edustandaard

Edukoppeling

M2M gegevensuitwisseling binnen het onderwijs

Profiel voor communicatie via RESTful API's

Edustandaard
Datum: 2 september 2026
Versie 1.0
Status: concept

Inhoudsopgave

1. Status van dit document

Dit document is een conceptversie van het Profiel voor communicatie via RESTful API's (hierna Communicatie-profiel).

De nieuwe Edukoppeling Architectuur versie 3.0 geeft context aan synchrone en asynchrone communicatie tussen ketenpartners, waarbij het Communicatie-profiel de communicatie standaardiseert. De nieuwe RESTful-API architectuur wijkt sterk af van de vorige versie[1]. In die versie onderkende we al RESTful API's, maar was nog sterk gericht op webservices en een Digikoppeling-indeling. Verder onderkende we al patronen die synchrone RESTful API's gebruikte om asynchrone communicatie te implementeren, maar werd niet beschreven hoe deze uitwisseling gestandaardiseerd en zoveel mogelijk ontkoppeld kon plaatsvinden.

In de nieuwe opzet staat asynchrone communicatie meer centraal. Deze architectuur sluit veel meer aan op ontkoppelde architecturen zoals een Event-driven architectuur (EDA), en wordt toegepast op ketenpartnerlandschap binnen het onderwijs.

Dit is zeer relevant in een speelveld waarin geïntegreerd wordt met systemen van de Nederlandse overheid, de Nederlandse onderwijssector (sectorpartners en onderwijsinstellingen) en verschillende andere derde partijen. Hoge ontkoppeling minimaliseert effect van verkeerspieken en (tijdelijke) (on)geplande onbeschikbaarheid op de rest van de keten.

1.1. Documenthistorie

Versie	Status	Auteur	Datum	Opmerking
1.0	Concept	D. Grootendorst	Mei 2026 Sep 2026 Okt 2026	Initiële versie profiel. Input werkgroep 20 mei verwerkt en split naar architectuur, OAuth-profiel en Communicatie profiel gemaakt. Focus op koppeling tussen 'intermediairs'. Feedback werkgroep september '26 verwerkt.

[1] https://www.edustandaard.nl/app/uploads/2021/10/2021-02-10-Edukoppeling-Architectuur-2.0-definitief.pdf

2. Inleiding

2.1. Doel van Edukoppeling

Het doel van Edukoppeling is standaardisatie van de technische afspraken op het M2M-koppelvlak waardoor het ontwikkelen van ketensamenwerkingen by design veilig en eenvoudiger wordt. De toepassing van Edukoppeling zorgt ervoor dat, op het niveau van de applicatielaag, gesloten data tijdens transport van de ene naar de andere organisatie niet ongeoorloofd kan worden ingezien of gemanipuleerd. De standaard gaat over de afhandeling van berichten (het transport) en niet over de inhoud van berichten.

Met Edukoppeling worden voor ketensamenwerking een aantal ketenfuncties in de applicatielaag geregeld (zie hoofdstuk Toepassingsscenario's).

2.2. Doelgroep

Dit document is bedoeld voor ICT-specialisten die betrokken zijn bij het ontwerpen en ontwikkelen van systeem-naar-systeem (M2M) koppelingen voor RESTful API's. Het gaat hier om werknemers (ontwikkelaars, architecten, projectmanagers, informatiemanagers etc.) werkzaam bij onderwijs gerelateerde organisaties, zowel in de publieke als private sector. De lezer van dit document willen wij vragen om zaken die ontbreken of onduidelijk zijn te melden bij de beheerorganisatie Edustandaard[2]. Edustandaard is een open platform waar partijen binnen het onderwijsveld bij elkaar komen om afspraken te maken. Hier vindt tevens de doorontwikkeling van de standaard plaats. Hiertoe is een werkgroep Edukoppeling[3] ingericht.

2.3. Organisatorisch werkingsgebied van Edukoppeling

Edukoppeling schrijft voor hoe onderwijsorganisaties, publieke uitvoeringsorganisaties, dienstverleners[4] en andere ketenpartners gegevensuitwisselingen opzetten. Edukoppeling heeft als scope alle werkingsgebieden vallend onder alle onderwijssectoren. Zie de werkingsgebieden in ROSA[5]. Hieronder vallen dus alle door de overheid erkende onderwijsorganisaties binnen de sectoren po, vo, bve en ho. Daar waar behoefte is aan technische afspraken op het M2M-koppelvlak, volgens het functioneel toepassingsgebied van Edukoppeling, zijn ketensamenwerkingen binnen deze sectoren verantwoordelijk voor de toepassing ervan. De toepassing buiten het organisatorisch werkingsgebied is toegestaan.

2.4. Functioneel toepassingsgebied

Het functioneel toepassingsgebied van Edukoppeling is de geautomatiseerde uitwisseling van gesloten data[6] tussen informatiesystemen van onderwijsorganisaties en ketenpartners

---

[2] https://www.edustandaard.nl/standaarden/afspraken/afspraak/edukoppeling/. Reageren kan via info@edustandaard.nl.

[3] Voor meer info over de Edukoppeling werkgroep, zie https://www.edustandaard.nl/standaard_werkgroepen/werkgroep-edukoppeling/

[4] Dienstverleners (ROSA: *Een organisatie die een dienst levert aan een organisatie of een natuurlijke persoon).*

[5] https://rosa.wikixl.nl/index.php/Werkingsgebieden

[6] Het gaat om gegevens waarvoor je geautoriseerd moet worden voor toegang. De gegevens zijn niet voor publiek hergebruik beschikbaar. Dit kan om verschillende redenen zijn, zie https://data.overheid.nl/gesloten-datasets

(onderling, met bedrijven of met de overheid). Deze uitwisseling betreft M2M point-to-point verbinding voor uitwisseling tussen een confidential client en een gesloten API.

Edukoppeling gaat over de afhandeling van berichten (het transport) en niet over de inhoud van berichten. Het is een functioneel technische standaard, maar zal ook aansluiting moeten vinden op kaders van andere architectuurlagen. Hoe Edukoppeling aansluit op bredere afspraken is aan ketensamenwerkingen[7] waarin de standaard als onderdeel van de afspraak of het afsprakenstelsel wordt gevat.

Edukoppeling regelt de volgende ketenfuncties: identificatie, authenticatie, autorisatie en routering, op de 'uitwisselingslaag'. Dit om de zorgen dat vertrouwelijke gegevens tijdens transport van de ene naar de andere organisatie, niet ongeoorloofd worden ingezien of gemanipuleerd.

2.5. Positionering van Edukoppeling in het Edustandaard vijflagen model

Het Edustandaard 5-lagenmodel[8] onderkent de volgende lagen:

1. Grondslagenlaag: borgt de juridische basis en beleidskaders waarbinnen gegevensuitwisseling is toegestaan;
2. Organisatorische laag: ketensamenwerking afspraken over wie welke rol heeft, welke gegevensdiensten, interfaces en interactiepatronen er zijn en welke gegevens onder welke condities uitgewisseld worden;
3. Informatielaag: semantiek, waaronder gegevensdefinities, informatiemodellen en de gebruikte identifiers voor rechtspersonen en natuurlijke personen;
4. Applicatielaag: API's en hun beveiligingsprofielen, berichtspecificaties, payload beveiliging, interactiepatronen en foutafhandeling;
5. IT-infrastructuurlaag: transportprotocollen en technische beveiligingsmechanismen zoals TLS.

Dit Edukoppeling-profiel heeft binnen het Edustandaard 5 lagen model met name betrekking op de applicatielaag (laag 4), levert een belangrijke ondersteuning aan de organisatorische laag (laag 2) en heeft een relatie met de IT-infrastructuurlaag (laag 5).

2.6. Notatiewijze voorschriften

Voor elk voorschrift wordt aangegeven in welke mate hier invulling aan moet worden gegeven. Hiermee kunnen we duidelijk aangeven wat de grenzen van dit profiel zijn ten opzichte van de mogelijke externe bron(nen) waar het voorschrift eventueel van wordt overgenomen. We gebruiken hiervoor de notatiewijze van RFC2119[9]. Deze gebruikt de volgende termen: "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL".

[7] Ketensamenwerkingen zijn bijvoorbeeld OKE, Edu-V, ROD ec. (zie ook: https://rosa-begrippenkader.wikixl.nl/index.php/Begrip:27a6accf-472d-4415-bc5b-1e9de17bf288#tab=Betekenis)
[8] AMIGO-methodiek-1.1.0-1.pdf en https://rosa.wikixl.nl/index.php/Interoperabiliteit_en_het_Edustandaard_lagenmodel#Opbouw_van_het_lagenmodel
[9] https://tools.ietf.org/html/rfc2119

2.7. Leeswijzer

Hoofdstuk 3 bevat een toelichting op de Edukoppeling voorschriften. In hoofdstuk 4 zijn de Edukoppeling voorschriften opgenomen. Dit zijn aanscherpingen op de onderliggende internationale open industrie standaarden (zoals CloudEvents)[10]. In hoofdstuk 5 worden scenario's beschreven die een aantal voorschriften in een bepaalde context plaatsen om ketensamenwerkingen te ondersteunen in de te maken keuzes.

2.8. Relevante ontwikkelingen

In deze paragraaf worden relevante ontwikkelingen binnen de overheid en IT beschreven die inspiratie bieden voor ontwikkeling binnen het onderwijs.

- Binnen het Groeifondsprogramma Npuls, en Edu-V worden nieuwe koppelvlakken ontwikkeld en/of koppelvlakken gemoderniseerd in lijn met Event Driven Architecturen (EDA).
- Er is een NL GOV profile for CloudEvents ontwikkeld. Er is recent een definitieve versie 1.1[11] gepubliceerd. Deze beschrijft een patroon voor gegevensuitwisseling binnen Event Driven Architecturen. Dit verwijst o.a. naar de internationale standaard CloudEvents v1.0.1 [12], een JSON Event Format v1.0.1[13] voor CloudEvents en Web Hooks v1.0.1[14] voor Event Delivery.
- AsyncAPI specificatie v3.1.0 initiatief voor het beschrijven van event-driven APIs [15]
- Een voorgestelde IETF-standaard, om idempotency [16] te implementeren voor HTTP-methoden die niet als veilig worden gezien, die toegevoegde waarde heeft voor robuuste end-to-end uitwisseling van gegevens.

2.9. Uitgangspunten voor het Communicatie-profiel

1. Dit Edukoppeling profiel beschrijft de voorschriften voor berichtuitwisseling tussen een verzender en ontvanger. Dit profiel plaatst dit in de context van een IT-landschap, maar legt niet op hoe het IT-landschap binnen de grenzen van een ketenpartner eruitziet.

2. Dit Edukoppeling-profiel is alleen van toepassing in de context van confidential clients.

3. Dit Edukoppeling-profiel volgt het OAuth client credentials profiel voor RESTful API's voor machine-to-machine (M2M) gegevensuitwisseling binnen het onderwijs.

4. Dit Edukoppeling-profiel schrijft het gebruik van Transport Layer Security (TLS) voor conform UBV TLS.

[10] https://cloudevents.io/
[11] NL GOV profile for CloudEvents 1.1
[12] https://github.com/cloudevents/spec/tree/v1.0.1
[13] https://github.com/cloudevents/spec/blob/v1.0.1/json-format.md
[14] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md
[15] https://www.asyncapi.com/docs/reference/specification/v3.1.0
[16] https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/

5. Dit Edukoppeling-profiel sluit aan op de transactiepatronen zoals gedefinieerd in de Edukoppeling architectuur.

6. Dit Edukoppeling-profiel hanteert de volgende aanvullende uitgangspunten voor asynchrone interacties:

    a. Het is gebaseerd op de internationale open standaarden rond CloudEvents. Het fundament is de CloudEvents specificatie, opgesteld door Cloud Native Computing Foundation (CNCF).
    b. Dit Edukoppeling-profiel is een aanvulling en volgt de richtlijnen zoals beschreven in het NL GOV profile for CloudEvents [17]:
        i. Context attributen (H3);
        ii. Event Data (H4);
        iii. Size Limits (H5);
    c. Gebruik van JSON, HTTP en webhook (bijlage A) zoals ook beschreven in de Guidelines for NL-GOV profile CloudEvents [18].

    Met uitzondering van:
    d. CloudEvent Security Options (H6);
    e. Abuse protection as described in the guidelines [19].

    In paragraaf 3.2.4 Beveiliging worden de beveiligingsopties binnen dit profiel beschreven.

[17] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/
[18] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/
[19] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/

3. Toelichting voorschriften

3.1. Inleiding

Dit hoofdstuk beschrijft de voorschriften die de basis vormt voor communicatie binnen het Edukoppeling-profiel. Het profiel biedt op een aantal punten keuzemogelijkheden. Hiermee beogen we een profiel dat een verplichte basis set van voorschriften bevat, maar ook genoeg ruimte biedt voor passende configuraties om voor ketenpartners binnen een bepaalde ketensamenwerking niet onoverkomelijke drempels op te werpen. Het is aan een ketensamenwerking om te bepalen welke opties van toepassing zijn.

Dit profiel moet worden toegepast wanneer er binnen een ketensamenwerking gegevensuitwisseling plaatsvindt tussen confidential clients en RESTful API's.

Het onderwijsveld bestaat uit verschillende ketenpartners die ieder verantwoordelijk zijn voor een deel van de keten en zijn systemen. Dit profiel moet daarom in de context van deze ketensamenwerking en ketenprocessen worden geplaatst.

Dit hoofdstuk is opgedeeld als volgt:

- De inleiding beschrijft plaatst het profiel verder in context van een IT-landschap en in relatie tot de Edukoppeling architectuur.
- Toelichting op voorschriften relevant voor alle communicatie, zowel synchroon en asynchroon.
- Toelichting op aanvullende voorschriften relevant voor asynchrone communicatie. Er zijn momenteel specifieke voorschriften voor synchrone communicatie.

3.1.1. Context en relatie tot Edukoppeling architectuur

De Edukoppeling architectuur beschrijft een ketensamenwerking tussen ketenpartners bestaande uit één of meer technische interacties. De architectuur beschrijft een client-server model als basis en onderkent een intermediair *per ketenpartner* (en meerdere servers) in het achterlandschap. Een intermediair per ketenpartner is een bewuste keuze: er is momenteel binnen de (semi-)overheid niet één centrale partij toe te wijzen die logischerwijs dit beheer op zich kan nemen én een gecentraliseerd hub-en-bespoke model waarin vergaande encryptie/ autorisatie maatregelen vereist zijn heeft niet de voorkeur.

In de context van ketenpartners en vaststellen van afspraken is met name belangrijk hoe communicatie aan de buitenrand van systemen van IT-landschappen met elkaar is. Dit profiel beschrijft de berichtuitwisseling tussen Intermediary A en Intermediary B en plaatst dit in context van een client en server. Echter, de ketenpartner kan zelf vormgeven aan de implementatie qua scheiding tussen de betreffende componenten, uiteraard zonder de richtlijnen in dit document uit het oog te verliezen.

Het patroon betekent dat een ontvangende partij een API-endpoint definieert waar berichten naar verstuurd kunnen worden. Het uitgangspunt voor dit API-endpoint is technologie agnostisch; losstaand van ketenpartner specifieke technologie keuzes voor implementatie van een client/ server. Dit patroon aan ontvanger zijde lijkt op een traditionele API-gateway patroon waarbij de gateway als enkelvoudige ingang dient voor meerdere servers daarachter.

Binnen Edukoppeling architectuur onderkennen we transactiepatronen voor technische interacties tussen ketenpartners binnen een ketensamenwerking en kenmerken van deze interacties: richting, overdrachtswijze push / pull, asynchroon / synchroon en hechte / losse koppeling. Bij al deze transactiepatronen vindt de communicatie plaats via synchrone berichten via RESTful API's. Het type verwerking (asynchroon of synchroon) is afhankelijk van het toegepaste transactiepatroon.

Binnen Edukoppeling is losse koppeling een belangrijk uitgangspunt, maar dit betekent niet dat alle interacties volledig ontkoppeld moeten zijn. Er kunnen situaties zijn waar synchrone uitwisseling wenselijk of zelfs noodzakelijk is. Het is context- en procesafhankelijk welke vorm van uitwisseling nodig is. Het Communicatie-profiel stelt daarom dat de keuze in eerste instantie context- en procesafhankelijk gemaakt dient te worden, maar stelt wel dat indien bij een context/proces óók asynchrone uitwisseling mogelijk is dat bij voorkeur een asynchrone uitwisseling wordt gerealiseerd. Het Communicatie-profiel stuurt daarmee wel op asynchrone uitwisseling (binnen karakteristieken van context en proces), maar sluit het gebruik van synchrone uitwisseling nadrukkelijk niet uit. Ketenpartners kunnen bewust kiezen om een synchrone/ hechte koppeling te accepteren.

Dit document focust in vervolg op hoe interacties vormgegeven kunnen worden. Dit kan dan worden toegepast binnen verschillende transactiepatronen.

3.2. Toelichting alle communicatie

3.2.1. Registratie

Registratie is een proces dat relevant is ongeacht welk type interactie. In dit proces zullen twee ketenpartners elkaar initieel identificeren, authenticeren en configureren zodat gegevensuitwisseling tussen beide mogelijk wordt.

De registratie omvat minimaal de volgende stappen:

- Ketenpartners spreken het voornemen uit naar elkaar om berichtuitwisseling op te zetten.
- De verzender registreert zich bij de ontvanger.
- De ontvanger accepteert de registratie en authenticeert de verzender voor het versturen van berichten aan de ontvanger. De ontvanger die de ontvangende API aanbiedt stelt is ervoor verantwoordelijk dat een portaal en/of proces beschikbaar gesteld voor registratie van de ketenpartners.
- De ontvanger registreert zijn API-endpoint(s) bij de verzender via een portaal en/of proces dat beschikbaar gesteld wordt door de verzender.

3.2.2. Technische contracten

De Edukoppeling architectuur beschrijft het bouwblok 'Contractregister'; de bron van waarheid voor de betreffende specificatie.

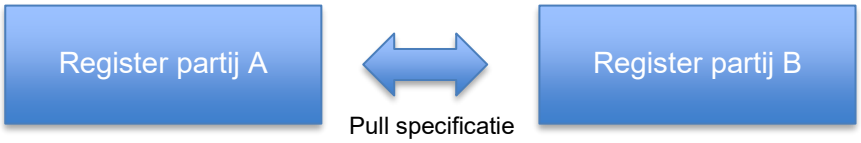
Ketenpartner bepalen altijd gezamenlijk wie welke contracten opstelt. Desondanks, stellen we belangrijk richtlijnen op die kunnen worden gehanteerd. We hanteren de volgende richtlijnen:

- De OpenAPI-specificatie (OAS) van de API wordt gedefinieerd door de ontvangende partij.

- De AsyncAPI-specificatie (AAS), i.e. de business data, wordt gedefinieerd door de producer, omdat deze eigenaar is van data en het gedrag. Hierbij stemt de producer de specificatie af op de eisen van de mogelijke consumers. Datacontracten hanteren idealiter waar mogelijk sectorstandaarden voor gegevensstructuren.

Zoals elke richtlijn kan daar met goede argumentatie van afgeweken worden, indien noodzakelijk. Zo kan het wenselijk zijn dat een partij die een centrale voorziening levert afspraken maakt over welke gegevens ontvangen kunnen worden. Daarom bepalen ketenpartners altijd gezamenlijk definitief *wie* de specificatie opstelt en onderhoudt.

Het is mogelijk om vanuit beveiligingsoogpunt eerst de specificatie over te zetten naar het 'eigen' intern register. Dit is weergegeven hieronder:

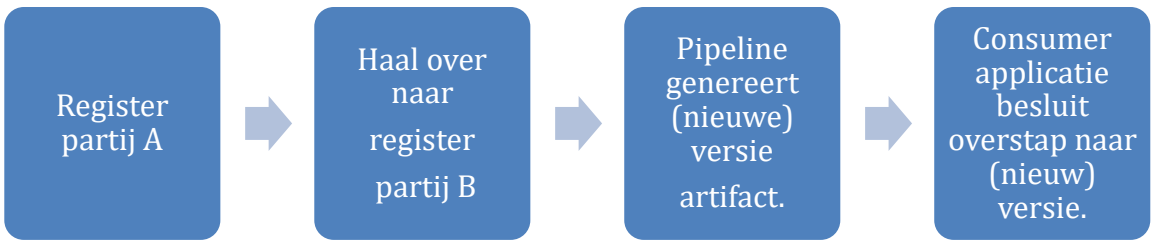


Figuur 1: Interactie tussen registers partijen.

Andere partijen kunnen deze specificatie dan handmatig en/of geautomatiseerd ophalen om de andere kant van de integratie op te zetten. Automatisering kan helpen om de berichtspecificatie op te halen uit de bron en daaruit direct, of via een intern register dat een kopie opslaat, automatisch code te generen.

Bijvoorbeeld als volgt:

- Register partij A bevat het AsyncAPI contract.
- Partij B haalt het contract over register A naar een eigen immutable register B.
- Een pipeline draagt zorg voor het ophalen van dit contract uit het interne register, genereren, compileren, packaging en publicatie van het artifact naar een interne repository.
- Een automatische dependency updater kan detecteren dat er een nieuwe versie beschikbaar is, de versie ophogen en een ontwikkelaar laten besluiten of dit direct doorgevoerd kan worden. Belangrijk: Een consumer applicatie beslist zelf wanneer een upgrade plaatsvindt.



Figuur 2: Voorbeeld proces over rol register en automatisering.

3.2.3. Aflevering semantiek

In een gedistribueerd landschap heb je traditioneel te maken met de uitdaging van *atomiciteit*. In de ketencontext heb je het over de communicatie tussen twee intermediaries

waarbij één intermediary als verzender optreedt en één intermediary als ontvanger optreedt. Echter, robuuste end-to-end verwerking vereist garanties van elke hop in de uitwisseling, dus van client naar Intermediary, van Intermediary naar Intermediary, en van Intermediary naar server.

3.2.3.1. Aflevergaranties

Als het gaat om aflevergaranties richting een Intermediary (van client naar Intermediary, of van Intermediary naar Intermediary) zijn er drie opties:

- At most once: een bericht kan worden afgeleverd, maar nooit meer dan één keer. Dit kan leiden tot verlies van berichten en wordt daarom zelden, zo niet nooit, gebruikt.
- At least once: een bericht wordt afgeleverd, maar kan meer dan één keer worden afgeleverd. Dit kan leiden tot dubbele berichten.
- Exactly once: een bericht wordt precies één keer afgeleverd.

De reikwijdte van deze garanties ligt echter alleen binnen een Intermediary en niet daarbuiten, zoals in een gedistribueerd IT-landschap. In een volledig robuust IT-landschap moeten we rekening houden met end-to-end robuuste aflevering en uitgaan van het worstcasescenario.

Een worstcasescenario is wanneer een verzender crasht/ herstart/ stopt tijdens de verwerking van een bericht. Met name in een schaalbare infrastructuur is dit een realistisch scenario. Afhankelijk van het exacte moment van de crash kunnen zich verschillende situaties voordoen:

- Als de verzender het bericht nog niet heeft geproduceerd: de verzender zal het bericht opnieuw proberen te leveren. Dit vormt geen probleem.
- Als de verzender het bericht heeft afgeleverd en geen andere status hoeft bij te houden (meestal wanneer de Intermediair de enige bron van waarheid is): dit vormt geen probleem.
- Als de verzender het bericht heeft geproduceerd, maar het bericht nog niet als afgeleverd heeft gemarkeerd: de verzender zal het bericht opnieuw proberen te produceren, wat leidt tot dubbele berichten. Zonder mitigerende maatregelen aan de ontvanger-zijde kan dit tot problemen leiden.

3.2.3.2. Ontvanger garanties

Wat betreft ontvanger-garanties zijn er drie opties:

- At most once: een bericht kan worden geconsumeerd, maar nooit meer dan één keer. Dit kan leiden tot verlies van berichten en wordt daarom zelden, zo niet nooit, gebruikt. Dit treedt typisch op wanneer een bericht als verwerkt wordt gemarkeerd vóór de verwerking, maar de verwerking daarna faalt.
- At least once: een bericht wordt geconsumeerd, maar kan meer dan één keer worden geconsumeerd. Dit treedt typisch op wanneer een bericht pas ná de verwerking als verwerkt wordt gemarkeerd, maar het markeren faalt en het bericht opnieuw wordt geconsumeerd. Zonder aanvullende maatregelen kan dit leiden tot dubbele verwerking.
- Exactly once: een bericht wordt exact één keer geconsumeerd én verwerkt (oftewel in één transactie de verwerking plaatsvinden en markeren dat het bericht is geconsumeerd). In onze gedistribueerde opzet is dit per definitie niet mogelijk, vanwege de uitdagingen rond atomiciteit.

3.2.3.3. End-to-end exactly-once verwerking

In deze sectie bekijken we hoe echte end-to-end exactly-once verwerking kan worden gerealiseerd binnen een Event-Driven Architecture (EDA). Daarbij zorgen we ervoor dat:

A. Een **client of intermediary** berichten **at least once** produceert. In normale omstandigheden produceert een client altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.
B. Een **server** berichten **at least once** consumeert. In normale omstandigheden consumeert een consumer altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.
C. De verwerking **exactly once** wordt uitgevoerd in de ontvangende service.

Dit zorgt voor een minimale extra impact op de performance aan de verwerkingskant en waarborgt uiteindelijke atomiciteit zonder dataverlies of duplicatie.

3.2.3.4. Toelichting voorschriften

Veel HTTP-methodes zijn in lijn met de NLGov API Design Rules [20] al van nature idempotent. Zo zullen voor veel synchrone bevragingen geen aanvullende maatregelen vereisen.

Voor uitwisselingen, synchroon en/of asynchroon, waar exactly-once verwerking niet gegarandeerd kan worden moet voor het kunnen bereiken van robuuste uitwisselingen de ketenpartners maatregelen treffen. Voor uitwisselingen waar robuustheid geen vereiste is en meerdere meldingen accepteerbaar zijn kan gekozen worden om daar geen maatregelen voor te treffen. De ketenpartners leggen dit bewust in de onderbouwing dan vast.

Indien de ketenpartners wel maatregelen moeten treffen, dan zouden de ketenpartner daar idempotentie-sleutels voor kunnen gebruiken om idempotentie te bereiken. Een ketenpartner mag ervoor kiezen om andere mitigatie of contingency maatregelen te nemen (bijv. door het introduceren van aanvullende business logica).

In een ketensamenwerking dragen de verzender én ontvanger beide bij aan de robuuste uitwisseling van berichten. Een best-practice voor robuuste uitwisseling is de ondersteuning van at-least once berichten produceren en exactly-once verwerking via idempotentie. Dit is algemene richtlijn die ketenpartners zouden moeten hanteren voor robuuste uitwisseling. In bijlage C worden richtlijnen voor het implementeren van idempotentie beschreven.

3.2.4. Beveiliging

De beveiliging vindt plaats op applicatielaag en IT-infrastructuurlaag (verplicht). Hiervoor wordt gebruikt gemaakt van het Edukoppeling OAuth client credentials profiel voor RESTful API's.

Dit Edukoppeling-profiel definieert geen aanvullende beveiligingsmechanismen. Zo geldt:

- De payload wordt niet aanvullend beveiligd. Dat is niet nodig omdat het uitgangspunt machine-to-machine (M2M) is. Vooralsnog wordt de payload alleen beveiligd in transport (TLS).

[20] https://gitdocumentatie.logius.nl/publicatie/api/adr/2.1.0/#/core/http-safety

- Abuse protection zoals beschreven in de webhook standaard[21] is niet nodig, omdat OAuth2 al de toegang regelt tot het webhook-endpoint. Indien de consumer toegang verleent aan de producer via OAuth2, wordt daarmee ook de producer goedgekeurd om berichten naar de consumer te sturen.

In de berichtuitwisseling maken we onderscheid tussen contextuele-attributen en domein specifieke data (data-attribuut). De CloudEvents specificatie maakt al expliciet onderscheid tussen deze soort attributen. Hiervoor geldt:

- Context-attributen: Attributen die zijn gemarkeerd als 'contextueel' zouden geen sensitieve informatie moeten bevatten aangezien producers, consumers en intermediairs deze attributen mogen loggen.
- Domein specifieke data (data-attribuut): Domein specifieke event data wordt niet (aanvullend) versleuteld.

Aanvullende beveiliging afspraken worden, waar noodzakelijk, afgestemd tussen de ketenpartners. Zo geldt:

a. Versleuteling is alleen noodzakelijk in scenario's waar (niet vertrouwde) tussenliggende componenten voor datalekken kunnen zorgen. Versleutelen is rekenintensief en het maakt het bovendien moeilijker voor beveiligingsmechanismen, zoals API-gateways, de payload te valideren en transformeren (indien nodig).
b. Ondertekening van gegevens is alleen nodig indien er een risico is dat de integriteit van de gegevens aangetast kan worden of als onweerlegbare overdracht vereist wordt.

3.3. Toelichting asynchrone communicatie

De basis voor asynchrone communicatie vormt het NL GOV for CloudEvents versie 1.1. Bij asynchrone communicatie stuurt een computersysteem een bericht of event en ontvangt hoogstens een *bevestiging* van ontvangst. De *verwerking* vindt losgekoppeld en later plaats en levert geen direct antwoord op.

3.3.1. Abonneren

Abonneren vindt plaats na registratie. Dit is het proces waarbij een ketenpartner aanmeldt voor specifieke gegevens of gebeurtenissen, zodat deze automatisch worden ontvangen wanneer relevante wijzigingen optreden. Alhoewel dit via synchrone interacties kan plaatsvinden, past het beter binnen de beschreven asynchrone interacties en bijbehorende patronen. De rest van deze paragraaf focust zich daarom op asynchrone interacties.

Na registratie abonneert de consumer zich bij de producer op de ontvangen berichten. De ketensamenwerking bepaalt gezamenlijk hoe het proces van abonneren plaatsvindt. In het algemeen geldt het volgende. Het is de verantwoordelijkheid van de consumer om zich te abonneren bij de producer voor de berichten die gegevensleveringen die noodzakelijk zijn. Het proces van abonneren hoeft niet altijd expliciet te gebeuren door de consumer, maar kan ook impliciet plaatsvinden (bijv. een abonnement volgt automatisch op een eerdere uitwisseling). De consumer levert een webhook endpoint aan voor berichtaflevering.

[21] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md

Bij abonneren is het van belang dat de producer alleen gegevensleveringen naar consumers mogelijk maakt die toegang mogen hebben tot de data die wordt uitgewisseld. Dit stelt de producer zeker via autorisaties. De producer mag het mogelijk maken via portaal/proces om een afleverlocatie te configureren/ aan te leveren.

Het niveau waarop abonnementen (en daarmee de scope van een abonnement) worden vastgelegd hangt nauw samen met verschillende aspecten waaronder de data die wordt uitgewisseld, het beoogde type berichten die moeten worden uitgewisseld en het niveau van autorisatie op de data die gewenst is. Ter illustratie zou je kunnen abonneren op ‘events’ van individuele personen of alle personen; dit maakt het verschil tussen respectievelijk een miljoen+ abonnement of één abonnement. Dit is erg use-case afhankelijk en we benoemen daarom hier het belang om afstemming hierover te laten plaatsvinden tussen ketenpartijen. Het is buiten de scope van dit document om dit op een use-case basis uit te werken.

3.3.2. Verzender (producer kant)

Een producer is een systeem dat voor één of meer gegevensuitwisselingen (in verschillende ketensamenwerkingen) CloudEvents produceert en aflevert (via zijn eigen gekoppelde Intermediary) bij geregistreerde Intermediaries. Voor simplificatie beschouwen we beide componenten hier als de verzender waarin de ketenpartner zelf vorm kan geven aan de implementatie qua scheiding producer/ intermediary, uiteraard zonder de richtlijnen uit deze paragraaf uit het oog te verliezen.

De verzender houdt bij het opstellen/versturen van requests rekening met aan welke geregistreerde partijen berichten dienen te worden aangeleverd (conform abonnementen) en stuurt de berichten naar het registreerde afleverpunt (i.e. een webhook) van de ontvanger. Hierbij houdt de verzender rekening met eventuele verschillende versies die een bericht kan hebben.

Indien de verzender eigenaar is van de berichtspecificatie, registreert de verzender de technische contracten bij het contract register (zie 3.2.2 Technische contracten).

3.3.3. Ontvanger (consumer kant)

Binnen de context van CloudEvents kent de ontvanger een Intermediary en Consumer.

A. De intermediary uit zich naar de buitenwereld als een beveiligd endpoint. Het endpoint verwacht een CloudEvents payload. Dit betreft daardoor twee varianten:
    a. Endpoint voor individuele CloudEvents berichten (verplicht).
    b. Endpoint voor CloudEvents in batch (optioneel).
B. De consumer is een systeem dat events ontvangt en een actie onderneemt op basis daarvan.

Voor simplificatie beschouwen we beide componenten hier als de ontvanger waarin de ketenpartner zelf vorm kan geven aan de implementatie qua scheiding consumer/ intermediary, uiteraard zonder de richtlijnen uit deze paragraaf uit het oog te verliezen.

De ontvanger heeft een aantal taken:

- Voert authenticatie en autorisatie uit;
- Accepteert het verzoek;
- Valideert het verzoek (minimaal de CloudEvents structuur);

- Persisteert het verzoek;
- Geeft een technische bevestiging dat het bericht is ontvangen;
- Zorgt voor verwerking door de consumer(s). Functionele verwerking kan asynchroon plaatsvinden door de consumer.

De ontvanger registreert het technische contract van de webhook bij het contract register (zie 3.2.2 Technische contracten).

4. Voorschriften Communicatie-profiel

Dit hoofdstuk beschrijft de voorschriften voor het Edukoppeling Communicatie-profiel voor synchrone en asynchrone communicatie via RESTful API's. Het geheel aan voorschriften biedt een Edukoppeling Communicatie-profiel die kan worden toegepast wanneer een confidential client via een M2M point-to-point verbinding interacteert met een gesloten API.

De basis voor asynchrone communicatie vormt het NL GOV for CloudEvents versie 1.1. De voorschriften in dit hoofdstuk zijn daarmee aanscherpingen op het NL GOV profile for CloudEvents. Bij asynchrone communicatie stuurt een computersysteem een bericht of event en ontvangt hoogstens een *bevestiging* van ontvangst. De *verwerking* vindt losgekoppeld en later plaats en levert geen direct antwoord op.

Het Edukoppeling Communicatie-profiel is bewust in context van een client en server geplaatst, maar focust zich op de uitwisseling tussen twee intermediairs van elke ketenpartner. Het dicteert niet het achterliggende landschap van een ketenpartner (bijv. van producer naar intermediary), maar focust op de interactie tussen de ketenpartners en hun intermediairs. Het is aan elke ketenpartner om die scheiding zelf goed te leggen. Wel geven de voorschriften aan waar een ketenpartner aan moet voldoen; in welke component dat precies wordt geïmplementeerd is aan de ketenpartner. Voor simplificatie, binnen de voorschriften wordt geschreven over 'verzender' en 'ontvanger': respectievelijk de combinatie van een intermediair, en client of server.

De voorschriften zijn verdeeld in algemene voorschriften, voorschriften die relevant zijn voor alle interacties (synchroon en asynchroon) en aanvullend voorschriften die specifiek zijn voor asynchrone interacties. Er zijn momenteel aanvullende voorschriften specifiek voor synchrone communicatie.

4.1. Algemene voorschriften

1. SHOULD: Dit profiel zou moeten worden toegepast wanneer er binnen een ketensamenwerking gegevensuitwisseling plaatsvindt tussen confidential clients en RESTful API's.

2. MUST: Dit Edukoppeling profiel is alleen van toepassing in de context van confidential clients. De client moet de vertrouwelijke gegevens voor authenticatie (client secret) veilig kunnen opslaan en hier veilig over kunnen beschikken.

3. SHOULD: Het Edukoppeling profiel maakt gebruik van RESTful API's.

4. MAY: Een verzender mag circuit breaking toepassen bij ongeplande niet-beschikbaarheid.

5. SHOULD: Voor het ontwerpen van RESTful API's worden de NLGov REST API Design Rules v2.2.1. gevolgd [22].

[22] https://gitdocumentatie.logius.nl/publicatie/api/adr/2.2.1/

6. MAY: Binnen de NLGov REST API Design Rules worden extensies aangeboden, waaronder een rate-limiting module. Deze wordt nog uitgewerkt en beschrijft daarom hier nu nog de richtlijnen. Een ontvanger mag rate-limiting toepassen om zijn systemen weerbaarder te maken [23]. Indien rate-limiting wordt toegepast:
    a. SHOULD: Ontvanger biedt rate-limiting informatie aan via de HTTP-header X-Rate-Limit. [24]
    b. SHOULD: Ontvanger informeert de verzender in geval van te veel verzoeken via een HTTP-status code 429 Too Many Requests [25].
    c. SHOULD: Ontvanger past rate-limiting toe per producer.
    d. SHOULD: Het wordt in het algemeen afgeraden om rate-limiting toe te passen op IP-adres. Verschillende producers kunnen immers vanaf hetzelfde IP-adres geserviced worden.
    e. MUST: De verzender moet rekening houden met rate-limiting van de ontvanger en throttling toepassen waar nodig.

7. SHOULD: De voorschriften voor uitwisseling worden toegepast binnen één van de transactiepatronen zoals gedefinieerd in de Edukoppeling architectuur.

8. SHOULD: Indien context/proces dit toelaat, stuur aan op asynchrone uitwisseling ten behoeve van ontkoppeling van systemen m.u.v. ‘Bevragingen’.

9. MAY: Synchrone uitwisseling mag worden toegepast indien wenselijk en acceptabel voor de ketenpartners.

    a. SHOULD: Indien synchrone uitwisseling wordt gekozen, onderbouw dit dan.

4.2. Alle interacties

4.2.1. Registratie

In deze paragraaf wordt aangenomen dat het ketenpartners het eens zijn over uitwisseling van gegevens; een bilaterale afspraak.

10. MUST: De verzender en ontvanger stellen, waar mogelijk, tijdens één interactie de benodigde gegevens van zowel de producer als de consumer vast. De gegevens registreren de ketenpartners via een portaal of het binnen de ketensamenwerking afgesproken proces.

11. MUST: De ontvanger stelt een API-endpoint beschikbaar aan de verzender.

    a. MAY: De ontvanger mag dit via een URL beschikbaar stellen aan de verzender, zodat verzender dit handmatig zelf kan verwerken in zijn administratie/systeem.

[23] https://docs.geostandaarden.nl/api/API-Strategie-ext/#api-44
[24] https://docs.geostandaarden.nl/api/API-Strategie-ext/#api-45
[25] https://httpwg.org/specs/rfc6585.html

b. MAY: Een verzender mag het mogelijk maken voor de ontvanger via portaal/proces om een afleverlocatie te configureren/ aan te leveren. Aandachtspunt voor de verzender is een juiste inrichting van Identity en access management (IAM) om gebruikers van andere organisaties (en hun rol) vast te leggen.

12. MUST: De ontvanger accepteert de registratie en authenticeert de verzender voor de betreffende API(s).

4.2.2. Technisch contract register

13. MUST: Ketenpartners bepalen gezamenlijk definitief wie de specificatie opstelt en onderhoudt.

14. SHOULD: Datacontracten hanteren idealiter waar mogelijk sectorstandaarden voor gegevensstructuren.

15. MUST: De ketenpartner die een specificatie opstelt en onderhoudt stelt een berichtenspecificatie beschikbaar. We onderkennen de volgende contractspecificaties:
    - De OpenAPI-specificatie v3.2.0[26] (OAS) van de webhook, oftewel van contextuele data, conform de CloudEvents specificatie.
    - De AsyncAPI-specificatie v3.1.0[27] (AAS) van de business data.

16. SHOULD: De technische contracten worden geregistreerd in een contract-register door de ketenpartner die de specificatie opstelt en onderhoudt.

    a. MUST: Een ketenpartner stelt het contract-register voor ketenpartners beschikbaar met een OpenAPI-specificatie (OAS).
    b. MUST: Een contract-register API maakt het mogelijk om contracten uit de lijsten, versie van contracten uit te lijsten én een individuele versie van een contract op te halen. Het contract bestaat minimaal uit de API-specificatie.
    c. SHOULD: Een contract-register API geeft de mogelijkheid om aanvullende metadata rondom status, eigenaarschap en contactpunt voor een contract op te halen.
    d. MAY: Het is mogelijk om vanuit beveiligingsoogpunt de specificatie over te zetten van het ‘eigen’ intern register naar het contract-register dat extern beschikbaar gesteld worden.
    e. MAY: Ketenpartners mogen specificaties uit het contract-register ophalen als onderdeel van handmatige en geautomatiseerde processen.

17. SHOULD: Ketenpartners in een ketensamenwerking bepalen gezamenlijk in welk centraal contract-register de specificaties worden vastgelegd.

[26] https://spec.openapis.org/oas/v3.2.0.html
[27] https://www.asyncapi.com/docs/reference/specification/v3.1.0

18. MAY: Bij gebrek aan een geschikt centraal contract-register of afspraken binnen een ketensamenwerking, mag een ketenpartner voor kiezen om specificaties in een eigen contract-register vast te leggen.

19. SHOULD: De OAS van het API-endpoint wordt gedefinieerd door de ontvangende partij.

    SHOULD: De AAS wordt gedefinieerd door de verzender, omdat deze eigenaar is van data en het gedrag. Hierbij stemt de verzender de specificatie af op de eisen van de mogelijke ontvanger(s).

4.2.3. Aflevering semantiek

20. MUST: In een ketensamenwerking dragen de verzender én ontvanger beide bij aan de robuuste uitwisseling van events. Ketenpartner moeten gezamenlijk een bewuste afweging maken of robuuste uitwisseling met exactly-once verwerking een vereiste is.

    a. SHOULD: Voor robuuste uitwisseling zou exactly-once verwerking gegarandeerd moeten worden.

    b. MAY: Voor uitwisselingen waar robuustheid geen vereiste is en meerdere meldingen accepteerbaar zijn kan gekozen worden om daar geen maatregelen voor te treffen. De ketenpartners leggen dit bewust in de onderbouwing dan vast.

21. SHOULD: Indien exactly-once verwerking gegarandeerd moet worden, dan worden voor operaties waar dit nog niet gegarandeerd voor is, de handreiking voor idempotency gevolgd zoals beschreven in bijlage C.

    a. MUST: De verzender produceert berichten at-least once, bijv. door gebruik van het transactional outbox patroon.
    b. MUST: De ontvanger verwerkt berichten exactly-once via idempotentie retries, DLQ en/of replays in de intermediary en/of consumer. In bijlage C wordt een handreiking voor het implementeren van idempotentie beschreven.
    c. SHOULD: Het is aan te raden bovenstaande maatregelen te treffen zo dicht bij het doelsysteem waar de verwerking plaatsvindt (i.e. consumer).
    d. MAY: Om moderne flows i.c.m. legacy applicaties voldoende ruimte te geven mag dit ook anders geïmplementeerd worden, bijv. in de Intermediary.

22. MAY: Een ketenpartner mag ervoor kiezen om andere mitigatie of contingency maatregelen te nemen om exactly-once verwerking te behalen (bijv. door het introduceren van aanvullende business logica).

4.2.4. Beveiliging

23. MUST: Beveiliging tussen intermediairs op basis van het Edukoppeling OAuth-profiel[28]. Dit Edukoppeling-profiel volgt het OAuth client credentials profiel voor RESTful API's voor machine-to-machine (M2M) gegevensuitwisseling binnen het onderwijs en schrijft daarmee ook het gebruik van Transport Layer Security (TLS) voor conform UBV TLS.

24. SHOULD: Sensitieve informatie zou niet opgenomen moeten worden in context-attributen aangezien producers, consumers en intermediairs deze attributen mogen loggen.

25. MAY: Aanvullende beveiliging afspraken worden, waar noodzakelijk, afgestemd tussen de ketenpartners.

4.3. Asynchrone communicatie

Events zijn van nature geschikt om te informeren over individuele gebeurtenissen/ entiteiten. Goed ontwerp van events zorgt dat niet volledige geaggregeerde objecten hoeven worden meegeleverd (bijv. een school event bevat informatie over de school, maar niet alle gekoppelde studenten en daaronder gekoppelde data). Het ontwerp van events en richtlijnen daarvoor is verder niet in scope van dit profiel.

26. SHOULD: *Indien* asynchrone communicatie wordt geïmplementeerd, dan zou dit moeten plaatsvinden door middel van CloudEvents.

    a. MAY: *Indien* voor asynchrone communicatie CloudEvents niet gebruikt wensen te worden, dan mag dit. De ketenpartners leggen dit bewust in de onderbouwing dan vast.

27. MUST: *Indien* voor asynchrone communicatie CloudEvents worden gebruikt dan worden volgende voorschriften hiervoor gevolgd:

    a. SHOULD: CloudEvents wordt niet alleen gebruikt voor simpele notificaties en signalen, maar ook voor het bereiken van ontkoppelde gegevensuitwisselingen; een event bevat naast de gebeurtenis ook informatie van het object.
    b. MUST: Dit Edukoppeling-profiel volgt de richtlijnen zoals beschreven in het NL GOV profile for CloudEvents v1.1 :
        i. Context attributen (H3);
        ii. Event Data (H4);
        iii. Size Limits (H5);
        iv. Gebruik van JSON, HTTP en webhook (bijlage A) zoals ook beschreven in de Guidelines for NL-GOV profile CloudEvents v1.0.

[28] Link opnemen naar gepubliceerd doc

Met uitzondering van:
i. CloudEvent Security Options (H6);
ii. Abuse protection as described in the guidelines[29].

c. MUST: Voor grote dataobjecten (lees: bestanden groter dan 'size limits') geldt dat een referentie naar de locatie van het object wordt opgenomen.
d. MUST: Een ketenpartner implementeert een Intermediair voor ontvangst en versturen van Events. De ketenpartner bepaalt zelf hoe de koppeling met het achterliggende landschap (de producer/ consumer) wordt gerealiseerd en gescheiden, uiteraard zonder de overige voorschriften uit het oog te verliezen.
e. MUST: De verzender voorschriften zoals beschreven in 4.3.1.
f. MUST: De ontvanger voorschriften zoals beschreven in 4.3.2.

4.3.1. Verzender voorschriften

28. MUST: Een verzender biedt een mogelijkheid voor ontvangers om te abonneren (expliciet of impliciet). Ketensamenwerking bepaalt gezamenlijk vast of abonneren expliciet of impliciet plaatsvindt. Bij abonneren is het van belang dat de verzender alleen gegevensleveringen naar ontvangers mogelijk maakt die toegang mogen hebben tot de data die wordt uitgewisseld. Dit stelt de verzender zeker via autorisaties.

29. MUST: Een verzender houdt bij het opstellen van requests rekening met aan welke geregistreerde partijen berichten dienen te worden aangeleverd (conform abonnementen) en stuurt de berichten naar het registreerde afleverpunt (i.e. een webhook) van de Intermediary (ontvanger kant). Hierbij houdt de verzender rekening met eventuele verschillende versies die een bericht kan hebben.

30. MAY: Een verzender mag CloudEvents bufferen en als batch aanleveren, indien de ontvanger CloudEvents in batch ondersteunt.

4.3.2. Ontvanger voorschriften

31. MUST: Een ontvanger abonneert zicht bij de producerende ketenpartner op de berichten die de ontvanger wenst te ontvangen (expliciet of impliciet). Ketensamenwerking bepaalt gezamenlijk vast of abonneren expliciet of impliciet plaatsvindt. De ontvanger levert hiervoor een webhook endpoint en Edukoppeling credentials aan.

32. MUST: Een ontvanger implementeert een endpoint voor individuele CloudEvents berichten.

33. MAY: Een ontvanger implementeert een endpoint voor CloudEvents in batch.

[29] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md#4-abuse-protection

34. MUST: Een ontvanger consumeert events at-least-once, via PUSH óf PULL vanuit de Intermediair. In normale omstandigheden consumeert een ontvanger altijd exactly-once, maar in foutscenario's kan dit vaker gebeuren.

35. MUST: Een ontvanger verwerkt events exactly-once door middel van idempotentie, retries, DLQ en/of replays in de intermediary en/of consumer.

    a. SHOULD: Het is aan te raden bovenstaande maatregelen te treffen zo dicht bij het doelsysteem waar de verwerking plaatsvindt (i.e. consumer).
    b. MAY: Om moderne flows i.c.m. legacy applicaties voldoende ruimte te geven mag dit ook anders geïmplementeerd worden, bijv. in de Intermediary.

36. MUST: Een ontvanger geeft een technische bevestiging aan de verzender dat het bericht is ontvangen. Functionele verwerking vindt asynchroon plaats aan de kant van de ontvanger.

5. Toepassingsscenario's

5.1. Probleemstelling - wat moet er met het profiel geregeld worden?

Voor onderwijsorganisaties, sectorvoorzieningen en ketenpartners die binnen een ketensamenwerking met elkaar gesloten data uitwisselen is het van belang dat er standaarden zijn die uitwisselingen van gegevens op een betrouwbare en robuuste manier mogelijk maakt zonder dat de systemen te nauw gekoppeld zijn aan elkaar. Hoge koppeling tussen systemen zorgt voor meer afhankelijkheid, meer afstemming en daarmee minder flexibiliteit en snelheid. Voor synchrone uitwisseling bood een Edukoppeling REST-profiel al de basis voor een gemeenschappelijke taal. Het CloudEvents [30] profiel is een specificatie voor formaat van events en beperkt hoe geautomatiseerde uitwisseling van events plaatsvindt. Het profiel bevat een specificatie, verschillende keuzevrijheden en richtlijnen die nog niet als standaard binnen de overheid worden gezien, en daarmee niet eenduidig toepasbaar zijn voor de onderwijssector.

Het nieuwe Edukoppeling Communicatie-profiel beoogt voor synchrone uitwisselingen en daarnaast voor asynchrone uitwisselingen met CloudEvents deze rol te vullen en daarmee te voorkomen dat partijen bij elke koppeling opnieuw discussie voeren over logistieke details. Binnen een ketensamenwerking is controle en afstemming over uitwisseling noodzakelijk. Zo geldt dat pieken bij één ketenpartner gecontroleerd door de keten moeten bewegen. Het Communicatie-profiel moet handvaten bieden die gebruikt kunnen worden om controle uit te oefenen binnen operationele afspraken die ketensamenwerkingen met elkaar maken.

Om binnen een ketensamenwerking betrouwbare en robuuste manier gegevens uit te wisselen zijn er richtlijnen nodig die dit ondersteunen zodat ook bij *tijdelijke storingen* het resultaat deterministisch is. Dit is niet mogelijk in een gedistribueerde ketensamenwerking zonder richtlijnen. Asynchrone uitwisseling die vaak worden gehanteerd bij producer – consumer patronen met events zoals in event driven architecturen werken van nature meer asynchrone, herhaalbare en mogelijke dubbele gebeurtenissen als gevolg van *at-least*-once *delivery*. Het toepassen van de juiste event levering semantiek, met de juiste aflever- en consumergaranties en bijbehorende handreiking voor idempotentie zijn daarom essentieel voor het juiste resultaat.

Het Communicatie-profiel gaat ketensamenwerkingen daarmee helpen richting betrouwbare uitwisselingen via een moderne architectuur.

5.2. Oplossing - wat biedt het profiel?

We bieden ketens een standaard voor asynchrone uitwisseling van gegevens met een hoog niveau van ontkoppeling. Ketenpartners behouden de flexibiliteit om hun eigen landschap naar eigen inzicht in te richten, maar worden óók gestuurd op interoperabiliteit binnen de ketensamenwerking. De interoperabiliteit wordt bereikt via duidelijke voorschriften die toegepast dienen te worden. Deze voorschriften sturen op betrouwbaarheid, ontkoppeling (flexibiliteit) en controle binnen de ketensamenwerking.

[30] NL GOV profile for CloudEvents 1.1

Het Edukoppeling Communicatie-profiel biedt structuur in de vorm van duidelijke scheiding tussen contextuele metadata en business data, en daarmee duidelijkheid en een stabiele API voor het transport van berichten. Hiermee kan een grote verscheidenheid aan berichten via één RESTful API overgebracht worden; hier is niet per berichttype afstemming nodig. Het Edukoppeling Communicatie-profiel specificeert *niet* de business data, onderliggende gegevensmodel en/of events die uitgewisseld dienen te worden; dit dient binnen de ketensamenwerking via andere (Edustandaard) standaarden geregeld te worden. Voorkeuren van richting van uitwisseling tussen producer en consumer worden niet door individuele ketenpartners opgelegd. Het profiel biedt juiste de flexibiliteit voor elke ketenpartner om dat binnen het eigen landschap naar wens in te richten. Zo geldt dat binnen een ketenpartner een consumer berichten aangeleverd kan krijgen óf kan ophalen.

6. Bijlage A: Begrippen

Abonneren: Abonneren is het proces waarbij een ketenpartner zich na registratie aanmeldt voor specifieke gegevens of gebeurtenissen, zodat deze automatisch worden ontvangen wanneer relevante wijzigingen optreden.

Antwoord: Een antwoord is de inhoudelijke reactie op een verzoek: het resultaat van een operatie of vraag.

API-aanbieder (ook wel API-provider): Ketenpartner die binnen een ketensamenwerking een RESTful API aanbiedt.

API-afnemer (ook wel API-provider): Ketenpartner die binnen een ketensamenwerking met een systeem een RESTful API afneemt.

Asynchroon: Bij asynchrone communicatie stuurt een computersysteem een bericht of event en ontvangt hoogstens een *bevestiging* van ontvangst. De *verwerking* vindt losgekoppeld en later plaats en levert geen direct antwoord op.

Atomiciteit: Binnen de grenzen van één enkel computersysteem zorgen transacties ervoor dat óf alle wijzigingen óf geen van de wijzigingen worden opgeslagen. Dit concept van atomiciteit zorgt ervoor dat een transactie “alles of niets” is.

Bevestiging: Een bevestiging van ontvangst geeft alleen aan dat het bericht technisch correct is ontvangen, niet dat het al is verwerkt.

Business data: Domein specifieke informatie (oftewel de payload). Dit kan informatie bevatten over de gebeurtenis zelf, details over gewijzigde gegevens of andere relevante gegevens.

Client secret: Een vertrouwelijke sleutel die alleen bekend is bij de client en afhankelijk van de vorm (symmetrisch (wachtwoord) of asymmetrisch (PKI)) ook bij de Authorization Server. Het wordt gebruikt om de client bij de Authorization Server te kunnen authenticeren via het token endpoint.

Confidential client[31]: Een confidential client is een client die draait in een omgeving waar vertrouwelijke gegevens (waaronder het client secret) veilig bewaard kunnen worden (bijvoorbeeld server-side webapplicaties).

Consumer: Een “consumer” ontvangt het event en onderneemt actie op basis daarvan. De consument gebruikt de context en de data om logica uit te voeren, wat kan leiden tot het optreden van nieuwe events.

Consumeren: Het lezen/verkrijgen van een event vanuit de Intermediary.

[31] https://datatracker.ietf.org/doc/html/rfc6749#section-2.1

Contextuele metadata: Contextuele metadata zijn vastgelegd in de contextattributen. Tools en applicatiecode kunnen deze informatie gebruiken om de relatie van events met onderdelen van het computersysteem of met andere events te identificeren.

Contract: Een (data) contract beschrijft hoe systemen met elkaar communiceren. Een data contract kan naast de berichtspecificaties van velden/datatypes/verplichte velden bestaan uit metadata zoals eigenaarschap, classificatie, ondersteuning, status (draft, stable), governance regels en eventueel schema evolutie voor migratie tussen versies.

Contractregister: Een contractregister maakt het mogelijk om afspraken over schemastructuren, eigenaarschap en kwaliteit te definiëren en centraal op te slaan en versioneren. Het zorgt ervoor dat systemen veilig met elkaar kunnen communiceren door afspraken over berichtenformaten, API-specificaties en events te beheren. Dit geldt dus voor synchrone en asynchrone API's.

DLQ (Dead Letter Queue): een aparte queue waarin events terechtkomen die na meerdere retries niet succesvol verwerkt konden worden.

Deterministisch: Het resultaat is hetzelfde, ongeacht hoe vaak de operatie wordt herhaald. Niet aan toeval overgelaten.

Idempotentie: Het meerdere keren verwerken van een identiek verzoek veroorzaakt geen schadelijke of onbedoelde neveneffecten of wijzigingen in de systeemtoestand en garandeert een deterministisch resultaat.

Intermediary: Een "intermediary" ontvangt een bericht dat een event bevat met als doel dit door te sturen naar de volgende ontvanger. Dit kan een andere intermediair of een consument zijn. Een typische taak van een intermediair is het routeren van events naar ontvangers op basis van de informatie in de context.

Ontvanger: Een ontvangende ketenpartner; de combinatie van Intermediary en Consumer.

Order guarantee: Order guarantee is het concept dat berichten worden verwerkt in de volgorde waarin ze zijn geproduceerd. Dit voorkomt dat een status wordt teruggezet naar een oude waarde terwijl een nieuwere waarde al is verwerkt.

Producer: De "producer" is een specifieke instantie, proces of apparaat dat de datastructuur aanmaakt die het CloudEvent beschrijft.
Produceren: Het creëren van een event en het afleveren ervan bij de Intermediary.

Registratie: Registratie is het proces waarbij twee ketenpartners elkaar initieel identificeren, authenticeren en configureren zodat gegevensuitwisseling tussen beide mogelijk wordt.

Replay: Het opnieuw aanbieden van eerder opgeslagen events om ze (opnieuw of alsnog) te verwerken.

Retry: Het opnieuw proberen te verwerken van een event nadat de eerste verwerking is mislukt, vaak vanuit een apart gezette queue.

RESTful API: Een RESTful API is een application programmable interface (API) die HTTP-methoden, zoals GET, POST, PUT, PATCH en DELETE, gebruikt om resources te beheren. Resources worden geïdentificeerd via URI's (Uniform Resource Identifiers) en worden doorgaans geretourneerd in JSON- of XML-formaat. We noemen ze RESTful omdat ze niet aan alle REST[32] principes[33] hoeven te voldoen.

Synchroon: Bij synchrone communicatie stuurt een systeem een verzoek en wacht op het *antwoord* (response). Dit antwoord wordt pas teruggestuurd nadat de *verwerking* is afgerond.

Tijdelijke storing: Een tijdelijke, kortdurende, vaak vanzelf herstellende fout, bijv. als gevolg voor een korte onderbreking in het netwerkverkeer (Engels: transient failures).

Verwerking: Verwerking is het daadwerkelijk uitvoeren van de businesslogica op basis van het ontvangen bericht of verzoek.

Verzender: Een verzendende ketenpartner; de combinatie van Intermediary en Producer.

[32] REST - Wikipedia
[33] Dit Edukoppeling-profiel gaat uit van vertrouwelijke gegevens waarbij ketenpartners weten wat ze van elke vragen binnen een bepaalde ketensamenwerking. Ondersteuning van HATEOAS lijkt niet noodzakelijk.

7. Bijlage B: Uitdagingen in een gedistribueerd landschap

Een gedistribueerd IT-landschap bestaat uit meerdere systemen die met elkaar communiceren door het uitwisselen van berichten. In die zin is er altijd sprake van een zender en een ontvanger van een bericht. We moeten ervoor zorgen dat er geen berichten verloren gaan of dubbel worden verwerkt. Dit zou namelijk leiden tot informatieverlies of dubbele data.

Bijvoorbeeld: in een IT-landschap met een ordersysteem en een betaalsysteem moeten we ervoor zorgen dat een order die in het ordersysteem wordt aangemaakt exact één keer financieel wordt verwerkt.

Robuuste afhandeling betekent dat alle berichten (verzonden door een zender) exact één keer door de ontvanger worden verwerkt, zonder aanvullende neveneffecten.

7.1. Traditionele uitdagingen in een gedistribueerd IT-landschap

Binnen de grenzen van één enkel systeem zorgen transacties ervoor dat óf alle wijzigingen óf geen van de wijzigingen worden opgeslagen. Dit concept van atomiciteit zorgt ervoor dat een transactie “alles of niets” is.

In een gedistribueerd IT-landschap is het echter, zonder aanvullende maatregelen die een gedistribueerde transactie over systeemgrenzen heen waarborgen, niet mogelijk om twee (of meer) systemen atomair bij te werken, omdat er in wezen sprake is van twee afzonderlijke transacties.

Consistentie over gedistribueerde systemen kan wel worden geïmplementeerd, maar dit gaat ten koste van de schaalbaarheid. In de volgende paragrafen verkennen we traditionele patronen om deze uitdagingen op te lossen binnen een microservices-architectuur of andere systeem-tot-systeemkoppelingen.

In paragraaf beschrijven we traditioneel gebruikte patronen in een traditioneel gedistribueerd IT-landschap.

7.2. Traditioneel gebruikte patronen in een gedistribueerd IT-landschap

7.2.1. Patroon: Two-phase commit (2PC)

Het two-phase commit-patroon wordt gebruikt om data op meerdere nodes in een gedistribueerd systeem atomair, volgens het alles-of-niets-principe, op te slaan. De twee fasen in dit patroon worden gecoördineerd door één centrale coördinator:

- Voorbereidingsfase (preparation phase): De coördinator stuurt een verzoek naar alle nodes en vraagt elke deelnemer te controleren of de transactie kan worden voltooid.
- Commitfase (commit phase): Als alle deelnemers positief hebben geantwoord, stuurt de coördinator een commit naar alle deelnemers. Als ten minste één deelnemer negatief heeft geantwoord, stuurt de coördinator een abort naar alle deelnemers.

Belangrijk in dit patroon is dat elke deelnemer in de voorbereidingsfase de duurzaamheid (durability) van de beslissing waarborgt.

7.2.2. Patroon: Saga-patroon

Het Saga-patroon is in essentie een reeks lokale transacties in afzonderlijke systemen. Deze reeks kan op twee manieren worden gecoördineerd:

- Elke lokale transactie publiceert berichten die lokale transacties in andere services activeren; een choreografie van berichtensequenties.
- Een orchestrator instrueert de deelnemers welke lokale transacties zij in welke volgorde moeten uitvoeren.

Binnen dit patroon moeten maatregelen worden genomen om een transactie “ongedaan te maken” wanneer een vervolgstap door een deelnemer niet succesvol kan worden verwerkt. Het ongedaan maken van een transactie is een compenserende actie, en geen strikte rollback.

7.2.3. Patroon: Transactioneel outbox-patroon

Tijdens de verwerking van een businessoperatie schrijft een microservice het ‘uitgaande bericht’ weg in een outbox-tabel binnen dezelfde transactie als de overige datawijzigingen. Een achtergrondproces leest de outbox en publiceert de berichten op betrouwbare wijze naar een message broker.

Dit zorgt ervoor dat er geen berichten verloren gaan wanneer de volledige businesslogica niet succesvol wordt afgerond. Hierdoor worden atomaire writes mogelijk binnen het systeem waarin de businessoperatie plaatsvond.

7.2.4. Trade-offs

We bekijken in deze paragraaf hoe traditionele patronen voor transactionaliteit (Two-phase commit en Saga, zie bijlage B) en moderne patronen (idempotentie, zie bijlage C) bijdragen aan end-to-end exactly-once verwerking setup.

	2PC	**Saga**	**Idempotentie**
Schaalbaarheid	---	+	+++
Complexiteit	- (coördinatie)	--- (compenserende acties, orkestratie)	+
Latency	---	-	+++
EDA-vriendelijk	---	+++	+++
Impact op leveranciers/keten	---	---	-
Eindscore	---	+-	++

Het implementeren van 2PC of een Saga patroon vereist dat interfaces en systemen met oog op deze patronen ontwikkeld worden. Voor bestaande systemen/ leveranciers in de keten kan dit daarmee flinke impact hebben. Daarnaast geldt dat significant meer berichtuitwisselingen nodig zijn om exactly-once verwerking te bewerkstelligen.

Bij idempotency geldt dit niet. Uitgangspunt is dat er altijd 1 bericht op de lijn gaat zoals dit ook plaatsvindt; er is dus 1 bericht nodig voor datauitwisseling. Met name in foutsituaties bij producer of consumers kunnen ervoor zorgen dat berichten meermaals respectievelijk geproduceerd of geconsumeerd worden.

8. Bijlage C: Handreiking voor idempotentie

In deze bijlage wordt een handreiking met praktische richtlijnen voor het implementeren van idempotentie beschreven. Eerst bespreken we *wanneer* dit moet worden toegepast en vervolgens *hoe*. Als laatst worden technische richtlijnen beschreven voor de implementatie.

8.1. Wanneer implementeren

Idempotentie zorgt ervoor dat het meerdere keren verwerken van hetzelfde bericht geen schadelijke of onbedoelde neveneffecten of wijzigingen in de systeemtoestand veroorzaakt en dat een deterministisch resultaat wordt gegarandeerd. Dit betekent ook dat niet alle service-operaties hierdoor worden beïnvloed. Sommige service-operaties zijn per definitie idempotent. Enkele voorbeelden:

- Elke GET-operatie is idempotent, omdat deze geen toestand wijzigt.
- Elke PUT-operatie (volgens de specificaties) vervangt of wijzigt een resource. *Voorbeeld:* PUT /account/123 met {balance: 100} moet het saldo van de rekening op 100 zetten. Opnieuw toepassen levert hetzelfde resultaat op. Order guarantee is hierbij belangrijk.
- Elke DELETE-operatie is idempotent, omdat een entiteit niet meer dan één keer kan worden verwijderd. Aandachtspunt is de response – zie sectie 8.2.1.
- POST-operaties kunnen wel of niet idempotent zijn, afhankelijk van de functionaliteit. *Voorbeeld:* POST /transactions met { amount: 100, account: 123 } kan bij herhaling opnieuw geld toevoegen aan rekening 123, wat leidt tot een onjuist saldo. *Voorbeeld:* POST /email/send zal opnieuw een e-mail versturen (er is dus een neveneffect).

8.2. Algemene richtlijnen

De aanbevolen manier om idempotentie te implementeren is door middel van een uniek event-ID in elk bericht. De consumer kan dan de volgende eenvoudige logica toepassen:

- Kent de consumer het message-ID al? → Verwerp het bericht.
- Kent de consumer het message-ID nog niet? → Verwerk het bericht en sla het message-ID op.

Om volledige atomiciteit te garanderen, wordt het message-ID opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

Een goede message-ID moet globaal uniek, stabiel, deterministisch en compact genoeg zijn. Dit message-ID moet door de producer worden gegenereerd.

8.2.1. Response

Elke idempotente service zorgt ervoor dat de response bij de eerste, tweede en derde poging identiek is. Dit is met name relevant voor services die:

- Data aanmaken: de client wil doorgaans het identificatienummer van de aangemaakte entiteit ontvangen.
- Data verwijderen: de client moet correcte foutafhandeling kunnen implementeren. Dit zou niet mogelijk zijn als bij een tweede DELETE-aanroep een foutresponse wordt teruggegeven.

8.3. Technische implementatierichtlijnen: Idempotentie

Idempotentie zorgt ervoor dat meerdere identieke verzoeken hetzelfde effect hebben als één enkel verzoek. Dit is cruciaal voor scenario's waarin clients verzoeken opnieuw proberen te versturen vanwege time-outs, netwerkfouten of onherstelbare fouten aan de clientzijde.

Deze technische richtlijnen zijn gebaseerd op een bestaand Internet Engineering Task Force (IETF)-concept [34] en bieden aanvullende richtlijnen om consistente implementatie en efficiënte systeemintegratie binnen de onderwijssector te ondersteunen. Eerst worden de idempotentieconventies binnen de sector beschreven, daarna volgt een gedetailleerd implementatievoorbeeld op basis van deze conventies.

8.3.1. Idempotentieconventies

Alle operaties die resources aanmaken of muteren, typisch POST- en PATCH-operaties, *moeten* een Idempotency-Key-header gebruiken. Hoewel DELETE-operaties per definitie [35] idempotent zijn, kunnen clients in een gedistribueerd systeem met retries inconsistente responses krijgen. Daarom *zouden* DELETE-operaties ook een idempotency key moeten gebruiken.

Elke operatie die niet voldoet aan de standaarden voor het gebruik van HTTP-methoden [36] en resources aanmaakt of wijzigt, *moet* eveneens worden meegenomen.

8.3.2. Uniekheid idempotentie sleutel

Een UUIDv4 [37] (RFC4122) *moet* worden gebruikt als idempotentie-sleutel.

8.3.3. Geldigheid en verloop van idempotentie sleutel

De geldigheid en vervaldatum van de idempotency key *zouden* rekening moeten houden met de tijd die nodig is voor automatische en/of handmatige retry-mechanismen en met de mogelijkheid van herhalingen (worstcasescenario na een restore door de client).

De time-to-live (TTL) voor een idempotency-entry *zouden* gebaseerd moeten zijn op het maximaal verwachte replay-venster: met andere woorden, het maximale tijdsvenster waarin een bericht opnieuw kan worden aangeboden. Een vervalperiode van 7 dagen wordt *aanbevolen*.

8.3.4. Idempotentie fingerprint

Voor extra veiligheid *moet* een idempotentie fingerprint worden aangemaakt. Aanbevolen wordt om de volledige request-payload te gebruiken om deze fingerprint te genereren.

[34] The Idempotency-Key HTTP Header Field
[35] RFC 9110: HTTP Semantics
[36] RFC 9114: HTTP/3
[37] RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

8.3.5. Verantwoordelijkheden client

Clients *moeten* de idempotentie sleutel opslaan voor de volledige levensduur van de operatie (oftewel de TTL die door de resource is gespecificeerd). Dit garandeert dat de sleutel uniek, stabiel, deterministisch en constant blijft gedurende de operatie. De client MAG hiervoor het transactional outbox-patroon gebruiken (zie bijlage A).

Clients *moeten* voor één idempotency key dezelfde request-payload versturen om HTTP 422-fouten van de resource te voorkomen.

Clients *zouden* automatische retries moeten stoppen nadat de TTL van de resource is verlopen. Daarna kan de client er niet langer van uitgaan dat de resource de idempotentie sleutel nog heeft opgeslagen. Handmatige bevestiging binnen de clientapplicatie *zou* dan moeten worden gestart en een nieuw verzoek *moet* een nieuwe sleutel bevatten.

8.3.6. Verantwoordelijkheden resource

De resource *moet* bij een duplicaatverzoek de response retourneren van de eerder voltooide operatie. Met name DELETE-verzoeken *zouden* ook dezelfde response moeten retourneren als eerder.

Als de resource meerdere clients ondersteunt, *moet* de opslag van idempotency keys per client worden gescheiden om het risico op botsingen te elimineren.

Om volledige atomiciteit te waarborgen, *zou* de idempotency key moeten worden opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

8.3.7. Foutafhandeling

De resource *moet* de idempotency key valideren voordat verdere verwerking plaatsvindt. Als een ongeldige idempotency key wordt aangeleverd, *zou* de resource moeten antwoorden met een HTTP 400-statuscode.

8.3.8. Client side implementatie (voorbeeld)

De volgende informatie kan in een outbox worden vastgelegd volgens het transactional outbox-patroon:

- event_id
- event_type
- payload
- schema_version
- created_at
- expires_at (moment waarop het event niet langer geldig wordt verklaard, bijvoorbeeld na het verstrijken van de TTL van de resource)
- idempotency_key (de UUIDv4)
- status (pending, acknowledged, failed)

De logica aan de clientzijde kan er als volgt uitzien:

1. Een business-event wordt vastgelegd inclusief alle metadata.
2. Zoek business-events met status *pending*.

3. Als expires_at > now(), zet de status op *failed*.
4. Anders:
   a. Lees het business-event.
   b. Exporteer het business-event en wacht op de response.
   c. Als de response een 4XX-fout aangeeft, los het probleem in de applicatie op.
   d. Als de response een 5XX-fout aangeeft, zet de status op *failed*.
   e. Als de response succesvolle verwerking aangeeft, verwijder het business-event of zet de status op *acknowledged*.

8.3.9. Resource implementatie (voorbeeld)

Alle API's die resources aanmaken of muteren, typisch POST, DELETE en PATCH-operaties, MOETEN een Idempotency-Key-header ondersteunen.

De volgende informatie kan worden opgeslagen in een idempotency key-tabel:

- idempotency_key
- idempotency_fingerprint (hash van het request)
- resource_id (indien van toepassing)
- client_id (indien van toepassing)
- response_status_code
- response_body
- created_at
- expires_at

De validatie aan de resourcezijde kan er als volgt uitzien:

1. Valideer de aanwezigheid én het formaat van de idempotency key. Indien deze ontbreekt of ongeldig is, retourneer een HTTP 400-fout.
2. Zoek de idempotency_key op voor resource_id + client_id in de tabel.
   a. Indien aanwezig én de idempotency fingerprint niet overeenkomt, retourneer een HTTP 422-fout.
   b. Indien aanwezig, retourneer de opgeslagen response.
3. Probeer een idempotency-lockrecord te verkrijgen inclusief vervaltijd. Als dit mislukt, verwerkt een ander proces deze key op dat moment en wordt een HTTP 409-fout geretourneerd.
4. Start een transactie:
   a. Verwerk de businesslogica.
   b. Voeg het idempotency-record inclusief response toe aan de tabel.
5. Commit de transactie en geef de idempotency-lock vrij.
6. Retourneer de response van de businesslogica.

Daarnaast kan een automatisch proces de idempotency key-entries opschonen waarvoor expires_at > now().

8.4. Voorschriften idempotentie

Dit hoofdstuk beschrijft de voorschriften voor het toepassen van idempotentie. De voorschriften zijn verdeeld in algemene voorschriften, voorschriften voor de client en resource.

8.4.1. Algemene voorschriften

1. MUST: De verzender produceert berichten at-least-once.

2. MUST: De ontvanger verwerkt berichten exactly-once via idempotentie.

3. MUST: Alle asynchrone uitwisselingen bevatten een Idempotency-Key-header.

4. MUST: Voor synchrone operaties die resources aanmaken of muteren, typisch POST- en PATCH-operaties, moet een Idempotency-Key-header worden meegegeven.

5. MUST: Elke synchrone HTTP-operatie die niet voldoet aan de standaarden voor het gebruik van HTTP-methoden [38] en resources aanmaakt of wijzigt, moet eveneens worden meegenomen.

6. SHOULD: Hoewel DELETE-operaties per definitie [39] idempotent zijn, kunnen clients in een gedistribueerd systeem met retries inconsistente responses krijgen. Daarom *zouden* DELETE-operaties in synchrone uitwisselingen ook een idempotency key moeten gebruiken.

7. MUST: Een UUIDv4 [40] (RFC4122) moet worden gebruikt als idempotentie-sleutel.

8. SHOULD: De geldigheid en vervaldatum van de idempotency key *zouden* rekening moeten houden met de tijd die nodig is voor automatische en/of handmatige retry-mechanismen en met de mogelijkheid van herhalingen (worstcasescenario na een restore door de client). De time-to-live (TTL) voor een idempotency-entry is 7 dagen.

9. MUST: Voor extra veiligheid moet een idempotentie fingerprint worden aangemaakt. Aanbevolen wordt om de volledige request-payload te gebruiken om deze fingerprint te genereren.

8.4.2. Voorschriften client

10. MUST: Clients *moeten* de idempotentie sleutel opslaan voor de volledige levensduur van de operatie (oftewel de TTL die door de resource is gespecificeerd). Dit garandeert dat de sleutel uniek, stabiel, deterministisch en constant blijft gedurende de operatie.

11. MAY: De client mag hiervoor het transactional outbox-patroon gebruiken (zie paragraaf 7.2.3).

12. MUST: Clients moeten voor één idempotency key dezelfde request-payload versturen. Bij synchrone operaties ontvangt de client anders een HTTP422-fout van de resource.

---

[38] RFC 9114: HTTP/3
[39] RFC 9110: HTTP Semantics
[40] RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

13. SHOULD: Clients zouden automatische retries moeten stoppen nadat de TTL van de resource is verlopen. Daarna kan de client er niet langer van uitgaan dat de resource de idempotentie sleutel nog heeft opgeslagen. Handmatige bevestiging binnen de clientapplicatie *zou* dan moeten worden gestart.

14. MUST: Clients moeten een nieuwe idempotentie sleutel genereren na het verzoek van de TTL; een nieuw verzoek moet een nieuwe sleutel bevatten.

8.4.3. Voorschriften resource

15. MUST: Bij synchrone operaties moet de resource bij een duplicaatverzoek de response retourneren van de eerder voltooide operatie. Met name DELETE-verzoeken *zouden* ook dezelfde response moeten retourneren als eerder.

16. SHOULD: Bij asynchrone operaties neemt de ontvanger het duplicate event in de Intermediary wel aan en negeert het event later in de keten als de consumer de verwerking doet.

17. MUST: Als de resource meerdere clients ondersteunt, moet de opslag van idempotency keys per client worden gescheiden om het risico op botsingen te elimineren.

18. SHOULD: Om volledige atomiciteit te waarborgen, zou de idempotency key moeten worden opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.
19. MUST: De resource *moet* de idempotency key valideren voordat verdere verwerking plaatsvindt. Als een ongeldige idempotency key wordt aangeleverd, antwoord de resource met een HTTP 400-statuscode.

9. Bijlage D: Referenties

- **NL GOV profile for CloudEvents 1.1 (LOGIUS STANDARD)**
  https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/

- **CloudEvents v1.0.1 (CNCF SPECIFICATION STANDARD)**
  https://github.com/cloudevents/spec/blob/v1.0.1/cloudevents/spec.md

- **HTTP 1.1 Web Hooks for Event Delivery v1.0.1 (CNCF SPECIFICATION STANDARD)**
  https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md

- **Guidelines for NL-GOV profile CloudEvents 1.0 (LOGIUS GUIDELINES)**
  https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/

- **AsyncAPI specification for defining asynchronous APIs v3.1.0 (LINUX FOUNDATION SPECIFICATION)**
  https://www.asyncapi.com/docs/reference/specification/v3.1.0

- **OpenAPI specification for defining synchronous APIs v3.2.0 (LINUX FOUNDATION SPECIFICATION)**
  https://spec.openapis.org/oas/v3.2.0.html

Idempotence:

- **Idempotency-key HTTP Header Field (IETF PROPOSED STANDARD)**
  https://www.ietf.org/archive/id/draft-ietf-httpapi-idempotency-key-header-07.txt

- **RFC 9110: HTTP Semantics, idempotent methods (IETF STANDARD)**
  https://www.rfc-editor.org/rfc/rfc9110#section-9.2.2

- **RFC 9114: HTTP/3 (IETF PROPOSED STANDARD)**
  https://www.rfc-editor.org/rfc/rfc9114

- **RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace (IETF PROPOSED STANDARD)**
  https://www.rfc-editor.org/rfc/rfc4122