Deelprogramma
Digitaal Stelsel Omgevingswet

Kaderstellende notities
API-strategie

Versie 1.1 Vastgesteld 12-03-2018

Dit document is vastgesteld door het Stelsel Architectuur Board (SAB). Hiermee is de richting op hoofdlijnen goedgekeurd. Omdat het deelprogramma DSO 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 een UI voor ontwikkelaars – hiervoor geldt, zoals bij elke UI, dat het belangrijk is dat de gebruikerservaring goed doordacht is.

Colofon

Titel	:	API-strategie
Versie	:	1.1 Vastgesteld
Datum	:	12-03-2018
Opdrachtgever	:	Programma Implementatie Omgevingswet
Opdrachtnemer	:	Deelprogramma DSO
Auteurs	:	Tony Sloos Stephen Oostenbrink Dimitri van Hees
Contactpersonen	:	A.J. (Tony) Sloos *Domeinarchitect Informatie & Kernfuncties* +31 6 1125 2597 tony.sloos@rws.nl

Versiehistorie

Versie	Status	Datum	Auteur(s)	Toelichting
0.1	Concept	25-01-2017	A.J. Sloos	Initiële versie.
0.2	Concept	27-01-2017	S. Oostenbrink	Aanvullingen en opmerkingen Stephen.
0.3	Concept	06-02.-2017	A.J. Sloos	Nieuwe versie ter bespreking met Stephen en Dimitri.
0.4	Concept	10-02-2017	S. Oostenbrink	Aanscherpen formulering en tekst gehele document volledig herzien.
0.45	Concept	15-02-2017	A.J. Sloos	Kleine tekstuele aanpassingen en correcties. Aanvulling Bas verwerkt.
0.46	Concept	17-03-2017	A.J. Sloos	Review PAT verwerkt. Review provincies verwerkt. Input Dimitri van Hees (PR10) verwerkt. Review Stephen verwerkt en openstaande punten toegevoegd. Versienummering in lijn gebracht met GAS-en: 0.5 → 0.45
0.5	Concept	12-04-2017	A.J. Sloos	Review 0.46 door provincies verwerkt. Review en afstemming open einden met Dimitri verwerkt.
0.51	Concept	15-05-2017	A.J. Sloos	Reviews 0.5 en input externe consultatie verwerkt. Laatste inzichten IHR in overleg met Dimitri verwerkt.
0.52	Concept	29-05-2017	A.J. Sloos	Aanvulling op eisen t.a.v. standaard taal.
0.53	Vastgesteld	19-06-2017	A.J. Sloos	Oplevering als bijlage van OGAS 1.5.
1.0	Vastgesteld	20-07-2017	A.J. Sloos	Vastgesteld met wijzigingen.
1.1	Vastgesteld	12-03-2018	A.J. Sloos	Vastgesteld met wijzigingen.

Goedkeuring

Functie	Naam	Versie	Datum	Handtekening
Programma Directeur Implementatie Omgevingswet	Ineke van der Hee	1.0, 1.1		
Programma Manager PDSO	Bert Uffen	1.0, 1.1		
Lead architect programma	Victorine Binkhorst	1.0		

Distributie

Functie/Orgaan	Versie	Opmerkingen
Lead architect programma	0.2, 0.45, 0.46, 0.5, 0.51, 1.0	
Opdrachtgevend Beraad Implementatie Omgevingswet	1.0, 1.1	Ter informatie
Programma Raad Implementatie Omgevingswet	1.0, 1.1	Ter informatie
Programma Team	0.45, 0.46, 0.5, 1.0, 1.1	

Functie/Orgaan	Versie	Opmerkingen
Programma Architectuur Team	0.45, 0.46, 0.5, 1.0, 1.1	
Architectuur Team Overleg	0.45, 0.46, 0.5, 1.0, 1.1	
Projectmanager	0.45, 0.46, 0.5, 1.0, 1.1	Ter informatie
Strategische Ontwikkelpartners	0.45, 0.46, 1.0, 1.1	
Open Stelsel (PR10) Bob Coret, Dimitri van Hees	0.1, 0.3, 0.45, 0.46, 1.0, 1.1	Ter informatie
Externe consultatie	0.5, 1.1	
Externe publicatie	1.0, 1.1	Ter informatie

Review

Naam	Versies
Andre Batenburg (AB), BLA provincies	0.45, 0.46, 0.5, 0.51, 1.0, 1.1
Jan van Langeveld (JvL), BLA gemeenten	0.45, 0.46, 0.5, 0.51, 1.0, 1.1
Paul de Frankrijker (PdF), BLA waterschappen	0.45, 0.46, 0.5, 0.51, 1.0, 1.1
Peter Visser (PV), BLA Rijk	0.45, 0.46, 0.5, 0.51, 1.0
Rene Kint (RK), Stelselarchitect	0.45, 0.46, 0.5, 0.51, 1.0, 1.1
Silvion Moesan (SM), Projectarchitect PR02	0.45, 0.5, 1.0, 1.1
Frank Terpstra (FT), Projectarchitect PR05	0.45, 0.5, 1.0
Jan Jaap Zoutendijk (JJZ), Projectarchitect PR05	0.45, 0.5, 1.0, 1.1
Eric van Capelleveen (EvC), Projectarchitect PR04	1.0
Frank Robijn (FR), Projectarchitect PR04	1.1
Lennart van Bergen (LvB), Projectarchitect PR06 en PR30	0.45, 0.5, 1.0, 1.1
Marco Brattinga (MB), opsteller GAS PR06	1.0, 1.1
Dimitri van Hees (DvH), Projectarchitect PR10	0.1, 0.3, 0.45, 0.46, 0.5, 1.0, 1.1
Rien Berkhout (RB), Projectarchitect PR12	0.45, 0.5, 1.0, 1.1
Nico Plat (NP), Projectarchitect PR12	1.1
Jeroen Ekkelenkamp (JE), Projectarchitect PR13	0.45, 0.5, 1.0
Jeroen de Hond (JdH), Projectarchitect PR13	1.1
Tony Sloos (TS), Domeinarchitect	0.3, 0.45
Bas Crompvoets (BC), Domeinarchitect	0.3, 0.45, 0.5, 1.0, 1.1
Mickel Langeveld (ML), Domeinarchitect	0.5, 1.0, 1.1
Rob van der Veer (RvdV), Sylvan Rigal (SR), Željko Obrenović (ZO), Software Improvement Group	1.0

Referenties

ID	Naam	Status	Versie
[1]	DSO - GAS: Overall GAS	VASTGESTELD	1.5
[2]	DSO - Kaderstellende notities: URI-strategie	VASTGESTELD	1.1
[3]	DSO - Kaderstellende notities: Tijdreizen	VASTGESTELD	1.0

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. Met de voorliggende kaders wordt invulling gegeven aan een gemeenschappelijke strategie voor de realisatie van Application Programming Interfaces ofwel API's. Hiermee wordt op technisch koppelvlakniveau invulling gegeven aan het leidende DSO-principe 'Alles is een service'.

1.1 Doelgroep

Dit document richt zich op de stelselarchitect, Business Liaison Architecten (BLA's), de projectarchitecten, projectmedewerkers en andere geïnteresseerden.

1.2 Doel

Het primaire doel van dit document is het meegeven van ontwikkelkaders voor API's. Afwijken van deze kaders 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. Hiermee wordt de leercurve en de Time To First Successful Call (TTFSC)[1] verkort doordat alle API's van het stelsel op een uniforme manier werken. Dit is niet alleen nodig voor intern gebruik maar ook voor het succes van het open stelsel voor derden.

> *Een API is een UI voor ontwikkelaars – hiervoor geldt, zoals bij elke UI, dat het belangrijk is dat de gebruikerservaring goed doordacht is.*

1.4 Afkortingen en begrippen

Relevante afkortingen en begrippen zijn te vinden in bijlage B en C.

1.5 Leeswijzer

Dit document dient in combinatie met de URI-strategie [2] document gelezen te worden, omdat beide documenten elkaar aanvullen.

Om dit kaderstellend document te begrijpen, op waarde te schatten en te beoordelen is het belangrijk om te begrijpen vanuit welke paradigma deze is opgesteld. De basis wordt gevormd door een architectuurstijl die door Roy Thomas Fielding in 2000 onder

[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.

de naam REpresentational State Transfer (REST) is geïntroduceerd. Het is belangrijk om de achtergrond en eigenschappen van REST te begrijpen. Bijvoorbeeld dat REST in tegenstelling tot SOAP geen standaard is. REST volgt de internetstandaarden, waardoor het veel eenvoudiger in gebruik is.

Een korte REST introductie (REST in 20 minuten) is hier te vinden:
http://www.restapitutorial.com/lessons/whatisrest.html

In maart 2016 heeft het Forum Standaardisatie een discussiedocument opgesteld waarin wordt uitgelegd wat API's zijn, hoe ze werken en wat de betekenis is voor standaarden. In deze discussie worden ook achtergronden en trends besproken die de gekozen richting in dit document ondersteunen.

Zie voor meer achtergrond informatie:
https://www.forumstandaardisatie.nl/thema/application-programming-interfaces-api

2 API's

2.1 Context

In de Visie (v1.0) 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 een ieder.*

Dit document beschrijft de strategie waarmee de API's op een open en robuuste manier in de vorm van consistente en uniforme API's binnen het stelsel, aan de keten en aan een ieder worden aangeboden.

2.2 De basis

Er zijn verschillende ontwikkelpartners die samen het stelsel ontwikkelen. Het stelsel bestaat uit een groot aantal stelselcomponenten. De functionaliteiten van deze stelsel-componenten worden beschikbaar gesteld middels API's. De ontwikkelaars van bijvoorbeeld gebruikerstoepassingen integreren functionaliteit van een groot aantal stelselcomponenten met behulp van deze API's. Om de integratie inspanning zo laag mogelijk te houden dient de leercurve van de API's zo kort mogelijk te zijn. Dit wordt o.a. bereikt door een goed API-ontwerp, herkenbaarheid over API's heen, toepassen van defacto standaarden en goede documentatie.

Het DSO hanteert het principe van "eat your own dogfood". Dit betekent dat alles in het stelsel een API is en deze API's door alle stelselonderdelen worden gebruikt. Er is geen andere manier van integratie toegestaan. Dit vereist dat alle API 's op een uniforme manier zijn opgezet en werken en goed gedocumenteerd zijn. 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 principes en uitgangspunten zijn afgeleid van het principe in de doelarchitectuur dat zegt: "Nieuwe standaarden sluiten aan op bestaande standaarden" (APDSO13) of komen uit de Overall Globale Architectuur Schets [1] en zijn leidend voor de API-strategie.

Alles is een service (Service 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 koppelvlak niveau invulling te geven aan het leidende DSO principe 'Alles is een Service' worden er gedetailleerde standaarden voor de programmeerinterface van services (API's of Application Programming Interfaces) voorgeschreven.

Geen extra services voor derden	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.

Intern = Extern (Eat your own dogfood)	Services werken voor interne en externe ontwikkelaars hetzelfde. Intern wordt gekozen voor REST API's. Deze 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.

Formele koppelvlakken zijn op basis van 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 het DSO naar een bevoegd gezag.

Open stelsel voor derden

Het DSO wordt ontwikkeld als "open stelsel". De overheid is verantwoordelijk voor de wettelijk bepaalde functionaliteit van het stelsel. De functionaliteit en data van het stelsel worden aan derden beschikbaar gesteld via het "open stelsel" principe. Hierdoor kunnen derden aanvullende toepassingen ontwikkelen op basis van de data en API's van het stelsel.

De API's moeten toegankelijk genoeg zijn om een brede ontwikkel-community aan te spreken (ontwikkelvriendelijk zijn). Deze 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 moeten toch door het DSO op een goede manier worden bediend. Dit wordt gerealiseerd door zoveel mogelijk aan te sluiten op de standaarden waarmee deze ontwikkelaars gewend zijn te werken.

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 goede Developer Experience (DX) is vereist. Dit wordt onder andere bereikt door goede API-ontwerpen en herkenbaarheid over API's heen. Dit geldt overigens, zoals eerder in dit document benoemd, ook voor de ontwikkelaars van het stelsel zelf.

2.3 Een API is een gebruikersinterface voor ontwikkelaars

De volgende vijf basiseisen zijn gevolgd bij het uitwerken van deze API-strategie. Een API:

1) Maakt gebruik van web-standaarden[2] *waar dit zinvol is*;
2) Is gebruiksvriendelijk voor ontwikkelaars en kan via de browser[3] verkend worden;
3) Is eenvoudig, intuïtief en consistent in gebruik waardoor het gebruik niet alleen gemakkelijk is maar ook aangenaam;
4) Is kanaalonafhankelijk[4] en voldoende flexibel om verschillende gebruiksscenario's te ondersteunen;
5) Is efficiënt, dit is in evenwicht met de voorgaande eisen.

2.4 Strategie

De hier beschreven strategie gaat uit van een API-first aanpak. Dit betekent dat de API ontworpen en gebouwd wordt onafhankelijk van de front-end(s) waarin deze gebruikt wordt. Een API is een product op zichzelf. Een API moet elk kanaal kunnen bedienen en niet één specifiek kanaal. Dit wil niet zeggen dat de API los van de werkelijkheid wordt ontwikkeld. Er wordt afgestemd met de verschillende partijen en hun input wordt gebruikt bij het ontwikkelen, beheren en onderhouden. Een API is een combinatie van het koppelvlak, documentatie en andere ondersteunende hulpmiddelen.

2.5 Bestaande API's en product specifieke API's

Bijzondere categorieën in relatie tot de API-strategie zijn bestaande API's en leverancier en of product specifieke API's. Het doel van de API-strategie is om stelselbreed een uniforme API ervaring te bieden. Wat betekent dit voor bestaande API's en product specifieke API's?

Bij bestaande API's wordt een kosten baten afweging gemaakt wat de inspanning is om deze om te zetten zodat deze wel voldoen aan de API-strategie. Hierbij zijn in ieder geval drie aanpakken mogelijk: 1) onderzoeken of de bestaande API (en wellicht zelfs de applicatie) nog wel nodig is; 2) opnieuw bouwen; 3) of een façade API.

API-01 ***Bestaande API's voldoen waar mogelijk 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.

2 Bij de meeste onderwerpen wordt verwezen naar de relevante standaard zoals JSON, HAL, HATEOAS specifieke W3C standaarden of IETF RFCS.

3 De API kan via de webbrowser (al dan niet met plug-in) worden verkend en uitgeprobeerd.

4 Geschikt voor gebruik in zowel web-, mobiele- als backend applicaties.

Product- of leverancier specifieke API's zijn een categorie op zich. Dit zijn meestal API's die onderdeel zijn van een product en daarmee niet leverancier- of productneutraal zijn. Het koppelvlak is dus leverancier- of product specifiek. Deze

API's worden nooit direct door afnemers aangeroepen maar altijd via een façade API. Hierdoor is het mogelijk om een onderliggend systeem en/of bijbehorende technische API's te vervangen met geen of minimale impact voor de afnemers.

API-02 ***Product- en leverancier-specifieke API's worden nooit direct aangeroepen***

Hierdoor 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 gestandaardiseerde API aanbieden.

2.6 Standaardisatie

Het resultaat van de gemeenschappelijke strategie voor API's is een stelselbrede standaardisatie van de volgende aspecten:

1. RESTful principes (inclusief HATEOAS-constraint[5] in een afgeslankte vorm)
2. Beveiliging (versleutelingen en authenticatie)
3. Documentatie
4. Versionering
5. Gebruik van JSON (inclusief HAL hypermedia controls)
6. Filteren, sorteren en zoeken
7. Tijdreizen
8. GEO-ondersteuning
9. Paginering
10. Caching
11. Beperken van het aantal verzoeken per tijdsperiode
12. Foutafhandeling (status codes)

2.6.1 RESTful principes

De 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)[6] heeft een specifieke betekenis.

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

[6] 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.

Operatie	CRUD	Toelichting
POST	Create	Wordt gebruik 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 (RFC 7386).
DELETE	Delete	Verwijdert een specifieke resource.

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.

Idempotent	Onder idempotent wordt verstaan dat meerdere identieke verzoeken exact hetzelfde effect hebben als één verzoek.

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

API-03 ***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 (zie tabel).

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 toestand (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, zonder gebruik te hoeven maken van toestand-informatie op de server.

API-04 ***Toestandsinformatie wordt nooit op de server opgeslagen***

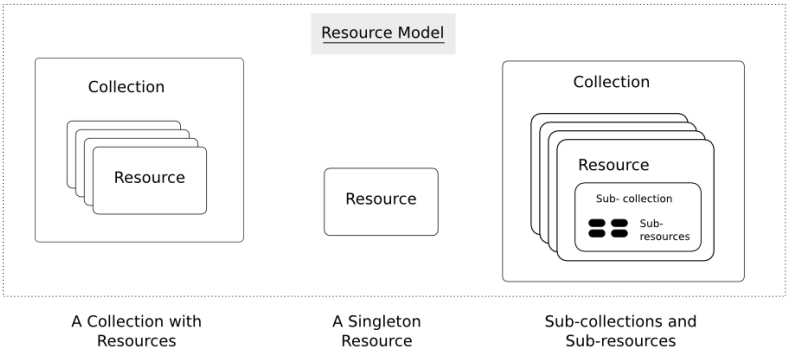
De client-toestand wordt volledig bijgehouden door de client zelf.

Alle stelsel API's zijn op het Stelselknooppunt beschikbaar. Het is niet toegestaan API's direct te benaderen. Stelsel API's zijn API's aangeboden door informatiehuizen, DSO projecten, door e-overheid bouwstenen en GDI. (gebruikt door het DSO en ontsloten via het Stelselknooppunt).

API-05 ***Communicatie met API's verloopt alleen via het Stelselknooppunt***

Het Stelselknooppunt is het centrale punt waarop alle API's beschikbaar worden gesteld. Het is niet toegestaan API's direct te benaderen. Het Stelselknooppunt is verantwoordelijk voor het verlenen van toegang en afdwingen van fair-use policies (indien van toepassing).

Wat zijn resources?
Het 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 (niet werkwoorden!) die relevant zijn vanuit het perspectief van de afnemer van de API. Dus resources zijn zelfstandige naamwoorden en operaties zijn werkwoorden. Operaties zijn acties die op resources worden uitgevoerd.



Het is mogelijk om interne datamodellen één-op-één toe te wijzen aan resources, maar dit hoeft niet per definitie zo te zijn. De crux is om alle niet relevante implementatiedetails te verbergen. Enkele voorbeelden van resources van het DSO zijn: aanvraag, activiteit, juridische regel, toepasbare regel, vergunning.

Als de resources geïdentificeerd zijn, wordt bepaald welke operaties van toepassing zijn en hoe deze worden ondersteund door de API. RESTful API's realiseren CRUD[7] operaties met behulp van HTTP-operaties:

`GET /aanvragen`	Haalt een lijst van aanvragen op
`GET /aanvragen/12`	Haalt een specifieke aanvraag op
`POST /aanvragen`	Creëert een nieuwe aanvraag
`PUT /aanvragen/12`	Wijzigt aanvraag #12 als geheel
`PATCH /aanvragen/12`	Wijzigt een gedeelte van aanvraag #12
`DELETE /aanvragen/12`	Verwijdert aanvraag #12

Het mooie van REST is dat er gebruik wordt gemaakt van de bestaande HTTP-operaties om de functionaliteit te implementeren met één enkele eindpunt. Hierdoor

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

zijn er geen aanvullende naamgevingsconventies nodig in de URI en blijft de URI-structuur eenvoudig.

> **API-06** ***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.

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

> **API-07** ***Definitie van het koppelvlak is in het Nederlands tenzij er sprake is van een officieel Engelstalig begrippenkader***
>
> Resources en de achterliggende entiteiten, velden, etc. (het informatiemodel en het externe koppelvlak) worden in het Nederlands gedefinieerd. Engels is toegestaan indien er sprake is van een officieel Engelstalig begrippenkader.

Naamgeving eindpunten in enkelvoud of meervoud?
De Keep It Simple Stupid (KISS) regel is hier van toepassing. Hoewel grammaticaal gezien het verkeerd aanvoelt om bij het opvragen van een enkele resource gebruik te maken van een resource naam in het meervoud, is het pragmatisch om te kiezen voor consistente eindpunten en altijd meervoud te gebruiken. Voor de afnemer is het gebruik veel eenvoudiger als er geen rekening gehouden hoeft te worden met enkel- en meervoud (aanvraag/aanvragen, regel/regels). Daarnaast is de implementatie eenvoudiger omdat de meeste ontwikkel frameworks het afhandelen van enkele resource (`/aanvragen/12`) en meervoudige resources (`/aanvragen`) met één controller kan oplossen.

> **API-08** ***Resource namen zijn zelfstandige naamwoorden in het meervoud***
>
> Namen van resources zijn zelfstandige naamwoorden en altijd in het meervoud, zoals aanvragen, activiteiten, vergunningen. Ook als het om het een enkel resource betreft.

Hoe omgaan 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 activiteit hoort bij één aanvraag. De activiteiten wordt als volgt benaderd via het eindpunt `/aanvragen`:

`GET /aanvragen/12/statussen`	Haalt een lijst van statussen op van aanvraag #12
`GET /aanvragen/12/statussen/5`	Haalt een specifieke status (#5) van aanvraag #12 op
`POST /aanvragen/12/statussen`	Creëert een nieuwe status voor aanvraag #12
`PUT /aanvragen/12/statussen/5`	Wijzigt statussen #5 van aanvraag #12
`PATCH /aanvragen/12/statussen/5`	Wijzigt een gedeelte van status #5 van aanvraag #12

`DELETE /aanvragen/12/statussen/5`	Verwijdert status #5 uit aanvraag #12

API-09 ***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 eindpunt gecreëerd. De afhankelijke resource heeft geen eigen eindpunt

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 /aanvragen/12/activiteiten
```

```
GET /activiteiten?aanvraag=12
```

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. De afnemers moet zelf het eindpunt van de gerelativeerde resource (relatie) aanroepen om deze op te vragen. Dit wordt ook wel aangeduid als lazy loading. De afnemer bepaalt zelf of de relatie geladen wordt en op welk moment.

De resource dient naast "lazy loading" (de standaard) ook "eager loading" te ondersteunen, zodat de afnemer kan aangeven dat relaties direct meegeladen moeten worden. Dit wordt gerealiseerd middels de standaard query-parameter `expand=true`. Dit voorkomt dat er twee of meer aparte aanroepen nodig zijn. In beide gevallen is de afnemer in control.

API-10 ***Resources ondersteunen "lazy" en "eager" laden van relaties***

Resources die een n-op-n relatie kennen ondersteunen zowel het teruggeven van identificaties van gerelateerde resources (lazy loading) als het teruggeven van de resources zelf (eager loading).

Automatische laden van gelinkte resources

Vaak wordt er 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 op te nemen in een resource. Op basis hiervan kan de afnemer, indien gewenst, de gelinkte resources zelf opvragen. Dit is vergelijkbaar met "lazy loading" in een ORM oplossing: resources worden alleen opgehaald als ze nodig zijn. In sommige gevallen, als de afnemer alle resources nodig heeft, is het efficiënter als de geneste resources in één keer opgehaald worden. Dit is vergelijkbaar met "eager loading" patroon in een ORM-oplossing.

Omdat dit tegen de REST principes in gaat, moet het toepassen van dit mechanisme een expliciete keuze zijn. De keuze wordt bij de afnemers van de API belegd, zij weten immers of ze extra informatie nodig hebben en welke informatie precies. Dit mechanisme wordt mogelijk gemaakt met de `expand` query-parameter.

Als `expand=true` wordt meegegeven, dan worden alle geneste resources geladen en embedded in het resultaat teruggegeven. Om de hoeveelheid informatie die geretourneerd wordt te beperken, is verplicht om te specificeren welke resources en zelfs welke velden van een resource teruggeven moeten worden. Hiermee wordt voorkomen dat de hele database wordt leeg getrokken.

Dit wordt gedaan door de resources als een komma's gescheiden lijst te specificeren, bijvoorbeeld: `expand=aanvrager,bevoegd_gezag`. De dot-notatie wordt gebruikt om specifieke velden van resources te selecteren. In het onderstaande voorbeeld wordt van de aanvrager alleen het veld "naam" teruggeven en van het bevoegd gezag de complete resource. Conform de HAL-standaard zijn de gelinkte resources embedded in de standaard representatie (zie ook aanpasbare representatie).

```
GET /aanvragen/12?expand=aanvrager.naam,bevoegd_gezag
```

Dit levert het volgende resultaat[8] op:

```
{
  "id": "https: //.../api/register/v1/aanvragen/12",
  "naam": "Mijn dakkapel",
  "samenvatting": "Ik wil een dakkapel bouwen!",
  "_embedded": {
    "aanvrager": {
      "naam": "Bob",
        "_links": {
          "self": {
            "href": "https: //.../api/register/v1/aanvragers/847",
            "title": "Bob"
          }
        }
    },
    "bevoegdGezag": {
      "id": "https: //.../api/register/v1/bevoegde-gezagen/42",
      "soort": "Gemeente",
      "naam": "Rotterdam",
      "_links": {
        "self": {
          "href": "https: //.../api/register/v1/bevoegde-gezagen/42",
          "title": "Rotterdam"
        }
      }
    }
  },
  "_links": {
    "self": {
      "href": "https: //.../api/register/v1/aanvragen/12",
      "title": "Mijn dakkapel"
    }
  }
}
```

Afhankelijk van de implementatie zal door het selectief kunnen laden van gelinkte resources in veel gevallen de overhead van database selecties, hoeveelheid serialisatie en hoeveelheid uitgewisselde data worden beperkt.

> **API-11** ***Gelinkte resources worden expliciet en selectief mee-geladen***
>
> Gelinkte resources worden expliciet en selectief mee-geladen als onderdeel van een resource verzoek middels de `expand` query-parameter.

Aanpasbare representatie

De gebruiker van een API heeft niet altijd de volledige representatie (lees alle velden) van een resource nodig. De mogelijkheid bieden om de gewenste velden te selecteren helpt bij het beperken van het netwerkverkeer (relevant voor lichtgewicht toepassingen), vereenvoudigt het gebruik van de API en maakt deze aanpasbaar (op maat). Om dit mogelijk te maken wordt de query-parameter `fields` ondersteund. De

[8] Dit resultaat volgt het HATEOAS concept en is op basis van HAL opgesteld.

query-parameter accepteert een door komma's gescheiden lijst met veldnamen. Het resultaat is een representatie op maat. Het volgende verzoek haalt bijvoorbeeld voldoende informatie om een gesorteerde lijst van open aanvragen te tonen.

In het geval van HAL zijn de gelinkte resources embedded in de standaard representatie. Met de hier beschreven fields parameter ontstaat de mogelijkheid om de inhoud van de body naar behoefte aan te passen.

```
GET /aanvragen?fields=id,onderwerp,aanvrager,wijzig_datum&status=open&sorteer=-
wijzig_datum
```

API-12 ***Representatie op maat wordt ondersteund***

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

Hoe om te gaan met acties die niet passen in het CRUD model?
Er zijn ook resource-acties die niet data manipulatie (CRUD) gerelateerd zijn. Een 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 aanvraag markeren met `PUT /aanvragen/12/markeringen` en verwijderen van de markering met `DELETE /aanvragen/12/markeringen`. Om de REST principes volledig te volgen moet ook de `GET methode` 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 service-eindpunt `/zoek` het meest logische. Gebruik werkwoorden in gebiedende wijs om het onderscheid t.o.v. "echte" resource end-points zo duidelijk mogelijk te maken. Dit is vanuit het perspectief van de gebruiker het meest logisch ontwerp. In de URI-strategie [2] is voor dit doel het fragment <service> opgenomen. Dit fragment volgt na de "/api" en maakt daarmee de routering eenvoudig.

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

API-13 ***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 eindpunt opgelost.

2.6.2 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 is versleuteld maakt het authenticatiemechanisme eenvoudiger. Hierdoor wordt het mogelijk om eenvoudige toegangstokens te gebruiken in plaats van toegangstokens met encryptie.

API-14 ***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-15 ***API's zijn alleen bruikbaar met behulp van een 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 wordt uitgegeven.

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 een token. Binnen het DSO is gekozen voor OAuth 2.0 als de standaarden voor het autorisatiemechanisme.

API-16 ***Tokens worden niet gebruikt in query parameters***

Er is een inherent beveiligingsprobleem bij het gebruik van een query parameter voor tokens omdat de meeste webservers queryparameters in de server logs wegschrijven.

Bij het gebruik van 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>`

Bij het ontbreken van de juiste headers zijn geen authenticatiedetails beschikbaar en dient de statuscode `403 Forbidden` terug te worden gegeven.

API-17 ***Autorisatie is gebaseerd op OAuth 2.0***

Een REST API mag geen state hebben. Elk verzoek moet daarom zijn voorzien van een token. OAuth 2.0 is hiervoor de voorgeschreven standaard.

API-18 ***Authenticatie voor API's met toegangsbeperking of doelbinding is gebaseerd op PKIoverheid***

In het geval van API's met toegangsbeperking of doelbinding zal er aanvullend sprake zijn van authenticatie op basis PKIoverheid certificaten en tweezijdig TLS. In de URI-strategie [2] is vastgelegd welke kaders gelden voor API's die gebruik maken van tweezijdig TLS.

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

BEST PRACTICE

Beslistabel
Met dit in het achterhoofd is het advies om voor statuscode `401 Unauthorized`, `403 Forbidden` en `404 Not Found`, de volgende regels te hanteren:

Bestaat de resource?	Kan de autorisatie worden bepaald?	Geautoriseerd?	HTTP statuscode
✔	✔	✔	`20x (200 OK)`
✔	✔	✖	`401 Unauthorized`
✔	✖	?	`403 Forbidden`
✖	✔	✔	`404 Not Found`
✖	✔	✖	`403 Forbidden`
✖	✖	?	`403 Forbidden`

Tabel 1 – 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 bijkomend voordeel van de strategie om eerst te bepalen of er toegang is, meer ruimte biedt om de access control logica te scheiden van de business code.

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

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 korter en gegarandeerd "web-veilig" zijn, is het advies om alleen de base64-gecodeerde variant van 22 tekens te gebruiken. 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 dit te voorkomen dienen in dit geval zogenaamde "restricted" API-key's te worden uitgegeven en gebruikt.

BEST PRACTICE

Gebruik "publieke" API-Key

In JavaScript dient alleen gebruik te worden gemaakt van een zogenaamde "restricted" API-key. Dit is API-key die gekoppeld is aan specifieke kenmerken van de aanroeper (web-app, mobile-app), waaronder een clientId en/of verwijzer-URL.

CORS-policy

Webbrowsers implementeren een zogenaamde "same origin policy", een belangrijk beveiligingsconcept om te voorkomen dat verzoeken naar een ander domein gaan dan waarop het is aangeboden. Hoewel dit beleid effectief is in het voorkomen van aanroepen in verschillende domeinen, voorkomt het ook legitieme interactie tussen een API's en clients van een bekende en vertrouwde oorsprong.

BEST PRACTICE

Controleer toegang en gebruik CORS-header

Controleer eerst het domein van de inkomende aanvraag en genereer de response-header afhankelijk van of dit domein verzoeken mag verzenden of niet (whitelist). In dat geval wordt alleen dit specifieke domein aan de respons-header `Access-Control-Allow-Origin` toegevoegd.

! Het is technisch mogelijk om in de `Access-Control-Allow-Origin` respons-header een wildcard ("*") terug geven en zo alle bronnen toe te staan. Dit is een onveilige praktijk!

2.6.3 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. Wanneer de documentatie is weggestopt in pdf-bestanden en achter een inlog, dan vormt dit een drempel voor ontwikkelaars om aan de gang te gaan en de documentatie is niet vindbaar met zoekmachines.

API-19 ***Documentatie is gebaseerd op OAS 2.0 of hoger***

Specificaties (documentatie) is beschikbaar als Open API Specification (OAS)[9] V2.0 of hoger. De nieuwere variant V3.0 is beschikbaar en het gebruik van deze versie wordt sterk aangeraden.

BEST PRACTICE

OAS via basis-URI beschikbaar in JSON-formaat

Om te zorgen dat de actuele documentatie altijd publiekelijk toegankelijk is, is het advies om de inhoud van de Open API Specification op het “root end-point” van de API beschikbaar te maken in JSON-formaat:

```
https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v1
```

Maakt de OAS behorend bij v1 van deze API beschikbaar.

Hiermee wordt een unieke plek (die altijd synchroon loopt met de features die de API biedt) gekoppeld aan de actuele documentatie.

API-20 ***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 is het Engels en als er sprake is van een officieel Engelstalig begrippenkader.

De documentatie dient voorzien te zijn van voorbeelden inclusief complete request en response 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-21 ***Documentatie wordt getest en geaccepteerd***

De API-manager van het Stelselknooppunt biedt de mogelijk om direct vanuit de documentatie de API te testen. Hier dient bij het opzetten van de documentatie rekening mee gehouden te worden en dit dient getest te worden.

[9] Formerly Known As (FKA) the Swagger Specification. The goal of The OpenAPI Specification is to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interfaces have done for lower-level programming, OpenAPI removes the guesswork in calling the service.

Als een API in productie is mag het "contract" (koppelvlak) niet zonder voorafgaande kennisgeving worden gewijzigd. De documentatie moet voorzien zijn van een uitfaseringsplanning (deprecation schedule) en alle details van de wijziging bevatten. Wijzigingen worden via een publiek toegankelijke blog als changelog bekendgemaakt en via een mailinglijst. De mailinglijst wordt beheerd binnen het Stelselknooppunt. Hierbij wordt primair gebruik gemaakt van de emailadressen die zijn gekoppeld aan de uitgifte van API-keys.

API-22 ***Wijzigingen worden gepubliceerd met een uitfaseringschema***

Koppelvlak wijzigingen worden met bijbehorende planning op een publiek toegankelijke blog als changelog bekendgemaakt en via een mailinglijst.

2.6.4 Versionering

API's zijn altijd geversioneerd. Versioneren zorgt voor een soepele overgang bij wijzigingen. De oude en nieuwe versies worden voor een beperkte overgangsperiode (één jaar) aangeboden. Er worden bovendien maximaal 3 versies van een API ondersteund. 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-23 ***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 (1 jaar) naast elkaar aangeboden.

Er zijn verschillende meningen over de vraag of de versie in de URI of in de header hoort. De URI-strategie [2] 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 2.3.

`https://.../api/register/v1/aanvragen/12` Vraagt via API v1 aanvraag #12 op

Het versienummer begint bij 1 en wordt met 1 opgehoogd voor elke major release waarbij het koppelvlak niet backward compatible wijzigt. De minor en patch versienummers staan altijd in de response-header van het bericht zelf in het 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`

Via de optionele request-header kan één minor/patch-versie worden aangewezen. Hiermee wordt bedoeld dat in (pre)productie- of aansluitomgeving naast bijvoorbeeld v1 en v2 (de aangewezen

versies die alleen bereikbaar zijn via de URI's) ook nog de mogelijkheid bestaat om één "oudere" of "nieuwere" versie van deze API's aan te wijzen en via de request-header te benaderen. De onderstaande URI's wijzen bijvoorbeeld naar de aangewezen productie-release van de API die alleen bereikbaar zijn via de URI:

```
https://service.omgevingswet.overheid.nl/publiek/catalogus/api/raadplegen/v1
```

v1.0.2

```
API-version: 1.0.2 (response-header)
```

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

v2.1.0

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

Het weglaten van de request-header (`API-version:x.y.z`**)** selecteert altijd de "aangewezen" productieversie. Indien er voor V2 ook één andere aangewezen versie beschikbaar is, met bijvoorbeeld versienummer V2.1.1, dan kan deze via hetzelfde basis end-point worden aangeboden en geselecteerd door het meegeven van de juiste request-parameter:

```
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)
```

Het toevoegen van een end-point of een niet verplichte attribuut aan de payload zijn voorbeelden van wijzigingen die backward compatible zijn.

API-24 ***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 wel.

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. Immers, als een nieuwe release van de API niet tot backward incompatibiliteit 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) clients deze nieuwe versie implementeren.

Omdat we geen clients willen breken kunnen we niet van de een op de andere dag overschakelen van de oude naar de nieuwe versie, zoals dat bijvoorbeeld bij een update van een website wel vaak gebeurt. Daarom is het noodzakelijk om na de livegang van de nieuwe versie óók de oude versie in de lucht te houden. Omdat we de oude versie niet tot in de eeuwigheid willen blijven onderhouden en juist iedereen

willen stimuleren om de nieuwe versie te gaan gebruiken, communiceren we een periode waarin clients de gelegenheid krijgen om hun code aan te passen aan de nieuwe versie. Deze periode noemen we de deprecation periode. De lengte van deze periode kan verschillen per API, vaak is dit zes maanden, maar 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 gecommuniceerd worden:

- Een link naar de (documentatie van de) nieuwe versie van de API;
- Deprecation periode met exacte datum waarop de deprecated versie offline wordt gehaald;
- Migratieplan om eenvoudig over te stappen naar de nieuwe versie;
- Welke features er toegevoegd, gewijzigd of verwijderd worden;
- Welke wijzigingen de huidige implementaties kunnen breken;
- Contactmogelijkheid om een verlenging van de deprecation periode aan te vragen.

Deze zaken dienen gecommuniceerd te worden via de volgende kanalen:

- Per e-mail van de clients (indien bekend);
- Duidelijk leesbaar in de API documentatie van de oude versie;
- Via een topic op het DSO forum;
- Met een Warning response-header in alle responses van de oude API.

Stap voor stap betekent dit het volgende:

1. Lanceren nieuwe versie;
2. Bepalen deprecation periode;
3. Schrijven migratieplan;
4. Communiceren in de API-documentatie van de oude versie;
5. Deprecation periode communiceren per e-mail, forum en eventuele andere kanalen;
6. Warning header toevoegen aan responses van de oude versie;
7. Logs checken om gebruik van de oude versie te monitoren gedurende deprecation periode;
8. End-point oude versie dichtzetten op geplande datum en feedback monitoren;
9. Indien er binnen twee weken geen feedback op de oude versie komt kan de oude versie (inclusief docs) verwijderd worden;

De Warning response-header
De Warning header (zie: RFC 7234) die we hier gebruiken heeft warn-code 299 ("Miscellaneous Persistent Warning") en het API end-point (inclusief versienummer) als de warn-agent van de warning, gevolgd door de warn-text met de human-readable waarschuwing. Voorbeeld:

```
Waarschuwing: 299 https://service.../api/.../v1 "Deze versie van de API is verouderd en
zal uit dienst worden genomen op 2018-02-01. 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.

API-25 ***Gebruikers van een 'deprecated' API worden actief gewaarschuwd***

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

2.6.5 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
  }
}
```

API-26 ***JSON first - API's ontvangen en versturen JSON***

API's ontvangen en versturen JSON.

API-27 ***API's zijn optioneel voorzien van een JSON Schema***

API's ondersteunen JSON Schema (http://json-schema.org), zodat validatie mogelijk (optioneel) is en vereenvoudigd wordt.

API-28 ***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.

API-29 ***API's controleren dat de Content-Type header is ingesteld***

Er wordt gecontroleerd of het Content-Type header is ingesteld op `application/json` of andere ondersteunde content types, anders wordt HTTP-status code `415 Unsupported Media Type` geretourneerd.

Veldnamen in snake_case, camelCase, UpperCamelCase of koppelteken?
Bij veldnamen wordt gebruik gemaakt van camelCase.

API-30 ***Woorden in veldnamen zijn gedefinieerd in camelCase***

Een veldnaam begint met kleine letters (eerste woord) en ieder opvolgend woord begint met een hoofdletter.

Pretty print
De meeste REST clients en browsers (al dan niet met extensies) kunnen JSON netjes geformatteerd weergeven, ook als de response geen white-space bevat.

API-31 ***Pretty print is standaard uitgeschakeld***

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

Gebruik geen envelop
Veel API's wikkelen antwoorden in enveloppen zoals hieronder is weergegeven:

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

Een toekomstbestendig API is vrij van enveloppen.

API-32 ***Een JSON-response heeft geen omhullende envelop***

Er worden standaard geen enveloppen toegepast.

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?

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

Resulteert in:

```
{"Name": "John Smith", "Age": 23}
```

En

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

Resulteert in:

```
Name=John+Smith&Age=23
```

API-33 ***API's ondersteunen 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.

2.6.6 Filteren, sorteren en zoeken

Er wordt gekozen om de basis URL's van resources zo eenvoudig mogelijk te houden. 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 je bijvoorbeeld een lijst met aanvragen wilt opvragen van het eindpunt `/aanvragen` en deze wilt beperken tot open aanvragen, dan wordt het verzoek `GET/aanvragen?status=open` gebruikt. Hier is `status` een veld waarop gefilterd kan worden.

API-34 ***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.

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

```
[
  {
   "id": 1,
   "status": "actief",
   "overheid": {
     "code": "0000",
     "naam": "Ministerie van BZK"
    }
  },
  {
    "id": 2,
    "status": "inactief",
    "overheid": {
    "code": "9901",
    "naam": "Provincie Gelderland"
    }
  }
]
```

Alle objecten met de status "actief" kunnen worden gefilterd met `/?status=actief`. Maar als daarnaast ook op objecten met code "0000" van de overheid gefilterd moeten worden, heeft dit betrekking op een geneste property. Hier kan dan de puntnotatie (zoals bij Javascript) voor worden gebruikt: `/?status=actief&overheid.code=0000`.

Sorteren

Voor sorteren wordt de query-parameter `sorteer` gebruikt. 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 sorteervolgorde gesorteerd. Een aantal voorbeelden:

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

API-35 ***Voor sorteren wordt de query-parameter `sorteer` gebruikt***

De generieke query-parameter `sorteer` 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.

Zoeken

Soms zijn eenvoudige filters onvoldoende en is de kracht van vrije-tekst zoekmachines nodig. Hiervoor is de bouwsteen Zoeken (technisch component ElasticSearch) beschikbaar. API's ondersteunen dit middels de query-parameter `zoek`. Een zoekopdracht die meegegeven wordt met deze query-parameter wordt 1-op-1 doorgegeven aan de bouwsteen Zoeken. Het resultaat wordt in dezelfde representatie teruggegeven.

API-36 ***Voor vrije-tekst zoeken wordt een query-parameter `zoek` gebruikt***

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

Voorbeelden van de combinatie filteren, sorteren en zoeken:

`GET /aanvragen?sorteer=-wijziging_datum`	Haalt een lijst van recente aanvragen op.
`GET / aanvragen?status=gesloten&sorteer=-wijziging_datum`	Haalt een lijst van recent gesloten aanvragen op.
`GET /aanvragen?zoek=urgent&status=open &sorteer=-prio,aanvraag_datum`	Haalt een lijst van aanvragen op met de status 'open' en waarin het woord 'urgent' voorkomt, gesorteerd van hoogste naar laagste prioriteit, en daarbinnen op aanvraagdatum van oud naar nieuw.

Wildcards

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

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-37 ***API's die vrije-tekst zoeken ondersteunen kunnen overweg met twee soorten wildcard karakters: * en ?***

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 wildcard karakters 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) worden gebruikt als woordscheiding en wildcard-matching werkt alleen binnen een enkel woord. Bijvoorbeeld, `r*te*` komt overeen met de **`r`**`uimte`**`te`**`lijk`, maar niet met **`r`**`uimte` **`te`**`kort`.
- Wildcards werken alleen op JSON-velden met tekst (string) waarden.

Aliassen voor terugkerende queries

Om de API ervaring verder te verbeteren is het mogelijk om terugkerende queries als end-points aan te bieden. Zo kunnen onlangs gesloten aanvragen als volgt worden benaderd:

```
GET …/aanvragen/recent_gesloten
```

2.6.7 Tijdreizen

Informatie over een resource is onderhevig aan verandering. Tijdreizen is een mechanisme waarmee het mogelijk is om op standaard manier informatie op te vragen over 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.

Het basisprincipe voor het tijdreizen i.r.t. de URI-strategie [2] 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 (gebaseerd op RFC3339 / ISO 8601):

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

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

In de Kaderstellende notities: URI-strategie [2] en Tijdreizen [3] 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 dus 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/1.1 403`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`"type": "…/id/<c>/parameter/TijdreizenNietOndersteund",`
	`"title": "{Het verzoek is begrepen, maar de tijdreisparameters worden niet ondersteund}",`
	`"status": 403,`
	`"detail": "{Tijdreizen wordt niet ondersteund, verwijder alle tijdreisparameters}",`
	`"instance": "/resource/#id"`
	`}`

Inwerkingtreding	`HTTP/1.1 403`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`"type": "…/id/<c>/parameter/InWerkingTredingNietOndersteund'",`
	`"title": "{Het verzoek is begrepen, maar de tijdreisparameter 'inWerkingOp' wordt niet ondersteund}",`
	`"status": 403,`
	`"detail": "{Tijdreizen met een inwerkingstredingsmoment wordt niet ondersteund, verwijder de parameter 'inWerkingOp'}",`
	`"instance": "/resource/#id"`
	`}`

Toekomst	`HTTP/1.1 403`
	`Content-Type: application/problem+json`
	`Content-Language: nl`
	`{`
	`"type": "…/id/<c>/parameter/ToekomstNietOndersteund",`
	`"title": "{Het verzoek is begrepen, maar tijdreisparameters mogen geen tijdstip in de toekomst bevatten"}",`
	`"status": 403,`
	`"detail": "{Tijdreizen naar de toekomst wordt niet ondersteund, verwijder de tijdreisparameters of gebruik een tijdstip in het verleden}",`
	`"instance": "/resource/#id"`
	`}`

Robuustheid
Bij de verwerking van tijdreisvragen dient ook rekening te worden gehouden met het ontbreken van historie en globale vragen in een resource-collectie. In volgende specifieke gevallen 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`
	`{`

Soort verzoek	HTTP statuscode en toelichting

	<pre> "type": "…/id/<c>/BestaatNiet",
	  "title": "De gevraagde versie bestaat niet.",
	  "status": 404,
	  "detail": "Versie 2017-01-01 van regeling 123 bestaat niet.",
	  "instance": "/regelingen/v1/123?geldigOp=2017-01-01"
	}</pre>

Resource-collectie	<pre>GET …/regelingen/v1?beschikbaarOp=2017-01-01T00:00:00
	
	HTTP/1.1 200
	Content-Type: application/json+hal
	{
	   " links": {
	      "self": {
	        "href": "/r-n/v1/123?beschikbaarOp=2017-01-01T00:00:00"
	      },
	      "items": [
	       { "href": "/r-n/v1/123?beschikbaarOp=2017-01-01T00:00:00" },
	       { "href": "/r-n/v1/456?beschikbaarOp=2017-01-01T00:00:00" },
	       { "href": "/r-n/v1/789?beschikbaarOp=2017-01-01T00:00:00" }
	      ]
	   }
	}</pre>

2.6.8 GEO-ondersteuning

REST API's voor het werken met geometrieën kunnen een filter aanbieden op basis van geografische gegevens. Het is hierbij belangrijk om een onderscheid te maken tussen een geometrie in het resultaat (response) en een geografische filter in de aanroep (request). Het is immers niet vanzelfsprekend dat als iemand wil weten in welk perceel ik 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 dit 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-38 ***GEO API's ontvangen en versturen GeoJSON***

Voor GEO API's wordt de standaard GeoJSON gebruikt.

Resultaat (response)
In een JSON API wordt een geometrie teruggegeven als GeoJSON. We kiezen er voor om dit binnen de application/json te 'wrappen' in een apart GeoJSON object.

API-39 ***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 ge-embed zijn.

Aanroep (requests)
Een geografisch filter kan erg complex en groot zijn. Het is daarom een best practice om dergelijke complexe vragen niet in de request URI, maar in de body mee te sturen. Omdat GET geen payload mag hebben (hoewel sommige clients dit wel ondersteunen) moet hier dus een andere methode voor gebruikt worden.

In analogie met API's zoals die van Elasticsearch, is een POST naar een apart end-point de juiste weg om te bewandelen. 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`, etc.) het GeoJSON object (in dit geval een Point, dus één coördinaat) bevat:

```
POST …/api/zoek/v1/panden
{                                                                    (request-body)
  "_geo": {
                "contains": {
                                "type": "Point",
                                "coordinates": [5.9623762, 52.2118093]
                             }
              }
}
```

API-40 ***Voor GEO queries is een POST end-point beschikbaar***

Geometrische queries worden in een POST naar een apart end-point verzonden.

Omdat we ons met het geo end-point beperken tot een GEO-query en we wellicht ook gecombineerde queries willen doen is het sterk aan te bevelen om een generiek query end-point in te richten:

```
POST …/api/zoek/v1/panden
{                                                                    (request-body)
  "_geo": {
                "contains": {
                                "type": "Point",
                                "coordinates": [5.9623762, 52.2118093]
                             }
              },
  "status": "Actief"
}
```

API-41 ***POST end-points zijn uitbreidbaar voor gecombineerde vragen***

De geometrie is uitbreidbaar met andere properties voor gecombineerde queries.

Naast `contains` kan er ook `intersects` (snijdt) of `within` (valt binnen) als operators gebruikt worden. De benamingen van deze operators komen uit de GEO-wereld en die willen we niet opnieuw uitvinden.

Omdat we voor de geometrie een GeoJSON object gebruiken hoeven we hier geen syntax meer voor te verzinnen. Daarnaast kunnen we in het GeoJSON object ook aangeven welk CRS (zie volgende paragraaf) de opgegeven geometrie hanteert.

API-42 ***Resultaten van een globale geometrische zoekvraag worden in de relevante geometrische context geplaatst***

In het geval van een globale zoekvraag …/api/zoek/v1 is het noodzakelijk om de resultaten in een relevante geometrische context te plaatsen.

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

```
POST …/api/zoek/v1
{                                                                    (response-body)
 "_embedded": {
               "results": [{
                            "type": "enkelbestemming",
                            "_links": {
                                       "self": {
                                                 "title": "Enkelbestemming 1234",
                                                 "href": ".../enkelbestemmingen/1234"
                                                }
                                      }
                           },
                           {
                            "type": "dubbelbestemming",
                            "_links": {
                                       "self": {
                                                 "title": "Dubbelbestemming 8765",
                                                 "href": ".../dubbelbestemmingen/8765"
                                                }
                                      }
                          }]
              }
}
```

CRS-negotiation

Het default CRS (Coordinate Reference System) 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 (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 ook WGS84 te ondersteunen.

API-43 ***Het voorkeur-coördinatenstelsels (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 expliciet worden opgegeven.

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, ETRS89 en WGS84.

OE-01 ***Het standaard coördinatenstelsels is nog niet definitief bepaald. Op dit moment wordt ETRS89 als het voorkeur CRS gehanteerd en gewerkt met de voorlopige uitgangspunten in de Overall GAS [1].***

Een nadere opsomming van de uitgangspunten voor het CRS in de OGAS [1]:

- Leg als bronsysteem binnenkomende formaat vast (juridische context);
- Coördinatenstelsels API-strategie: vraag/antwoord in RD; ETRS89; WGS84;
- Overweeg no-regret: vastleggen in ETRS89 én RD i.p.v. realtime bepalen;
- Hanteer RDNAPTRANS™ 2018 bij transformatie RD versus ETRS89 (correctiegrid);
- Presentatie afhankelijk van context (o.a. gebruikerswensen);
- Uitwisselingsformaat (notatie) ETRS89 en WGS84 X Y in decimale graden; DD.ddddddddd:
- voorbeeld: 52.255023450
- Uitwisselingsformaat (notatie) RD X Y in meters (niet afgerond): 195427.5200 311611.8400

API-44 ***Het coördinatenstelsel (CRS) van de vraag en het antwoord wordt in de header meegestuurd***

Het gekozen coördinatenstelsel (CRS) voor zowel de vraag als het antwoord worden meegestuurd als onderdeel van de request-header en de response-header. Bij het ontbreken van deze headers wordt `412 Precondition Failed` teruggegeven. Hiermee wordt voorkomen dat per abuis met het verkeerde CRS wordt gewerkt.

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 een inwindatum.

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:3856	WGS84, Wereld (Web-Mercator-projectie)
Content-Crs	EPSG:4258	ETRS89, Europees
Content-Crs	EPSG:28992	RD, Nederlands

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:3856	WGS84, Wereld (Web-Mercator-projectie)
Accept-Crs	EPSG:4258	ETRS89, Europees
Accept-Crs	EPSG:28992	RD, Nederlands

CRS-transformatie

Voor het transformeren tussen coördinaatreferentiesystemen is binnen de Rijksoverheid software met een keurmerk beschikbaar.

BEST PRACTICE

Transformatiediensten

Hieronder worden de aanbevolen diensten en transformatiestappen per specifieke transformatie weergegeven:

Van	Naar	Transformatiedienst(en)	Toepassingsgebied
WGS84	ETRS89	PCTRANS (EPSG:3856→ EPSG:4258)	zee
ETRS89	WGS84	PCTRANS (EPSG:4258 → EPSG:3856)	zee
RD	ETRS89	RDNAPTRANS (EPSG:28992 → EPSG:4258)	land
ETRS89	RD	RDNAPTRANS (EPSG:4258 → EPSG:28992)	land
WGS84	RD	RDNAPTRANS (EPSG:3856→ EPSG:28992)	land
RD	WGS84	RDNAPTRANS (EPSG:28992 → EPSG:3856)	land

OE-02 ***De ondersteuning van CRS-transformaties vereist nader onderzoek met daarin twee hoofdvragen: (1. welke diensten dienen welke transformaties te ondersteunen? (2. welke nauwkeurigheid is voor deze transformaties vereist?***

Advies NSGI t.a.v. de nauwkeurigheid is 0,0001 m / 0,000 000 001 graad (0,1 mm). Een nauwkeurigheid beter dan 1 mm is vereist voor de transformatie tussen ETRS89 en RD om de merknaam RDNAPTRANS(TM) te mogen gebruiken.

API-45 ***Het gewenste coördinatenstelsel 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.

2.6.9 Paginering

Voor paginering wordt aangesloten op Hypertext Application Language (HAL). Dit is een standaard voor het uitdrukken van hyperlinks met JSON [RFC4627]. Aan geretourneerde objecten worden twee gereserveerde velden (gedefinieerd door RFC5988) `_links` (verplicht) en `_embedded` (optioneel) toegevoegd. Deze velden vertegenwoordigen respectievelijk hyperlinks en embedded resources.

Hier is een voorbeeld van een JSON+HAL representatie:

```
{
  "_links": {
      "self": {
        "href": "https://.../api/registratie/v1/aanvragen?pagina=3"
      },
      "first": {
        "href": "https://.../api/registratie/v1/aanvragen"
      },
      "prev": {
        "href": "https://.../api/registratie/v1/aanvragen?pagina=2"
      },
      "next": {
        "href": "https://.../api/registratie/v1/aanvragen?pagina=4"
      },
      "last": {
        "href": "https://.../api/registratie/v1/aanvragen?pagina=5"
      }
  },
  "id": "https://.../api/registratie/v1/aanvragen/12",
  "naam": "Mijn dakkapel",
  "samenvatting": "Ik wil een dakkapel bouwen!",
  "_embedded": {
     "aanvrager": {
       "naam": "Bob"
     }
   }
}
```

Indien het “plain” JSON, GeoJSON of iets anders dan HAL betreft zijn er geen _links. Deze kunnen dan opgenomen worden in de link response headers. Naast de representatie wordt de volgende metadata teruggegeven als HTTP-headers.

HTTP-header	Toelichting
X-Total-Count (optioneel)	Totaal aantal resultaten.
X-Pagination-Count (optioneel)	Totaal aantal pagina’s.

X-Pagination-Page	Huidige pagina.
X-Pagination-Limit	Aantal resultaten per pagina.

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.

API-46 ***Paginering wordt gerealiseerd op basis van JSON+HAL***

Aan de representatie worden twee gereserveerde velden (gedefinieerd door RFC5988) "_links" (verplicht) en `_embedded` (optioneel) toegevoegd. Daarnaast wordt paginerings-metadata als HTTP-headers meegegeven.

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

2.6.10 Caching

Voor sommige resources kan het nuttig zijn om caching toe te passen. HTTP biedt voor caching standaard mechanismes. Door deze mechanismes te gebruiken wordt gebruik gemaakt van standaard caching oplossingen van de client en in de infra-structuur.

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 cached de resource en de ETag. 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. 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 RFC 1123 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 dan 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-47 ***Waar relevant wordt caching toegepast***

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`).

2.6.11 Beperken van het aantal verzoeken per tijdsperiode

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

API-48 ***Beperken van het aantal verzoeken per tijdsperiode wordt centraal opgelost door het Stelselknooppunt***

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.

RFC 6585 introduceerde een HTTP-statuscode `429 Too Many Requests` die wordt gebruikt om het overschrijden van het aantal verzoeken te melden aan de gebruiker.

API-49 ***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.

OE-03 ***Het is nog onduidelijk of de `X-Rate-Limit` headers door het Stelselknooppunt ondersteund zullen worden.***

2.6.12 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 alleen met een eigen set van velden.

De API moet altijd zinvolle HTTP-statuscodes teruggeven. HTTP-statuscodes zijn opgesplitst in twee categorieën:

- 400 reeks: voor inhoudelijke fouten
- 500 reeks: voor server problemen

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

- Een uniek fouttype in de vorm van een URI die verwijst naar informatie in externe (HTML) documentatie,
- Een korte maar nuttig foutmelding,
- 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 (de fout-instantie). Het strekt de voorkeur om een URN te gebruiken waarmee alleen daartoe gerechtigde gebruikers details kunnen opzoeken in de fout-log.

De basis voor deze standaardformaten is: https://tools.ietf.org/html/rfc7807.
Een JSON-representatie van een fout ziet er als volgt uit:

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

OE-04 ***De collectie <c> waarin fouten kunnen worden ondergebracht zal in overleg met de beheerder van de Stelselcatalogus worden bepaald.***

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

API-50 ***Foutafhandeling is gestandaardiseerd***

API's ondersteunen de gestandaardiseerde foutmeldingen van de HTTP-statuscodes 400 en 500 reeks inclusief een verwerkbare JSON representatie in lijn met RFC7807.

In bijlage A staan de gestandaardiseerde foutmeldingsformaten in een verwerkbare 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-51 ***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, 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. 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.

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).

Bijlage A: Standaard foutmeldingsformaten

Een API moet minimaal de onderstaande standaard foutmeldingsformaten 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: https://tools.ietf.org/html/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/ValidatieMislukt",
  "title": "Validatie mislukt.",
  "status": 400,
  "invalid-params":
   [
     {
      "type": "https://content.../id/<c>/validatie/Emailadres",
      "name": "email",
```

```
            "reason": "Ongeldig emailadres"
          },
          {
            "type": "https://content.../id/<c>/validatie/Telefoonnummer",
            "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/id/<c>/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/id/<c>/ValidatieFout",
    "title": "Validatie mislukt.",
    "status": 400,
    "invalid-params":
    [
      {
         "type": "https://content.../id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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/id/<c>/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 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/id/<c>/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"
}
```

13. 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/id/<c>/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"
}
```

14. 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/id/<c>/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).

Bijlage B: Afkortingen

Afkorting	Toelichting
API	Application Programming Interface
DSO	Digital Stelsel Omgevingswet
GDI	Generieke Digitale Infrastructuur
HAL	Hypertext Application Language
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
ORM	Object Relational Mapping
REST	Representational State Transfer
RFC	Request For Change
SAB	Stelsel Architectuur Board
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Bijlage C: Begrippen

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: `{ 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.
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 D: Principes en Kaders

Principes

Kaders

Kaders

Kaders

Bijlage E: Open Einden

Backlog	
OE-01	HET STANDAARD COÖRDINATENSTELSELS IS NOG NIET DEFINITIEF BEPAALD. OP DIT MOMENT WORDT ETRS89 ALS HET VOORKEUR CRS GEHANTEERD EN GEWERKT MET DE VOORLOPIGE UITGANGSPUNTEN IN DE OVERALL GAS [1]. 33
OE-02	DE ONDERSTEUNING VAN CRS-TRANSFORMATIES VEREIST NADER ONDERZOEK MET DAARIN TWEE HOOFDVRAGEN: (1. WELKE DIENSTEN DIENEN WELKE TRANSFORMATIES TE ONDERSTEUNEN? (2. WELKE NAUWKEURIGHEID IS VOOR DEZE TRANSFORMATIES VEREIST? ADVIES NSGI T.A.V. DE NAUWKEURIGHEID IS 0,0001 M / 0,000 000 001 GRAAD (0,1 MM). EEN NAUWKEURIGHEID BETER DAN 1 MM IS VEREIST VOOR DE TRANSFORMATIE TUSSEN ETRS89 EN RD OM DE MERKNAAM RDNAPTRANS(TM) TE MOGEN GEBRUIKEN. 35
OE-03	HET IS NOG ONDUIDELIJK OF DE `X-RATE-LIMIT` HEADERS DOOR HET STELSELKNOOPPUNT ONDERSTEUND ZULLEN WORDEN. 37
OE-04	DE COLLECTIE <C> WAARIN FOUTEN KUNNEN WORDEN ONDERGEBRACHT ZAL IN OVERLEG MET DE BEHEERDER VAN DE STELSELCATALOGUS WORDEN BEPAALD. 38