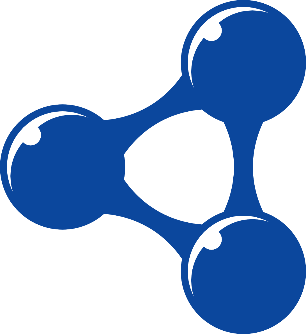
Procesdocumentatie



**Linked Data Theater Informatiehuis Water**

Colofon

Titel: Procesdocumentatie Linked Data Theater Informatiehuis Water

Auteurs: Ordina VisionWorks:

Brattinga, Marco

Schollmeijer, Wessel

Reuijl, Ad

© Copyright Ordina 2018

Niets uit dit document mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand of openbaar gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of op enig andere manier, zonder voorafgaande schriftelijke toestemming van Ordina. De enige hierop toegestane uitzondering is verspreiding onder uitsluitend die eigen medewerkers van de geadresseerde van het document, die voor de beoordeling van het document verantwoordelijk zijn.

Inhoudsopgave

[1 Inleiding 4](#_Toc515030992)

[1.1 Linked Data Theatre 4](#_Toc515030993)

[2 Uitgangspunten 5](#_Toc515030994)

[3 Architectuur 9](#_Toc515030995)

[3.1 Ldt 2.0 Framework 9](#_Toc515030996)

[3.2 Configuratie 11](#_Toc515030997)

[4 Opbouw Configuratie LDT 2.0 14](#_Toc515030998)

[4.1 Frontend 14](#_Toc515030999)

[4.2 Backend 15](#_Toc515031000)

[4.3 Transaction 15](#_Toc515031001)

[4.4 System 16](#_Toc515031002)

[4.5 Elmo vocabulary 17](#_Toc515031003)

[5 Datamodel 18](#_Toc515031004)

[5.1 Uri Strategie 18](#_Toc515031005)

[5.2 Vocabulaires 19](#_Toc515031006)

[5.2.1 SKOS 19](#_Toc515031007)

[5.2.2 SKOS-THES 20](#_Toc515031008)

[5.2.3 RDFS 21](#_Toc515031009)

[5.2.4 DC 21](#_Toc515031010)

[5.2.5 DCTERMS 21](#_Toc515031011)

[5.2.6 DCAT 22](#_Toc515031012)

[5.2.7 ADMS 22](#_Toc515031013)

[5.2.8 Aquo vocabulaire 22](#_Toc515031014)

[5.3 Model Validatie 22](#_Toc515031015)

[5.4 Keuzen domeintabel 23](#_Toc515031016)

[6 Beheren en Historie 24](#_Toc515031017)

[6.1 UITLEG STAGING EN TRANSACTIE PER ONDERDEEL 24](#_Toc515031018)

[6.2 Functionaliteit 24](#_Toc515031019)

[6.3 Uploaden 24](#_Toc515031020)

[6.4 Navigeren 24](#_Toc515031021)

[6.5 Beheren 24](#_Toc515031022)

[6.6 Delen 24](#_Toc515031023)

[7 Nawoord 25](#_Toc515031024)

# Inleiding

Op het moment dat deze documentatie is geschreven, maakte het Kadaster gebruik van een Linked Data Theater (LDT 2.0 óf Dotwebstack) voor verschillenden onderdelen van het Digitaal stelsel omgevingswet. Het Informatie Huis Water (IHW) heeft Ordina, opdracht gegeven voor het maken van een soortgelijk systeem. Wanneer de Omgevingswet in 2018 in werking treedt, dient de datahuishouding van IHW op orde te zijn.

Het Informatiehuis Water (IHW) werkt momenteel met twee specifiek voor IHW gebouwde applicaties om begrippen en domeintabellen in te kunnen zien, namelijk Aquo LOC (voor begrippen) en Aquo DS (voor domeintabellen).

Deze applicaties bieden toegang tot het Aquo-lex, ookwel het ‘waterwoordenboek’ (<http://www.aquo.nl/over-aquo/aquo-onderdelen/aquo-lex/>), waar ongeveer 7500 definities van termen in staan. Momenteel is er één persoon verantwoordelijk voor beheer, wat complicaties oplevert bij uitbreiding of aanpassing van functionaliteiten. ook zijn er performanceproblemen bij exporteer functies en geavanceerd zoeken binnen Aquo LOC. Data van Aquo LOC en Aquo DS is vrij toegankelijk voor bezoekers en voor het LDT 2.0 geëxtraheerd en geconverteerd vanuit de twee applicaties.

Een goede oplossing voor het IHW is Linked Data. Het sluit aan bij de best practices voor publiceren van data op het web. Aanpassing en uitbreiding van data is tevens eenvoudiger en sneller. Lees voor een goede introductie van Linked data ook http://bp4mc2.org/.

Het Linked Data Theater 2.0 (LDT 2.0) is een snel te configureren presentatie laag voor Linked Data en doordat deze vanuit Ordina en gelieerde instanties wordt doorontwikkeld de geschikte keuze. Het bouwen van de presentatie laag, en het maken van het onderliggende data model neemt naar schatting 3 maanden in beslag. Dit document bevat de procesbeschrijving van het tot stand komen van het Aquo Linked data Theatre en de generale opbouw van de applicatie en haar configuratie. De nieuwe applicatie zal zowel de data van Aquo Loc en Aquo Ds omvatten en daarmee de volledige Aquo-Lex bedienen.

## Linked Data Theatre

HIER IETS OVER DE ORSPRONG VAN HET LDT EN DE BENAMINGEN ZOALS STAGE ETC.

**HIER NOG IETS OVER DE EINDGEBRUIKERS VAN DE APPLICATIE**

# Uitgangspunten

Bij het tot stand brengen van het Aquo Linked data Theater zijn er een aantal uitgangspunten vastgesteld en onderverdeeld in volgorde van importantie, te weten: Must Have, Should Have, Could have, Won’t Have.

|  |  |  |  |
| --- | --- | --- | --- |
| No | Eis | Omschrijving | Prioriteit |
| D.25 | Zoeken (algemeen) | Termen worden bij invoeren aangevuld tot lijst bestaande waarden. (zoals zoekveld van Google) | Could have | |
| D.0 | Wijzigingen worden direct gepubliceerd | Aanpassingen in domeintabellen en woordenboek worden direct na invoer gepubliceerd. | Must have | |
| D.1 | Toevoegen domeinwaarde |  | Must have | |
| D.2 | Verwijderen domeinwaarde |  | Must have | |
| D.3 | Aanpassen domeinwaarden |  | Must have | |
| D.4 | Importeren van lijst domeinwaarden | Domeinwaarden kunnen in bulk worden ingevoerd door het importeren van een file. | Must have | |
| D.6 | Historie van een domeinwaarde | Voor versiebeheer moet de historie van een domeinwaarde worden bijgehouden. Dus datum van invoer en laatste wijziging. | Must have | |
| D.7 | Historie van een domeinwaarde | Ook oude waardes en de periode van geldigheid worden bijgehouden. | Must have | |
| D.11 | Toevoegen kolom aan domeintabel | Beheerders moeten zelf kolommen binnen domeintabellen kunnen bewerken. In linked data context bestaan kolommen niet als zodanig. Maar de essentie is dat een informatieveld moet worden kunne toegekend aan een gegevensdefinitie. | Must have | |
| D.12 | Verwijderen kolom uit domeintabel | Zie boven. | Must have | |
| D.13 | Aanpassen kolom in domeintabel | Zie boven. | Must have | |
| D.14 | Zoeken naar een domeinwaarde binnen een domeintabel | Zoeken naar een tekstveld | Must have | |
| D.23 | Zoeken naar een domeintabel | Zoeken op tekst | Must have | |
| D.29 | Mappings | Er zijn domeintabellen die een relatie tussen domeinwaarden in verschillende tabellen weergeven. Of: er kan een relatie gelegd worden tussen verschillende domeinwaarden, ook voor domeinwaarden in andere systemen. | Must have | |
| D.32 | Export domeintabel als XML |  | Must have | |
| D.34 | Export domeintabel als CSV |  | Must have | |
| D.5 | Wijzigen in bulk | Mogelijkheid om meer waarden in één keer te wijzigen na selectie. | Should have | |
| D.8 | Audit-trail | De historie van een domeinwaarde, wanneer deze gecreerd en gewijzigd is kan getoond worden per domeinwaarde. | Should have | |
| D.9 | Audit-trail | Bij invoering, wijziging of beeindiging van een domeinwaarde kan aangegeven worden welk wijzigingsvoorstel ten grondslag ligt aan de definitie. | Should have | |
| D.15 | Zoeken naar een domeinwaarde binnen een domeintabel | Zoeken op status | Should have | |
| D.16 | Zoeken naar een domeinwaarde binnen een domeintabel | Zoeken op begin of einddatum | Should have | |
| D.19 | Zoeken over alle domeinwaarden | Zoeken op status | Should have | |
| D.20 | Zoeken over alle domeinwaarden | Zoeken op begin of einddatum | Should have | |
| D.24 | Zoeken naar een domeintabel | Zoeken op relaties met andere tabellen | Should have | |
| D.26 | Sorteren | Domeintabellen kunnen worden gesorteerd op naam of op datum laatste wijziging. | Should have | |
| D.27 | Overzicht toepassing domeintabellen | Er is een volledige overzicht van gebruik van domeintabellen binnen de informatiemodellen. Zie ook Beheer Informatiemodel | Should have | |
| D.35 | Export als SKOS |  | Should have | |
| D.39 | Beeldmateriaal | Het is mogelijk in een defintite te verwijzen naar een beeld. | Should have | |
| D.36 | Export als HTML |  | Won't have | |
| D.38 | Parameterlijst | Domeintabel parameter is opgesplitst opgenomen in de DS inclusief lijsten voor typeringen en objecten. M.a.w. er is per soort parameter een aparte domeintabel. De huidige tabel parameter wordt opgebouwd uit de onderliggende aparte domeintabellen. | Won't have | |
| D.10 | Status | De status van een domeinwaarde wordt getoond. |  | |
|  |  |  |  | |
| D.17 | Zoeken naar een domeinwaarde binnen een domeintabel | Zoeken op code of ID |  | |
| D.18 | Zoeken over alle domeinwaarden | Zoeken naar een tekstveld |  | |
| D.21 | Zoeken over alle domeinwaarden | Zoeken op code of ID |  | |
| D.22 | Zoeken naar beeindigde domeinwaarden | Zoeken naar beeindigde domeinwaarden |  | |
| D.28 | Code en ID relatie | In de Aquo DS is de code op dit moment de sleutel (uniek en verplicht). Daarnaast is het ID opgenomen in de domeintabellen deze is ook uniek en verplicht. Wijzigen van de code heeft nu tot gevolg dat ook het ID moet wijzigen. Alleen met een truc is dit op te lossen. |  | |
| D.30 | Termen in lexicon met meer definities | Er zijn termen met meer dan één definitie. Voorbeelden van termen met definities vanuit verschillende gezichtspunten: Oppervlaktewaterlichaam, Waterbodem |  | |
| D.31 | Export vanuit gezichtspunt | Relatie met O.1, O.2 en O.3 |  | |
| D.37 |  | Bij iedere domeinwaarde kan een relatie met de (LOC) defintie getoond worden. |  | |
|  |  |  |  | |

Deze uitgangspunten zijn veelal een uitbreiding op bestaande functionaliteit in de huidige Aquo DS en Aquo LOC. De huidige functionaliteit die in ieder geval moet terugkomen in de nieuwe applicatie omvatten:

* Eenvoudig toegang tot de data via een SOAP Service
* Een HTML presentatielaag
* Beheer functionaliteit om verschillende onderdelen van begrippen, categorieën én domeintabellen zelf aan te kunnen passen.
* Historie bijhouden voor begrippen en domeintabellen

Toegang tot de data via een Soap service wordt daarom bijvoorbeeld niet genoemd omdat deze functionaliteit al aanwezig is in de huidige omgevingen.

Eveneens kan worden gesteld dat het Linked data Theater functionaliteit met zich meebrengt die wellicht niet in eerste instantie is benodigd maar wel wat toevoegt aan de functionaliteit zoals bijvoorbeeld de Graph appearance in de begrip detail overzichten.

**EVENTUEEL VERWIJZEN NAAR DE EXCEL**

# Architectuur

Om aan de wensen en eisen van het IHW te kunnen voldoen is gekozen voor de volgende technische architectuur:

Technische architectuur:

Platform: LDT 2.0 met embedded Tomcat server

Database server: Apache Jena Fuseki

Opmaakt: CSS

Upload data: Curl & .bat

De applicatie architectuur bestaat uit het LDT 2.0 framework met bijbehorende configuratiebestanden. In de configuraties bepaal je hoe het systeem eruit moet zien, welke pagina’s we willen aanmaken, welke query’s welke data moeten ophalen om te tonen op de pagina’s en welke functionaliteit van het framework moet worden aangesproken.

Deze tweedeling van configuraties en framework verdiend meer uitleg en zullen daarom beide worden besproken. Het framework in mindere mate en de configuraties ietwat uitgebreider.   
  
Eventuele aanpassingen in de toekomst zullen veelal plaatsvinden in de configuraties, terwijl wensen voor additionele functionaliteit (zoals we bij de wens tot een SOAP service zijn tegengekomen) veelal een doorontwikkeling van het LDT 2.0 framework betreffen.

## Ldt 2.0 Framework

Het LDT 2.0 framework is -zoals de naam al doet vermoeden- een doorontwikkeling op het oudere LDT framework. Het oude framework was geprogrammeerd in XSLT, een XML gebaseerde taal die 2 nadelen had, weinig mensen kennen de taal en het framework kon niet overweg met de nieuwere REST API’s.

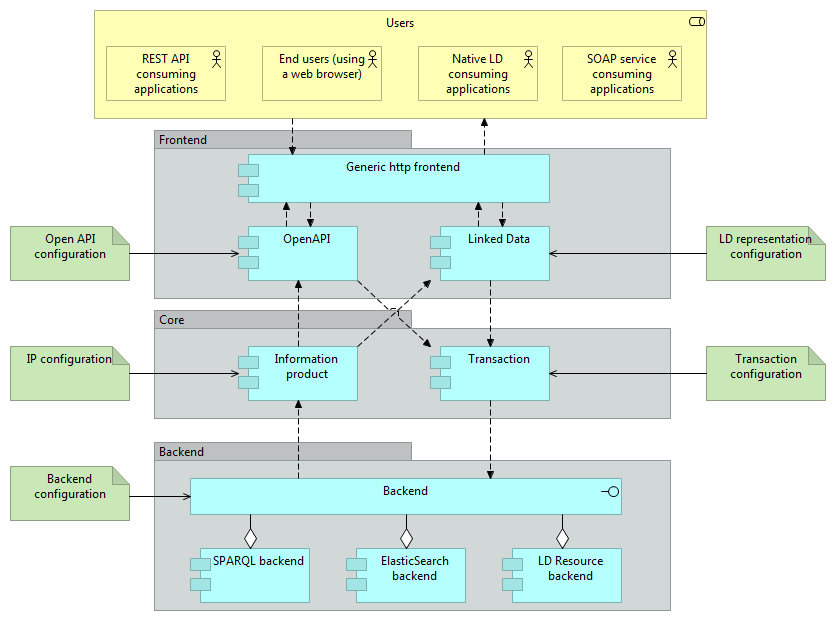
Bij het ontwikkelen van het nieuwe framework stond de onderhoudbaarheid en mogelijkheid tot doorontwikkeling hoog op de prioriteitenlijst evenals de mogelijkheid om naast een Linked data Backend ook in gegevensuitwisseling via API’s te kunnen voorzien. Het resultaat is een doordacht framework, geschreven in java met optimale connectiviteit.

Het framework bestaat kortgezegd uit 3 delen, een frontend, een backend en een core.   
De frontend zorgt voor alle communicatie naar buiten, dit kan op verschillende manier (zie figuur 1). Zo bestaat er de mogelijkheid om via html (een internetpagina) de data te bekijken maar ook via een API, SOAP service óf Linked data backend gegevens op te vragen. De verschillende manieren hebben natuurlijk hun voor en nadelen, zo zal een API altijd een bepaald antwoord teruggeven bij een bepaalde vraag (deze API specificatie leg je vast) en een Linked data endpoint zal altijd Linked data retourneren. Het hangt van de eindgebruiker af wat de gewenste manier is om met elkaar te communiceren.

Het backend gedeelte van het framework doet de communicatie met de database server zo kunnen we een eigen sparql backend aanroepen op onze server maar we kunnen ook een backend aanspreken van een andere server zoals we in de Aquo applicatie doen naar de GWSW Dataset. Ook bestaat de mogelijk om verbinding te maken met ElasticSearch maar deze functionaliteit blijft voor het IHW voor nu onbenut.

De core van de applicatie doet het rekenwerk, de in de configuratie aangeleverde querys worden hier door het framework afgehandeld. De query’s worden in het LDT 2.0 Informatie producten genoemd vanuit hun onmisbare functionele eigenschappen; het leveren van informatie. Naast informatieproducten vinden we een tweede variant te weten de Transaction. Wijzigingen in data en uploads van data worden aangeboden middels een transactie en in dit onderdeel afgehandeld.

Zoals het schema laat zien benodigd dit framework input op alle onderdelen van de frontend, backend en core middels configuratie bestanden. Laten we verder inzoomen op deze configuratie.

*Figuur 1*

## Configuratie

De configuratie bestanden kunnen worden opgesplits in 5 delen, te weten:

*Framework Configuratie*

1. Backend configuratie backend system
2. Transactie configuratie core transactie
3. Informatie producten configuratie core backend
4. Open API configuratie frontend openApi
5. Linked data representatie configuratie frontend frontend

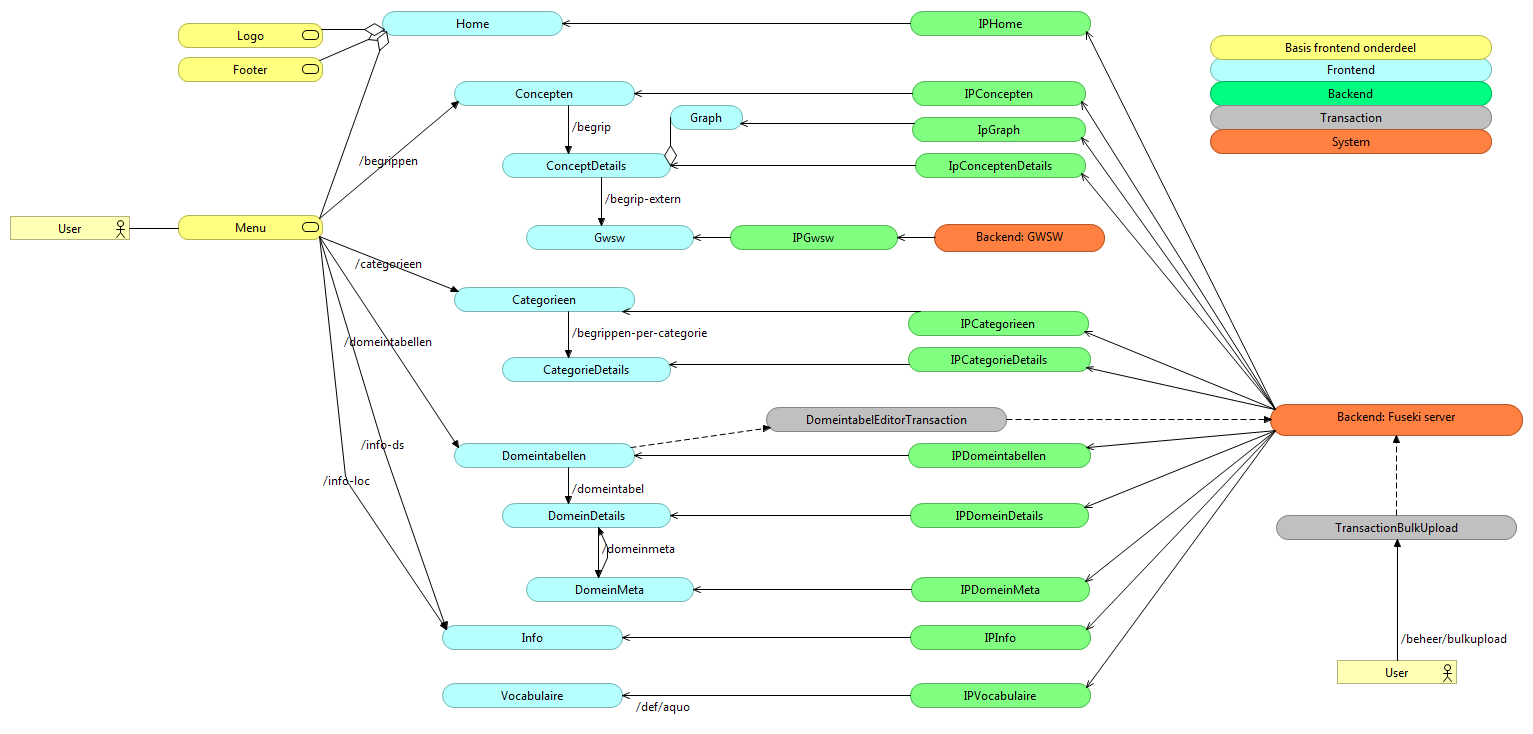
De bestanden staan verdeelt in de volgende mapstructuur op de ontwikkel omgeving (<https://github.com/bp4mc2/bp4mc2-ihw/tree/develop>).

Via het pad: ‘/config/model’ vinden we 4 mappen:

* frontend
  + Bevat alle Linked data representatie configuraties
* backend
  + Bevat alle Informatieproducten (dit kan verwarrend zijn met bovenstaande framework architectuur waarin de informatieproducten zich als onderdeel van de ‘core’ laten omschrijven, echter vanuit een configuratie oogpunt is het informatieproduct een backend voor het frontend gedeelte, hier wordt namelijk de data opgehaald tbv de frontend, zie Figuur 2)
* transactions
  + Bevat alle Transactions voor het uploaden van data en editen van data
* system
  + Bevat alle benodigde onderdelen van het Linked Data Theatre zoals de Stage, Sites, Parameters, Prefixes en natuurlijk de Backends (de configuratie van de gebruikte backends, hierin staat het pad naar de betreffende backend in).

Zie figuur 2.

Figuur 2



Figuur 2 geeft een totaal overzicht van alle configuratie bestanden en hoe ze met elkaar in relatie staan. Het LDT 2.0 laadt bij het opstarten alle configuratie bestanden in waarna de applicatie draait en benaderbaar is. De volgorde van inladen maakt voor het LDT niet uit, in principe zouden alle configuraties binnen in 1 TRIG bestand kunnen staan, dat zou echter het beheer onoverzichtelijk maken.

Een gebruiker gaat naar de homepage van de applicatie via de link ‘/begrippen’. Daar vindt hij een pagina met een overzichtstabel van alle begrippen, met een menu, een footer en een logo. Op de achtergrond vinden er meerdere stappen plaats:

* De link triggert via een elmo:pathPattern de config:ConceptenLijst.
* Deze config:ConceptenLijst is van het type elmo:Representation en er wordt dus een (door het elmo-vocabulaire gedefinieerde) representatie aangeroepen.
* Deze representatie bevat een aantal onderdelen die ergens anders in de code zijn gedefinieerd, te weten:
  + elmo:contains config:Image óf config:Conceptimage; 🡪 dit is het Logo óf het tijdelijke logo.
  + elmo:contains config:Menu; 🡪 dit is het menu.
  + elmo:contains config:Footer; 🡪 dit is de footer.
  + elmo:contains config:ExportIndex; 🡪 dit is de funtionaliteit om een export uit te voeren.
  + elmo:contains config:ConceptenIndex; 🡪 dit is de index om te kunnen filteren binnen de tabel, deze code staat overigens in hetzelfde frontend bestand beschreven omdat we deze filter alleen hier tegen komen.
* Naast deze onderdelen heeft de representatie-pagina ook een hoofd element, namelijk de tabel met begrippen. Voor die tabel hebben we 2 dingen nodig, een informatieproduct die de informatie ophaalt vanuit de database in de vorm waarin wij de data willen hebben. Daarnaast heeft de representatie een elmo:appearance nodig waarin we aangeven welk onderdeel van het framework we willen aanroepen voor het weergeven van de data, en eventueel een verdere finetuning daarvan.
* In de elmo:appearance vinden we vervolgens de definiering van het onderdeel, namelijk een elmo:TableAppearance. Daarbij specificeren we de fragmenten waaruit de tabel moet bestaan, de volgorde en de namen van de kolommen. Andere verborgen trucjes die hierin plaatsvinden zijn bijvoorbeeld het groeperen van de categorielabels en het klikbaar maken van de inhoud van de tabel via het onderderdeel xhtml:link.

Al deze code staat in het frontend bestand ‘Concepten.trig’ beschreven, we verwijzen in dit bestand echter wel naar een InformatieProduct (in het bestand IPConcepten.trig) en naar andere onderdelen (Footer, Menu, Logo) die ergens anders geconfigureerd zijn (een uitzondering hierbij is de index appearance die zich in hetzelfde bestand bevind). Figuur 2 geeft inzicht in welk bestand er gebruikt wordt en welke functie het dient.

De figuur laat 3 soorten pijlen zien:

* Serving relation ingekleurde pijl
* Triggering relation pijl
* Influence relation gestippelde pijl

De pijlen geven de functie weer, namelijk:

* het dienen (**serving**) van een frontend configuratie
* het **triggeren** van de juiste frontend configuratie
* Of, in het meest complexe geval, het wijzigen van de backend/database via het triggeren van een transactie, deze noemen we de **influencing** relationship.

# Opbouw Configuratie LDT 2.0

De configuraties gebruiken (zoals in voorgaand voorbeeld in paragraaf 3.2) verschillende onderdelen van het framework en van het elmo vocabulaire. Dit hoofdstuk bespreekt de verschillende gebruikte onderdelen en hun functies. Technische specificaties zijn te vinden op <https://github.com/dotwebstack/dotwebstack/tree/master/docs/design>.

## Frontend

* Endpoint
  + Elmo:Stage
  + Elmo:pathPattern
  + Elmo:getRepresentation
* Representation
  + Elmo:contains
  + Elmo:appearance
  + Elmo:stage
  + Elmo:informationProduct
* Appearance
  + Fragments
    - Elmo:Applies to
    - Xhtml:link
    - Elmo:name
    - Rdfs:label
    - elmo:index

Zoals bovenstaand is af te lezen bestaat een frontend bestand uit 3 delen. Het endpoint zorgt voor een bereikbare ingang (elmo:Pathpattern) en het opzetten van een ‘Stage’. Op de Stage wordt vervolgens de Representatie gezet.

De Representatie is de inhoud van de stage en beantwoord de vraag: wat wil ik op deze pagina hebben staan? De elmo:contains statement zorgt voor het insluiten van andere gedefinieerde onderdelen zoals een menu, dit kan tevens een hele representation zijn zoals we bij de begrip-details-pagina zien met het insluiten van een GraphAppearance. De elmo:appearance statement verwijst naar het hoofdelement van de pagina die verderop in het bestand wordt gedefinieerd. Als laatste komen we het Informatieproduct tegen die wordt gebruikt voor het ophalen van data ter behoeve van het hoofdelement van de pagina (de Appearance).   
  
De Appearance laat, zoals de naam al doet vermoeden, de data ‘verschijnen’. Daarin bestaat de keuze uit vele vormen van representeren, zie voor een totaal overzicht <https://github.com/architolk/Linked-Data-Theatre/wiki/Appearances> (WAT IS HIERVAN NOG UP TO DATE). Na het kiezen van de geschikte Appearance kan er op detail niveau worden gespecificeerd hoe de verschillende stukken data zich moeten tonen, dit gebeurd in de fragment(s). Zo kan er een link verwijzing naar een andere Frontend-endpoint worden gemaakt met een xhtml:Link waarin het element waarop geklikt wordt als parameter kan worden meegegeven naar de betreffende pagina. Ook kan de volgorde van de data elementen worden bepaald met het elmo:index statement. Eventueel kan er een label worden meegegeven via een rdfs:label statement. Per fragment geeft je aan op welk stuk data hij betrekking moet hebben via een elmo:appliesTo (deze is in het dotwebstack framework verplicht). Als laatste is er de elmo:name statement waarin ik niet weet hoe hij werkt.

Eveneens: hoe zorg je dat uri en label over elkaar heen worden gelegd in de 2 typen queries?

## Backend

* Informatieproduct
  + Elmo:backend
  + Elmo:resulttype
  + Elmo:requiredParameter
  + Elmo:optionalParameter
  + Elmo:query

Het informatieproduct zorgt voor het aanleveren van de data in een formaat die we kunnen gebruiken voor de verschillende Appearances. Er zijn 2 typen query’s namelijk de **Select**-query en de **Construct**-query. In het geval van een select query stellen we een tabel op met daarin de betreffende data. In het geval van een Construct query stellen we een nieuwe set triples op in de vorm zoals we hem nodig hebben voor de Appearance. Welk type geschikt is kan worden teruggevonden in deze tabel: <https://github.com/architolk/Linked-Data-Theatre/wiki/Appearances>. Wanneer we een Select-query gebruiken moeten we ook aangeven dat het een elmo:result type een Tuple-result is. Standaard gaat het LDT uit van een Graph result type welke van toepassing is op een Construct query.

Bij het informatieproduct geven we aan welke backend we willen aanspreken voor het ophalen van de data, veelal is dat onze eigen backend maar dat kan ook een andere backend zijn zoals bijvoorbeeld de GWSW

In de query kunnen we ook parameters meegeven die we vanuit de frontend meekrijgen of los kunnen invoeren in de url mbv een parameter mapper. Deze parameters kunnen verplicht of optioneel zijn voor de query, dit dient aangegeven te worden via een elmo:requiredParameter of elmo:optionalParameter in het informatieproduct.

## Transaction

* Elmo:endpoint
  + Elmo:pathPattern
  + Elmo:stage
  + Elmo:putService /elmo:postService
* Elmo:service
  + Elmo:stage
  + Elmo:transaction
* Transaction
  + Sequential flow
    - Update step
      * Elmo:backend
      * Elmo:query
    - Persistence step
      * Elmo:persistenceStrategy
      * Elmo:backend
      * Elmo:targetgraph

De Transaction is één van de laatste toevoegingen aan het LDT 2.0 framework en voorziet in het uitvoeren van wijzigingen of toevoegingen op de dataset. De transactie heeft 4 mogelijke stappen die duidelijk worden uitgelegd op de volgende pagina: <https://github.com/dotwebstack/dotwebstack/blob/master/docs/design/transactions.md>

Naast de elmo:Transaction, bestaat de transactie-configuratie uit een endpoint met daarin het pathpattern waarmee je de transactie aanroept, verplicht hierbij zijn de Stage en de Service die moeten worden gebruikt, de service wijst op haar beurt weer door naar de elmo:Transaction.

## System

* Backends
  + Elmo:endpoint
* Parameters
  + Elmo:name
  + Elmo:shape
* Sites
  + Elmo:domain
* Stages
  + Stage
    - Elmo:site
    - Elmo:basepath
    - Elmo:layout
    - Dc:title
  + Layout
    - Xhtml:stylesheet
* Prefixes

In de system folder vinden we alle benodigdheden voor de werking van de configuratie. Zo moeten we de Stage(s) definiëren waar we bepalen wat het basepath is om de applicatie te benaderen via de browser. Ook bepalen we in de Stage welke layout er moet worden gebruikt voor het tonen van de inhoud, zodoende kan de applicatie in eigen huisstijl worden getoond. De Site geeft de verwijzing naar het domein, in ons geval ‘Localhost’, welke zal moeten worden gewijzigd voor de livegang naar de Aquo omgeving. In het Parameter bestand vinden we alle gebruikte Parameters voor de query’s gebruikt in de Informatieproducten en Transacties. Hierbij definiëren we de naam van de parameter maar ook de shape (vorm) die de parameter moet teruggeven, dit kan bijvoorbeeld een string zijn maar ook een datum of een IRI. In de Shape kan je ook een default value meegeven, handig als de parameter bijvoorbeeld verplicht is. Om niet in ieder configuratiebestand de prefixes te hoeven herhalen vinden we ook een prefix bestand in de system folder met daarin alle mogelijke prefixes die we tegenkomen in de gecombineerde configuratie bestanden; de query’s die we vinden in de informatie producten dienen nog wel prefix statements te gebruiken.

Verder vinden we in de system folder ook de backend verwijzingen naar alle gebruikte backends in de Transacties en Informatieproducten. Momenteel gebruiken we er 2:

* de localBackend voor alle query’s naar onze Fuseki query backend met daarin de data van het IHW. <http://localhost:3030/ihw/query>
  + Deze zien we het meest in de informatieproducten terug komen.
* de UpdateBackend voor alle query’s tbv het uploaden van data naar onze Fuseki backend. <http://localhost:3030/ihw/update>
  + Deze zien we het meest bij de Transactions terug komen.

Fuseki biedt de mogelijkheid om verschillende backends aan te spreken, de backend hebben hun eigen functie, zoals een update, upload of query. Via <http://localhost:3030/dataset.html> kan er een overzicht worden opgevraagd van de backend onder de info-tab. Dit werkt alleen wanneer je Fuseki actief laat draaien.

## Elmo vocabulary

Voor het configureren van het Linked Data Theatre wordt ook RDF gebruikt; dezelfde systematiek als het vastleggen van de data, namelijk met zogenaamde triples. Lees voor een vollediger begrip van deze systematiek <http://bp4mc2.org/>.

Het ELMO (Extendible lightweight model ontology) vocabulaire is tegelijk met het LDT tot stand gekomen als configuratie vocabulaire voor het LDT. In dit vocabulaire vinden we onderdelen waarmee we het framework kunnen laten reageren zoals wij dat willen; zie voor de volledige technische omschrijving: <https://github.com/architolk/Linked-Data-Theatre/tree/master/vocabulary>.

Daarin valt snel op dat er via SHACL ook restricties zijn opgelegd aan de objecten en classes van de ELMO-vocabulaire.

Om een dergelijke vocabulaire te maken bestaat er OWL, <https://www.w3.org/TR/owl-guide/>   
Daarin staat uitgelegd hoe men Classen kan maken en eigenschappen daaraan kan toewijzen.

# Datamodel

Het data model is tot stand gekomen door goed inzicht te krijgen in de verschillende onderdelen van de data die gerepresenteerd moet worden. Na het bestuderen van de data onderdelen is bepaald welke bestaande vocabulaires een exacte match zijn om te gebruiken. In de vocabulaires staan namelijk beschrijvingen van dataonderdelen en de vraag is dan ook altijd: welke omschrijving past het beste bij het stukje data dat we hier hebben?

De best practice in Linked data is dan ook om zoveel mogelijk vocabulaires te gebruiken die al bestaan zonder gebruik te maken van hele exotische vocabulaires die specifiek voor een applicatie zijn gemaakt. Beter is om bij het tegenkomen van een stuk data waar we geen vocabulaire bij kunnen vinden een eigen vocabulaire te maken die de missende stukken invult. Dat is ook wat erbij het IHW nodig was om het datamodel compleet te maken. Het nieuw gemaakte Aquo vocabulaire zal daarom ook worden besproken met daarbij de afwegingen die zijn gemaakt.

Naast het bepalen van de juiste vocabulaires is er ook een URI strategie nodig om de data goed op te slaan.

## Uri Strategie

Voor het maken van een datamodel in Linked data is het nodig om een strategie te bedenken voor het vastleggen van alle stukken data. Omdat alle data bij Linked data een URI is dienen we te bepalen hoe deze eruit gaat zien.   
  
Voor een basis begrip van de onderdelen van een URI kan hier meer informatie worden gevonden: <https://aandeslagmetdeomgevingswet.nl/digitaal-stelsel/documenten/documenten/api-uri-strategie/>

<https://en.wikipedia.org/wiki/Uniform_Resource_Identifier>

De juiste definities van een URI vinden we in de terug in het specificatie document van internet standaard RFC3986 (https://www.ietf.org/rfc/rfc3986.txt) .

De Nederlandse omschrijving die het Kadaster heeft gemaakt geeft een mooie samenvatting:

*“Een Uniform Resource Identifier (URI) is een gestandaardiseerde manier om bronnen van informatie (webpages, tekst, afbeeldingen, etc.) op het Internet te identificeren. Een URI identificeert een bron van informatie ("resource") aan de hand van een hiërarchische beschrijving (reeks tekens) die meestal een locatie op een netwerk representeert. URI’s zijn unieke verwijzingen naar digitale objecten. Identificatie aan de hand van een URI maakt interactie met verschillende bronnen van informatie via een netwerk (zoals het World Wide Web) mogelijk. De meest voorkomende vorm van een URI is het uniform resource locator (URL), vaak aangeduid als een webadres. URL en URN zijn beiden een specifieke vorm van een URI.”*

Voor het maken van de URI’s volgt het IHW de internet standaard RFC3986, dat betekend dat de basis opbouw van een URI er als volgt uitziet:

foo://example.com:8042/over/there?name=ferret#nose

\\_/ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_/ \\_\_/

| | | | |

scheme authority path query fragment

Een URI identificeert een bron van informatie aan de hand van deze hiërarchische beschrijving.

Voor het IHW bestaat de URI van een begrip uit onderstaand path. Deze URI is echter niet ook een URL; een via HTML benaderbaar object en we kunnen er dus niet via de browser heen navigeren.

http://standaard.aquo.nl/id/begrip/nummer

\\_/ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_/ \\_\_/

| | | | |

scheme authority path query fragment

Wel kunnen we via het LDT een URL aanroepen die de juiste data (en daarmee URI’s) kan tonen.

Dit is uitgebreid besproken in voorgaande hoofdstukken.

De volgende basispaden worden gebruikt voor het registreren van de data:

Voor het vastleggen van begrippen: <http://standaard.aquo.nl/id/begrip>

Voor het vastleggen van categorieën: <http://standaard.aquo.nl/id/category>

Voor het vastleggen van domeintabellen: <http://standaard.aquo.nl/id/dataset/domeintabel>

Voor het vastleggen van domeinmetadata: <http://standaard.aquo.nl/id/dataset/domeinmeta>

Voor het vastleggen van de publiher: <http://standaard.aquo.nl/id/publisher>

Voor het vast;eggen van het thema: <http://standaard.aquo.nl/id/thema>

## Vocabulaires

Om de dataset van IHW te beschrijven is er gekozen voor een combinatie van SKOS, SKOSTHES, RDFS, DC, DCTERMS, DCAT en ADMS. Daarnaast was het nodig om voor een paar onderdelen een eigen vocabulaire te ontwikkelen. Die wordt in het volgende hoofdstuk behandeld.

### SKOS

SKOS staat voor “Simple Knowledge organization system” en is een woordenboek voor het uitdrukken van de inhoud van concept schema’s en thesauri en daarmee uitermate geschikt voor het beschrijven van veel onderdelen van de dataset(s) van het IHW. Een overzicht van de Nederlandse definitie van de onderdelen is hier in rdf te vinden : <http://www.w3.org/2004/02/skos/core_nl>

En hier als html <https://www.w3.org/TR/swbp-skos-core-spec/>

Meer achtergrond over SKOS is hier te vinden: <https://www.w3.org/TR/skos-reference/>.

Ook bij het Kadaster is gekozen voor het SKOS vocabulaire en daarmee kunnen toekomstige koppelingen tussen begrippen zeer eenvoudig worden gerealiseerd. Voor de Aquo applicatie gebruiken we de volgende onderdelen van SKOS, de laatste bullet is steeds een specifieke uitleg voor de Aquo applicatie.

* Concept
  + Definitie: “Een abstract idee of notie”
  + Het skos:Concept wordt gebruikt om het (rdfs) type van een begrip mee aan te geven. Dit maakt een begrip een begrip; Wanneer we een pagina opvragen die informatie over een begrip bevat, is dit technisch gezien informatie over een skos:Concept.
* Broader
  + Definitie: “Een verwant concept met een algemenere betekenis”
  + De skos:Broader eigenschap is een inverse van de skos:Narrower eigenschap
  + De skos:Broader eigenschap wordt gebruikt om een relatie te leggen naar een ander begrip (skos:Concept) die een meer algemene betekenis heeft.
* Narrower
  + Definitie: “Een verwant begrip met een specifiekere betekenis”
  + De skos:Narrower eigenschap is een inverse van de skos:Broader eigenschap
  + De skos:Narrower eigenschap wordt gebruikt om een relatie te leggen naar een ander begrip (skos:Concept) die een meer specifieke betekenis heeft
* Preflabel
  + Definitie “Het geprefereerde lexicale label van een onderwerp in een bepaalde taal.”
  + Een prefLabel mag niet 2 keer (in dezelfde taal) voorkomen in hetzelfde conceptSchema
  + Het skos:prefLabel wordt gebruikt om een uniek label vast te leggen per begrip waarin de naam van het begrip staat uitgedrukt. In de Aquo applicatie is dit label altijd Nederlands, eventuele extra/optionele labels worden in de skos:altLabel vastgelegd.
* altLabel
  + Definitie: “Een alternatief label”
  + Acroniemen, afkortingen, afwijkende schrijfwijzen en onregelmatige enkelvouds/meervoudsvormen zijn mogelijk alternatieve labels van een concept. Verkeerd gespelde woorden worden meestal als verborgen labels aangegeven (zie skos:hiddenLabel).
  + Het skos:altLabel wordt gebruikt voor het vastleggen van labels in andere talen; Duits, Engels en Vlaams.
* Note
  + Definitie: “Een algemene notitie zonder specifiek gebruiksdoel.”
  + De skos:Note wordt gebruik voor het geven van een toelichting bij een begrip (skos:Concept).
* Definition
  + Definitie: “Een uitleg of formele definitie van een concept.”
  + De skos:definition wordt gebruikt om een formele definitie te geven van het begrip (skos:Concept).
* Related
  + Definitie: “heeft gerelateerd concept”
  + De skos:related eigenschap wordt gebruikt om een gerelateerde term aan te geven. Dit is een relatie die dus geen grootte of richting aangeeft maar gewoonweg zegt dat de begrippen in algemene zin gerelateerd zijn.
* RelatedMatch
  + Definitie “Heeft gerelateerd concept in ander schema”
  + De skos:relatedMatch word gebruikt om een relatie naar een begrip in een ander schema te leggen. Concreet wordt deze relatie gebruikt voor een verwijzing naar de GWSW dataset.
* SemanticRelation
  + Definitie: “Een in zijn betekenis verwant begrip.”
  + De skos:SemanticRelation wordt gebruikt om een verbinding te leggen tussen de begrippen en de onderdelen van de domeintabellen.
* Notation
  + Definitie: “A notation is a string of characters such as "T58.5" or "303.4833" used to uniquely identify a concept within the scope of a given concept scheme.”
  + Wat staat er in de skos:notation?? 🡪 niet uniek 🡪 data issue?

### SKOS-THES

SKOS-THES biedt een aanvulling op SKOS vocabulaire om aan veelvoorkomende eigenschappen van thesauri te kunnen voldoen (<https://www.w3.org/2004/02/skos/extensions/spec/2004-10-18.html>) . In de aquo applicatie lenen we 2 eigenschappen van deze toevoeging op het SKOS vocabulaire.

* narrowerPartitive
  + Definitie: “*An extension of the “narrower” property to specify a partitive (part of) relationship between two concepts.*”
  + De skosthes:narrowerPartitive eigenschap wordt toegekend wanneer het begrip waarnaar wordt verwezen een nauwer onderdeel is van het begrip waaraan deze eigenschap wordt toegekend.
* broaderPartitive
  + Definitie: “*An extension of the “broader” property to specify a partitive (part of) relationship between two concepts.*”
  + De skosthes:broaderPartitive eigenschap wordt toegekend wanneer het begrip waarnaar wordt verwezen een breder onderdeel is van het begrip waaraan deze eigenschap wordt toegekend.

### RDFS

RDFS staat voor RDF (Resource Description Framework) Schema. Het is een vocabulaire voor het modeleren van RDF data. Zie <https://www.w3.org/TR/rdf-schema/>.

* Type
  + Definitie:” rdf:type is an instance of [rdf:Property](https://www.w3.org/TR/rdf-schema/" \l "ch_property) that is used to state that a resource is an instance of a class.
  + Rdf:type wordt gebruikt om bijvoorbeeld aan te geven dat een begrip van het rdf:type (ook wel ‘a’ genoemd in de code) skos:Concept is.
* Label
  + Definitie: “rdfs:label is an instance of [rdf:Property](https://www.w3.org/TR/rdf-schema/" \l "ch_property) that may be used to provide a human-readable version of a resource's name.”
  + Wij gebruiken rdfs:label om alle begrippen, collecties en domeintabellen te voorzien van een label.
* Comment
  + Definitie: “rdfs:comment is an instance of [rdf:Property](https://www.w3.org/TR/rdf-schema/" \l "ch_property) that may be used to provide a human-readable description of a resource.”
  + De rdfs:comment dient als uitleg bij een begrip (skos:Concept).

### DC

Dublin Core (DC) is een standaard voor het beschrijven van content op het internet, als zodanig gebruiken wij. Lees meer over DC en DC-Terms hier <http://dublincore.org/documents/dcmi-terms/>

* Source
  + Definitie: “A related resource from which the described resource is derived.”
  + Geeft de bron van het begrip weer.

### DCTERMS

* Created
  + Definitie: “Date of creation of the resource.”
* Modified
  + Definitie: “Date on which the resource was changed.”
* Subject
  + Definitie: “The topic of the resource.”
  + Dct:Subject wordt gebruikt om een link te leggen naar de categorie van het begrip, dit kunnen er meer dan 1 tegelijk zijn.

### DCAT

De dcat vocabulaire richt zich op het laten samenwerken van verschillende catalogussen op het web. Lees er meer over op <https://www.w3.org/TR/vocab-dcat/>

* Theme
  + Definitie: “De hoofdcategorie van een dataset, een dataset kan meerdere themes hebben.
  + Bij het uploaden van domeintabellen registreren we de betreffende domeinmetadata-uri onder een dcat:theme.

### ADMS

Asset description metadata schema (ADMS) is ontwikkeld om semantische begrippen te kunnen beschrijven. Lees er meer over op <https://www.w3.org/TR/vocab-adms/>

* Status
  + Definitie: “De status van een ding in de context van een specifieke workflow”
  + Adms:Status wordt gebruikt om de juridische geldigheid van een begrip aan te duiden en omvat 2 waarden: ‘Ingediend’ en ‘Geldig’.

### Aquo vocabulaire

Toevoeging eigen vocabulaire: <http://localhost:8080/def/aquo>

* Status
  + De status die wordt toegekend tijdens het wijzigen, toevoegen of verwijderen van onderdelen van de domeintabel. De status is benodigd om een staging laag te kunnen realiseren waarin een laatste goedkeuring kan plaatsvinden van de wijzigingen.
* Abbreviation
  + De Aquo:abbreviation wordt gebruikt om (logische) afkortingen vast te leggen bij begrippen.
* englishLabel
  + Het Engelse label gedestilleerd vanuit het engelse skos:altLabel, @en.
* flamishLabel
  + Het Vlaamse label gedestilleerd vanuit het vlaamse skos:altLabel, @fl.
* germanLabel
  + Het Duitse label gedestilleerd vanuit het duitse skos:altLabel, @de.
* dutchLabel
  + Het Nederlandse label gedestilleerd vanuit het nederlandse skos:altLabel, @nl. Deze wordt gebruikt om een synoniem aan te geven binnen de begrippendetail pagina.
* Omschrijving
  + Geeft de omschrijving van een object uit de domeintabel.
* Cijfercode
  + Legt de unieke code van ieder object uit de domeintabel vast.
* Groep
  + **Nog geen toepassing voor?**
* CASnummer
  + Het CASnummer geeft een unieke identificatie voor chemische stoffen, getoond in de domeintabel.

## Model Validatie

Voor het valideren van de configuratiebestanden is gekozen voor SHACL.

SHACL biedt de mogelijkheid om vast te leggen aan welk patroon de configuratie bestanden moeten voldoen om binnen de -via SHACL ingevoerde- ontologie richtlijnen te blijven.

Het voor het Informatie Huis Water opgestelde SHACL model ligt vast in het bestand

bp4mc2-ihw\model\aquo.ttl

Configuratiebestanden die ingaan tegen dit vastgestelde model kunnen op verschillende – zelf te definiëren- manieren worden afgehandeld. Per SHACL onderdeel is er namelijk de mogelijkheid om de ernst van een eventuele mismatch aan te geven, en daarmee ook te bepalen hoe de verdere afhandeling gaat.

Lees alles over SHACL op: <https://www.w3.org/TR/shacl/>

Het SHACL model die voor de Aquo is samengesteld bestaat uit NodeShapes en PropertyShapes, te weten:

NoteShapes:

* ConceptShape
* DomeintabelShape

PropertyShapes:

* OmschrijvingShape
* PlaatjesShape
* BeginDatumGeldigheidShape
* EindDatumGeldigheidShape
* DatumLaatstePublicatieShape
* DomeinShape
* LabelShape
* NameShape
* DefinitieShape
* ToelichtingShape
* BronShape
* AfkortingShape
* CategorieShape
* BredereTermShape
* NauwereTermShape
* GerelateerdTermShape
* StatusShape

### Domeintabel validatie

In de bestaande aquo-ds applicatie vinden we de domeintabellen en de definities van de kolommen die bij iedere domeintabel hoort.

Voor iedere domeintabel zal er daarom een losse shacl definitie komen die bepaald welke waarden er in de domeintabel kunnen voorkomen en waaraan die waarden moeten voldoen.

Vervolgens kan er in de upload transaction een validation step worden aangemaakt welke de toegevoegde data tegen de domeinspecifieke shacl definitie valideert. Bij eventuele tekortkomingen of mismatches verschijnt er dan een melding welk stuk data er niet voldoet en waarom niet.

Er is hiervoor een voorbeeld bestand gemaakt voor de domeintabel “AfsluitmiddelType”.

Het uploaden van de shacl definitie gaat op eenzelfde manier als andere data bestanden; via een upload .bat bestand. De shacl definitie komt in haar eigen graph te staan; te weten <http://standaard.aquo.nl/id/dataset/shacl> .

## Keuzen domeintabel

Iedere domeintabel is een klasse. De onderdelen/inhoud binnen deze klasse zijn de properties van de klasse. Bij het uploaden wordt op basis van de bestandnaam de domeintabel/klasse aangemaakt, de inhoud komt tevens in een aparte graph. Bijvoorbeeld <http://standaard.aquo.nl/id/dataset/domeintabel/AfsluitmiddelType> .

VERDERE OMSCHRIJVING VAN DE KEUZEN HIERVOOR 🡪 Marco ?

# Beheren en Historie

DIT ONDERDEEL MOET EERST NOG VERDER WORDEN DOORONTWIKKELD IN DE APPLICATIE + UITLEG MARCO OVER EDITTORTRANSACTIE. TEVENS NOG WAT VERTELLEN OVER DE GLOBALE STAPPEN DIE GEMAAKT WORDEN BIJ EEN EDIT; AANROEPEN TRANASCTIE EN STEPS TOELICHTEN; PERSISTENCY STEPS ETC.

## UITLEG STAGING EN TRANSACTIE PER ONDERDEEL

UITLEG HISTORIEMODEL

## Functionaliteit

DIT ONDERDEEL PAS MAKEN ALS DE APPLICATIE HELEMAAL KLAAR IS

## Uploaden

UITLEG + SCREENSHOTS

## Navigeren

UITLEG + SCREENSHOTS

## Beheren

UITLEG + SCREENSHOTS

## Delen

UITLEG + SCREENSHOTS

SOAP EN API

* Beheer domeintabellen
* Beheer categorieën
* Beheer begrippen

# Nawoord

NIET VERGETEN:

* UITLEG SOAP SERVICE/ DOCUMENTATIE
* UITLEG API

Ordina implementeert strategie in bedrijfsprocessen en ICT. Wij bedenken, bouwen en beheren oplossingen voor een duurzame digitale wereld. Samenwerking met onze klanten, partners en leveranciers leidt tot de beste oplossingen. Dit doen wij vanuit gespecialiseerde divisies voor consulting, solutions en ICT. Ons werkterrein is de Benelux. Daarbinnen bieden we (herhaalbare) oplossingen en kennis voor de markten financiële dienstverlening, overheid, zorg en een aantal specifieke segmenten binnen de industriemarkt. Ordina N.V. is opgericht in 1973 en het aandeel is genoteerd aan NYSE Euronext Amsterdam. Het bedrijf maakt onderdeel uit van de Small Cap index.

**Ordina Consulting B.V.**

**Adres ,**

**Post ,**

**Telefoon**

**Fax**

**Email   
Internet**