string	time	RFC3339	"06:43:40.252Z"
string	uri	RFC3986	"https://www.overheid.nl/"
string	uri-template	RFC6570	"/documenten/{id}"
string	uuid	RFC4122	"550e8400-e29b-41d4-a716-…"

ⓘ *Het gebruik van gestandaardiseerde datatypen vergroot de interoperabiliteit.*

[6] ISO 8601 (Information interchange – Representation of dates and times).

API-B08 ***Voor de uitwisseling van datum en tijd in JSON wordt de RFC3339 gevolgd***

Voor de uitwisseling van datum en tijd in JSON wordt de [RFC3339] gevolgd. Deze IETF-standaard (RFC3339) is het internetprofiel van de ISO 8601 standaard.

Veldnamen in snake_case, camelCase, UpperCamelCase of koppelteken?
Bij veldnamen wordt gebruik gemaakt van camelCase. Zie ook de URI-strategie [5] voor alle naamconventies in URL's, URI's en URN's.

API-B09 ***Veldnamen in camelCase en enumeraties in UPPER_SNAKE_CASE***

Een veldnaam begint met kleine letters (eerste woord) en ieder opvolgend woord begint met een hoofdletter. Veldnamen zijn beperkt tot de alfanumerieke reeks, waarbij gereserveerde namen beginnen met een underscore (zoals: `_links`). Waarden van enumeraties worden met hoofdletters gedefinieerd, bijvoorbeeld: `WAARDE` of `EEN_ANDERE_WAARDE`.

ⓘ *Met deze benadering kunnen gereserveerde namen en waarden van enumeraties duidelijk worden onderscheiden van velden of andere elementen.*

Sommige API's wikkelen antwoorden niet in een object en retourneren primitieven of arrays die alleen in een specifieke context nog betekenis hebben. Dit is in Voorbeeld 4 aan de rechterkant weergegeven. Gebruik alleen betekenisvolle objecten (links), want dat maakt de context duidelijk en bovendien bevordert het de uitbreidbaarheid.

```
// object met eigenschap en waarde
{
  "geboortejaar": 1983
}
```

```
// betekenisloze waarde
1983
```

```
// betekenisvolle lijst van objecten (array)
{
  "personen": [
    {
      "naam": "Jan",
      "geboortejaar": 1983
    },
    {
      "naam": "Piet",
      "geboortejaar": 1981
    }
  ]
}
```

```
// betekenisloze lijst van objecten (array)
[
  {
    "naam": "Jan",
    "geboortejaar": 1983
  },
  {
    "naam": "Piet",
    "geboortejaar": 1981
  }
]
```

Voorbeeld 4 - JSON-responses met en zonder betekenisvolle "enveloppen"

Een toekomstbestendige API gebruikt alleen betekenisvolle enveloppen.

API-B10 ***Een JSON-response heeft alleen betekenisvolle enveloppen***

In een JSON-respons worden alleen betekenisvolle enveloppen toegepast:

- Een omhullende envelop heeft altijd een naam die feitelijk fungeert als namespace voor objecten en arrays,
- Omdat arrays meerdere waarden bevatten wordt de naam altijd in meervoudsvorm gedefinieerd.
- Objectnamen krijgen tot slot een naam in enkelvoud.

ⓘ *Met betekenisvolle enveloppen voor objecten en arrays wordt de uitbreidbaarheid vergroot en is de API toekomstbestendiger.*

Wel of niet werken met "pretty print"?
De meeste REST-clients en browsers (al dan niet met extensies) kunnen JSON netjes geformatteerd weergeven, ook als de response geen "white-spaces" bevat.

API-B11 ***Pretty print is standaard uitgeschakeld***

Het uitgangspunt is dat REST clients en browsers (al dan niet met extensies) JSON netjes geformatteerd kunnen weergeven.

2.5.2 Beveiliging

API's zijn vanaf elke locatie vanaf het internet te benaderen. Om uitgewisselde informatie af te schermen wordt altijd gebruik gemaakt van een versleutelde verbinding op basis van TLS. Geen uitzonderingen, dus overal en altijd. Doordat de verbinding altijd versleuteld is, maakt dat het authenticatiemechanisme eenvoudiger. Hierdoor wordt het mogelijk om eenvoudige toegangstokens te gebruiken in plaats van toegangstokens met encryptie.

API-B12 ***De verbinding is ALTIJD versleuteld met minimaal TLS V1.2***

De verbinding is ALTIJD versleuteld op basis van minimaal TLS V1.2. Geen uitzonderingen, dus overal en altijd. In het geval van toegangsbeperking of doelbinding wordt tweezijdig TLS toegepast.

API-B13 ***API's zijn alleen bruikbaar met een geldige API-key***

Voor alle DSO API's wordt minimaal een registratie inclusief acceptatie van de fair use voorwaarden vereist. Op basis hiervan zal dan een API-key worden uitgegeven.

Autorisatiefouten
In een productieomgeving is het wenselijk om voor het (kunnen) autoriseren zo min mogelijk informatie weg te geven.

BEST PRACTICE - 02 ***Gebruik beslistabel voor toepassen statuscodes: 401, 403 en 404***

Beslistabel
Om te zorgen dat voor het (kunnen) autoriseren zo min mogelijk informatie wordt weggegeven, is het advies om voor statuscode `401 Unauthorized`, `403 Forbidden` en `404 Not Found`, de volgende regels te hanteren:

Geauthentiseerd?	Geautoriseerd?	Bestaat de resource?	HTTP-statuscode
✔	✔	✔	`20x (200 OK)`
✔	✖	?	`403 Forbidden`
✖	?	?	`401 Unauthorized`
✔	✔	✖	`404 Not Found`
✔	✖	?	`403 Forbidden`
✖	?	?	`401 Unauthorized`

Tabel 2 - Regels voor bepalen van statuscode bij autorisatie

Het idee van deze regels is dat eerst wordt bepaald of de aanroeper (principal) gerechtigd is voor een resource. Is het antwoord 'nee' of kan dat niet worden bepaald, bijvoorbeeld omdat de resource nodig is om deze beslissing te kunnen nemen en de resource niet bestaat, dan wordt 403 Forbidden teruggegeven. Op deze manier wordt geen informatie teruggegeven over het al dan niet bestaan van een resource aan een niet-geautoriseerde principal.

ⓘ *Een strategie waarmee eerst wordt bepaald of er toegang is, biedt meer ruimte om de 'access control logic' te scheiden van de 'business logic'.*

Openbare identifiers
Openbaar zichtbare identifiers (ID's), zoals die veelal in URI's van RESTful API's voorkomen, zouden onderliggende mechanismen (zoals een nummergenerator) niet bloot moeten leggen en zeker geen zakelijke betekenis moeten hebben.

BEST PRACTICE - 03 ***Gebruik Universally-Unique IDentifier (UUID's) voor de identificatie van resources met een vertrouwelijk karakter***

UUID's
Het wordt aanbevolen om voor resources die een vertrouwelijk karakter hebben het concept van een UUID (Universally-Unique IDentifier) te gebruiken. Dit is een 16-bytes (128-bits) binaire representatie, weergegeven als een reeks van 32 hexadecimale cijfers, in vijf groepen gescheiden door koppeltekens en in totaal 36 tekens (32 alfanumerieke tekens plus vier afbreekstreepjes):

```
550e8400-e29b-41d4-a716-446655440000
```

Om te zorgen dat de UUID's hanteerbaarder zijn, kan een base64-gecodeerde variant van 22 tekens worden gebruikt. De bovenstaande UUID ziet er dan als volgt uit:

```
abcdEFh4520juieUKHWgJQ
```

Blootstellen API-key
De API-key's die standaard worden uitgegeven zijn "unrestricted". Dat wil zeggen dat er geen gebruiksbeperkingen op zitten en ze niet blootgesteld mogen worden via een webapplicatie. Door API-key's zonder gebruiksbeperkingen toe te passen in JavaScript, is er een reële kans op misbruik en quotum-diefstal. Om quotum-diefstal te voorkomen dient voor dergelijke toepassingen een zogenaamde "restricted" API-key's te worden uitgegeven en gebruikt.

BEST PRACTICE - 04 ***Gebruik "restricted" API-key's in publieke clients***

Restricted API-key's
In JavaScript en andere publieke apps dient alleen gebruik te worden gemaakt van zogenaamde "restricted" API-key's. Dit zijn API-key's die zijn gekoppeld aan specifieke kenmerken van de aanroeper (webapplicatie, mobiele app, domein), waaronder een clientId, 'referer-URL' en/of IP-adressen. Dit zijn helaas geen waterdichte oplossingen, maar het wordt aanzienlijk lastiger om de API-key in willekeurige web-toepassingen te misbruiken. Het is daarom ook noodzakelijk om:

- De scope van API-key's te beperken (gebruik per applicatie onafhankelijke API-key's);
- Niet-gebruikte API-key's zo snel als mogelijk te verwijderen;
- Het gebruik van de API's te bewaken en alle API-key's die afwijkend of ongeoorloofd gedrag vertonen te blokkeren.

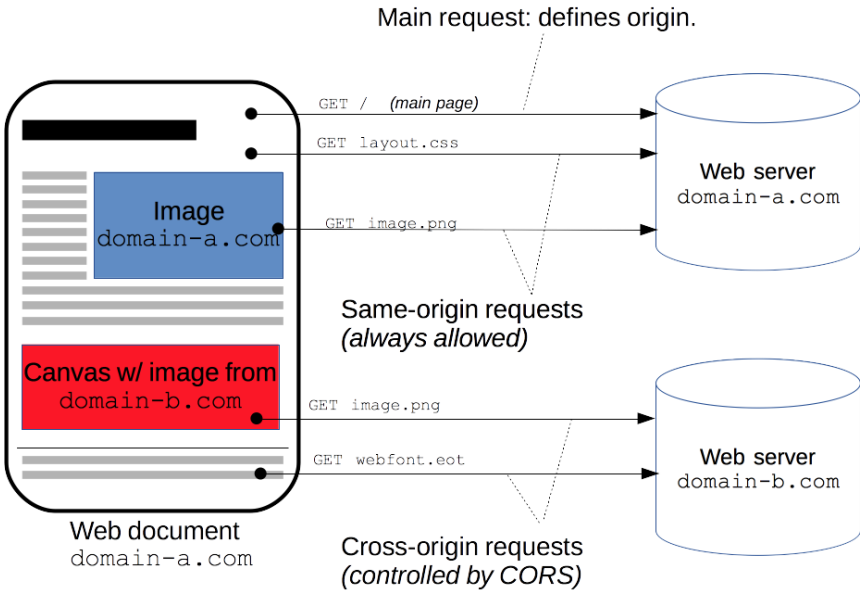
Gebruik van 'restricted API-keys' is nooit een oplossing om de toegang te beperken. Hiervoor dient gebruik te worden gemaakt van API's met toegangsbeperking via het Knooppunt en/of authenticatie en autorisatie zoals beschreven in paragraaf 2.5.2 en 2.5.3.

API-E01 ***API's kunnen via het Knooppunt van DSO-LV werken met "restricted" API-key's***

Het Knooppunt van DSO-LV maakt het mogelijk om een API-key uit te geven en deze te koppelen aan specifieke kenmerken van de aanroeper (webapplicatie, mobiele app, domein), waaronder een client-id, 'referer-URL' en/of IP-adressen. Daarnaast biedt het Knooppunt alle faciliteiten die nodig zijn voor het beheren, bewaken en blokkeren van API-key's.

Cross Origin Resource Sharing (CORS)-policy
Volgens de definitie maakt een client een HTTP-verzoek met een andere oorsprong (cross-origin) wanneer het een resource aanvraagt uit een ander domein of poort dan het domein dat de eerste resource leverde. Webbrowsers implementeren een

zogenaamde "same origin policy", een belangrijk beveiligingsconcept om te voorkomen dat verzoeken ongemerkt naar een ander domein kunnen gaan. Hoewel dit beleid effectief is in het voorkomen[7] van aanroepen in verschillende domeinen, voorkomt het potentieel ook alle legitieme interactie tussen API's en clients van een bekende en vertrouwde oorsprong.



Figuur 4 - Illustratie van Cross-Origin Resource Sharing (CORS) in een webbrowser

Het CORS-mechanisme ondersteunt verzoeken met een andere oorsprong inclusief de gegevensoverdracht tussen browsers en servers. Moderne browsers gebruiken CORS-headers om de risico's van cross-origin HTTP-verzoeken naar API's te beperken.

BEST PRACTICE - 05 ***Gebruik CORS-headers en controleer domein-toegang op basis van een whitelist***

CORS-policy
Controleer eerst het domein van de inkomende aanvraag en genereer de response-header afhankelijk van het gegeven of dit domein verzoeken mag verzenden of niet (whitelist). Alleen in dat geval wordt voor dit specifieke domein aan de respons-header Access-Control-Allow-Origin toegevoegd. Maak daarnaast gebruik van de W3C aanbevelingen: https://www.w3.org/TR/cors/.

ⓘ *De `Access-Control-Allow-Origin` dient te worden teruggegeven door de API. Als dit in de browser niet overeenkomt met de `Origin-header` en geen `*` is, zal de browser het verzoek weigeren.*

CORS is een versoepeling van de zogenaamde "same origin policy", maar probeert dit op een veilige manier te ondersteunen. Met een wildcard (`*`) worden de meeste beveiligingsregels van CORS uitgeschakeld. Er zijn gevallen waarin wildcard oké is, zoals een open API die in veel websites van derden kan worden geïntegreerd. Maar in veel gevallen vormt het een beveiligingsrisico. In plaats daarvan dient te worden gewerkt met een lijst van vertrouwde domeinen (whitelist).

API-B14 ***API's gebruiken geen wildcard in CORS-header***

API's moeten het retourneren van wildcards (`*`) in de `Access-Control-Allow-Origin` in de response-header ten alle tijden vermijden.

[7] Het is bekend dat CORS kan leiden tot problemen door hybride en multilevel-aanvallen met combinaties zoals XSS (Cross-Site Scripting) and CSRF (Cross-Site Request Forgery). Met de juiste CORS-policy kunnen dit soort kwaadaardige aanvallen worden voorkomen.

De API op de server heeft controle over het al dan niet toestaan van het verzoek, afhankelijk van de oorsprong van het verzoek. De browser garandeert dat de inhoud van de `Origin-Request` header betrouwbaar is.

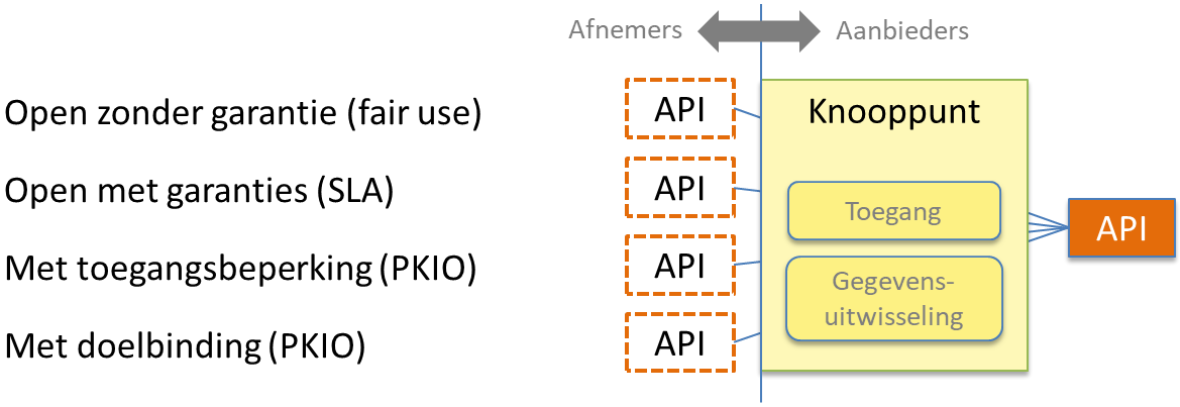
> *API-B15* ***API's die een CORS-policy implementeren gebruiken een whitelist***
>
> API's retourneren alleen het specifieke domein in de `Access-Control-Allow-Origin` response-header op basis van een whitelist.

API's kunnen uiteraard individueel een CORS-policy implementeren, maar het is beter om dit eenduidig te doen en dit centraal te faciliteren en beheren.

> *API-E02* ***Het Knooppunt geeft CORS-headers desgewenst transparant door***
>
> Het Knooppunt van DSO-LV maakt het mogelijk om te configureren dat alle CORS-headers, die door een API worden geretourneerd, transparant wordt doorgegeven naar de aanroeper.

> *API-E03* ***API's kunnen de CORS-policy ook realiseren via een whitelist van het Knooppunt***
>
> Het Knooppunt van DSO-LV maakt het mogelijk om een whitelist te beheren van domeinen die verzoeken mogen verzenden in de context van de betrokken API. Het Knooppunt retourneert alleen het specifieke domein in de `Access-Control-Allow-Origin` response-header, wanneer het domein in de `Origin-Request` header in de whitelist (aka allowed origin-list) voorkomt.

Het Knooppunt kan API's die vanuit DSO-LV worden aangeboden virtualiseren. Daarmee kan op een eenvoudige en veilige manier, dezelfde API met verschillende toegangsniveaus worden aangeboden. Het Knooppunt kan in het geval van fair-use, ook de daarvoor geldende quota afdwingen.



Figuur 5 - Virtualisatie van API's door het Knooppunt van DSO-LV

> *API-E04* ***Het Knooppunt voorziet in het virtualiseren van API's met verschillende toegangsniveaus***
>
> Het Knooppunt van DSO-LV maakt het mogelijk om een API te virtualiseren en dezelfde API's met vier verschillende toegangsniveaus aan te bieden:
>
> 1. Open zonder garanties (fair-use)
> 2. Open met garanties (SLA)
> 3. Gesloten met toegangsbeperking (beveiligd met dubbelzijdig TLS op basis van PKIO)
> 4. Gesloten met doelbinding (beveiligd met dubbelzijdig TLS op basis van PKIO)
>
> ⓘ *De API is zelf (aan de achterkant) via dubbelzijdig TLS verbonden met het Knooppunt.*

2.5.3 Authenticatie, propagatie van identiteit en autorisatie

Authenticatie en autorisatie
Een REST API mag geen toestand (state) bijhouden. Dit betekent dat authenticatie en autorisatie van een verzoek niet mag afhangen van cookies of sessies. In plaats daarvan wordt elk verzoek voorzien van één of meer tokens. Binnen het DSO is gekozen voor OAuth 2.0 en OpenID Connect als de standaarden voor het autorisatiemechanisme.

API-I01 ***Autorisatie is gebaseerd op OAuth 2.0***

Een REST-API mag geen state hebben. Elk verzoek voor de toegang tot resources die autorisatie vereisten, moet daarom minimaal zijn voorzien van een access-token. OAuth 2.0 is hiervoor de voorgeschreven standaard.

API-E05 ***API's kunnen via het Knooppunt van DSO-LV gebruik maken van OAuth 2.0 in combinatie met OpenID Connect***

Het Knooppunt (onderdeel Toegang) van DSO-LV speelt een sleutelrol bij het gebruik van OAuth 2.0 en OpenID Connect. Waaronder de dienst waarmee tokens kunnen worden uitgegeven. Afhankelijk van de scope van het verzoek en de rechten van de gebruiker levert het Knooppunt een access-token of een combinatie van een ID-token en een access-token.

Bij het gebruik van access-tokens wordt onderscheid gemaakt tussen geauthentiseerde en niet-geauthentiseerde services met de bijhorende headers:

	HTTP-header
Geauthentiseerd	`Authorization: Bearer <token>`
Niet-geauthentiseerd	`X-Api-Key: <api-key>`

Tabel 3 - Definitie van access-tokens

API's moeten de aanwezigheid en de inhoud van de genoemde headers contoleren.

API-I02 ***API's controleren altijd of de juiste headers met authenticatiedetails beschikbaar zijn***

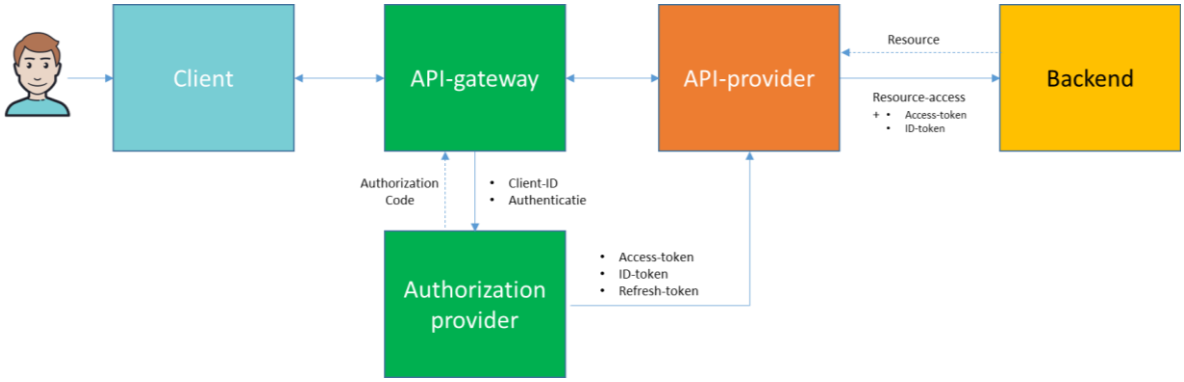
Bij het ontbreken van de juiste headers zijn geen authenticatiedetails beschikbaar en dient de statuscode `401 Unauthorized` terug te worden gegeven. Zie ook best practice - 02.

API-I03 ***Authenticatie voor API's met toegangsbeperking of doelbinding is gebaseerd op PKIO***

In het geval van API's met toegangsbeperking of doelbinding zal er aanvullend sprake zijn van authenticatie op basis PKI-overheid certificaten (PKIO) ofwel tweezijdig TLS. In de URI-strategie [5] is vastgelegd welke kaders gelden voor URI's van API's die tweezijdig TLS gebruiken.

Identiteit propageren in de keten
Voor het end-to-end propageren (doorgeven in de keten) van (pseudo)identiteiten wordt aanvullend gebruik gemaakt van ID-tokens. Wanneer er sprake is van een ingelogde eindgebruiker wordt daarom een ID-token toegevoegd, maar tegelijk ook een nieuw access-token aangemaakt en toegevoegd. Deze tokens zijn met elkaar verbonden.

Het uitgangspunt bij 'identity propagation' is telkens dat een externe gebruiker ingelogd is of wordt ingelogd met eHerkenning, DigiD of een ander standaard beschikbaar authenticatiemiddel.



Figuur 6 - Globale context en token-uitwisseling bij identity-propagation

De API-gateway maakt gebruik van OpenID Connect om de identiteit van een gebruiker door (of terug) te geven in een ID-token. Dit token bevat dus informatie over de eindgebruiker in de vorm van een pseudo-identiteit. Deze informatie is echter zonder toegang tot de identity server niet terug te leiden tot persoonsgegevens. Het is informatie die alleen binnen de stelselcontext een eindgebruiker identificeert.

API-I04 ***API's maken gebruik van identiteitpropogatie voor resources die fijnmazige autorisatie vereisen***

Eindgebruikers moeten identificeerbaar zijn voor de aanbieder van API's (API-providers) om fijnmazige autorisatie, zoals "Claim/Attribute Based Access Control" ABAC/CBAC te kunnen doen. In die gevallen kan met identiteitpropogatie een ID-token worden doorgeven en kan, waar van toepassing, de gebruiker worden geïdentificeerd.

ⓘ *Alleen API-providers met expliciete toestemming (doelbinding) hebben toegang tot de identiteitsgegevens die zijn afgeschermd binnen de "Identity Server".*

ⓘ *Wanneer het een organisatie betreft, is het Organisatie Identificatienummer (OIN) wel altijd beschikbaar.*

Het grote verschil tussen de twee genoemde tokens, is dus dat een access-token een zogenaamde bearer-token is, ofwel een ondoorzichtige reeks karakters die niet bedoeld is om enige betekenis te hebben voor clients. Een ID-token is daarentegen een JSON Web Token (JWT) en bestaat uit een gescheiden header, body en handtekening.

API-I05 ***Tokens worden niet gebruikt in query-parameters***

Er is een inherent beveiligingsprobleem bij het gebruik van alle soorten tokens in query-parameters, omdat de meeste webservers query-parameters in de server logs wegschrijven.


```
{
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "refresh_token": "8xLOxBtZp8",
  "expires_in": 3600,
  "id_token":  "eyJ4NXQiOiJObUptT0dVeE16WmxZak0yWkRSaE5UWmxZVEExWXpkaFpUUmlPV0UwTldJMk0ySm1PVGMxWkEiL
                CJraWQiOiJkMGVjNTE0YTMyYjZmODhjMGFiZDEyYTINDA2OTliZGQzZGViYTlkIiwiYWxnIJoiUlMyNTYifQ.
                eyJhdF9oYXNoIjoiQ2tZbFd1WDdLdzVPVTZaSmdwYi0yQSIsInN1YiI6ImFkbWluIiwiYXVkIjpbIlYxMWExY
                mhQZjIzMF9VYWd0T2Z1bXhCcDk5Y2EiXSwiYXpwIjoVjExYTFiaFBmMjMwX1VhZ3RPZnVteEJwOTljYSIsImF
                1dGhfdGltZSI6MTUxMDY2NzQxMywiaXNzIjoiaHR0cHM6XC9cL2xvY2FsaG9zdDo5NDQzXC9vYXV0aDJcL3Ra
                2VuIiwiZXhwIjoxNTEwNjcxMDE1LCJub25jZSI6IjNFZmdUN182IiwiaWF0IjoxNTEwNjY3NDE1fQ.LWiefiA
                KGuEOiw2NZPWI-pcba_ekUyd0Qg4JSjhnLfW0D12klJsN7yLraoIAG0iPtwjiIB8TtHfftlE1Hd86rGkOM4pP
                MIaA-BzZE71Ps-HbZBrdLDuiqedAONHi0V3hEfoawie44MDEq2DUWtkDX38oJWKT6Y2-iWdixb5dWM"
}
```


Voorbeeld 5 - JWT ID-token

Een bearer geeft alleen aan of de eigenaar toegang kan krijgen tot een backend service. Het is slechts een sleutel voor de toegang tot een reeks toegewezen resources.

De eigenaar wordt hierbij niet geauthentiseerd. Dat wil zeggen er wordt niet gecontroleerd dat de eigenaar van het token de partij is waar het token aan uitgereikt is [RFC6750].

Het ID-token wordt echter voorzien van de digitale handtekening [RFC7519] en daarmee kan gevalideerd worden of hij daadwerkelijk door de bewuste instantie is uitgegeven. De API-provider is dan ook verplicht de digitale handtekening van het token te controleren, en wel voordat de identiteit in het token als voor waar wordt aangenomen.

API-I06 ***API-providers die een ID-token ontvangen moet altijd de digitale handtekening controleren***

ID-tokens zijn digitaal ondertekend. API-providers zijn verplicht de digitale handtekening van het token te controleren, en wel voordat de identiteit in het token als voor waar wordt aangenomen.

Na het decoderen ziet een ID-token (zonder handtekening) er als volgt uit:

```
{ "alg": "RS256" }                          // Algoritme handtekening: RSA signature SHA-256
{
  "auth_time": 1452170176,                  // Tijd waarop de eindgebruiker is geauthentiseerd (epoch[8])
  "exp": 1452173776,                        // Tijd waarop het access-token verloopt (epoch)
  "sub": "DjOIotSop0qF34upWDns2g==",        // Identiteit van de geautoriseerde gebruiker (pseudo-id)
  "azp": "1yL1_fznzGYutX5gB03L6tXGyjga",    // Geautoriseerde partij (die het ID-token mag doorgeven)
  "at_hash": "Yicl1e529ZXY8OsT9o3y-w",      // Hash om access-token mee te kunnen valideren
  "aud": [
         "1yL1_fznzGYutX5gB03L6tXGyjga"     // Voor deze partijen is het access + ID-token bedoeld
  ],
  "iss": "https://localhost:9443/oauth2/token", // Uitgever van het token (Authorization Provider)
  "iat": 1452170176                             // Het uitgifte moment van het token (epoch)
}
```

Voorbeeld 6 - Een gedecodeerd JWT ID-token

Een ID-token bevat verschillende parameters waarmee het ID-token en het access-token gecontroleerd moeten worden door de API-provider.

API-I07 ***API-providers die een ID-token ontvangen controleren alle parameters***

ID-tokens bevatten verschillende parameters die gecontroleerd moeten worden. De API-provider controleert of het token aan de volgende eisen voldoet:

- De "issuer-identifier" claim (iss) moet overeenkomen met de vooraf opgegeven OpenID Provider waarde;
- De "audience" claim (aud) moet overeenkomen met de eigen waarde voor de identiteit;
- Indien meerdere "audiences" claims (aud) aanwezig zijn, dan moet er een "azp" claim zijn;
- Indien er een "azp" claim is, dan moet deze overeenkomen met de eigen waarde voor de identiteit;
- Omdat TLS wordt gebruikt mag in plaats van het checken van de handtekening ook het certificaat van de autorisatie-server worden gecontroleerd;
- De "algoritim" claim (alg) moet de waarde RS256 bevatten;
- De huidige tijd (nu) moet eerder zijn dan de tijd in de "expiration" claim (exp).

API-I08 ***API-providers die een ID-token ontvangen controleren daarmee ook het ontvangen access-token***

De "access-token hash" claim (at_hash) uit de ID-token dient te worden gebruikt om het access-token te controleren.

[8] Epoch is een systeem voor het beschrijven van een tijdstip. Unix-Epoch is de tijd 00:00:00 UTC op 1 januari 1970, zonder schrikkelseconden.

2.5.4 RESTful principes

Het belangrijkste principe van REST is het scheiden van de API in logische resources ("dingen"). De resources beschrijven de informatie van het "ding". Deze resources worden gemanipuleerd met behulp van HTTP-verzoeken en HTTP-operaties. Elke operatie (GET, POST, PUT, PATCH, DELETE)[9] heeft daarbij een specifieke betekenis.

Operatie	CRUD	Toelichting
POST	Create	Wordt gebruikt als een "create" voor resources die collecties representeren (ofwel POST voegt een resource toe aan de collectie).
GET	Read	Wordt gebruikt om een resource op te vragen van de server. Data wordt alleen opgevraagd en niet gewijzigd.
PUT	Update	Wordt gebruikt om een specifieke resource te vervangen. Is óók een "create" wanneer de resource op aangegeven identifier/URI nog niet bestaat.
PATCH	Update	Wordt gebruikt om een bestaande resource gedeeltelijk bij te werken. Het verzoek bevat de gegevens die gewijzigd moeten worden en de operaties die de resource muteren in het daarvoor bedoelde JSON merge patch formaat [RFC7386].
DELETE	Delete	Verwijdert een specifieke resource.

Tabel 4 - Overzicht van veelgebruikte HTTP-operaties

Per operatie is tevens bepaald of deze gegarandeerd "veilig" en/of "idempotent" moet zijn. Dit is van belang omdat afnemers en tussenliggende middleware hierop rekenen.

Veilig (alleen-lezen)
Veilig betekent in dit geval dat de semantiek is gedefinieerd als alleen-lezen. Dit is van belang als afnemers en tussenliggende systemen gebruik willen maken van caching. Daarnaast kan in een API-gateway een policy zijn ingesteld die slechts 'alleen-lezen' operaties doorlaat.

Idempotent
Onder idempotent wordt verstaan dat meerdere identieke verzoeken exact hetzelfde effect hebben als één verzoek. Dit is van belang wanneer in het geval van bijvoorbeeld een falende verbinding, dezelfde berichten opnieuw worden aangeboden.

Operatie	Veilig	Idempotent
POST	Nee	Nee
GET, OPTIONS, HEAD	Ja	Ja
PUT	Nee	Ja
PATCH	Nee	Optioneel
DELETE	Nee	Ja

Tabel 5 - Overzicht van veilige en idempotente operaties

API-B16 ***API's garanderen dat operaties "veilig" en/of "idempotent" zijn***

Operaties van een API zijn gegarandeerd "veilig" en/of "idempotent" als dat zo is bepaald voor de gebruikte http-operatie (zie overzicht in Tabel 4 en Tabel 5).

REST maakt gebruik van het client-stateless-server ontwerpprincipe dat is afgeleid van client-server met als aanvullende beperking dat het niet toegestaan is om de toestand

[9] HTTP definieert ook operaties als HEAD, TRACE, OPTIONS en CONNECT. Deze worden echter in de context van REST vrijwel niet gebruikt en zijn daarom in de verdere uitwerking weggelaten.

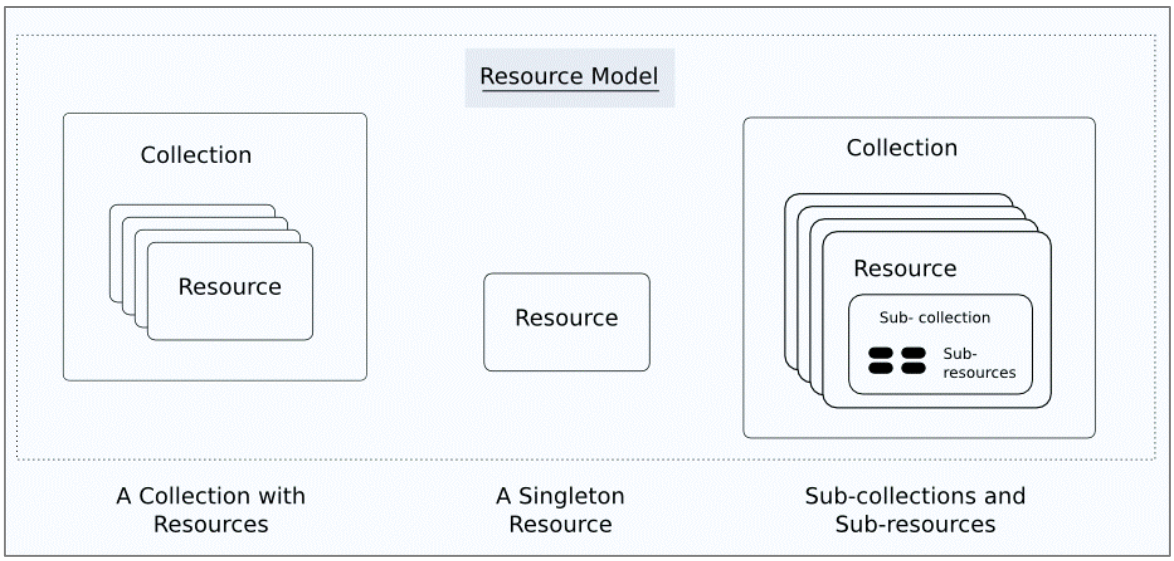
(state) op de server bij te houden. Elk verzoek van de client naar de server moet alle informatie bevatten die nodig is om het verzoek te verwerken, dus zonder gebruik te hoeven maken van toestandsinformatie op de server.

> *API-B17* ***Toestandsinformatie wordt nooit op de server opgeslagen***
>
> De client-toestand wordt volledig bijgehouden door de client zelf.

Alle stelsel-API's zijn via het Knooppunt beschikbaar. Het is niet toegestaan API's direct te benaderen. Stelsel-API's zijn API's aangeboden door onderdelen van de landelijke voorzieningen, door leveranciers van omgevingsinformatie, door e-overheid bouwstenen en GDI-voorzieningen (gebruikt door DSO-LV en ontsloten via het Knooppunt).

> *API-B18* ***Gegevensuitwisseling met API's verloopt alleen via het Knooppunt***
>
> Het Knooppunt is het centrale punt waarop alle API's beschikbaar worden gesteld. Het is niet toegestaan API's direct te benaderen. Het Knooppunt is verantwoordelijk voor het verlenen van toegang en afdwingen van fair-use policies (indien van toepassing).

Wat zijn resources?
Een fundamenteel concept in elke RESTful API is de resource. Een resource is een object met een type, bijbehorende data, relaties met andere resources en een aantal operaties om deze te bewerken. Resources worden aangeduid met zelfstandige naamwoorden die relevant zijn vanuit het perspectief van de afnemer van de API. Dus resources zijn zelfstandige naamwoorden en operaties zijn werkwoorden. Operaties zijn feitelijk de acties die op resources worden uitgevoerd.



Figuur 7 - Resourcemodel voor RESTful API's

Het is mogelijk om interne datamodellen één-op-één toe te wijzen aan resources, maar dit is vaak geen goed idee. Het is beter om te werken met een abstracter, zogenaamd conceptueel informatiemodel (CIM) [10]. Conceptuele informatiemodellen (CIM's) zijn bovendien zeer geschikt om domeinconcepten, zowel in en tussen CIM's, met elkaar te verbinden en te zorgen dat API's in samenhang worden ontworpen en de interoperabiliteit wordt vergroot. De crux is om alle niet-relevante implementatiedetails te verbergen. Enkele voorbeelden van resources van het DSO zijn: verzoek, activiteit, juridische regel, toepasbare regel, (afwijk)vergunning. Deze komen allemaal voort uit CIM's.

Naast het kiezen van zinvolle resourcenamen voor API's is het ook van belang om het aantal resources van één API te beperken.

BEST PRACTICE - 06 ***Gebruik functionele scheiding als ontwerpcriterium voor API's bij het bepalen van resources***

Beperk het aantal resources per API
API's moeten worden opgezet rond een set sterk gerelateerde resources, zoals een resource-collectie, de onderdelen van de resource-collectie en eventuele directe sub-resources. Dat betekent onder andere dat verschillende soorten bedrijfsfuncties niet in dezelfde API dienen te worden gecombineerd. Een goede API heeft ergens tussen de 5 en 10 resources.

ⓘ *Een API moet geen Zwitsers zakmes worden en daarom is functionele scheiding ook voor API's een essentieel ontwerpcriterium. Het aantal resource moet daarom worden beperkt.*

Wanneer de resources geïdentificeerd zijn, kan worden bepaald welke operaties van toepassing zijn en hoe deze worden ondersteund door de API. RESTful API's realiseren CRUD[10] operaties met behulp van HTTP-operaties:

`GET /verzoeken`	Haalt een lijst van verzoeken op
`GET /verzoeken/12`	Haalt een specifiek verzoek op
`POST /verzoeken`	Creëert een nieuw verzoek
`PUT /verzoeken/12`	Wijzigt verzoek #12 als geheel
`PATCH /verzoeken/12`	Wijzigt een gedeelte van verzoek #12
`DELETE /verzoeken/12`	Verwijdert verzoek #12

Voorbeeld 7 - Resources en HTTP-operaties

Het mooie van REST is dat er gebruik wordt gemaakt van de bestaande HTTP-operaties om de functionaliteit te implementeren met slecht één "endpoint". Hierdoor zijn geen aanvullende naamgevingsconventies nodig voor de URI's en blijft de URI-structuur eenvoudig. Zie tevens de URI-strategie [5] voor informatie over de opbouw van URI's.

API-B19 ***Alleen standaard HTTP-operaties worden toegepast***

Een RESTful API is een application programming interface die de standaard HTTP-operaties GET, PUT, POST, PATCH en DELETE gebruikt.

ⓘ *Dit is van groot belang omdat een API-gateway een policy kan hanteren die alleen specifieke operaties doorlaat.*

JSON gecodeerde POST, PUT en PATCH payloads

API's ondersteunen minimaal JSON gecodeerde POST, PUT en PATCH payloads. Encoded form data (`application/x-www-form-urlencoded`) payloads worden niet ondersteund. Wat is het verschil?

[10] CRUD = Create, Read, Update, Delete.

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

Resulteert in:

```
{ "Naam": "Jan Klaassen", "Leeftijd": 44 }
```

en

```
Content-Type: application/x-www-form-urlencoded
```

Resulteert in:

```
Naam=Jan+Klaassen&Leeftijd=44
```

API-B20 ***API's ondersteunen alleen JSON gecodeerde POST, PUT en PATCH payloads***

API's ondersteunen minimaal JSON gecodeerde POST, PUT en PATCH payloads. Encoded form data (application/x-www-form-urlencoded) wordt niet ondersteund.

Welke taal?

Omdat de exacte betekenis van concepten en begrippen vaak verloren gaan in een vertaling, worden resources en de achterliggende entiteiten, velden, etc. (feitelijke het conceptuele informatiemodel en het externe koppelvlak) in het Nederlands gedefinieerd.

API-B21 ***De definitie van een koppelvlak is in het Nederlands tenzij er sprake is van een officieel Engelstalig begrippenkader***

Resources en de achterliggende entiteiten, velden, etc. (het conceptuele informatiemodel en het externe koppelvlak) worden in het Nederlands gedefinieerd. Engels is alleen toegestaan indien er sprake is van een officieel Engelstalig begrippenkader.

Naamgeving "endpoints" in enkelvoud of meervoud?

Het is pragmatisch om te kiezen voor consistente "endpoints" en altijd de meervoudsvorm te gebruiken. Voor de afnemer is het gebruik ook veel eenvoudiger als er geen rekening gehouden hoeft te worden met enkel- en meervoud (verzoek/verzoeken, locatie/locaties).

Daarnaast is de implementatie eenvoudiger omdat de meeste ontwikkel-frameworks het afhandelen van een enkele resource: `/verzoeken/12` en meervoudige resources: `/verzoeken` transparant met één controller kan oplossen.

Toch valt er iets voor te zeggen om een uitzondering te maken voor zogenaamde singletons[11] of collecties met een kardinaliteit van n:1 of 1:1. Bijvoorbeeld één specifiek verzoek dat altijd maar bij één project hoort: `/verzoeken/12/project`.

API-B22 ***Resource-namen zijn zelfstandige naamwoorden in het meervoud***

Namen van resources zijn zelfstandige naamwoorden en altijd in het meervoud, zoals `verzoeken`, `activiteiten`, `locaties`. Een uitzondering hierop is de situatie waarin een resource een zogenaamde singleton of een collectie met een kardinaliteit van n:1 of 1:1 betreft. Resource-namen zijn beperkt tot de alfanumerieke reeks en beginnen altijd met een letter.

[11] Singleton is een ontwerppatroon om het aantal objecten van een bepaalde klasse tot één te beperken.

Hoe om te gaan met acties die niet passen in het CRUD-model?
Er zijn ook resource-acties die niet gerelateerd zijn aan data-manipulatie (CRUD). Voorbeeld van dit soort acties zijn: het wijzigen van de status (activeren en deactiveren) van een resource of het markeren (star) van een resource.
Afhankelijk van het type actie zijn er drie manieren om dit aan te pakken:

1) Herstructureer de actie zodat deze onderdeel wordt van een resource. Dit werkt als de actie geen parameters nodig heeft. Bijvoorbeeld een activeeractie kan worden toegewezen aan een booleaans veld `geactiveerd` dat bijgewerkt wordt via een PATCH op de resource.

2) Behandel de actie als een sub-resource. Bijvoorbeeld, een verzoek kan worden gemarkeerd met: `PUT /verzoeken/12/markeringen` en het verwijderen van de markering met: `DELETE /verzoeken/12/markeringen`. Om de REST-principes volledig te volgen moet ook `GET` voor deze sub-resource beschikbaar zijn.

3) Soms is er geen logische manier om een actie aan een bestaande resource te koppelen. Een voorbeeld hiervan is een zoekopdracht over meerdere resources heen. Deze actie kan niet worden toegekend aan een specifieke resource. In dit geval is de keuze voor een zelfstandig "endpoint": `/_zoek` de meest logische aanpak. Het onderscheid t.o.v. "echte" resources moet duidelijk te herkennen zijn en daarom wordt een underscore als prefix gebruikt.

In de DSO API-strategie wordt gekozen voor manier 2 en 3.

> *API-B23* ***Acties die niet passen in het CRUD-model worden een sub-resource***
>
> Acties die niet passen in het CRUD-model worden op de volgende manieren opgelost:
>
> - Behandel een actie als een sub-resource.
> - Alleen in uitzonderlijke gevallen wordt een actie met een eigen "endpoint" opgelost. In dat geval wordt gebruik gemaakt van een werkwoord in gebiedende wijs dat vooraf wordt gegaan door een underscore, bijvoorbeeld: `_zoek`

De semantiek van `null`[12]
Om tot eenduidig gedrag van een RESTful API te komen, is het van belang dat zowel het gebruik als de betekenis van de waarde `null` voor een aantal specifieke situaties wordt gedefinieerd. Voor een veld dat is gedefinieerd als niet verplicht en/of null-baar, geldt hetzelfde gedrag voor het toekennen van de waarde `null` als voor het geheel weglaten van het bewuste veld. Alle voorkomende combinaties en het toegestane gebruik van de waarde `null` is daarom als volgt gedefinieerd:

Verplicht veld	Null-baar veld	`{}`	`{ "veld": null }`
JA	JA	☒ ONJUIST	☑ JUIST
NEE	JA	☑ JUIST	☑ JUIST
JA	NEE	☒ ONJUIST	☒ ONJUIST
NEE	NEE	☑ JUIST	☒ ONJUIST

Tabel 6 - Definitie voor gebruik van waarde "null"

[12] Een nul-referentie: "NULL", wordt in de computerwereld veelal beschouwd als een gereserveerde waarde die aangeeft dat een referentie niet naar een geldige waarde of object verwijst.

API-B24 ***Het gedrag dat hoort bij het toekennen van de waarde `null` of het geheel weglaten van een veld is eenduidig***

Een veld dat kan worden gedefinieerd als niet verplicht en/of null-baar heeft voor alle voorkomende combinaties een eenduidige representatie.

Verplicht veld	Null-baar veld	
JA	JA	`{ "veld": null }`
NEE	JA	`{}` of `{ "veld": null }`
JA	NEE	`{ "veld": "Een geldige waarde" }`
NEE	NEE	`{}`

BEST PRACTICE - 07 ***Gebruik voor de representatie van een "niet-gemaakte keuze" niet de waarde: `null`***

Gebruik een "leeg object" voor niet-gemaakt keuze
Stel dat een gebruiker kan kiezen uit drie opties: "`Goed`", "`Redelijk`", "`Slecht`" of de keuze kan overslaan of uitstellen. In dat geval verdient het de voorkeur om geen keuze te representeren als een "leeg" object.

LIEVER ZO	DAN ZO

`// gemaakte keuze`
`{`
`  "waardering": { "keuze": "Goed" }`
`}`
	`// gemaakte keuze`
	`{`
	`  "waardering": "Goed"`
	`}`

`// niet-gemaakte keuze`
`{`
`  "waardering": {}`
`}`
	`// niet-gemaakte keuze`
	`{`
	`  "waardering": null`
	`}`

ⓘ *Uitzonderingen op deze regel zijn:*

1. *JSON Merge-Patch [RFC 7396] die `null` gebruikt om expliciet te geven dat een waarde verwijderd moeten worden, terwijl afwezige velden worden genegeerd, ofwel niet worden gewijzigd.*
2. *Wanneer een veld niet null-baar en verplicht is of als een gebruiker er expliciet om vraagt.*

API-B25 ***De waarde `null` wordt nooit gebruikt voor een lege lijst***

Om tot eenduidig gebruik en voorspelbaar gedrag van een lege lijst te komen is het van belang om nooit de waarde `null` toe te kennen. Maak alleen gebruik van: `[]`.

JUIST	ONJUIST

`{`
`  "namen": []`
`}`
	`{`
	`  "namen": null`
	`}`

Voorbeeld 8 - Juist en onjuiste representatie van een lege array

API-B26 ***De waarde `null` wordt nooit gebruikt voor een veld dat is ontworpen om een booleaanse waarde te bevatten***

Om tot eenduidig gebruik en voorspelbaar gedrag van booleaanse velden te komen, is het van belang om nooit de waarde `null` toe te kennen. De engine geldige waarden zijn `true` en `false`.

De waarde `null` wordt nooit gebruikt voor een lege array of een veld dat is ontworpen om een booleaanse waarde te bevatten.

Hoe wordt omgegaan met relaties?
Als een relatie alleen kan bestaan binnen een andere resource (1-op-n relatie), dan kan de afhankelijke resource (kind) alleen via de ouder benaderd worden. Het volgende voorbeeld maakt dit duidelijk. Een verzoek hoort bij één project. De verzoeken worden dan als volgt benaderd via het "endpoint" `/projecten`:

`GET /projecten/12/verzoeken`	Haalt een lijst van verzoeken op van project #12
`GET /projecten/12/verzoeken/5`	Haalt een specifiek verzoek (#5) van project #12 op
`POST /projecten/12/verzoeken`	Creëert een nieuw verzoek voor project #12
`PUT /projecten/12/verzoeken/5`	Wijzigt verzoek #5 van project #12
`PATCH /projecten/12/verzoeken/5`	Wijzigt een gedeelte van verzoek #5 van project #12
`DELETE /projecten/12/verzoeken/5`	Verwijdert verzoek #5 uit project #12

Voorbeeld 9 - Resources met 1-n relaties

API-B27 ***Relaties van geneste resources worden binnen het eindpunt gecreëerd***

Als een relatie alleen kan bestaan binnen een andere resource (geneste resource), wordt de relatie binnen het "endpoint" gecreëerd. De afhankelijke resource heeft geen eigen "endpoint"

BEST PRACTICE - 08 ***Beperk het aantal geneste sub-resources tot drie***

Beperkte nesting van sub-resources
Het nesten van sub-resources moet worden beperkt tot maximaal drie niveaus diep. Enerzijds omdat te diepe nesting de werking ondoorzichtig maakt, anderzijds omdat de totale lengte van een URL beperkt is tot 2048 karakters.

ⓘ *De lengte van een URL is beperkt tot 2048 karakters. URL's die te lang zijn kunnen worden afgebroken en dit kan weer leiden tot onvoorspelbaar gedrag.*

Hoe wordt omgegaan met n:n relaties?
Indien er sprake is van een n-op-n relatie zijn er verschillende manieren om de resources te benaderen. De onderstaande verzoeken leveren hetzelfde resultaat op:

```
GET /projecten/12/samenwerkingen
```

```
GET /samenwerkingen?project=12
```

Voorbeeld 10 - Resources met n-n relaties

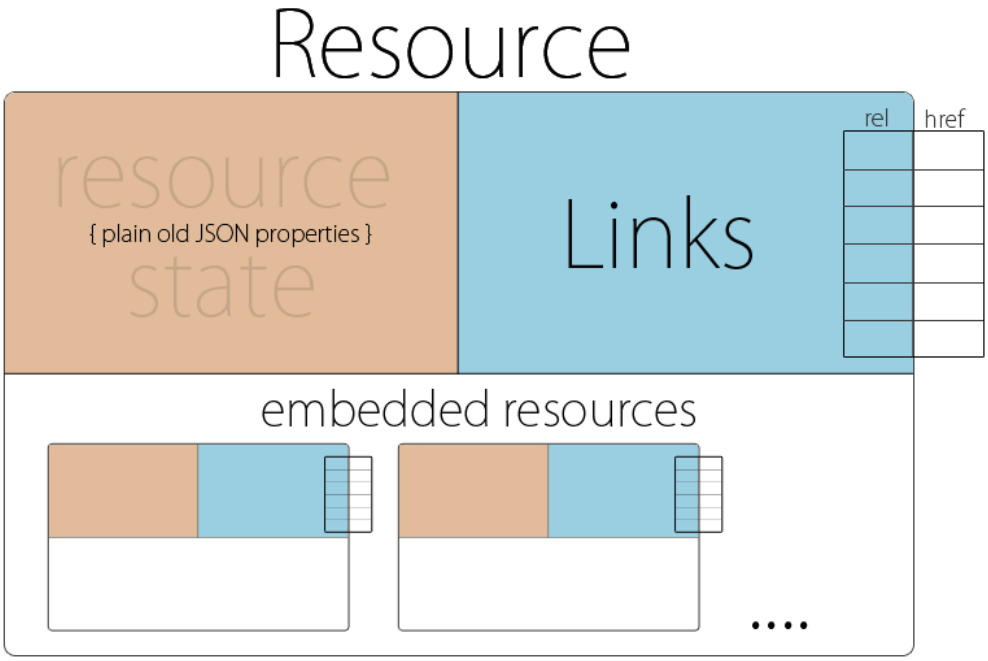
Bij een n-op-n relatie wordt het opvragen van de individuele resources sowieso ondersteund, waarbij minimaal de identificatie van gerelateerde resources (relatie) wordt teruggegeven. Voor dit doel, maar feitelijk ook voor 1-op-n relaties, biedt de Hypertext Application Language (HAL) een set conventies voor het opnemen van hyperlinks in JSON.

HAL is ontworpen om API's te bouwen waarin clients over verschillende resources kunnen navigeren door links te volgen. Links worden geïdentificeerd door linkrelaties ofwel hyperlinks.

API-H01 ***Voor het realiseren van hypermedia controls wordt de Hypertext Application Language (HAL) gebruikt***

HAL is ontworpen voor het bouwen van API's waarin clients door resources navigeren door (hyper)links te volgen. De HAL-representatie voor JSON dient te worden gebruikt indien gerelateerde resources (relaties) worden teruggegeven als hyperlinks.

Links vormen de ruggengraat van een hypermedia-API: ze geven aan welke resources beschikbaar zijn en hoe de ontwikkelaar het te volgen pad kan selecteren. Een HAL-representatie bestaat uit de "ruwe" JSON-inhoud aangevuld met links en eventueel ingebedde resources. De structuur is opgebouwd zoals weergegeven in Figuur 8.



Figuur 8 - Structuur van een HAL-representatie

Links zijn in HAL niet alleen identificerende ID's. Het zijn URI's die door ontwikkelaars kunnen worden verkend. Dit staat bekend als "discoverable". HAL vereenvoudigt het gebruik van link-relaties door:

- Koppelingen te identificeren en desgewenst de resources in te bedden;
- De verwachte structuur en betekenis af te leiden van doel-resources;
- Expliciet te maken welke verzoeken kunnen worden ingediend bij doel-resources en voor welke representaties.

HAL heeft een specifiek mediatype voor JSON met de naam: `application/hal+json`.

API-H02 ***Voor het opnemen van hyperlinks in JSON wordt het HAL-mediatype voor JSON gebruikt***

Voor het opnemen van hyperlinks in JSON biedt de Hypertext Application Language (HAL) een set conventies. Voor het gebruik van deze conventies in JSON dient het volgende mediatype gebruikt te worden: `application/hal+json`

Dit levert voor een project waarin twee verzoeken zitten (fictief voorbeeld) het volgende resultaat op:

```
{
 "id": "12",
 "naam": "Mijn huis",
 "samenvatting": "Ik wil mijn huis verbouwen!",
 "_links": {
  "verzoeken": [{
    "href": "https://.../api/indienen/v1/projecten/12/verzoeken/34",
    "title": "Melding dakkapel"           // optioneel, te gebruiken voor een leesbare lijstweergave
   },
    "href": "https://.../api/indienen/v1/projecten/12/verzoeken/35",
    "title": "Aanvraag kappen boom"
   }],
    "self": {
    "href": "https://.../api/indienen/v1/projecten/12",
    "title": "Mijn huis"
  }
 }
}
```

Voorbeeld 11 - JSON/HAL-respons met hypermedia controls

Afnemers moeten in dit geval via de hyperlinks zelf het "endpoint" van de gerelateerde resource (relatie) aanroepen om deze op te vragen. Dit wordt het "lazy loading" patroon genoemd. Dit heeft voordelen, want de afnemer bepaalt zelf of de relatie geladen wordt en op welk moment.

API-B28 ***Resources gebruiken "lazy loading" tenzij expliciet wordt gevraagd om de gerelateerde resources mee te laden***

Resources die relaties kennen gebruiken standaard het "lazy loading" patroon. Het teruggeven van ingebedde resources t.b.v het "eager loading" patroon, gebeurt voor collectie-resources op en voor gerelateerde resources op basis van een expliciet verzoek. Wanneer het om veel resources gaat, dient de link te verwijzen naar een gepagineerde resource (zie paragraaf 2.5.6).

Automatische laden van gelinkte resources

Vaak wordt vanuit één resource verwezen (gelinkt) naar andere (geneste) resources. De RESTful manier om dit op te lossen is de verwijzing naar andere resources als URI (hyperlinks) op te nemen in een resource. Op basis hiervan kan de afnemer, indien gewenst, de gelinkte resources zelf opvragen. In sommige gevallen, als de afnemer alle resources meteen nodig heeft, is het efficiënter als de geneste resources in één keer opgehaald worden. Dit wordt het "eager loading" patroon genoemd. In een RESTful-aanpak zou dit geen standaardbenadering moeten zijn. Daarom moet het toepassen van dit mechanisme een expliciete keuze zijn. De keuze wordt daarom bij de afnemers van de API gelegd, zij weten immers op welke momenten ze extra informatie nodig hebben en welke informatie dat precies is. Kortom, een resource kan naast "lazy loading" (de standaard) ook "eager loading" ondersteunen, in dat geval kan de afnemer aangeven dat relaties direct meegeladen moeten worden. Dit dient dan te worden ondersteund via de query-parameter: `_expand=[false|true]`. Om verwarring of "botsingen" met resource-properties te voorkomen wordt een underscore als prefix toegepast.

API-B29 ***Gelinkte resources kunnen op verzoek worden meegeladen ("eager loading")***

Gelinkte resources kunnen op verzoek worden meegeladen als onderdeel van een resource-verzoek. Dit dient dan te worden aangegeven via de `_expand` query-parameter. Geldige waarden voor deze parameter zijn `false` en `true`.

LETOP

DEP-01 De query-parameter `expand (boolean)` is vervangen door `_expand (boolean)`

In de API-strategie V1.1 werd voor het op verzoek en het selectief meeladen van gelinkte resources dezelfde query-parameter `expand` gebruikt. Deze parameter is vervangen door `_expand`. Indien API's deze parameter gebruiken dient deze uitgefaseerd te worden.

Het is ook een keuze voor de aanbieder om te bepalen in welke mate dit wordt ondersteund om te voorkomen dat het resultaat onverantwoord groot wordt. Door het gebruik van `_expand=true` kan worden vermeden dat er twee of meer aparte aanroepen nodig zijn. In alle gevallen is en blijft de afnemer in control. Wanneer `_expand=true` wordt meegegeven, worden alle geneste resources geladen en ingebed in het terug te geven resultaat.

Deze "expand all" optie is daarom alleen zinvol en verantwoord als het aantal relaties beperkt is en het resultaat gecontroleerd binnen de perken kan blijven.

> *API-H03* ***Voor het inbedden van resources in JSON wordt HAL toegepast***
>
> Voor het opnemen van zogenaamde "embedded resource" in JSON biedt de Hypertext Application Language (HAL) een set conventies. Dit is een aanvulling op het opnemen van hyperlinks en dit dient daarom in samenhang te worden toegepast.

Het gebruik van `_expand=true` levert bijvoorbeeld het volgende resultaat op:


```
{
 "id": "12",
 "naam": "Mijn huis",
 "samenvatting": "Ik wil mijn huis verbouwen!",
 "_embedded": {
   "verzoeken": [
    {
      "id": "34",
      "naam": "Mijn dakkapel",
      "_links": {
        "self": {
          "href": "https://.../api/indienen/v1/projecten/12/verzoeken/34",
          "title": "Melding dakkapel"
      }
    },
    {
      "id": "35",
      "naam": "Boom kappen",
      "_links": {
        "self": {
          "href": "https://.../api/indienen/v1/projecten/12/verzoeken/35",
          "title": "Aanvraag kappen boom"
      }
    }, → ]
  },
  "bevoegdGezag": {
    "oin": "00000001001876387000",
    "type": "GM",
    "code": "0599",
    "naam": "Rotterdam",
    "_links": {
     "self": {
       "href": "https://portaal.digikoppeling.nl/registers/api/v1/organisaties/00000001001876387000",
       "title": "gemeente Rotterdam"
       }
     }
   }
 },
 "_links": {
  "self": {
   "href": "https://.../api/indienen/v1/projecten/12",
   "title": "Mijn huis"
  }
 }
}
```


Voorbeeld 12 - JSON/HAL-respons met alle gerelateerde resources volledig ingebed

Om de hoeveelheid informatie die geretourneerd te kunnen te beperken, kan het wenselijke zijn om exact te specificeren welke resources en zelfs welke velden van een resource teruggeven moeten worden. Hiermee wordt de scope van de eerdere genoemde expand-operatie beperkt en kan worden voorkomen dat de hele registratie wordt leeggetrokken.

API-B30 ***Gelinkte resources kunnen selectief worden meegeladen ("eager loading")***

Gelinkte resources kunnen selectief worden meegeladen als onderdeel van een verzoek. Dit dient dan te worden gespecificeerd via de `_expandScope` query-parameter. Dit specificeren wordt gedaan door resources in een door komma's gescheiden lijst te specificeren. De dot-notatie kan aanvullend worden gebruikt om specifieke velden van resources te selecteren.

LETOP

DEP-02 ***De query-parameter `expand (string)` is vervangen door `_expandScope (string)`***

In de API-strategie V1.1 werd voor het selectief meeladen van gelinkte resources de query-parameter `expand` gebruikt. Deze parameter is vervangen door `_expandScope`. Indien API's deze parameter gebruiken dient deze uitgefaseerd te worden.

Het specificeren van de "scope" wordt gedaan door de resources in een door komma's gescheiden lijst te specificeren, bijvoorbeeld:

```
GET /projecten/12?_expand=true&_expandScope=bevoegdGezag
```

Dit levert dan het volgende resultaat op:


```
{
 "id": "12",
 "naam": "Mijn huis",
 "samenvatting": "Ik wil mijn huis verbouwen!",
 "_embedded": {
  "bevoegdGezag": {
   "oin": "00000001001876387000",
   "type": "GM",
   "code": "0599",
   "naam": "Rotterdam",
   "_links": {
    "self": {
     "href": "https://portaal.digikoppeling.nl/registers/api/v1/organisaties/00000001001876387000",
     "title": "gemeente Rotterdam"
    }
   }
  }
 },
 "_links": {
  "verzoeken": [
   {
    "href": "https://.../api/indienen/v1/projecten/12/verzoeken/34"
    "title": "Melding dakkapel"
   },
    "href": "https://.../api/indienen/v1/projecten/12/verzoeken/35"
    "title": "Aanvraag kappen boom"
   }
  ],
  "self": {
   "href": "https://.../api/indienen/v1/projecten/12",
   "title": "Mijn huis"
  }
 }
}
```


Voorbeeld 13 - JSON/HAL-respons met één volledig ingebedde resource

De dot-notatie kan aanvullend worden gebruikt om specifieke velden van resources te selecteren. In het onderstaande voorbeeld wordt van het bevoegd gezag alleen het veld "naam" teruggeven en van de het verzoek de complete resource.

Conform de HAL-standaard zijn de gelinkte resources ingebed in de standaard representatie (zie ook Query-projectie in paragraaf 2.5.7).

```
GET /projecten/12?_expand=true&_expandScope=verzoek,bevoegdGezag.code
```

Dit levert het volgende resultaat op:

```
{
 "id": "12",
 "naam": "Mijn huis",
 "samenvatting": "Ik wil mijn huis verbouwen!",
 "_embedded": {
  "verzoeken": [{
    "id": "34",
    "naam": "Mijn dakkapel",
    "_links": { →
   }, → ]
  },
  "bevoegdGezag": {
   "code": "0599"
   "_links": {
    "self": {
     "href": "https://portaal.digikoppeling.nl/registers/api/v1/organisaties/00000001001876387000",
     "title": "gemeente Rotterdam"
    }
   }
  }
 },
 "_links": { →
 }
}
```


Voorbeeld 14 - JSON/HAL-respons met één volledig en één gedeeltelijk ingebedde resource

2.5.5 Filteren, sorteren en zoeken

De basis-URL's van resources wordt zo eenvoudig mogelijk gehouden. Complexe resultaatfilters, sorteren en geavanceerd zoeken (wanneer dit beperkt blijft tot een enkele resource) worden geïmplementeerd als query-parameters bovenop de basis-URL.

Filteren
Om te filteren wordt gebruik gemaakt van unieke query-parameters die gelijk zijn aan de velden waarop gefilterd kan worden. Als bijvoorbeeld een lijst met verzoeken opgevraagd wordt van het "endpoint": `/verzoeken` en deze beperkt moet worden tot de openstaande verzoeken, dan wordt het verzoek: `GET /verzoeken?status=open` gebruikt. Hier is: `status` dus een veld waarop gefilterd kan worden.

API-B31 ***Filter query-parameters zijn gelijk aan de velden waarop gefilterd kan worden***

Gebruik unieke query-parameters die gelijk zijn aan de velden waarop gefilterd kan worden.

Aanvullend kan gebruik worden gemaakt van filter-operatoren om bijvoorbeeld een bepaald bereik te selecteren of om een negatief filter toe te passen. Dit kan eenvoudig met zogenaamde "Left Hand Size brackets" (LHS-brackets) worden geïmplementeerd. Hierbij wordt gebruik gemaakt van de mogelijkheid om vierkante haken aan de linker kant van "=" symbool te kunnen plaatsen:

Operator	Notatie	Toepassing in query-parameter
<	[lt]	GET /verzoeken?verzoekDatum[lt]=2020-01-01
>	[gt]	GET /verzoeken?verzoekDatum[gt]=2020-01-01
<=	[lte]	GET /verzoeken?verzoekDatum[lte]=2020-01-01
>=	[gte]	GET /verzoeken?verzoekDatum[gte]=2020-01-01
!=	[not]	GET /verzoeken?status[not]=open
Ook een bereik kan worden geselecteerd door een parameter simpelweg twee keer op te nemen met verschillende operatoren. In het onderstaande voorbeeld gebeurt dit om alle verzoeken tussen 1 januari 2019 en 1 januari 2020 te selecteren: GET /verzoeken?verzoekDatum[gte]=2019-01-01&verzoekDatum[lte]=2020-01-01		

Tabel 7 - definitie van LHS-brackets

Dezelfde systematiek kan ook worden gehanteerd voor geneste properties. Zoals uitgewerkt met een voorbeeld de volgende collectie:


```
[
  {
    "id": 1,
    "status": "actief",
    "overheid": {
      "type": "mnre",
      "code": "1034",
      "naam": "ministerie van Binnenlandse Zaken en Konikrijksrelaties"
    }
  },
  {
    "id": 2,
    "status": "inactief",
    "overheid": {
      "type": "pv",
      "code": "25",
      "naam": "provincie Gelderland"
    }
  }
]
```


Voorbeeld 15 - JSON-collectie gefilterd op twee niveaus

Alle objecten met de status "actief" kunnen dan worden geselecteerd met: `/?status=actief`. Maar als daarnaast ook op resources van het type "overheid" met code "pv" geselecteerd moet worden, dan heeft dit betrekking op een geneste property. Hier kan dan de puntnotatie (zoals ook gebruikelijk bij JavaScript) voor worden gebruikt: `/?status=actief&overheid.type=pv`.

API-B32 ***Een "filter" query-parameters met niet-bestaande veldnamen resulteert in een fout***

Als niet-bestaande veldnamen worden meegegeven in een query-parameter wordt een 400 Bad Request teruggegeven.

Sorteren

Voor het sorteren wordt de query-parameter `_sort` gebruikt. Om verwarring of "botsingen" met resource-properties te voorkomen wordt een underscore als prefix toegepast. Deze query-parameter accepteert een lijst van velden waarop gesorteerd moet worden gescheiden door een komma. Door een minteken ("-") voor de veldnaam te zetten wordt het veld in aflopende volgorde gesorteerd. Een aantal voorbeelden:

`GET /verzoeken?_sort=-prio`	Haalt een lijst van verzoeken op gesorteerd in aflopende volgorde van prioriteit.
`GET /verzoeken?_sort=-prio,verzoekDatum`	Haalt een lijst van verzoeken op in aflopende volgorde van prioriteit. Binnen een specifieke prioriteit, komen oudere aanvragen eerst.

Voorbeeld 16 - Gebruik van query-parameters voor sorteren

API-B33 ***Voor sorteren wordt de query-parameter `_sort` gebruikt***

De generieke query-parameter `_sort` wordt gebruikt voor het specificeren van door komma's gescheiden velden waarop gesorteerd moet worden. Door een minteken ("-") voor de veldnaam te zetten wordt het veld in aflopende sorteervolgorde gesorteerd.

LETOP

DEP-03 ***De query-parameter `sorteer` is vervangen door `_sort`***

In de API-strategie V1.1 werd voor sorteren de query-parameter `sorteer` gebruikt. Deze parameter is vervangen door `_sort`. Indien API's deze parameter gebruiken dient deze uitgefaseerd te worden.

Zoeken

Soms zijn eenvoudige filters onvoldoende en is de kracht van vrije-tekst zoekmachines nodig. Hiervoor is binnen het stelsel de bouwsteen Zoeken (technisch component Elasticsearch) beschikbaar. API's ondersteunen het gebruik van een zoekmachine middels de query-parameter `_find`. Om verwarring of "botsingen" met resource-properties te voorkomen wordt een underscore als prefix toegepast.

Een zoekopdracht die meegegeven wordt met deze query-parameter wordt 1-op-1 doorgegeven aan de zoekmachine. Het resultaat wordt in dezelfde representatie teruggegeven.

API-B34 ***Voor vrije-tekst zoekopdrachten wordt de query-parameter `_find` gebruikt***

API's die vrije-tekst zoeken ondersteunen doen dit middels de query-parameter `_find`.

LETOP

DEP-04 ***De query-parameter `zoek` is vervangen door `_find`***

In de API-strategie V1.1 werd voor zoeken de query-parameter `zoek` gebruikt. Deze parameter is vervangen door `_find`. Indien API's deze parameter gebruiken dient deze uitgefaseerd te worden.

Voorbeelden van de combinatie filteren, sorteren en zoeken:

`GET /verzoeken?_sort=-wijzigDatum`	Haalt een lijst van recente verzoeken op.
`GET /verzoeken?status=gesloten&_sort=-wijzigDatum`	Haalt een lijst van recent gesloten verzoeken op.
`GET /verzoeken?status=open&_find=urgent&&_sort=-prio,verzoekDatum`	Haalt een lijst van verzoeken op met de status 'open' en waarin het woord 'urgent' voorkomt, gesorteerd van de hoogste naar de laagste prioriteit, en daarbinnen op verzoekdatum van oud naar nieuw.

Voorbeeld 17 - Combinatie van query-parameters voor filteren, sorteren en zoeken

Wildcards

Wanneer door een API vrije-tekst zoeken wordt ondersteund worden twee soorten wildcard karakters toegestaan:

* `*` Komt overeen met nul of meer (niet-spatie) karakters
* `?` Komt precies overeen met één (niet-spatie) karakter

Bijvoorbeeld, `he*` zal overeenkomen met elk woord dat begint met `he`, zoals **`he`**`k`, **`he`**`melwaterafvoer`, enzovoort. In het geval van `he?` komt dit alleen overeen met drie letterwoorden die beginnen met `he`, zoals **`he`**`k`, **`he`**`g`, enzovoort.

API-B35 ***API's die vrije-tekst zoeken ondersteunen kunnen overweg met twee soorten wildcard karakters***

API's die vrije-tekst zoeken ondersteunen kunnen overweg met twee soorten wildcard karakters:
* Komt overeen met nul of meer (niet-spatie) karakters
? Komt precies overeen met één (niet-spatie) karakter

Hieronder volgen nog een aantal basisregels voor wildcards in zoekopdrachten:

- Er kan meer dan één wildcard in één zoekterm of zin voorkomen;
- De twee wildcardkarakters kunnen in combinatie worden gebruikt. Bijvoorbeeld `m*??` komt overeen met woorden die beginnen met `m` en drie of meer tekens hebben;
- Spaties (URL-encoded als %20) die worden gebruikt als woordscheiding en wildcard-matching werkt alleen binnen een enkel woord. Bijvoorbeeld, `r*te*` komt overeen met de `ruimtelijk`, maar niet met `ruimte tekort`;
- Wildcards werken alleen op JSON-velden met tekst (string) waarden.

Aliassen voor terugkerende queries
Om de Developer eXperience verder te verbeteren is het mogelijk om terugkerende queries als "endpoints" aan te bieden. Zo kunnen onlangs gesloten verzoeken ook als volgt worden benaderd:

```
GET …/verzoeken/recent_gesloten
```

2.5.6 Paginering

Voor paginering wordt gebruik gemaakt van hypermedia controls op basis van HAL of headers. Dit laatste is vooral gericht op de ondersteuning van paginering met "plain" JSON.

Hier is een voorbeeld van een JSON/HAL-representatie:

```
{
   "_embedded":{
      "activiteiten":[ // Inhoude van de lijst
         {
            "identificatie":"nl.imow-gm0599.activiteit.ModernWinkelenEnDrukOntmoeten",
            "naam":"Bouwen",
            "groep":{
               "code":"acg",
               "waarde":"Recreatie"
            },
```

```
         }
      ]
   },
   "_links":{ // Links voor de paginabesturing
      "self":{
         "href":"https://.../api/omgevingsdocumenten/v4/activiteiten"
      },
      "first":{
         "href":"https://.../api/omgevingsdocumenten/v4/activiteiten?page=2",
         "title":"Eerste pagina"
      },
      "last":{
         "href":"https://.../api/omgevingsdocumenten/v4/activiteiten?page=5",
         "title":"Laatste pagina"
      },
      "prev":{
         "href":"https://.../api/omgevingsdocumenten/v4/activiteiten?page=1",
         "title":"Vorige pagina"
      },
      "next":{
         "href":"https://.../api/omgevingsdocumenten/v4/activiteiten?page=3",
         "title":"Volgende pagina"
      }
   },
   "page":{ // Metadata
      "size":20,
      "totalElements":99,
      "totalPages":5,
      "number":0
   }
}
```

Voorbeeld 18 - HAL-representatie voor een collectie met paginering

Om ook met "plain" JSON of een andere representatie zonder hypermedia controls te kunnen werken, dient de pagineerinformatie ook te worden opgenomen in de response headers. Hierin wordt de metadata teruggegeven als HTTP-headers.

API-H04 ***Paginering via hypermedia controls wordt ondersteund met HAL***

Paginering met hypermedia controls wordt gerealiseerd met HAL. De JSON-respons bestaat hierbij altijd uit drie delen:

- De "embedded" collectie
- De navigatie links
- De metadata met pagineerinformatie

Alle links in HAL zijn absoluut. Dit in verband met mogelijke externe links (naar andere "endpoints", linked-data resources, etc.) en eenvoudigere navigatie door clients die dan niet zelf de URL hoeven op te bouwen.

API-B36 ***Paginering in "plain" JSON wordt ondersteund door HTTP-headers met metadata***

Voor de ondersteuning van paginering in "plain" JSON, wordt de paginerings-metadata in de vorm van HTTP-headers meegegeven.

HTTP-header	Toelichting	Toepassing
X-Pagination-Page	Huidige pagina.	Verplicht
X-Pagination-Limit	Aantal resultaten per pagina.	Verplicht
X-Pagination-Count	Totaal aantal pagina's.	Optioneel
X-Total-Count	Totaal aantal resultaten.	Optioneel

Bij grote datasets kunnen de berekeningen voor `X-Total-Count` en `X-Pagination-Count` behoorlijke impact hebben op de performance, voornamelijk als er niet of nauwelijks gefilterd wordt.

2.5.7 Query-projectie

Gebruikers van een API hebben niet altijd de volledige representatie (lees alle velden) van een resource nodig. Daarom helpt het als de mogelijkheid bestaat om de gewenste velden te selecteren, we noemen dit een projectie. Hierdoor kan het netwerkverkeer worden beperkt (relevant voor lichtgewicht toepassingen), vereenvoudigt het gebruik van de API en is de uitvoer aanpasbaar (op maat).

Indien dit mogelijk wordt gemaakt kan daarvoor de query-parameter `_fields` worden ondersteund. Om verwarring of "botsingen" met resource-properties te voorkomen wordt een underscore als prefix toegepast. Deze query-parameter accepteert een door komma's gescheiden lijst met veldnamen. Het resultaat is een op maat gesneden "projectie". Het volgende verzoek haalt bijvoorbeeld voldoende informatie om een gesorteerde lijst van openstaande verzoeken te tonen:

```
GET /verzoeken?_fields=onderwerp,datumLaatsteWijziging,aanvrager →
               &status=open&_sort=-datumLaatsteWijziging
```

Dit levert de volgende projectie op:

```
{
  "verzoeken": [
    {
     "onderwerp": "Bouw dakkapel",
     "datumLaatsteWijzing": "2019-11-12",
     "aanvrager": "Bob de brouwer"
    },
    {
     "onderwerp": "Bouw schuur",
     "datumLaatsteWijzing": "2019-12-31",
     "aanvrager": "Klusbedrijf Goudeerlijk"
    }
  ]
}
```

Voorbeeld 19 - JSON-respons van een collectie met een simpele projectie voor de resources

API-Q01 ***Query-projectie wordt ondersteund met de query-parameter `_fields`***

Het is mogelijk om een door komma's gescheiden lijst van veldennamen op te geven met de query-parameter `_fields` om een representatie op maat te krijgen.

LETOP

DEP-05 ***De query-parameter `fields` is vervangen door `_fields`***

In de API-strategie V1.1 werd voor query-projectie de query-parameter `fields` gebruikt. Deze parameter is vervangen door `_fields`. Indien API's deze parameter gebruiken dient deze uitgefaseerd te worden.

Om te voorkomen dat verkeerd gebruik van veldnamen ongewild tot een verkeerde projectie leidt, dient het gebruik van niet-bestaande veldnamen via een foutmelding te worden gemeld.

API-Q02 ***Query-projectie met niet-bestaande veldnamen resulteert in een fout***

Als niet-bestaande veldnamen worden meegegeven in een `_fields` query-parameter wordt een 400 Bad Request teruggegeven.

In het geval van HAL worden de gelinkte resources normaliter ingebed in de standaard (volledige) representatie. Met de eerder beschreven query-parameter ontstaat ook de mogelijkheid om de inhoud van ingebedde resources naar behoefte aan te passen. Om naamconflicten te voorkomen moeten de namen volledig gekwalificeerd zijn op basis van de dot-notatie (zie ook paragraaf 2.5.5):

```
GET /verzoeken?_fields=verzoeken.onderwerp,                                    →
                      verzoeken.aanvrager,verzoeken.datumLaatsteWijziging      →
              &status=open&_sort=-datumLaatsteWijziging
```

Dit levert de volgende projectie op:


```
{
 "_embedded": {
  "verzoeken": [{
   "onderwerp": "Bouw dakkapel",
   "aanvrager": "Bob de bouwer",
   "datumLaatsteWijzing": "2019-11-11",
   "_links": {
    "self": {
     "href": "https://.../api/indienen/v1/projecten/12/verzoeken/34",
     "title": "Melding dakkapel"
    }
   },
   "onderwerp": "Bouw schuur",
   "aanvrager": "Klusbedrijf Goudeerlijk",
   "datumLaatsteWijzing": "2019-12-22",
   "sortlinks": {
    "self": {
     "href": "https://.../api/indienen/v1/projecten/14/verzoeken/66",
     "title": "Aanvraag voor het bouwen van een schuurtje"
    }
   }]
  },
  "_links": { →
  },
  "page": {
    "size": 20,
    "totalElements": 2,
    "totalPages": 1,
    "number": 0
  }
}
```


Voorbeeld 20 - JSON/HAL-respons met een collectie en een projectie voor de ingebedde resources

API-Q03 ***Query-projectie kan met HAL worden toegepast op ingebedde resources***

Query-projectie kan met HAL worden toegepast om de inhoud van ingebedde resources naar behoefte aan te passen.

Uiteraard zijn er ook alternatieve manieren om query-projectie toe te passen, één daarvan gebruikt een zogenaamd "post-endpoint" en wordt beschreven in het onderdeel API-profielen (2.5.10). Daarnaast biedt de query-taal GraphQL [11] interessante mogelijkheden om geavanceerde projectie over meerdere resources heen te ondersteunen. GraphQL is een (data)querytaal en specificatie die in 2012 intern door Facebook is ontwikkeld voordat deze in 2015 voor het publiek in open source beschikbaar kwam. GraphQL biedt feitelijk een alternatief voor REST-gebaseerde architecturen met als doel de productiviteit van ontwikkelaars te verhogen en de hoeveelheid uit te leveren gegevens te minimaliseren of beter gezegd op maat te kunnen samenstellen. GraphQL wordt hier niet geïntroduceerd als alternatief voor REST, maar omdat het binnen één API de mogelijkheid kan bieden om voor een aantal resources een flexibele projectie toe te staan. Op zeer beperkte schaal dus en gebruikmakend van de bestaande resources.

Een GraphQL-service wordt gedefinieerd door types en velden voor die types. Een GraphQL-service voor een *Project* met daarin 0 of meer *Verzoeken*, kan er als volgt uitzien:

```
type Query {
  project(id: String): Project
}
```

```
type Project {
  id: String!
  naam: String!
  verzoeken: [Verzoek]
}
type Verzoek {
  id: String!
  naam: String!
  datum: String!
}
```

Voorbeeld 21 - GraphQL schemadefinitie (query en typen)

Een GraphQL-service die een *Project* met lijst van *Verzoeken* uitlevert met daarin de naam van het verzoek kan er zo uitzien:

```
query {
  project(id: "12345") {
    naam
    verzoeken {
      naam
    }
  }
}
```

```
{
  "data": {
    "project": {
      "naam": "Mijn huis",
      "verzoeken": [
        {
          "naam": "Melding dakkapel"
        },
        {
          "naam": "Aanvraag bouw berging"
        }, → ]
    }
  }
}
```

Voorbeeld 22 - GraphQL query en het resultaat

GraphQL heeft een eigen "runtime" die services beschikbaar maakt waarmee GraphQL-query's kunnen worden aangeboden om te valideren en uit te voeren. Een ontvangen query wordt eerst gecontroleerd om ervoor te zorgen dat deze alleen naar de gedefinieerde resources en properties verwijst en voert vervolgens de aangeboden query uit om een resultaat te produceren. Dit alles gebeurt feitelijk ook via een specifiek "post-endpoint". GraphQL heeft een strikte scheiding tussen structuur en gedrag. De structuur van een API wordt bepaald door de GraphQL-schemadefinitie (zoals in Voorbeeld 21). De schemadefinitie is een abstracte beschrijving van de services die worden ondersteund. Het gedrag van een GraphQL API wordt echter bepaald door zogenaamde "resolvers". Elk veld in het schema wordt ondersteund door precies één "resolver" die weet hoe de gegevens voor dat specifieke veld moeten worden opgehaald.

GraphQL is niet voor alles een goede oplossing en REST-API's zijn dan ook niet. Maar voor het flexibel uitleveren van data, kunnen ze elkaar in specifieke gevallen wel versterken. In de context van een RESTful benadering is een laag tussen afnemers en de bestaande REST API's een slimme strategie om GraphQL als een aanvullende functie te integreren. Met deze aanpak kunnen niet alle mogelijkheden van GraphQL worden benut, maar in veel gevallen kan er toch controleerbaar en gericht meer flexibiliteit mee worden aangeboden.

API-Q04 ***Query-projectie op basis van GraphQL verloopt via een gereserveerd "endpoint" van het "host" component***

Query-projectie kan met GraphQL worden toegepast door een specifiek daarvoor gereserveerd (extra) "endpoint" aan te bieden. Bijvoorbeeld: `https://.../verzoeken/api/graphql/v1/`. Omdat de REST API en de GraphQL-implementatie gekoppeld zijn, dienen de versienummers bij voorkeur in sync gehouden te worden.

ⓘ *Door deze aanpak kunnen de "endpoints" een eigen versie hanteren, maar blijft de routering inclusief de toegangscontrole die via het Knooppunt wordt gerealiseerd, gewoon intact.*

De integratie van GraphQL kan vrij eenvoudig worden gerealiseerd door in de "resolvers" de REST-API van het "host"" component ("verzoeken") aan te roepen en de JSON-response direct te retourneren. Dit ziet er dan als volgt uit:

```
const baseURL = `https://service.omgevingswet.overheid.nl/.../verzoeken/api/indienen/v1 `
const resolvers = {
  Query: {
    project: (args) => {
      const { id } = args
      return fetch(`${baseURL}/projecten/${id}`).then(res => res.json())
    },
    verzoeken: () => {
      return fetch(`${baseURL}/verzoeken`).then(res => res.json())
    }
  }
}
```

Voorbeeld 23 - GraphQL REST API-resolver

Met deze aanpak kunnen, zoals hierboven als is vermeld, niet alle mogelijkheden van GraphQL worden benut. Daarnaast kan er bij grote datasets te veel "overhead" ontstaan. Desalniettemin zijn er veel situaties waarin deze aanpak goed kan werken.

API-Q05 ***GraphQL-integratie wordt gerealiseerd met REST-API "resolvers" wanneer de situatie zich daarvoor leent***

De integratie van GraphQL en een REST-API kan goed worden gerealiseerd door in "resolvers" de REST-API van het "host" component aan te roepen en de JSON-response direct te retourneren. Uiteraard moet de situatie zich daarvoor lenen, wat betekent dat de "resolvers" eenvoudig zijn en de "overhead" gecontroleerd binnen de beperken blijft.

2.5.8 GEO-ondersteuning

REST API's voor het werken met geometrieën kunnen een filter aanbieden op basis van geografische gegevens. Het is hierbij belangrijk om onderscheid te maken tussen een geometrie in het resultaat (API-response) en een geografisch filter in de aanroep (API-request). Het is immers niet vanzelfsprekend dat als iemand wil weten in welk perceel hij/zij zich momenteel bevindt, dat ook de geometrie in de response wordt teruggegeven; een naam of nummer kan dan al voldoende zijn.

Het is wél belangrijk dat het antwoord juist is en de brondata dus zeer gedetailleerde geometrieën bevat; een gebruiker wil immers geen fout antwoord krijgen. Mocht iemand andersom alle percelen met `status=actief` willen plotten op een kaartje, dan wil hij/zij juist de geometrieën in de response ontvangen, maar is het detailniveau weer niet zo belangrijk.

> *API-G01* ***GEO API's ontvangen en versturen GeoJSON***
>
> Voor GEO API's wordt de standaard GeoJSON gebruikt.

Resultaat (API-response)
In een JSON API wordt een geometrie teruggegeven als GeoJSON. Om niet volledig afhankelijk te worden van GeoJSON is ervoor gekozen het geo-gedeelte binnen de `application/json` te 'wrappen' in een apart GeoJSON-object.

> *API-G02* ***GeoJSON is onderdeel van de embedded resource in de JSON-response***
>
> GeoJSON wordt in een JSON-response (application/json) geplaatst waarbij geo attributen als GeoJSON-compatible object in de resource ingebed zijn.

Aanroep (API-request)
Een geografisch filter kan erg complex en groot zijn. Het is daarom noodzakelijk om het in de request-body mee te sturen. Omdat een GET-methode geen payload kan hebben moet hiervoor een POST-methode worden gebruikt. We noemen dit "POST-endpoint.

Het onderstaande voorbeeld doet een zogenaamde GEO-query naar alle panden waarin het veld `geo` (er kunnen ook andere velden zijn, zoals `hoofdgeometrie`, `binnenOmtrek`, `buitenOmtrek`) het GeoJSON object (in dit geval een Point, dus één coördinaat) bevat:

```
POST https://...omgevingsdocumenten/api/opvragen/v4/locaties/_zoek
{                                                                    // request-body
  "geo": {
    "contains": {
       "type": "Point",
       "coordinates": [5.9623762, 52.2118093]
    }
  }
}
```

Voorbeeld 24 - GEO-query

> *API-G03* ***Voor GEO-queries is een POST-endpoint beschikbaar***
>
> Geometrische queries worden in een POST-request naar een apart zogenaamd POST-endpoint van de API verzonden.

Een POST-endpoint voor GEO-queries hoeft niet beperkt te blijven tot GEO-eigenschappen en kan in principe ook gecombineerde queries verwerken. Het is daarom sterk aan te bevelen om een generiek query POST-endpoint in te richten:

```
POST https://...omgevingsdocumenten/api/oprvagen/v4/locaties/_zoek
{                                                                    // request-body
  "geo": {
    "contains": {
       "type": "Point",
       "coordinates": [5.9623762, 52.2118093]
    }
  },
  "locatieType": "gebied"
}
```

Voorbeeld 25 - GEO-query met gecombineerde zoekvraag

> *API-G04* ***POST endpoints zijn uitbreidbaar voor gecombineerde vragen***
>
> De post-body is uitbreidbaar met andere properties (dan GEO) voor uitvoeren van gecombineerde queries.

Naast `contains` kunnen ook andere operations, zoals `intersects` (snijdt) of `within` (valt binnen) gebruikt worden. De benamingen van deze operators komen uit de GEO-wereld en worden daarom slim hergebruikt.

Omdat voor de geometrie een GeoJSON object wordt gebruikt hoeft ook voor de syntax niks nieuws verzonnen te worden. Daarnaast kan in het omhullende JSON-object bijvoorbeeld ook worden aangeven wat de hoogte t.o.v. NAP is, wat de bron en het originele coordinaatreferentiesysteem van de geometrie is.

```
{
  "identificatie": "nl.imow-gm0599.gebied.Centrumgebied",
  "noemer": "Centrum",
  "locatieType": "gebied",

  "geometrie": {                    // GEO-data in GeoJSON-formaat
    "type": "Polygon",
    "coordinates": [
                    [],[],[], ...
                  ]
  },

  "hoogte":                         // Aanvullende data
    {
      "waarde": 5,
      "eenheid": "meter + NAP"
    }
  ],

  "bron": [                         // Relevante metadata
    {
      "code": "BGT",
      "waarde": "12345678"
    }
  ],
  "origineelCoordinaatSysteem": "EPSG:4258"
}
```

Voorbeeld 26 - GEO-query resultaat

In het geval van een globale zoekvraag zoals:
`POST …/api/opvragen/v1/locaties/_zoek/`, is het noodzakelijk om de resultaten in een relevante geometrische context te plaatsen door bij iedere gevonden locatie een typeaanduiding en/of andere eigenschappen op te nemen.

> *API-G05* ***Resultaten van een globale geometrische zoekvraag worden in de relevante geometrische context geplaatst***
>
> In het geval van een globale zoekvraag (bijvoorbeeld: `…/api/oprvagen/v1/locaties/_zoek`) is het noodzakelijk om de resultaten in een relevante geometrische context te plaatsen, bijvoorbeeld door een typeaanduiding en/of andere eigenschappen op te nemen.

In het volgende voorbeeld wordt aangegeven hoe dit kan worden gerealiseerd:

```
POST https://.../api/opvragen/v1/locaties/_zoek/
{                                                                    // request-body
 "_embedded": {
    "resultaten": [
      {
        "type": "enkelbestemming",
        "_links": {
          "self": {
            "title": "Enkelbestemming 1234",
            "href": ".../enkelbestemmingen/1234"
          }
        }
      },
      {
        "type": "dubbelbestemming",
          "_links": {
            "self": {
              "title": "Dubbelbestemming 8765",
              "href": ".../dubbelbestemmingen/8765"
            }
         }
      }
    ]
  }
}
```

Voorbeeld 27 - GEO-query resultaat van een globale geometrische zoekvraag

CRS-negotiation

Het default coordinaatreferentiesysteem (CRS) van GeoJSON is WGS84. Dit is het globale coördinatenstelsel dat vanwege de verschuiving van de tektonische platen minder nauwkeurig is dan lokale coördinatenstelsels zoals ETRS89 (EPSG:4258, Europees) of RD New[13] (EPSG:28992, Nederlands).

Omdat de meeste client-bibliotheken met WGS84 werken, schrijft de W3C/OGC-werkgroep "Spatial Data on the Web" voor om dit standaard te ontsluiten. Dit kan direct op een kaart geplot worden zonder moeilijke transformaties. De API-strategie voorziet hierin door naast ETRS89 en RD New ook Pseudo-Mercator op basis van WGS84 te ondersteunen. De hierbij te gebruiken projectiemethode is EPSG:3856 op basis van WGS84.

API-G06 ***Het voorkeur-coordinaatreferentiesysteem (CRS) is ETRS89, maar het CRS wordt niet impliciet geselecteerd***

Algemeen gebruik van het Europese ETRS89 coördinatenstelsel (CRS) heeft sterk de voorkeur, maar dit wordt niet door API's afgedwongen als de "default". Derhalve moet het te gebruiken CRS in elke aanroep via de `Content-Crs` header expliciet worden opgegeven. Bij het ontbreken van deze header wordt `412 Precondition Failed` teruggegeven.

Het is mogelijk om het CRS voor vraag en antwoord via headers afzonderlijk te specificeren. Hierin zijn vervolgens drie opties (met voorgeschreven projecties) voorhanden: RD New, ETRS89 en WGS84.

API-G07 ***Het coordinaatreferentiesysteem (CRS) van de vraag en het antwoord worden in de response-header meegestuurd***

Het gekozen coördinatenstelsel (CRS) voor zowel de vraag als het antwoord moeten worden meegestuurd als onderdeel van de response-header. Hiermee wordt expliciet gemaakt in welk CRS zowel de vraag als het antwoord zijn uitgedrukt.

[13] RD New is een geprojecteerd CRS dat voor het laatst is herzien op 26 oktober 2019. Het is geschikt voor gebruik in Nederland, op land inclusief de Waddenzee, de Waddeneilanden en 12 mijl uit de kust. RD New vervangt EPSG28991 (RD Oud).

De hier genoemde headers zijn puur bedoeld voor de onderhandeling tussen de client en de server. Afhankelijk van de toepassing zal naast de geometrieën ook specifieke metadata-onderdeel vormen van het antwoord, bijvoorbeeld de oorspronkelijke realisatie inclusief, de inwindatum, etc.

Vraag en antwoord kunnen op een ander coördinatensysteem zijn gebaseerd. Hiervoor wordt het HTTP-mechanisme voor content negotiation gebruikt. Het CRS van de geometrie in de vraag (request body) wordt aangeduid met de header `Content-Crs`.

HTTP-header	Waarde	Toelichting
Content-Crs	EPSG:4258	ETRS89, Europees
Content-Crs	EPSG:28992	RD New, Nederlands
Content-Crs	EPSG:3856	WGS84/Pseudo-Mercator, Wereld (een geprojecteerd CRS)

Tabel 8 - Definities Content-Crs

Het gewenste CRS voor de geometrie in het antwoord (response body) wordt aangeduid met de header `Accept-Crs`.

HTTP-header	Waarde	Toelichting
Accept-Crs	EPSG:4258	ETRS89, Europees
Accept-Crs	EPSG:28992	RD New, Nederlands
Accept-Crs	EPSG:3856	WGS84/Pseudo-Mercator, Wereld (een geprojecteerd CRS)

Tabel 9 - Definities Accept-Crs

CRS-transformatie

Voor het transformeren tussen coördinaatreferentiesystemen is binnen de Rijksoverheid de RDNAPTRANS™ procedure en de benodigde certificering beschikbaar.

API-G08 ***Voor het transformeren tussen CRS-en wordt de geldende RDNAPTRANS™ procedure gebruikt***

Voor het transformeren tussen coördinaatreferentiesystemen wordt de geldende RDNAPTRANS™ procedure gebruikt. Dit om te zorgen dat er altijd een eenduidige transformatie plaatsvindt.

ⓘ *Binnen het stelsel wordt gebruik gemaakt van een generieke dienst op basis van de RDNAPTRANS™ procedure en de benodigde certificering.*

API-G09 ***Uitlevering van WGS84/Pseudo-Mercator is geen formele transformatie***

Voor de uitlevering via API's kan Pseudo-Mercator op basis van WGS84 worden ondersteund. Dit is een geprojecteerd CRS (EPSG:3856) en gebruikt de WGS84 geografische 2D CRS als basis-CRS voor de projectie.

ⓘ *Met de ondersteuning van WGS84 kan bijvoorbeeld beter worden voorzien in services en gebruikerstoepassingen en apps die aansluiten op Google Maps™.*

API-G10 ***Het gewenste coordinaatreferentiesysteem wordt op basis van content negotiation overeengekomen***

Het gewenste CRS voor de geometrie in het antwoord (response body) wordt aangeduid met een "Accept Header". Als het gewenste CRS niet geleverd kan worden zal er een `406 Not Acceptable` worden teruggegeven.

2.5.9 Tijdreizen

Informatie is onderhevig aan verandering. Tijdreizen is een mechanisme waarmee het mogelijk is om op standaard manier informatie op te vragen op een bepaald moment in de tijd. De drie belangrijkste tijdsmomenten vanuit het perspectief van tijdreizen via een API zijn:

Geldig	Dit is een tijdstip waarop de teruggegeven gegevens in de werkelijkheid geldig zijn.
Beschikbaar	Dit is een tijdstip waarop geldt dat de teruggegeven gegevens beschikbaar waren via dezelfde API.
In werking (getreden op)	Dit is een tijdstip waarop een besluit (of delen daarvan), dan wel de daarvan afgeleide gegevens (zoals de definitie van een begrip) juridische werking krijgt.

API-T01 ***Tijdreizen wordt ondersteund via gestandaardiseerde query-parameters***

Tijdreizen wordt in API's ondersteund met drie gestandaardiseerde query-parameters:

`geldigOp`	`YYYY-MM-DD`
`inWerkingOp`	`YYYY-MM-DD`
`beschikbaarOp`	`YYYY-MM-DDThh:mm:ss.s`

Het basisprincipe voor het tijdreizen i.r.t. de URI-strategie [5] is daarbij als volgt:

1. Indien er *geen* specifieke datum wordt meegegeven, dan wordt de *meest actueel geldige* informatie teruggegeven, zoals gebruikelijk op het internet. Indien er nog geen enkele geldige versie bestaat, dan wordt de meest actueel bekende versie teruggegeven;
2. Indien een specifieke datum wordt meegegeven, wordt deze meegegeven als query-parameter die onderdeel is van de URI.
3. Bij het opgeven van een specifieke datum wordt onderscheid gemaakt tussen:
   a. welke gegevens op die datum geldig waren (geldigOp);
   b. welke gegevens op die datum beschikbaar waren (beschikbaarOp);
   c. welke regels op die datum juridisch in werking waren getreden (inWerkingOp).

De waarden van de drie query-parameters in de URI zijn als volgt opgebouwd en gebaseerd op [RFC3339]:

`geldigOp`	`YYYY-MM-DD`
`inWerkingOp`	`YYYY-MM-DD`
`beschikbaarOp`	`YYYY-MM-DDThh:mm:ss.s`

Tabel 10 - Definitie tijdreis-parameters

`YYYY`	Viercijferig jaar
`MM`	Tweecijferige maand (01 = januari, enz.)
`DD`	Tweecijferige dag van de maand (01 tot en met 31)
`Hh`	Twee cijfers van het uur (00 tot 23) → (am / pm niet toegestaan)
`Mm`	Twee cijfers van de minuut (00 tot en met 59)
`Ss`	twee cijfers van de seconden (00 tot en met 59)
`S`	Eén of meer cijfers die een decimale fractie van een seconde vertegenwoordigen

Tabel 11 - Legenda datum- en tijdformaat

API-T02 ***De waarden van de query-parameters voor tijdreizen volgen RFC3339***

De waarden van de query-parameters `geldigOp`, `beschikbaarOp` en `inWerkingOp` zijn opgebouwd zoals beschreven in de Stelselafspraken [7] en volgen [RFC3339].

In de URI-strategie [5], de notitie Tijdreizen [6] en de Stelselafspraken [7] wordt meer context gegeven en zijn tevens voorbeelden te vinden waarbij de benoemde parameters worden gebruikt.

Mate van ondersteuning
Tijdreizen is een optioneel mechanisme met optionele features. Het kan voorkomen dat:

- tijdreizen in het geheel niet wordt ondersteund;
- het begrip inwerkingtreding niet wordt ondersteund;
- tijdreizen naar de toekomst niet wordt ondersteund.

Bij de verwerking van tijdreisvragen dient de afnemer geïnformeerd te worden over ongeldige/niet ondersteunde verzoeken. In de volgende specifieke gevallen wordt de bijbehorende statuscode en toelichting teruggegeven:

Wat wordt niet ondersteund?	http-statuscode en toelichting

Tijdreizen	`HTTP/1.1 400`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`"type": "…/fout/id/concept/NietOndersteund_Parameter",`
	`"title": "{Het verzoek is begrepen, maar één of meer parameters worden niet ondersteund}",`
	`"status": 400,`
	`"invalid-params": [`
	`{`
	`"type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietOndersteund_Parameter ",`
	`"name": "beschikbaarOp",`
	`"reason": "De parameter beschikaarOp wordt niet ondersteund."`
	`}, →`
	`],`
	`"instance": "urn:uuid:4017fabc-1b28-11e8-accf-0ed5f89f718b"`
	`}`

Inwerkingtreding	`HTTP/1.1 400`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`"type": "…/fout/id/concept/InWerkingTredingNietOndersteund_Parameter",`
	`"title": "{Het verzoek is begrepen, maar de tijdreisparameter 'inWerkingOp' wordt niet ondersteund}",`
	`"status": 400,`
	`"invalid-params": [`
	`{`
	`"type": "https://content.omgevingswet.overheid.nl/fout/id/concept/InWerkingTredingNietOndersteund_Parameter ",`
	`"name": "InWerkingOp",`
	`"reason": "De parameter InWerkingOp wordt niet ondersteund."`
	`}, →`
	`],`
	`"instance": "urn:uuid:4017fabc-1b28-11e8-accf-0ed5f89f718b"`
	`}`

Toekomst	`HTTP/1.1 400`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`"type": "…/fout/id/concept/ToekomstNietOndersteund_Parameter",`
	`"title": "{Het verzoek is begrepen, maar tijdreisparameters mogen geen tijdstip in de toekomst bevatten" }",`
	`"status": 400,`
	`"invalid-params": [`
	`{`
	`"type": "https://content.omgevingswet.overheid.nl/fout/id/concept/ToekomstNietOndersteund_Parameter ",`
	`"name": "geldigOp",`
	`"reason": "De parameter geldigOp mag geen datum in de toekomst zijn."`
	`}, →`
	`],`
	`"instance": "urn:uuid:4017fabc-1b28-11e8-accf-0ed5f89f718b"`
	`}`

Tabel 12 - Definitie foutafhandeling voor niet ondersteunde parameters

API-T03 ***Het tijdreis-mechanisme van een API is robuust en houdt rekening met de optionele parameters***

Bij de verwerking van tijdreisvragen dient ook rekening te worden gehouden met de optionele parameters. Het kan voorkomen dat:

- tijdreizen in het geheel niet wordt ondersteund;
- het begrip inwerkingtreding niet wordt ondersteund;
- tijdreizen naar de toekomst niet wordt ondersteund.

Voor de beschreven gevallen wordt de gedefinieerde statuscode en toelichting teruggegeven.

Robuustheid

Bij de verwerking van tijdreisvragen dient ook rekening te worden gehouden met het ontbreken van historie. In dit specifieke geval wordt de bijbehorende statuscode en toelichting teruggegeven:

Soort verzoek	http-statuscode en toelichting

Specifieke resource	`HTTP/1.1 404`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`  "type": "…/fout/id/concept/BestaatNiet",`
	`  "title": "De gevraagde versie bestaat niet.",`
	`  "status": 404,`
	`  "detail": "Versie 2017-01-01 van regeling 123 bestaat niet.",`
	`  "instance": "urn:uuid:4017fabc-1b28-11e8-accf-0ed5f89f718b"`
	`}`

Tabel 13 - Definitie foutafhandeling/response voor tijdreisvragen zonder historie

API-T04 ***Het tijdreis-mechanisme van een API is robuust en houdt rekening met het ontbreken van historie***

Bij de verwerking van tijdreisvragen dient ook rekening te worden gehouden met het ontbreken van historie. Voor de beschreven situaties wordt de gedefinieerde statuscode en toelichting teruggegeven.

Bij de verwerking van tijdreisvragen kan ook rekening te worden gehouden met het ondersteunen van vragen naar het aantal beschikbare voorkomens in een resource-collectie. In dit specifieke geval wordt de volgende bijbehorende respons teruggegeven:

Soort verzoek	http-statuscode en toelichting

Resource-collectie	`GET …/opvragen/v1/activiteiten/123/voorkomens?beschikbaarOp=2019-01-01`
	
	`HTTP/1.1 200`
	`Content-Type: application/json+hal`
	`{`
	`  "_embedded": {`
	`  "voorkomens": [`
	`  {`
	`  "_links": {`
	`  "self": {`
	`  "href": "…/v1/actviteiten/123?geldigOp=2018-12-31"`
	`  }`
	`  },`
	`  "naam": "Exploiteren horeca"`
	`  },`
	`  {`
	`  "_links": {`
	`  "self": {`
	`  "href": "…/v1/actviteiten/123?geldigOp=2017-01-01"`
	`  }`
	`  },`
	`  "naam": "Horecabedrijf exploiteren"`
	`  }, →`
	`  ]`
	`  }`
	`  }`

Tabel 14 - Respons voor tijdreisvraag met beschikbare voorkomens

2.5.10 API-profielen

Met een API-profiel [12] kan een API-provider aangeven dat de API een standaard bevragingskoppelvlak ondersteunt. Dit is relevant in een context waarin één afnemer met veel aanbieders (en API's) moet werken en dynamisch moet kunnen bepalen of een aanbieder het vereiste bevragingskoppelvlak ondersteunt, ofwel kan zoeken naar beschikbare relevante API's.

API-A01 ***API's die een standaard bevragingskoppelvlak moeten aanbieden, doen dit op basis van het API-profiel mechanisme***

Het API-profiel mechanisme is een manier om dynamisch te werken met één of meer standaard bevragingskoppelvlakken. Een API-profiel schrijft letterlijk voor in welke vorm de vraag wordt gesteld en hoe het antwoord wordt verwacht. Aanbieders kunnen aangeven dat de API een profiel ondersteunt en afnemers kunnen aanbieders hierop selecteren.

Een API-profiel schrijft letterlijk voor in welke vorm de vraag wordt gesteld en hoe het antwoord wordt verwacht. API's kunnen zich aan één of meer profielen conformeren en dit centraal kenbaar maken.

API-A02 ***Een API-profiel heeft een unieke naam en de specificatie wordt vastgelegd op basis Open API Specification (OAS) 3.0 of hoger***

Een API-profiel wordt met een unieke naam geregisteerd. Iedere registratie (zoals "voorinvullen" [13], "orienteren" of "x-metadata") verwijst naar de bijbehorende specificatie op basis van de Open API Specification (OAS) 3.0 of hoger.

API-A03 ***De registratie van API-profielen is opgenomen in de Stelselcatalogus***

Een API-profiel wordt onder een unieke naam geregistreerd in de Stelselcatalogus. De registratie verwijst naar de Open API Specification in JSON-formaat. De URI van een API-profiel dient als volgt te worden gemunt:

`http://standaarden.omgevingswet.overheid.nl/api-profiel/id/concept/{naam}`

Concrete voorbeelden:

`http://standaarden.omgevingswet.overheid.nl/api-profiel/id/concept/voorinvullen`
`http://standaarden.omgevingswet.overheid.nl/api-profiel/id/concept/orienteren`
`http://standaarden.omgevingswet.overheid.nl/api-profiel/id/concept/x-metadata`

Alle API-profielen zijn opgenomen in een waardelijst met de volgende URI:

`http://standaarden.omgevingswet.overheid.nl/id/waardelijst/Api-profiel`

API-A04 ***Een API kan zich conformeren aan één of meer API-profielen door dit centraal te registeren***

Een API conformeert zich aan een API-profiel door dit centraal te registeren. De registratie bestaat uit de id (URI) van het profiel en de URI van een "endpoint" voor het standaard bevragingskoppelvlak dat beschikbaar wordt gesteld.

API-A05 ***API's die API-profielen implementeren zijn opgenomen in de Stelselcatalogus***

Een API conformeert zich aan API-profiel door dit in de Stelselcatalogus re registeren als onderdeel van de aansluitinformatie. Dit betreft de metadata die een informatieleverancier dient vast te leggen conform de geldende aansluitvoorwaarden.

Een API-aanbieder die een API-profiel implementeert stelt resources beschikbaar voor:

- Introspectie [12]: op basis van (zelf)reflectie de eigen structuur en diensten kenbaar maken;
- Query-projectie [12]: op basis van een zoekvraag met een projectie de gevonden resultaten (op maat) beschikbaar stellen.

2.5.11 Caching

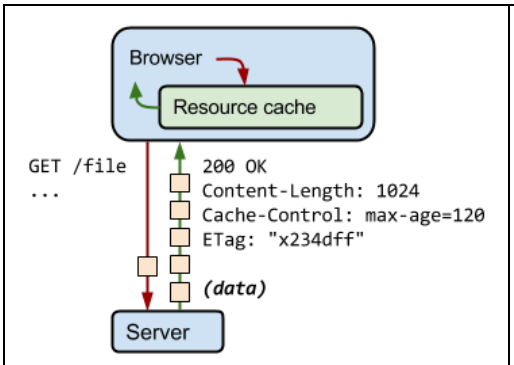
Voor sommige resources kan het nuttig zijn om caching toe te passen. HTTP biedt voor caching standaard mechanismes aan. Door deze mechanismes te gebruiken kan op een eenvoudig manier gebruik worden gemaakt van de standaard caching oplossingen in de client en in de infrastructuur. Dit is één van de grote voordelen van REST-API's.

Het enige wat nodig is om hiervan gebruik te maken is:

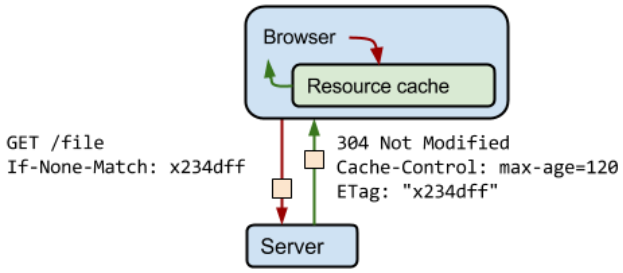
- Een aantal extra uitgaande HTTP-headers toevoegen;
- Functionaliteit om een aantal specifieke inkomende HTTP-headers af te handelen.

Er zijn 2 manieren om caching te realiseren: `ETag en Last-Modified`.

ETag
Een ETag (Entity Tag) is een hashcode of checksum van een resource. Als de resource wijzigt ontstaat een andere ETag. Een ETag is dus uniek voor een bepaalde versie van een resource. De ETag wordt als de HTTP-header `ETag` teruggegeven met de resource. De afnemer bewaart de resource en de ETag in de cache. Als de afnemer dezelfde resource opvraagt dan stuurt deze de ETag mee in de HTTP-header `If-None-Match`. De server controleert of de ETag in de HTTP-header `If-None-Match` gelijk is aan de eigen ETag. Indien dit het geval geeft de server een HTTP-statuscode `304 Not Modified` terug. De afnemer laadt dan de resource uit de eigen cache.



De server geeft aan dat het antwoord gedurende 120 seconden in het cachegeheugen mag blijven.



De client verstrekt automatisch een ETag-verzoek en weet indien de server antwoordt met "`not modified`" dat de cache nog valide is.

Figuur 9 - Werking http-cache "besturing" m.b.v. ETag

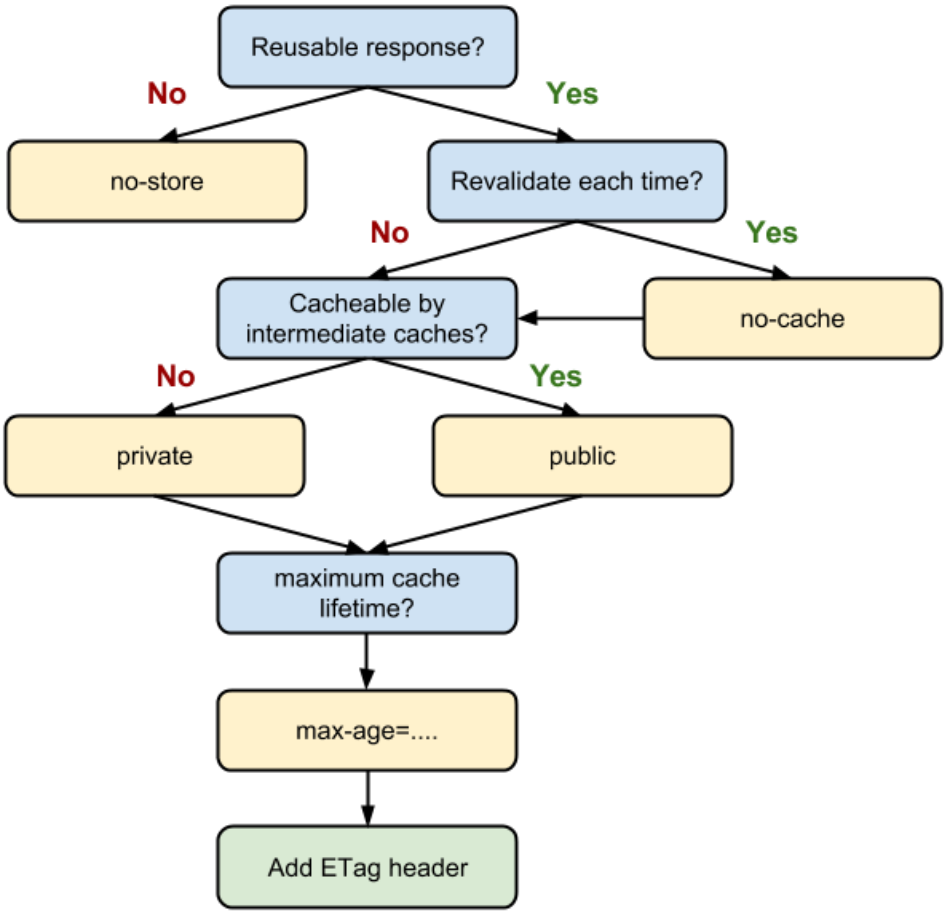
Dit gaat alleen op over client-side caching, dus is alleen van toepassing als de client ook daadwerkelijk de request headers meestuurt, anders altijd een 200 OK.

Last-Modified
Dit werkt in principe net als ETag, behalve dat het gebruik maakt van tijdstempels. De HTTP-header `Last-Modified` bevat een tijdstempel in het [RFC1123] formaat die wordt gevalideerd tegen de HTTP-header `If-Modified-Since`. De server controleert of de resource gewijzigd is sinds het aangegeven tijdstip. Indien dit niet het geval is geeft de server een HTTP-statuscode `304 Not Modified` terug. De afnemer laadt dan de resource uit de eigen cache. Dit gaat alleen op over client-side caching, dus is alleen van toepassing als de client ook daadwerkelijk de request headers meestuurt, anders altijd een 200 OK.

API-B37 ***Waar relevant wordt caching toegepast via standaard http-mechanismen***

Voor caching wordt gebruikt van de HTTP standaard caching mechanismes door het toevoegen van een aantal extra uitgaande HTTP-headers (`ETag` of `Last-Modified`) en functionaliteit om te bepalen of een aantal specifieke inkomende HTTP-headers (`Is-None_Match` of `Is-Modified-Since`).

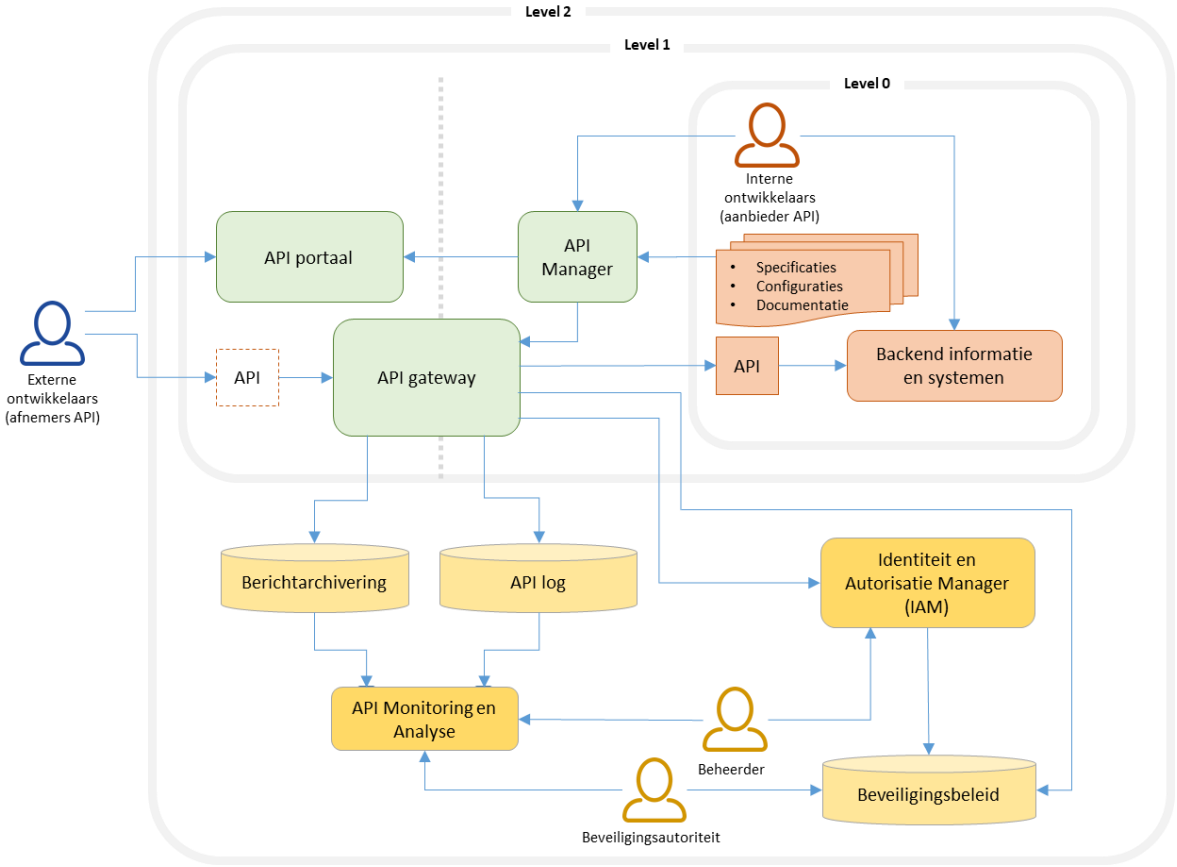
De ideale strategie is om zoveel mogelijk antwoorden zo lang mogelijk op de client op te slaan en voor elk antwoord een validatietoken te verstrekken voor efficiënte revalidatie. Voor het bepalen van het optimale cache-control-beleid kan echter het beste de beslisboom in Figuur 10 - Beslisboom voor het bepalen van een optimaal cache-beleid worden gebruikt.



Figuur 10 - Beslisboom voor het bepalen van een optimaal cache-beleid

2.5.12 Ecosysteem en ecosysteem-bewustzijn (context awareness)

In een stelsel staan API's niet op zichzelf en zelfs in één logisch koppelvlak kan sprake zijn van meerdere samenhangende API's. Dit is inherent aan het principe 'alles is een service'. API's maken dus onderdeel uit van een ecosysteem. Binnen deze context kunnen API's en API-providers ook vanuit het ecosysteem worden ontzorgd. Dat kan met ondersteunende voorzieningen en op verschillende niveaus. In Figuur 11 zijn er drie weergegeven: level 0, 1 en 2. DSO richt zich minimaal op level 2.



Figuur 11 - Globale weergave API-ecosysteem: level 0, 1 en 2

Wanneer API's een keten vormen of samen een koppelvlak realiseren, is bewustzijn van de omgeving van belang. Dit betekent onder andere dat applicaties op ieder moment:

- kunnen weten (als afnemer) en laten weten (als aanbieder via een API) dat de dienst geleverd kan worden;
- kunnen weten (als afnemer) en laten weten (als aanbieder van een API) wat en hoe de geleverd kan worden;
- kunnen signaleren en melden dat er een (potentieel) probleem is.

API-E06 ***API's dragen bij aan het ecosysteem-bewustzijn (context awareness)***

In een stelsel staan API's niet op zichzelf. Wanneer API's een keten vormen of samen een koppelvlak realiseren dragen ze bij aan het ecosysteem-bewustzijn. API's dragen daarom actief bij aan "context awareness" door het beschikbaar stellen van metadata over:

- De applicatie-eigenschappen, zoals: naam, release-informatie (een versienummer), een beschrijving en de gebruikte standaarden met versienummers;
- De applicatiestatus, te beginnen bij een eenvoudige waarde als: "UP/DOWN".

Om te zorgen dat afnemers van API's op een voorspelbare en uniforme manier gebruik kunnen maken van de genoemde metadata is het noodzakelijk dat er twee extra metadata "endpoints" beschikbaar worden gesteld stellen:

Soort metadata	Standaard metadata "endpoints"
De applicatie-eigenschappen	`https://.../api/opvragen/v1/`**`app-info`**
De applicatiestatus	`https://.../api/opvragen/v1/`**`app-health`**

Tabel 15 - Definitie API-metadata "endpoints"

API-E07 ***API's stellen een metadata "endpoint" `app-info` beschikbaar voor het opvragen van de actuele applicatie-eigenschappen***

Een API draagt actief bij aan "context awareness" door het beschikbaar stellen van metadata over de applicatie-eigenschappen via een standaard metadata "endpoint":

`https://.../api/opvragen/v1/`**`app-info`**

De applicatie-eigenschappen worden geretourneerd in de vorm van een object waarin alle relevante informatie is gegroepeerd. De inhoud is uitbreidbaar, maar bestaat minimaal uit de naam, release-informatie (een versienummer), een beschrijving en de gebruikte standaarden met versienummers.

```
{
  "app-info": {
    "name": "Objectgericht Ontsluiten Omgevingsdocumenten (Ozon)",
    "version": "1.0",
    "description": "Ozon levert functionaliteit ten behoeve van domeinspecifieke validaties,
                    waaronder geometrie-validaties, objectvorming en objectregistratie,
                    objectgerichte uitlevering van omgevingsdocumenten, uitlevering van totaalstanden
                voor specifieke toestanden van een regeling",
    "stopVersion": "0.98-KERN",
    "imowVersion": "0.98.3-KERN"
  }
}
```

Voorbeeld 28 - JSON-response van metadata-verzoek voor de actuele applicatie-eigenschappen

API-E08 ***API's stellen een metadata "endpoint" `app-health` beschikbaar voor het opvragen van de actuele applicatiestatus***

Een API draagt actief bij aan "context awareness" door het beschikbaar stellen van metadata over de applicatiestatus via een standaard metadata "endpoint":

`https://.../api/opvragen/v1/`**`app-health`**

De applicatie-status worden geretourneerd in de vorm van een object. De inhoud is uitbreidbaar, maar bestaat minimaal uit een veld "status" met de waarde: "UP" of "DOWN".

```
{
  "app-health": {
    "status": "UP"
  }
}
```

Voorbeeld 29 - JSON-response van metadata-verzoek voor de actuele applicatiestatus

2.5.13 Documentatie

Een API is zo goed als de bijbehorende documentatie. De documentatie moet gemakkelijk te vinden, te doorzoeken en publiekelijk toegankelijk zijn. De meeste ontwikkelaars zullen eerst de documenten doornemen voordat ze starten met de implementatie.

API-B38 ***Documentatie is gebaseerd op OAS 3.0 of hoger***

Specificaties (documentatie) is beschikbaar als Open API Specification (OAS) V3.0 of hoger.

API-B39 ***Technische documentatie (OAS) voorziet in kruisverwijzingen naar documentatie in algemene zin en voor andere doelgroepen***

API's staan meestal niet op zichzelf. Documentatie voorziet daarom in kruisverwijzingen van technische documentatie (OAS) naar documentatie in algemene zin en voor ander doelgroepen.

API-B40 ***OAS is via het "(root) endpoint" van de API beschikbaar in JSON-formaat***

Om te zorgen dat de actuele documentatie altijd publiekelijk toegankelijk is, dient de Open API Specification (OAS) via het "root endpoint" van de API beschikbaar te zijn in JSON-formaat. Hiermee ontstaat een directe koppeling met de actuele documentatie.

```
https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v1
```
Maakt de OAS behorend bij v1 van deze API beschikbaar.

API-B41 ***Documentatie is in het Nederlands tenzij er sprake is van bestaande documentatie in het Engels of er sprake is van een officieel Engelstalig begrippenkader***

De voertaal voor de API's is Nederlands. Het is wel toegestaan om te verwijzen naar bestaande documentatie in het Engels en als er sprake is van een officieel Engelstalig begrippenkader.

De documentatie dient te zijn voorzien van werkende voorbeelden, dus inclusief complete request- en respons-cycli. Het moet mogelijk zijn om direct vanuit de documentatie verzoeken te testen (uit te voeren). Daarnaast is elke fout beschreven en voorzien van een unieke foutcode die gebruikt kan worden om de fout op te zoeken.

API-B42 ***Documentatie wordt getest en geaccepteerd***

Bij het opzetten van de documentatie dient rekening te worden gehouden met de mogelijkheid om te testen, bijvoorbeeld in een ontwikkelaarsportaal. Dit moet worden getest en geaccepteerd.

Wanneer een API in productie is, mag het "contract" niet zonder voorafgaande kennisgeving worden gewijzigd. De documentatie moet zijn voorzien van een uitfaseringsplanning (deprecation-schedule) en alle details van de wijziging bevatten. Wijzigingen worden via een publiek toegankelijke website of blog als changelog bekendgemaakt of "ge-pushed", bijvoorbeeld via een mailinglijst.

API-B43 ***Wijzigingen worden gepubliceerd met een uitfaseringschema***

Wijzigingen in API's worden met de bijbehorende planning op een publiek toegankelijke website of blog als changelog bekendgemaakt. Bij voorkeur wordt de bekenmaking ook ge-pushed naar bekende afnemers.

API-E09 ***Het Knooppunt biedt de mogelijkheid om informatie naar bekende afnemers van een API te "pushen"***

Het Knooppunt houdt een registratie bij van de afnemers van een API en biedt de mogelijkheid om een berichten to "pushen".

ⓘ Nuttig voor het "pushen" van informatie over releases, gepland onderhoud en incidenten.

2.5.14 Versionering

API's zijn altijd geversioneerd. Versioneren zorgt voor een soepele overgang bij wijzigingen. De oude en nieuwe versies worden voor een beperkte overgangsperiode (6 maanden tot één jaar) aangeboden. Er worden bovendien maximaal 3 versies van een API ondersteund, waarvan twee major versies. Afnemers kiezen zelf het moment dat ze overschakelen van de oude naar de nieuwe versie van een API, als ze het maar voor het einde van de overgangsperiode is.

> *API-B44* ***De overgangsperiode bij een nieuwe API-versie is maximaal 1 jaar***
>
> Oude en nieuwe versies (max. 3) van een API worden voor een beperkte overgangsperiode (maximaal tot 1 jaar) naast elkaar aangeboden, waarvan twee major versies.

Er zijn verschillende meningen over de vraag of de versie in de URI of in de header hoort. De URI-strategie [5] heeft hier een keuze in gemaakt: alleen het major versienummer wordt in de URI opgenomen. Hierdoor is het mogelijk om verschillende versies van een API via de browser te verkennen. Hiermee wordt ook voldaan aan het derde punt van de vijf basiseisen in paragraaf 2.2.

`https://.../api/verzoeken/v1/indienen/12`	Vraagt via API v1 ingediend verzoek #12 op

Het versienummer begint bij 1 en wordt met 1 opgehoogd voor elke major release waarbij het koppelvlak niet backward compatible wijzigt. De major, minor en patch versienummers staan altijd in de response-header van het bericht en wel in het volgende formaat: `major.minor.patch`. De header (zowel request als response) is als volgt gedefinieerd:

HTTP-header	Toelichting
API-Version	Geeft een specifieke API-versie aan in de context van een specifieke aanroep. Bijvoorbeeld: `API-version: 1.2.56`

Tabel 16 - Definitie API-Version header

Via de optionele request-header kan één minor/patch-versie worden aangewezen. Hiermee wordt bedoeld dat bijvoorbeeld in een (pre)productie- of aansluitomgeving naast bijvoorbeeld v1 en v2, ook nog de mogelijkheid bestaat om één "oudere" of juist een "nieuwere" minor/patch-versie van deze API's aan te wijzen. Deze versie is dan via de request-header te benaderen. De onderstaande URI's wijzen bijvoorbeeld naar de door de aanbieder aangewezen release van de API die alleen bereikbaar zijn via de URI:

`https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v1` `API-version: 1.0.2` **`(response-header)`**	V1.0.2
`https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v2` `API-version: 2.1.0` **`(response-header)`**	V2.1.0

Voorbeeld 30 - Werking API-version response-header

Het weglaten van de request-header `API-version` selecteert altijd de door de aanbieder aangewezen versie. Indien er voor V2 ook één andere minor/patch-versie beschikbaar is, met bijvoorbeeld versienummer V2.1.1, dan kan deze via hetzelfde "endpoint" worden aangeboden en worden geselecteerd door het meegeven van de juiste request-header:

```
API-version: 2.1.1 (request-header)
https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v2
```

V2.1.1

```
API-version: 2.1.1 (response-header)
```

```
Geen header
https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v2
```

V2.1.0

```
API-version: 2.1.0 (response-header)
```

Voorbeeld 31 - Werking API-version request-header

API-B45 ***Alleen het major versienummer is onderdeel van de URI***

In de URI wordt alleen het major versienummer opgenomen. Minor versienummer en patch versienummer worden in de header van het bericht zelf opgenomen. Het uitgangspunt dat hierbij wordt gehanteerd is dat minor versies en patches geen impact hebben op bestaande code, maar major versies geldt dit wel.

ⓘ *Een API is backward compatible wanneer er sprake is van bijvoorbeeld een extra resource of één of meer nieuwe optionele parameters.*

Een API zal nooit helemaal stabiel zijn. Verandering is onvermijdelijk. Het is belangrijk hoe met deze verandering wordt omgegaan. Goed gedocumenteerde en tijdig gecommuniceerde uitfaseringsplanningen zijn in het algemeen voor veel API-gebruikers werkbaar.

Uitfaseren van een major API-versie

Major-releases van API's zijn altijd "backward incompatible", ofwel bevatten "breaking changes". Immers, als een nieuwe release van de API niet tot "breaking changes" leidt, is er geen reden om een hele versie omhoog te gaan en spreken we van een minor release. Op het moment dat er een major release plaatsvindt, is het de bedoeling dat alle (potentiële) afnemers deze nieuwe versie implementeren. Bestaande afspraken kunnen echter niet van de een op de andere dag worden aangepast, dit geldt zeker voor het overschakelen van een oude naar een nieuwe versie van een API. Daarom is het noodzakelijk om na de livegang van de nieuwe versie óók de oude versie een tijdje in de lucht te houden. Omdat de oude versie niet tot in de eeuwigheid onderhouden kan worden, moeten alle afnemers worden gestimuleerd om de nieuwe versie te gaan gebruiken. Daarom is het van belang om te communiceren hoe lang de periode is waarin afnemers de gelegenheid krijgen om hun applicatie aan te passen en aan te sluiten op de nieuwe versie. Deze periode wordt vaak de "deprecation period" genoemd. De lengte van deze periode kan verschillen per API, vaak is dit zes maanden, maar standaard niet meer dan één jaar. Met het oog op beheersbaarheid is het ten zeerste aan te bevelen om maximaal twee major versies (waarvan één de deprecated variant) naast elkaar te draaien. In deze fase is communicatie met clients van de oude versie cruciaal. De volgende zaken moeten worden gecommuniceerd:

- Een link naar de (documentatie van de) nieuwe versie van de API;
- De "deprecation period" met exacte datum waarop de "deprecated" versie offline wordt gehaald;
- Welke features er toegevoegd, gewijzigd of verwijderd worden;

- Welke wijzigingen de huidige implementaties kunnen breken;
- Een migratieplan om eenvoudig over te stappen naar de nieuwe versie;
- Contactmogelijkheid om een verlenging van de "deprecation" periode aan te vragen.

De genoemde zaken dienen bij voorkeur gecommuniceerd te worden via de volgende kanalen:

- Duidelijk leesbaar in de API-documentatie van de oude versie;
- Per e-mail naar de afnemers (indien bekend);
- Via een push-bericht (voor afnemers die zich daarvoor hebben aangemeld);
- Via een topic op een website of via relevante fora;
- Met een Warning response-header in alle responses van de oude API.

Stap voor stap betekent dit het volgende:

1. Lanceren van de nieuwe versie (publiciteit);
2. Bepalen van de "deprecation period";
3. Schrijven van een migratieplan;
4. Communiceren via de API-documentatie van de oude versie;
5. De "deprecation period" communiceren per e-mail, websites, fora en eventuele andere kanalen;
6. Warning header toevoegen aan responses van de oude versie;
7. Logs regelmatig uitlezen om het gebruik van de oude versie te monitoren gedurende de "deprecation period";
8. Het "endpoint" van de oude versie "uit zetten" op geplande datum en feedback monitoren;
9. Indien er binnen twee weken geen feedback op de oude versie komt kan de oude versie (inclusief de documentatie) gearchiveerd worden.

De Warning response-header
De Warning header [RFC7234] die hier wordt voorgeschreven heeft warn-code 299 ("Miscellaneous Persistent Warning") en het API "endpoint" (inclusief versienummer) als de "warn-agent" van de "warning", gevolgd door de "warn-tekst" met de mensleesbare waarschuwing. Voorbeeld:

```
Waarschuwing: 299 https://service.../api/.../v1 "Deze versie van de API is verouderd en
zal uit dienst worden genomen op 2022-01-01. Raadpleeg voor meer informatie hier de
documentatie: https://omgevingswet.../api/.../v1".
```

Gebruikers moeten voldoende tijd hebben om de oude API uit te faseren. Een periode van 6 tot 12 maanden wordt aanbevolen. Voor semantische standaarden wordt een maximale overgangstermijn van 2 jaar gehanteerd, dus het kan voorkomen dat de oude API's in deze context ook langer in de lucht moeten blijven.

API-B46 ***Gebruikers van een 'deprecated' API worden actief gewaarschuwd***

Met een Warning response-header [RFC7234] in alle responses van de oude API's worden gebruikers actief gewaarschuwd voor de aanstaande uitfasering.

2.5.15 Ontwerppatronen

Een ontwerppatroon is een generieke oplossing voor een terugkerend probleem. Het patroon geeft geen concrete oplossing, maar biedt een sjabloon waarmee het ontwerpprobleem kan worden aangepakt. In de context van het digitaal stelsel wordt grootschalig ingezet op API's en is zinvol om gebruik te kunnen maken van een reeks specifiek voor REST-API's ontwikkelde ontwerppatronen. De ontwerppatronen die hier worden beschreven zijn nuttig voor het realiseren van betrouwbare, schaalbare en veilige toepassingen. Daarnaast is binnen het ecosysteem voor bepaalde patronen reeds voorzien dat ze centraal zullen worden ondersteund, onder andere door het Knooppunt van DSO-LV. Elk patroon beschrijft het probleem dat met het patroon wordt opgelost, overwegingen voor het toepassen van het patroon en een voorbeeld.

RESTful asynchroon verzoek/antwoord patroon

Context en probleem

In veel zogenaamde "system-to-system" koppelingen draait het om informatieoverdracht. Het komt echter ook voor dat via zo'n koppeling complexe taken worden uitbesteed.

Het is niet ongewoon dat een backend-systeem er minuten of zelfs uren over doet om een taak uit te voeren en dat er sprake kan zijn van een wachtrij of menselijke interventie. In zo'n geval is het niet haalbaar om te wachten tot het werk is voltooid. Een standaard RESTful-aanpak op basis van synchrone (http-request/response) verwerking levert in een dergelijke context al snel problemen op, want het resulteert in time-outs op en het is niet schaalbaar.

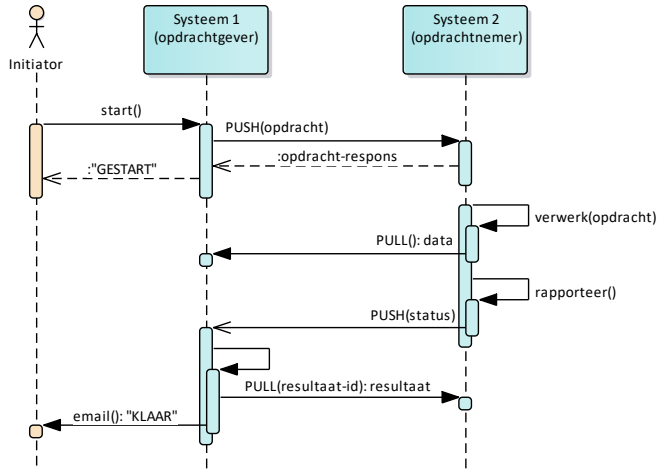
Oplossing

Een standaardoplossing voor dit probleem is het "opknippen" in losse verzoek- en antwoordberichten, ofwel asynchrone verwerking. Dit patroon kan echter in verschillende varianten worden uitgewerkt, waaronder de volgende drie:

	Opdracht	**Data** (opdracht)	**Status**	**Data** (resultaat)
Variant 1	*Push*		*Pull*	
Variant 2	*Push*		*Push*	
Variant 3	*Push*	*Pull*	*Push*	*Pull*

Tabel 17 - Overzicht met varianten van het verzoek/antwoord patroon

De 3e variant wordt hieronder uitgewerkt op basis van RESTful-aanpak. Met deze variant is er sprake van een extra scheiding, namelijk tussen de overdracht van de metadata en de data. Dit geldt bovendien voor zowel het verzoek als het antwoord.



Figuur 12 - Visualisatie asynchroon verzoek/antwoord patroon

Iedere PUSH en PULL is hier een API-call. Dat betekent dus dat zowel de opdrachtgever als de opdrachtnemer een REST-API beschikbaar stelt. Deze API's werken globaal als volgt:

STAP 1 De opdrachtgever doet een "push" van de opdracht (metadata) naar de opdrachtnemer:

```
{
  "identificatie": "Mijn opdracht",
  "checksumAlgoritme": "SHA256",
  "checksum": "7b7315b4918ad2e2b5d28859194be8d13ad0849f165458f06d670f07d8005f17",
  "grootte": 74260,
  "dataUrl": "https://.../opdrachtgever/data/mijn-opracht.zip",
  "responsUrl": "https://.../opdrachtgever/api/taakuitbesteder/v1/resultaten"
}
```

Voorbeeld 32 - Push van opdracht naar verzoek/antwoord API opdrachtnemer

STAP 1a De opdrachtgever krijgt direct een respons (met o.a. een status-URL):

```
{
  "tijdstempel": "2019-04-12T23:20:50.52Z",
  "status": 0, // ontvangen, voorbeelden van andere waarden zijn: 1) waarschuwing, 2 fout, etc.
  "fout": "Geen",
  "bericht": "Opdracht succesvol ontvangen",
  "statusUrl": "https://.../opdrachtnemer/api/taakverwerker/v1/statussen/opdracht1234567"
}
```

Voorbeeld 33 - Opdrachtrespons van de verzoek/antwoord API van de opdrachtnemer

Stap 2 De opdrachtnemer start de verwerking en doet een "pull" van de opdracht (data):

Dit is feitelijk niets meer dan een GET naar de `dataUrl` die door de opdrachtgever is meegestuurd met de opdracht. In dit voorbeeld een ZIP-bestand, maar dit kan ook een ander bestand of een JSON-respons zijn.

Stap 3 De opdrachtnemer is klaar en maakt een rapportage en doet een "push" van de status.

Dit betekent een POST naar de API van de opdrachtgever door gebruik te maken van de bij de opdracht meegegeven `responseUrl`:

```
{
  "identificatie": "resultaat1234567",
  "referentie": "Mijn opdracht",
  "status": "Gereed",
  "meldingen": [
     {
       "code": "200",
       "omschrijving": "Uw opdracht is succesvol verwerkt.",
       "detail": "Het resultaat van uw opdracht staat klaar en kan worden opgehaald."
     }
   ]
}
```

Voorbeeld 34 - Push van resultaat naar de verzoek/antwoord API van de opdrachtgever

Stap 4 De opdrachtgever doet een "pull" van het resultaat:

Dit is feitelijk niets meer dan een GET naar de API van de opdrachtnemer door gebruik te maken van de identificatie die is ontvangen in Stap3:

`GET https://.../opdrachtnemer/api/taakverwerker/v1/resultaten/resultaat1234567`

Beperkingen en overwegingen

Omdat in de oplossing gekozen is om te werken met een "callback" API, is deze specifieke uitwerking van dit patroon niet geschikt voor gebruikerstoepassingen.

Toepassingsgebied

Dit patroon is vooral goed inzetbaar wanneer een systeem- en/of stelselonderdeel regelmatig complexe verwerkingsopdrachten uitbesteedt aan een ander systeem. Het systeem dat uitbesteed heeft niet direct het resultaat nodig, maar wil wel direct door het andere systeem genotificeerd worden zodra een opdracht is voltooid of is afgebroken.

API-O01 ***API's die asynchrone verwerking ondersteunen volgen het beschreven "RESTful asynchroon verzoek/antwoord patroon"***

Wanneer een API-aanbieder diensten levert die zijn gebaseerd op asynchrone verwerking, wordt hiervoor het "RESTful asynchroon verzoek/antwoord patroon" gebruikt. Specifiek gaat het hier om een patroon waarin niet alleen de opdracht en het resultaat zijn gesplitst, maar ook de metadata- en de data-overdacht van elkaar zijn gescheiden.

RESTful pub/sub patroon

Context en probleem

Het digitaal stelsel bestaat uit een grote verzameling gedistribueerde toepassingen met onderdelen die elkaar regelmatig gericht informatie moeten verstrekken als bepaalde gebeurtenissen plaatsvinden. Hierbij valt te denken aan:

- Nieuwe informatie die beschikbaar komt;
- Bestaande informatie die wijzigt of vervalt;
- Een toestand die vraagt om directe opvolging (eventueel inclusief ontvangstbevestiging).

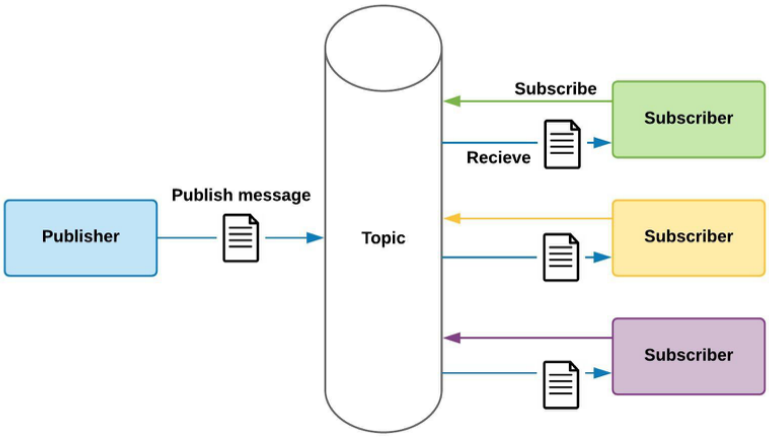
Vaak wordt dit opgelost door via API's periodiek de status op te bevragen (pollen) of door gebruik te maken van lokaal beschikbare middleware. Dit levert potentieel de volgende problemen op:

- De betrokken toepassingen worden gekoppeld en daardoor van elkaar afhankelijk;
- De middleware beperkt het toepassingsbereik (beperkte interoperabiliteit);
- De betrouwbaarheid en schaalbaarheid is sterk afhankelijk van de individuele inrichting.

Oplossing

Een standaardoplossing voor dit probleem is het ontkoppelen van de zender (publisher) en de ontvanger (subscriber) door een soort prikbord met rubrieken tussen beiden te plaatsen. Ontvangers abonneren zich bij het prikbord op de rubrieken waarin ze zijn geïnteresseerd, de zenders doen niets anders dan boodschappen onder een bepaalde rubriek op het prikbord plaatsen (publiceren).

Dit patroon wordt publish/subcribe (publiceren en abonneren) genoemd.



Figuur 13 - Globale werking pub/sub patroon

Met een RESTful-aanpak wordt het prikboard gerealiseerd door een REST-API met globaal de volgende functionaliteit:

- Beheren (met autorisatie) en opvragen van de rubrieken (lijst van beschikbare topics en hun context)
- Beheren van abonnementen (subscribe/unsubscribe)
- Opvragen van abonnementen (subscriptions per topic)
- Publiceren van berichten (publish)

De belangrijkste motivatie om hier een RESTful API voor in te zetten, is het technologie-neutrale koppelvlak dat het oplevert. REST API's hebben een veel grotere reikwijdte dan de onderliggende middleware (die nog wel nodig blijft, maar beter eenmalig centraal kan worden ingericht).

Zowel het prikbord als de ontvanger moet voor de gekozen oplossing berichten kunnen ontvangen. Dat betekent dus dat zowel het prikbord als de ontvangers een REST-API beschikbaar stellen. Deze API's werken globaal als volgt:

STAP 1 Een abonnee meldt zich bij het prikbord aan voor een (voor gedefinieerd) onderwerp:

```
{
  "onderwerp": "Storingen",
  "responsUrl": "https://.../gebruiker/api/abonnee/v1/berichten"
}
```

Voorbeeld 35 - Registratieverzoek van een 'abonnee' voor een specifiek onderwerp

STAP 2 Een zender meldt zich bij het prikbord en stuurt een bericht met een passend onderwerp:

```
{
  "onderwerp": "Storingen",
  "bericht": "De server ligt weer plat!"
}
```

Voorbeeld 36 - Bericht van een 'zender' voorzien van een specifiek onderwerp

Stap 3 Het prikbord verzendt het ontvangen bericht naar alle abonnees die zich voor het aangegeven onderwerp hadden aangemeld:

Dit is feitelijk niets meer dan een POST naar de API van de relevante abonnees door gebruik te maken van de bij de aanmelding opgegeven `responsUrl`:

```
POST https://.../gebruiker/api/abonnee/v1/berichten
```

```
{
  "tijdstempel": "2019-04-12T23:20:50.52Z",
  "onderwerp": "Storingen",
  "bericht": "De server ligt weer plat!",
}
```

Voorbeeld 37 - Bericht van het prikbord gericht aan de abonnee

Beperkingen en overwegingen

Omdat in de oplossing gekozen is om te werken met een "callback" API, is deze specifieke uitwerking van dit patroon niet geschikt voor gebruikerstoepassingen.

Toepassingsgebied

Dit patroon is vooral goed inzetbaar wanneer er sprake is van n-n relaties waarin gemeenschappelijk "topics" een rol spelen in de gegevensuitwisseling. Daarnaast past een RESTful aanpak goed bij een heterogeen en open stelsel, want in dat geval is het van belang dat de dienst laagdrempelig en technologieneutraal wordt aangeboden.

API-O02 ***API's die via een "callback" relevante berichten bij afnemers moeten afleveren, volgen het beschreven "RESTful pub/sub patroon"***

Het pub/sub-patroon wordt vaak toegepast en standaard door middleware ondersteund. De belangrijkste motivatie om hier een RESTful API voor in te zetten, is het technologie-neutrale koppelvlak. REST API's hebben een veel grotere reikwijdte dan de onderliggende middleware (die nog wel nodig blijft, maar beter eenmalig centraal kan worden ingericht).

API-E10 ***API's kunnen via het Knooppunt van DSO-LV gebruikmaken van het "RESTful pub/sub patroon"***

Het Knooppunt van DSO-LV maakt het mogelijk om het RESTful pub/sub patroon toe te passen. De onderstaande URL's zijn hiervoor gereserveerd:

```
https://service.omgevingswet.overheid.nl/overheid/knooppunt/api/pubsub/v1
https://service.omgevingswet.overheid.nl/publiek/knooppunt/api/pubsub/v1
```

ⓘ *Een centrale pub/sub dienst is feitelijk een stelselbreed "prikbord" waarop aanbieders berichten kunnen plaatsen, die vervolgens op basis van het onderwerp worden afgeleverd bij de afnemers die op dat onderwerp zijn geabonneerd.*

2.5.16 Gecontroleerde degradatie

API's beperken het aantal verzoeken dat per tijdsperiode gedaan kan worden, om te voorkomen dat de servers overbelast worden. Dit is nodig om een hoog serviceniveau te garanderen.

De API's van DSO-LV hanteren een bevragingslimiet (quota) die per maand wordt bijgehouden en die wordt afgedwongen per tijdsperiode van 60 seconden. HTTP-headers worden gebruikt om de bevragingslimit naar de gebruiker te communiceren.

HTTP-header	Toelichting
X-Rate-Limit-Limit	Geeft aan hoeveel verzoeken een applicatie mag doen per tijdsperiode. *Voor het DSO is dit 60 seconden.*
X-Rate-Limit-Remaining	Geeft aan hoeveel seconden over zijn in de huidige tijdsperiode.
X-Rate-Limit-Reset	Geeft aan hoeveel verzoeken nog gedaan kunnen worden in de huidige tijdsperiode.

Tabel 18 - Overzicht van X-Rate-Limit headers

De [RFC6585] introduceerde een HTTP-statuscode `429 Too Many Requests` die wordt gebruikt om het overschrijden van het aantal verzoeken te melden aan de gebruiker.

API-B47 ***Begrenzingen worden proactief gemeld***

Gebruik `X-Rate-Limit` headers om limieten aan de gebruiker te communiceren en HTTP-statuscode `429 Too Many Requests` als de limieten overschreden worden.

API-E11 ***Beperken van het aantal verzoeken per tijdsperiode wordt centraal opgelost door het Knooppunt van DSO-LV***

Alle verzoeken naar API's lopen via het Stelselknooppunt. Het Stelselknooppunt lost het beperken van het aantal verzoeken per tijdsperiode centraal op om overbelasting van servers te voorkomen om een hoog serviceniveau te garanderen.

ⓘ *Quota zijn per API instelbaar.*

2.5.17 Foutafhandeling (status codes)

Net als een webpagina een bruikbare foutmelding toont aan bezoekers als een fout optreedt, moet een API een bruikbare foutmelding in een bekend en verwerkbaar formaat teruggeven.

De representatie van een fout is niet anders dan de representatie van een willekeurige resource, maar dan met een eigen set velden. De API moet daarom altijd zinvolle HTTP-statuscodes en meldingen teruggeven. HTTP-statuscodes zijn opgesplitst in verschillende ranges en categorieën:

Range	Categorie	Voorbeelden
100-199	Informatieberichten	`100 Continue`
200-299	Succesvolle verwerking	`200 OK` `201 Created`
300-399	Verwijzingen	`301 Moved Permanently` `303 See Other`
400-499	Inhoudelijke fouten	`400 Bad Request` `404 Not Found`
500-599	Server problemen	`500 Internal Server Error` `503 Service Unavailable`

Tabel 19 - HTTP-statuscodes: ranges en categorieën

Een JSON-representatie van een fout moet een aantal onderdelen bevatten om een ontwikkelaar, beheerder en eindgebruiker te helpen, waaronder:

Onderdeel JSON (RFC7807) foutmelding
• Een uniek fouttype in de vorm van een URI die verwijst naar informatie in externe (HTML) documentatie.
• Een korte maar nuttig foutmelding.
• Een HTTP-statuscode.
• Een gedetailleerde beschrijving van de fout (die helpt bij het oplossen).
• Een unieke identificatie in de vorm van een URI die hoort bij het specifieke voorkomen van de fout. *Het heeft de voorkeur om een URN te gebruiken waarmee alleen daartoe gerechtigde gebruikers details kunnen opzoeken in de fout-log.*

Tabel 20 - Onderdelen in een JSON-representatie van een fout

De basis voor dit standaardformaat is gedefinieerd in [RFC7807].
Een JSON-representatie van een fout ziet er als volgt uit:

```
{
  "type": "URI: https://content.omgevingswet.overheid.nl/fout/id/concept/{fout}["_"{categorie}]",
  "title": "Hier staat wat er is misgegaan…",
  "status": 401,                                          // foutcode
  "detail": "Meer details over de fout staan hier....",
  "instance": "urn:uuid:ebd2e7f0-1b27-11e8-accf-0ed5f89f718b"         // fout-instantie
}
```

Voorbeeld 38 - JSON-representatie van een fout-respons

Validatiefouten voor POST-, PUT- en PATCH-verzoeken worden per veld gespecificeerd. De volledige lijst met fouten wordt in één keer teruggegeven.

Dit wordt opgezet met een vast hoofdniveau en foutcode voor validatiefouten en extra foutvelden met gedetailleerde fouten per veld. Dit ziet er dan als volgt uit:

```
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/ValidatieFout",
  "title": "Hier staat wat er is misgegaan…",
  "status": 400,                                                    // foutcode
  "invalid-params":
  [
    {
       "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/Voornaam_Validatie",
       "name": "voornaam",
       "reason": "De voornaam mag geen speciale karakters bevatten."
    },
    {
       "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/Wachtwoord_Validatie",
       "name": "wachtwoord",
       "reason": "Het wachtwoord is verplicht."
    }
  ]
  "instance": "urn:uuid:4017fabc-1b28-11e8-accf-0ed5f89f718b"         // fout-instantie
}
```

Voorbeeld 39 - JSON-representatie van een 400 fout-respons

API-B48 ***Foutafhandeling is gestandaardiseerd***

API's ondersteunen de gestandaardiseerde foutmeldingen van de HTTP-statuscodes 400 en 500 reeks inclusief een verwerkbare JSON-representatie conform [RFC7807].

In Bijlage H: Standaardformaat foutmeldingen staan de gestandaardiseerde foutmeldingsformaten in de JSON-representatie die API's moeten implementeren.

HTTP-statuscodes

HTTP definieert een hele reeks gestandaardiseerde statuscodes die gebruikt dienen te worden door API's. Deze helpen de gebruikers van de API's bij het afhandelen van fouten.

API-B49 ***API's passen de verplichte HTTP-statuscodes toe***

De volgende http-statuscodes worden minimaal toegepast: 200, 201, 204, 304, 400, 401, 403, 405, 406, 409, 410, 412, 415, 422, 429, 500, 503.

Samenvatting HTTP-operaties in combinatie met de primaire HTTP-statuscodes.

Operatie	CRUD	Gehele collectie (bijvoorbeeld /resource) Specifieke item (bijvoorbeeld /resource/<id>)
POST	Create	• 201 (Created), HTTP-header Location met de URI van de nieuwe resource (/resource/<id>). • 405 (Method Not Allowed), 409 (Conflict) als de resource al bestaat.
GET	Read	• 200 (OK), lijst van resources. Gebruik pagineren, filteren en sorteren om het werken met grote lijsten te vereenvoudigen. • 200 (OK) enkele resource, 404 (Not Found) als ID niet bestaat of ongeldig is.
PUT	Update	• 405 (Method Not Allowed), behalve als het de bedoeling is om elke resource in een collectie te wijzigen of vervangen. • 409 als een wijziging niet mogelijk is vanwege de huidige toestand van de instantie. • 200 (OK) of 204 (No Content), 404 (Not Found) als ID niet bestaat of ongeldig is.
PATCH	Update	• 405 (Method Not Allowed), behalve als het de bedoeling is de gehele collectie te vervangen.

Operatie	CRUD	Gehele collectie (bijvoorbeeld /resource) Specifieke item (bijvoorbeeld /resource/<id>)
		• 409 als een wijziging niet mogelijk is vanwege de huidige toestand van de instantie. • 200 (OK) of 204 (No Content), 404 (Not Found) als ID niet bestaat of ongeldig is.
DELETE	Delete	• 405 (Method Not Allowed), behalve als het de bedoeling is de gehele collectie te verwijderen. • 200 (OK) of 404 (Not Found) als ID niet bestaat of ongeldig is.

Tabel 21 - HTTP-operaties in combinatie met de primaire HTTP-statuscodes

Hieronder een korte lijst met een beschrijving van de HTTP-statuscodes die minimaal worden toegepast:

`200 OK`	Reactie op een succesvolle GET, PUT, patch of DELETE. Ook geschikt voor POST die niet resulteert in een creatie.
`201 Created`	Reactie op een POST die resulteert in een creatie. Moet worden gecombineerd met een locatie-header die wijst naar de locatie van de nieuwe resource.
`204 No Content`	Reactie op een succesvol verzoek die geen inhoud zal teruggeven (zoals een DELETE).
`304 Not Modified`	Gebruikt wanneer HTTP caching headers worden toegepast.
`400 Bad Request`	Het verzoek is onjuist, bijvoorbeeld als het verzoek (body) niet kan worden geïnterpreteerd.
`401 Unauthorized`	Als er geen of ongeldige authenticatie details worden verstrekt. Ook handig om een authenticatie-venster te tonen als de API wordt gebruikt vanuit een browser.
`403 Forbidden`	Als de authenticatie gelukt is maar de geverifieerde gebruiker geen toegangsrechten heeft voor de resource.
`404 Not Found`	Wanneer een niet-bestaande resource is opgevraagd.
`405 Method Not Allowed`	Wanneer een HTTP-methode wordt gebruikt die niet is toegestaan voor de geauthentiseerde gebruiker.
`406 Not Acceptable`	Wordt teruggegeven als het gevraagde formaat niet ondersteund wordt (onderdeel van content negotiation).
`409 Conflict`	Het verzoek kon ik niet worden verwerkt als het gevolg van een conflict met de huidige toestand van de resource.
`410 Gone`	Geeft aan dat de resource niet langer op het eindpunt beschikbaar is. Nuttig als een overkoepelend antwoord voor oude API versies.
`412 Precondition Failed`	De preconditie die wordt gegeven door één of meer velden in de request-header, ontbraken of zijn na validatie op de server afgekeurd.
`415 Unsupported Media Type`	Als een verkeerd content-type als onderdeel van het verzoek werd meegegeven.
`422 Unprocessable Entity`	Gebruikt voor een verzoek (body) dat correct is maar dat de server niet kan verwerken.
`429 Too Many Requests`	Wanneer een aanvraag wordt afgewezen als het aantal verzoeken per tijdsperiode is overschreden.
`500 Internal Server Error`	Wanneer een onverwachte fout optreedt en het beantwoorden van het verzoek wordt verhinderd.
`503 Service Unavailable`	Wordt gebruikt als de API niet beschikbaar is (bijv. door gepland onderhoud).

Tabel 22 - HTTP-statuscodes die minimaal worden toegepast

Bijlage A: Bronnen

In deze bijlage worden de voor dit document gebruikte bronnen beschreven.

Referentie	Document	Omschrijving
[1]	Bestuurlijk Overleg (2016). Visie: 1.0	Visie Digitaal Stelsel Omgevingswet
[2]	Bestuurlijk Overleg (2019). GPvE 2.3	Globaal Programma van Eisen DSO-LV
[3]	Bestuurlijk Overleg (2019). Doelarchitectuur: 3.11	Doelarchitectuur DSO-LV
[4]	ADSMO (2020). OGAS: 2.0	Overall GAS DSO-LV
[5]	ADSMO (2020). DSO URI-strategie: 2.0	Kaderstellende notitie URI-strategie
[6]	ADSMO (2017). DSO Kaderstellende notitie Tijdreizen: 1.0	Kaderstellende notitie Tijdreizen zoals vastgesteld door SAB op 27-11-2017.
[7]	ADSMO (2020). Stelselafspraken: 2.0	Kaderstellende notitie met stelselbrede afspraken
[8]	Roy Thomas Fielding (200). REpresentational State Transfer (REST) https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm	
[9]	REST introductievideo (REST in 20 minuten) http://www.restapitutorial.com/lessons/whatisrest.html	
[10]	ADSMO (2018). Notitie doel en noodzaak CIM: 1.1	Doel en noodzaak Conceptueel informatiemodel (CIM
[11]	A query language for your API: Prerelease https://graphql.org/	
[12]	ADSMO (2019). API-profielen: V1.0	Definitie van API-profielen voor het vinden - verkennen en bevragen van informatiebronnen.
[13]	ADSMO (2019). Uitgangspunten en kaders voor het voorinvullen van vragen bij toepasbare regels: V1.0	Notitie met de uitgangspunten en kaders voor het voorinvullen van vragen bij toepasbare regels
[RFC3339]	Date and Time on the Internet: Timestamps https://tools.ietf.org/html/rfc3339	
[RFC1123]	Requirements for Internet Hosts -- Application and Support https://tools.ietf.org/html/rfc1123	
[RFC7234]	Hypertext Transfer Protocol (HTTP/1.1): Caching https://tools.ietf.org/html/rfc7234	
[RFC6585]	Additional HTTP Status Codes https://tools.ietf.org/html/rfc6585	
[RFC7807]	Problem Details for HTTP APIs https://tools.ietf.org/html/rfc7807	
[RFC5322]	Internet Message Format https://tools.ietf.org/html/rfc5322	
[RFC6750]	The OAuth 2.0 Authorization Framework: Bearer Token Usage https://tools.ietf.org/html/rfc6750	
[RFC7519]	JSON Web Token (JWT) https://tools.ietf.org/html/rfc7519	
[RFC7386]	JSON Merge Patch https://tools.ietf.org/html/rfc7386	

Bijlage B: Afkortingen en begrippen

Afkorting	Toelichting
ADSMO	Aan De Slag Met de Omgevingswet
API	Application Programming Interface
CIM	Conceptueel Informatie Model
CRS	Coordinaatreferentiesysteem
DSO	Digital Stelsel Omgevingswet
GAS	Globale Architectuur Schets
GDI	Generieke Digitale Infrastructuur
GPvE	Globaal Programma van Eisen
HAL	Hypertext Application Language
LvO	Leverancier van Omgevingsinformatie
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
OGAS	Overall Globale Architectuur Schets
ORM	Object Relational Mapping
OSvD	Open Stelsel voor Derden
PAT	Project Architectuur Team
RD	Rijksdriehoek
REST	REpresentational State Transfer
RFC	Request For Change
SAB	Stelsel Architectuur Board
SAT	Stelsel Architectuur Team
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
TTFSC	Time To First Successful Call
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XML	eXtensible Markup Language

Begrip	Toelichting
Authenticatie	Dit is het proces van het vaststellen van de gebruikersidentiteit die elektronisch is voorgelegd aan een informatiesysteem.
Body	De berichttekst in een HTTP-verzoek dat onmiddellijk na de headers wordt verzonden. Ook wel aangeduid als de payload.
Eindpunt	Een verwijzing naar een URI die web verzoeken accepteert.
Enveloppen	Letterlijk een omhulsel van een bericht in de vorm van tags of symbolen, zoals:

Begrip	Toelichting
	`{ bericht }`.
HAL	HAL staat voor Hypertext Application Language en is een eenvoudig formaat om op een consistente en eenvoudige manier hyperlinks tussen resources en API's te leggen. API's die HAL gebruiken kunnen gemakkelijk worden aangeboden en afgenomen met behulp van open source bibliotheken die beschikbaar zijn voor de meeste grote programmeertalen.
HATEOAS	HATEOAS staat voor Hypermedia As The Engine Of Application State en is een beperking van de REST-applicatie-architectuur en het onderscheidt van de meeste andere netwerkapplicatie-architecturen. Het principe houdt in dat een client met een netwerktoepassing interacteert via hypermedia die dynamisch wordt toegewezen door applicatieservers. Een REST-client heeft zodoende geen voorkennis over hoe te communiceren met een bepaalde toepassing of server buiten een algemeen begrip van hypermedia. In sommige servicegeoriënteerde architecturen (SOA), interacteren clients en servers echter via een vaste interface die wordt beschreven door middel van API-documentatie (OAS) of een interface definitiestaal (IDL).
Methode	HTTP-verzoek bestaat uit de HTTP-verzoek methode, de URL, de headervelden en eventueel de body. Een overzicht van de HTTP-verzoek methoden: • GET • HEAD • POST • PUT • DELETE • TRACE • OPTIONS • CONNECT • PATCH
ORM	Een zogenaamde Object Relational Mapping is een "mapping" die de relatie beschrijft tussen een object-model in een applicatie en de opslag in een relationele database.
Pretty print	Dit is het toepassen van opmaak-conventies op tekstuele bestanden en berichten.
Query	Een vraag die een gebruiker invoert in een zoekmachine om zijn of haar informatie-behoeften te bevredigen.
Request-header	Koptitelvelden van een HTTP-verzoek.
Resource	Dit is een primitieve binnen de web-architectuur en wordt gebruikt in de definitie van de fundamentele elementen.
RESTful	Een term die wordt gebruik om API's aan te duiden die volgens de REST architectuur werken.
Response-header	Koptitelvelden van een HTTP-antwoord.
Serialisatie	Het "serieel" opslaan van een gegevensobject in de vorm van een reeks bytes. Binair of als leesbare tekst zoals bijvoorbeeld het geval is bij JSON.
Statuscode	Een HTTP-status code is een code die wordt gebruikt door servers en browsers. Wanneer een verzoek wordt verstuurd, geeft de server een reactie op dat verzoek door middel van de HTTP-status code.
Token	Beveiligingstoken dat in het bezit is van een aanroepende client en wordt gebruikt om een identiteit te bewijzen.

Bijlage C: Figuren, tabellen en voorbeelden

Lijst van figuren

Lijst van tabellen

Lijst van voorbeelden

Lijst van voorbeelden

Bijlage D: Overzicht principes, ontwerppatronen en "best practices"

Overzicht van basisprincipes API-strategie V2.0

Overzicht van ontwerppatronen API-strategie V2.0

Overzicht van "best practices" API-strategie V2.0

Bijlage E: Overzicht eisen

Eisen (BASIS) API-strategie V2.0

Eisen (IDENTITEIT) API-strategie V2.0

Eisen (HYPERMEDIA) API-strategie V2.0

Eisen (QUERY-PROJECTIE) API-strategie V2.0

Eisen (GEO) API-strategie V2.0

Eisen (GEO) API-strategie V2.0

Eisen (TIJDREIZEN) API-strategie V2.0

Eisen (ECOSYSTEEM) API-strategie V2.0

Eisen (API-PROFIELEN) API-strategie V2.0

Eisen (API-PROFIELEN) API-strategie V2.0	
API-A03	DE REGISTRATIE VAN API-PROFIELEN IS OPGENOMEN IN DE STELSELCATALOGUS 51
API-A04	EEN API KAN ZICH CONFORMEREN AAN ÉÉN OF MEER API-PROFIELEN DOOR DIT CENTRAAL TE REGISTEREN 51
API-A05	API'S DIE API-PROFIELEN IMPLEMENTEREN ZIJN OPGENOMEN IN DE STELSELCATALOGUS 51

Eisen (ONTWERPPATRONEN) API-strategie V2.0	
API-O01	API'S DIE ASYNCHRONE VERWERKING ONDERSTEUNEN VOLGEN HET BESCHREVEN "RESTFUL ASYNCHROON VERZOEK/ANTWOORD PATROON" 63
API-O02	API'S DIE VIA EEN "CALLBACK" RELEVANTE BERICHTEN BIJ AFNEMERS MOETEN AFLEVEREN, VOLGEN HET BESCHREVEN "RESTFUL PUB/SUB PATROON" 64

Bijlage F: Migratie eisen V1.1 → V2.0

Eisen API-strategie V1.1	Eisen API-strategie V2.0	Toelichting
-	API-B01	Nieuwe eis.
API-01	API-B02	Identificatie gewijzigd, maar inhoudelijk identiek.
API-02	API-B03	Identificatie gewijzigd, maar inhoudelijk identiek.
API-03	API-B16	Identificatie gewijzigd, maar inhoudelijk identiek.
API-04	API-B17	Identificatie gewijzigd, maar inhoudelijk identiek.
API-05	API-B18	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-06	API-B19	Identificatie gewijzigd, maar inhoudelijk identiek.
API-07	API-B21	Identificatie gewijzigd, maar inhoudelijk identiek.
API-08	API-B22	Identificatie gewijzigd, maar inhoudelijk identiek.
API-09	API-B27	Identificatie gewijzigd, maar inhoudelijk identiek.
API-10	API-B28	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-11	API-B29	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
-	API-B30	API-11 is opgesplitst in API-B26 en API-B27.
API-12	API-Q01	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-13	API-B23	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
	API-B24	Nieuwe eis.
	API-B25	Nieuwe eis.
	API-B26	Nieuwe eis.
API-14	API-B12	Identificatie gewijzigd, maar inhoudelijk identiek.
API-15	API-B13	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
-	API-B14	Nieuwe eis, was voorheen aanbeveling.
-	API-B15	Nieuwe eis, was voorheen aanbeveling.
API-16	API-I05	Identificatie gewijzigd, maar inhoudelijk identiek.
API-17	API-I01	Identificatie gewijzigd, maar inhoudelijk identiek.
-	API-I02	Nieuwe eis.
API-18	API-I03	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
-	API-I04	Nieuwe eis.
-	API-I06	Nieuwe eis.
-	API-I07	Nieuwe eis.
-	API-I08	Nieuwe eis.
API-19	API-B38	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
-	API-B39	Nieuwe eis, was voorheen aanbeveling.
-	API-B40	Nieuwe eis, was voorheen aanbeveling.
API-20	API-B41	Identificatie gewijzigd, maar inhoudelijk identiek.
API-21	API-B42	Identificatie gewijzigd, maar inhoudelijk identiek.
API-22	API-B43	Identificatie gewijzigd, maar inhoudelijk identiek.
API-23	API-B44	Identificatie gewijzigd, maar inhoudelijk identiek.
API-24	API-B45	Identificatie gewijzigd, maar inhoudelijk identiek.
API-25	API-B46	Identificatie gewijzigd, maar inhoudelijk identiek.

Eisen API-strategie V1.1	Eisen API-strategie V2.0	Toelichting
API-26	API-B04	Identificatie gewijzigd, regel uitgebreid met restricties op arrays en primitieve datatypen als “top-level” elementen.
API-27	API-B05	Identificatie gewijzigd, maar inhoudelijk identiek.
API-28	API-B06	Identificatie gewijzigd, maar inhoudelijk identiek.
API-29	API-B07	Identificatie gewijzigd, maar inhoudelijk identiek.
-	API-B08	Nieuwe eis. Was alleen expliciet voor tijdreizen, maar dit geldt in het algemeen.
API-30	API-B09	Identificatie gewijzigd, regel uitgebreid met restrictie voor namen tot de alfanumerieke en aanvullingen t.b.v. enumeraties.
API-31	API-B11	Identificatie gewijzigd, maar inhoudelijk identiek.
API-32	API-B10	Identificatie gewijzigd, maar inhoudelijk identiek.
API-33	API-B20	Identificatie gewijzigd, maar inhoudelijk identiek.
API-34	API-B31	Identificatie gewijzigd, maar inhoudelijk identiek.
-	API-B32	Nieuwe eis.
API-35	API-B33	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-36	API-B34	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-37	API-B35	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-38	API-G01	Identificatie gewijzigd, maar inhoudelijk identiek.
API-39	API-G02	Identificatie gewijzigd, maar inhoudelijk identiek.
API-40	API-G03	Identificatie gewijzigd, maar inhoudelijk identiek.
API-41	API-G04	Identificatie gewijzigd, maar inhoudelijk identiek.
API-42	API-G05	Identificatie gewijzigd, maar inhoudelijk identiek.
API-43	API-G06	Identificatie gewijzigd, maar inhoudelijk identiek.
API-44	API-G07	Identificatie gewijzigd, maar inhoudelijk identiek.
-	API-G08	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-G09	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
API-45	API-G10	Identificatie gewijzigd, maar inhoudelijk identiek.
-	API-H01	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-H02	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-H03	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
API-46	API-H04	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
-	API-B36	API-46 is opgesplitst in API-H04 en API-B33.
API-47	API-B37	Identificatie gewijzigd, maar inhoudelijk identiek.
API-48	API-E11	Identificatie en naam gewijzigd, maar inhoudelijk identiek.
API-49	API-B47	Identificatie gewijzigd, maar inhoudelijk identiek.
API-50	API-B48	Identificatie gewijzigd, maar inhoudelijk identiek.
API-51	API-B49	Identificatie gewijzigd, maar inhoudelijk identiek.
-	API-T01	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-T02	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-T03	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-T04	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-A01	Nieuwe eis, afkomstig van onderwerp API-profielen.
-	API-A02	Nieuwe eis, afkomstig van onderwerp API-profielen.

Eisen API-strategie V1.1	Eisen API-strategie V2.0	Toelichting
-	API-A03	Nieuwe eis, afkomstig van onderwerp API-profielen.
-	API-A04	Nieuwe eis, afkomstig van onderwerp API-profielen.
-	API-A05	Nieuwe eis, afkomstig van onderwerp API-profielen.
-	API-Q02	Nieuwe eis, afkomstig van onderwerp Query-projectie.
-	API-Q03	Nieuwe eis, afkomstig van onderwerp Query-projectie.
-	API-Q04	Nieuwe eis, afkomstig van onderwerp Query-projectie.
-	API-Q05	Nieuwe eis, afkomstig van onderwerp Query-projectie.
-	API-O01	Nieuwe eis, afkomstig van onderwerp Ontwerppatronen.
-	API-O02	Nieuwe eis, afkomstig van onderwerp Ontwerppatronen.
-	API-E01	Nieuwe eis, afkomstig van onderwerp Ontwerppatronen.
-	API-E02	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-E03	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-E04	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-E05	Nieuwe eis, was voorheen impliciet in tekst opgenomen.
-	API-E06	Nieuwe eis, afkomstig van onderwerp Ecosysteem en ecosysteem-bewustzijn.
-	API-E07	Nieuwe eis, afkomstig van onderwerp Ecosysteem en ecosysteem-bewustzijn.
-	API-E08	Nieuwe eis, afkomstig van onderwerp Ecosysteem en ecosysteem-bewustzijn.
-	API-E09	Nieuwe eis, afkomstig van onderwerp Ecosysteem en ecosysteem-bewustzijn.
-	API-E10	Nieuwe eis, afkomstig van onderwerp Ecosysteem en ecosysteem-bewustzijn.

Bijlage G: Uitfasering eisen (deprecated)

Eisen (uit V1.1) om uit te faseren	
DEP-01	DE QUERY-PARAMETER `EXPAND (BOOLEAN)` IS VERVANGEN DOOR `_EXPAND (BOOLEAN)` 32
DEP-02	DE QUERY-PARAMETER `EXPAND (STRING)` IS VERVANGEN DOOR `_EXPANDSCOPE (STRING)` 34
DEP-03	DE QUERY-PARAMETER `SORTEER` IS VERVANGEN DOOR `_SORT` 36
DEP-04	DE QUERY-PARAMETER `ZOEK` IS VERVANGEN DOOR `_FIND` 37
DEP-05	DE QUERY-PARAMETER `FIELDS` IS VERVANGEN DOOR `_FIELDS` 41

Bijlage H: Standaardformaat foutmeldingen

Een API moet minimaal de onderstaande standaard foutmeldingsformaten kunnen ondersteunen. De genoemde combinatie van operatie – response code zijn verplicht. Foutmeldingen zijn in het Nederlands. Bij iedere foutmelding wordt ter illustratie een voorbeeld gegeven.

De basis voor deze standaardformaten is [RFC7807].

Standaard foutmeldingsformaten

1. GET - HTTP Response Code: **404**

```
HTTP/1.1   404
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/BestaatNiet",
  "title": "De aanvraag bestaat niet.",
  "status": 404,
  "detail": "Het aanvraagnummer 123 heeft geen resultaat opgeleverd.",
  "instance": "urn:uuid:608aa448-1b28-11e8-accf-0ed5f89f718b"
}
```

2. DELETE - HTTP Response Code: **404/405**

```
HTTP/1.1   404
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/BestaatNiet",
  "title": "De aanvraag bestaat niet.",
  "status": 404,
  "detail": "De aanvraag met nummer 123 kon niet worden verwijderd.",
  "instance": urn:uuid:70e0f766-1b28-11e8-accf-0ed5f89f718b"
}

HTTP/1.1   405
Content-Type: application/problem+json
Content-Language: nl
{
  "type": " https://content.omgevingswet.overheid.nl/fout/id/concept/NietToegestaan",
  "title": "De aanvraag kan niet worden verwijderd",
  "status": 405,
  "detail": "De geauthentiseerde gebruiker mag aanvraag 123 niet verwijderen",
  "instance": "urn:uuid:77de3c0e-1b28-11e8-accf-0ed5f89f718b"
}
```

3. POST - HTTP Response Code: **400/405**

```
HTTP/1.1   400
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/ValidatieMislukt",
  "title": "Validatie mislukt.",
  "status": 400,
  "invalid-params":
   [
     {
      "type": "https://content.../fout/id/concept/Emailadres_Validatie",
      "name": "email",
         "reason": "Ongeldig emailadres"
       },
       {
         "type": "https://content.../fout/id/concept/Telefoonnummer_Validatie",
         "name": "telefoonnummer",
         "reason": "Ongeldig telefoonnummer"
       }
    ]
    "instance": "urn:uuid:8fd04816-1b28-11e8-a834-0ed5f89f718b"
}
```

```
HTTP/1.1   405
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietToegestaan",
  "title": "De contactgegevens kunnen niet worden verwerkt",
  "status": 405,
  "detail": "De geauthentiseerde gebruiker mag de contactgegevens niet wijzigen",
  "instance": "urn:uuid:9fc8bd20-1b28-11e8-accf-0ed5f89f718b"
}
```

4. PATCH - HTTP Response Code: **400**/**404**/**405**

```
 HTTP/1.1   400
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/ValidatieFout",
  "title": "Validatie mislukt.",
  "status": 400,
  "invalid-params":
  [
    {
       "type": "https://content.../fout/id/concept/validatie/Wachtwoord",
       "name": "wachtwoord",
       "reason": "Wachtwoord moet minimaal 8 karakters lang zijn en
                  minimaal bestaan uit 1 hoofdletter, 1 kleine letter en 1 en
                  1 leesteken."
    }
  ]
  "instance": "urn:uuid:c22a0202-1b28-11e8-accf-0ed5f89f718b"
}

HTTP/1.1   401
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/BestaatNiet",
  "title": "De aanvraag bestaat niet.",
  "status": 401,
  "detail": "De aanvraag met nummer 123 kon niet worden gewijzigd.",
  "instance": "urn:uuid:d07b5450-1b28-11e8-accf-0ed5f89f718b"
}

HTTP/1.1   405
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietToegestaan",
  "title": "De aanvraag kon niet worden gewijzigd.",
  "status": 405,
  "detail": "De aanvraag met nummer 123 mag niet worden gewijzigd.",
  "instance": "urn:uuid:db4c0762-1b28-11e8-accf-0ed5f89f718b"
}
```

5. VERB Unauthorized - HTTP Response Code: **401**

```
HTTP/1.1   401
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/Ongeautoriseerd",
  "title": "Authenticatiegegevens ontbreken of zijn onjuist.",
  "status": 401,
  "detail": "Corrigeer de authenticatiegegevens en probeer het opnieuw.",
  "instance": "urn:uuid:ebcd8cbe-1b28-11e8-accf-0ed5f89f718b"
}
```

6. VERB Forbidden - HTTP Response Code: **403**

```
HTTP/1.1   403
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/Verboden",
  "title": "{Het verzoek is begrepen, maar is geweigerd of niet toegestaan}",
  "status": 403,
  "detail": "{Toelichting die gebruikers helpt om dit probleem op te lossen}",
  "instance": "urn:uuid:f6a9acda-1b28-11e8-accf-0ed5f89f718b"
}
```

7. VERB Method Not Allowed - HTTP Response Code: **405**

```
HTTP/1.1   405
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietToegestaan",
  "title": "{Het verzoek is niet toegestaan voor deze resource}",
  "status": 405,
  "detail": "{Toelichting die gebruikers helpt om dit probleem op te lossen}",
  "instance": "urn:uuid:04141b08-1b29-11e8-accf-0ed5f89f718b"
}
```

8. VERB Method Not Acceptable - HTTP Response Code: **406**

```
HTTP/1.1   406
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietAcceptabel",
  "title": "{De representatie is niet beschikbaar}",
  "status": 406,
  "detail": "{Toelichting die gebruikers helpt om juiste representatie te kiezen}",
  "instance": "urn:uuid:0f70ee4a-1b29-11e8-accf-0ed5f89f718b"
}
```

9. VERB Conflict - HTTP Response Code: **409**

```
HTTP/1.1   409
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/Conflict ",
  "title": "{Omschrijving van het conflict}",
  "status": 409,
  "detail": "{Toelichting die gebruikers helpt om het conflict op te lossen}",
  "instance": "urn:uuid:1ad59fd8-1b29-11e8-accf-0ed5f89f718b"
}
```

10. VERB Gone - HTTP Response Code: **410**

```
HTTP/1.1   410
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/Verdwenen",
  "title": "{Omschrijving van de verwijderde/verdwenen resource}",
  "status": 410,
  "detail": "{Toelichting die gebruikers helpt om het probleem op te lossen}",
  "instance": "urn:uuid:34e257d6-1b29-11e8-accf-0ed5f89f718b"
 }
```

11. VERB Precondition Failed - HTTP Response Code: **412**

```
HTTP/1.1   412
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/MetadataOntbreekt",
  "title": "Het verzoek kan niet worden afgehandeld omdat een header ontbreekt",
  "status": 412,
  "detail": "{Gebruik voor GeoJSON-verzoeken altijd de Content-Crs header}",
  "instance": "urn:uuid:42382028-1b29-11e8-accf-0ed5f89f718b"
}
```

12. VERB Unsupported Media Type - HTTP Response Code: **415**

```
HTTP/1.1   415
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietOndersteundMediatype",
  "title": "Mediatype niet ondersteund",
  "status": 412,
  "detail": "{Het gevraagde mediatype wordt niet ondersteund}",
  "instance": "urn:uuid:42382028-1b29-11e8-accf-0ed5f89f718b"
}
```

13. VERB Unprocessable Entity - HTTP Response Code: **422**

```
HTTP/1.1   422
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/EntiteitNietVewerkbaar",
  "title": "De entiteit is niet verwerkbaar",
  "status": 429,
  "detail": "{De entiteit is correct maar kan niet worden verwerkt}",
  "instance": "urn:uuid:5b254f8e-1b29-11e8-accf-0ed5f89f718b"
}
```

14. VERB Too Many Requests - HTTP Response Code: **429**

```
HTTP/1.1   429
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/TeVeelVerzoeken",
  "title": "Het verzoek kan niet worden afgehandeld omdat het maximum aantal
            verzoeken is overschreden voor {naam van de resource}",
  "status": 429,
  "detail": "{Toelichting die gebruikers helpt om het probleem op te lossen}",
  "instance": "urn:uuid:5b254f8e-1b29-11e8-accf-0ed5f89f718b"
}
```

15. VERB Internal Server Error - HTTP Response Code: **500**

```
HTTP/1.1   500
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/ServerFout",
  "title": "Iets is misgegaan.",
  "status": 500,
  "detail": "{Toelichting die gebruikers helpt om het probleem te melden of op te
             lossen}",
  "instance": "urn:uuid:66fdb742-1b29-11e8-accf-0ed5f89f718b"
}
```

16. VERB Service Unavailable - HTTP Response Code: **503**

```
HTTP/1.1   503
Content-Type: application/problem+json
Content-Language: nl
{
  "type": "https://content.omgevingswet.overheid.nl/fout/id/concept/NietBeschikbaar",
  "title": "De server is tijdelijk niet beschikbaar. Probeer het later nog eens.",
  "status": 503,
  "detail": "{Toelichting die gebruikers helpt om het probleem te melden of op te
             lossen}",
  "instance": "urn:uuid:70031b20-1b29-11e8-accf-0ed5f89f718b"
}
```

Aanvullend wordt ook een response header gebruikt om een “retry-interval” aan te geven:

HTTP-header	Toelichting
Retry-After	Indicatie van het tijdstip waarop de gevraagde dienst opnieuw door de client kan worden aangeroepen. Het tijdstip is in het formaat dat ook wordt gebruikt door het Internet Message Format [RFC5322]. Bijvoorbeeld: `wo, 21 oktober 2015 07:28:00 GMT` of het aantal seconden (bijvoorbeeld `3000` als de dienst naar verwachting weer binnen 50 minuten beschikbaar zal zijn).