

DHRD BIM-Validator

Technische Specificatie

Project 'Duurzame Herbruikbaarheid Ruimtelijke Data'

Versie: 0.2 – Second Draft
0.1 – Draft (21 Feb 2022)

Datum: 24 augustus 2022

Auteurs: **Pieter Pauwels, TU Eindhoven**
Jakob Beetz, RWTH Aachen
Chiel van der Pas, TU Eindhoven

Executive Summary

Dit document bevat de technische specificaties die werden opgemaakt binnen het project 'Duurzame Herbruikbaarheid Ruimtelijke Data (DHRD)' ter opmaak van een BIM-validator tool die het mogelijk moet maken om bestanden uit BIM-tools en BIM-processen duurzaam op te slaan en zo bruikbaar te houden voor later gebruik (beoogde termijn: 10-20 jaar).

Inhoudstafel

| | |
|---|----|
| Executive Summary | 2 |
| Inhoudstafel | 3 |
| 1. Inleiding..... | 4 |
| 1.1 Doelstelling | 4 |
| 1.2 Scope | 4 |
| 1.3 Termen en Definities..... | 4 |
| 1.3.1. Definities | 4 |
| 1.3.2. Afkortingen | 5 |
| 1.4 Referenties | 6 |
| 1.5 Overzicht | 6 |
| 2. Systeemarchitectuur | 7 |
| 2.1 Introductie | 7 |
| 2.2 Algemene opstelling | 8 |
| 3. Implementatiedetails | 10 |
| 3.1 Server backend..... | 10 |
| 3.1.1 Hosting | 10 |
| 3.1.2 Backend code | 11 |
| 3.1.3 Autorisatie..... | 14 |
| 3.1.4 File storage..... | 15 |
| 3.1.5 Triple databank | 16 |
| 3.2 Interfaces | 18 |
| 3.2.1 User Interface | 18 |
| 3.2.2 Programming Interface (API) | 24 |
| 3.3 De validatie-scripts..... | 27 |
| 3.2.1 Types controle..... | 27 |
| 3.2.2 Metadata-controle | 28 |
| 3.2.3 Inhoudelijke controle | 29 |
| 3.2.4 ICDD Cross-check | 29 |
| 4. Getting Started..... | 31 |

1. Inleiding

1.1 Doelstelling

In dit document worden de systeemarchitectuur en technische specificaties uiteengezet voor de DHRD BIM-Validator tool. Deze volgen in grote lijnen het User Requirements Document (URD) dat in de eerste fase van dit onderzoeksproject werd uitgewerkt (WP1). Op basis van verschillende workshops met stakeholders werd dit URD uitgeschreven om de exacte vereisten van de gebruiker op te stellen ten aanzien van de software. In het URD werden de gebruikerseisen (user requirements) voor de software gedocumenteerd.

Het document met technische specificaties documenteert de te ontwikkelen software in meer detail. De algemene architectuur wordt toegelicht en de specifieke onderdelen worden in meer detail besproken. Ter referentie wordt in de rest van deze inleiding de kernpunten van het URD herhaald.

1.2 Scope

De DHRD BIM-Validator is een validatietool die de herbruikbaarheid van opgeleverde data van gebouwen (i.c. BIM-modellen) kan controleren. Deze tool moet algemeen beschikbaar zijn, zodat die breed gebruikt kan worden, en ook onderdeel kan gemaakt worden van een aantal bestaande systemen. De uitkomst van deze tool is een oordeel in hoeverre het opgeleverde materiaal herbruikbaar is of niet (niveau van herbruikbaarheid).

De BIM-Validator is een softwaretool welke vrij verkrijgbaar is en welke door eenieder gebruikt kan worden om de duurzaamheid van een zelf aan te leveren BIM-project, in de vorm van een ZIP-folder met daarin minimaal een IFC-bestand en informatieleveringsspecificaties in PDF-vorm, te toetsen. De validator rapporteert, op basis van de aangeleverde IFC-bestand(en) een oordeel over de duurzame herbruikbaarheid van het aangeleverde BIM-project, maar oordeelt niet over of de duurzame herbruikbaarheid van een voldoende niveau is. De validator beoordeelt de duurzame herbruikbaarheid van de aangeleverd bestanden op basis van (1) metadata en (2) data in de aangeleverde bestanden. De validator levert enkel een herbruikbaarheidsrapport op basis van ingevoerde bestanden. Dit rapport hoort door de gebruiker opgeslagen te worden voor eigen gebruik, zodat bijvoorbeeld een opdrachtgever dit rapport kan doorgeven aan aannemers, en omgekeerd. Het rapport wordt echter niet opgeslagen op de server(s) – geen opslag van gebruikersgevoelige data.

De beoogde validatietool toetst of de export van een BIM-model uit een informatiesysteem de volledige informatie bevat: data (inhoud) en metadata (gegevens over de opgeslagen informatie). Het doel is om deze tool zelfstandig te gebruiken. De tool wordt binnen het project binnen een e-depot (archivering)omgeving getest. Gekozen wordt voor een open source applicatie die laagdrempelig beschikbaar is, zodat de gebruikersgroep in Nederland gestaag kan groeien. De validatie-tool wordt ingezet voor het toetsen van kerncriteria van een duurzame toegankelijk BIM-model:

- metagegevens,
- kwaliteit van informatie,
- standaardformaat,
- volledigheid,
- integriteit.

1.3 Termen en Definities

1.3.1. Definities

| Term | Definitie |
|-------------------------|--|
| As-built | Het gebouw of model zoals het daadwerkelijk is uitgevoerd. |
| As-designed | Het gebouw of model zoals het aan het eind van de ontwerpfase is uitgewerkt en zoals het bedoeld is om uitgevoerd te worden. |
| aspectmodel | Een BIM-model voor een specifiek onderdeel van het BIM-Project. Samengevoegd vormen verschillende aspectmodellen een gecoördineerd BIM-model. Synoniem: disciplinemodel. |
| BIM-model | Een digitaal model welke het virtuele ontwerp van een gebouw of bouwwerk bevat. Een BIM-model bestaat uit objecten, bestaande uit een geometrie en aanvullende informatie, welke samen een digitale kopie van een gebouw vormen. Afkorting voor Building Information Model. |
| BIM-Project | Het project dat digitaal opgeleverd wordt bij oplevering van een gebouwde asset. Dit project bestaat uit een ZIP-folder met daarin de digitale bestanden die opgeleverd worden (met minstens 1 IFC-bestand en 1 informatieleveringsspecificatie of ILS). |
| Duurzaam herbruikbaar | Data is duurzaam herbruikbaar wanneer het opgeslagen wordt op zo een manier dat het op een later moment (termijn: 10-20 jaar) gemakkelijk te benaderen en gebruiken is |
| Eigenaar | De partij welke eigenaar is van het BIM-Project welke aangeboden wordt ter validatie. Het eigendom van het BIM-Project kan mogelijks veranderen na verloop van tijd (e.g. overdracht) |
| Gebruiker | Een persoon die communiceert met de gebruikersinterface van de BIM-Validator om een BIM-Project te valideren. |
| Gecoördineerd BIM-model | Samenvoeging van verschillende aspectmodel |
| IFC | Afkorting voor Industry Foundation Classes. Een open standaard voor het maken en delen van BIM-modellen |
| ILS | Afkorting voor informatieleveringsspecificatie. Synoniem voor Information Delivery Manual (IDM) en Exchange Information Requirements (EIR). Een document waarin is vastgelegd welke informatie de opdrachtgever van een BIM-project verwacht te vinden in het BIM-project en hoe deze informatie wordt beschreven in de digitale bestanden. Kan machine-interpreteerbaar zijn (JSON of TTL). |
| Metadata | Data welke de karakteristieken van een bepaald bestand of dataobject beschrijven |
| Opdrachtgever | De persoon die opdracht geeft voor een BIM-Project |
| Oplevering | Het moment waarop de bestanden van het BIM-Project door de Eigenaar worden overgedragen aan de Opdrachtgever. Dit is tevens het moment van Validatie. Na Oplevering wordt de Opdrachtgever Eigenaar van het BIM-project. |
| User story | Hypothetisch scenario welke beschrijft waarom, hoe, en waarvoor een Gebruiker de voorgestelde BIM-Validator zou willen gebruiken. |
| BIM-Validator | De software die in dit project wordt uitgewerkt. |

1.3.2. Afkortingen

| Acroniem | Betekenis |
|----------|--|
| BIM | Building Information Model |
| DHRD | Duurzame Herbruikbaarheid Ruimtelijke Data |
| EIR | Exchange Information Requirements |

| | |
|------|---------------------------------|
| IDM | Information Delivery Manual |
| IFC | Industry Foundation Classes |
| ILS | Informatieleveringsspecificatie |
| JSON | Javascript Object Notation |
| TTL | Terse RDF Triple Language |

1.4 Referenties

-

1.5 Overzicht

In hoofdstuk 2 wordt de systeemarchitectuur in algemeen overzicht beschreven. Hoofdstuk 3 gaat in op de verschillende onderdelen uit de architectuur in meer detail.

2. Systeemarchitectuur

2.1 Introductie

De bouwsector, real estate sector, asset managementsector en infrastructuursector worden aan versnellend tempo gedigitaliseerd. Vandaag de dag wordt voor elk groot bouwproject een 3D BIM-model gemodelleerd door de verschillende partijen die deelnemen aan het bouwproject (ingenieursbureau, architect, aannemer, onderaannemer, productie, etc.). Meer zelfs, typisch wordt een hele set aan BIM-modellen gemodelleerd (aspectmodellen of disciplinmodellen) die samen één of meerdere gecoördineerde BIM-modellen vormen.

Met andere woorden, voor het ontwerpen en realiseren van een nieuw gebouw (e.g. een ziekenhuis) gaan verschillende mensen in team aan de slag in de opmaak van een gecoördineerd 3D BIM-model. Dit wordt idealiter gedaan met inachtnaam van een informatieleveringsspecificatie (ILS) die oplijst hoe de Opdrachtgever uiteindelijk de informatie verwacht te ontvangen. Elke partner in het project is hierbij modelleur en eigenaar van zijn specifieke onderdeel of aspectmodel (bijvoorbeeld een BIM-model met daarin enkel de HVAC-installaties). Net voor de start van het bouwproject worden de verschillende BIM-modellen samengevoegd tot een gecoördineerd model waarmee het bouwproject kan starten. Dit model wordt het 'as-designed' BIM-model genoemd. Idealiter wordt het BIM-model tijdens dit proces bijgewerkt volgens wat daadwerkelijk werd uitgevoerd, zodat aan het eind van het bouwproces een up-to-date 'as-built' BIM-model beschikbaar is.

Bij oplevering van het bouwproject worden de digitale bestanden (as-designed en as-built BIM-modellen) overgedragen aan de Opdrachtgever. Op dat moment gaat de Opdrachtgever na in hoeverre deze bestanden voldoen aan de vooraf opgegeven ILS. De digitale bestanden worden gebruikt als bewijsvoering van de opgeleverde opdracht en voor toekomstig beheer van het bouwwerk. Vervolgens worden deze data gebruikt om asset management systemen of facility management informatiesystemen (FMIS) te voeden met een afgeleide voorstelling van het gebouwde asset. Bijvoorbeeld wordt een SQL-databank gevoed met informatie van de ruimten in een gebouw zodat FM-software vervolgens gebruikt kan worden op basis van deze informatie. Er wordt zelden tot nooit een live link onderhouden met het originele opgeleverde as-built of as-designed BIM-model. Ook wordt het BIM-model zelden onderhouden overheen de tijd (in geval van renovaties, groot of klein). Wel wordt het BIM-model bewaard voor mogelijks later gebruik (= archivering).

In de praktijk zijn er echter geen richtlijnen of handvaten voor de duurzame toegankelijkheid van dergelijke BIM-data. Bijgevolg zijn er heel erg veel maten en gewichten: er worden Revit-bestanden bewaard die niet meer geopend kunnen worden in toekomstige versies van de software; er worden IFC-bestanden bewaard waar allerhande data in blijkt te ontbreken; er worden PDFs bewaard die niet voldoende machine-readable zijn; metadata ontbreekt waardoor data onbetrouwbaar wordt; etc. Daarom werden dergelijke *richtlijnen of handvaten voor duurzame toegankelijkheid van ruimtelijke data* ontwikkeld als onderdeel van WP1 in dit DHRD-project ontwikkeld.

In een tweede stap wordt een BIM-validator tool ontwikkeld die kan nagaan of een Oplevering van een BIM-Project voldoet aan de gestelde richtlijnen. Deze BIM-validator tool wordt in onderstaand document in meer detail beschreven vanuit IT-technisch oogpunt. *Deze BIM-validator kan nagaan of opgeleverde data voldoet aan voldoende kwaliteitseisen voor een duurzame herbruikbaarheid. Het doel van het product is dat het nagaat of data die ingeleverd wordt, nadien duurzaam herbruikbaar is.*

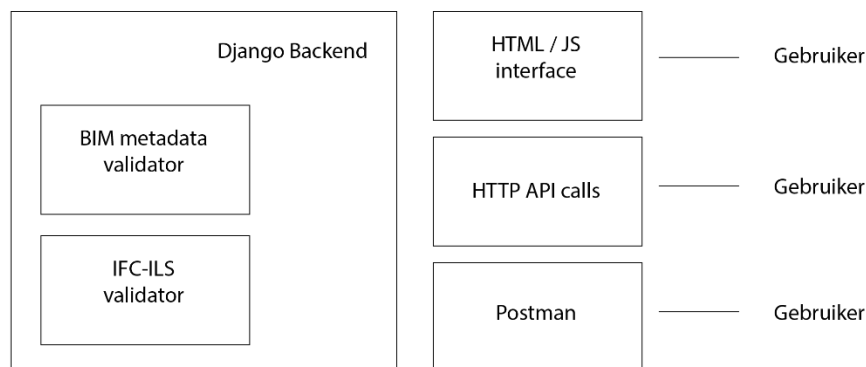
De tool is gericht op een aantal user stories en stakeholders / gebruikers. De meest vooraanstaande gebruikers op lange termijn zijn:

1. Asset manager
2. Burger
3. Ontwerper openbare ruimte
4. Hoofdaannemer
5. Wetenschappelijk onderzoeker

De verwachtingen voor de BIM-Validator tool liggen vooral in het validatiescript. Een script moet beschikbaar zijn dat op verschillende plaatsen (softwareproducten) kan ingebed worden, zodat de validatie vroeg kan uitgevoerd worden door een Gebruiker voor een snelle controle, maar ook laat, voor de eigenlijke validatie door de Opdrachtgever die zal archiveren. De meest belangrijke component is dus de eigenlijk controle-functionaliteit. Controle moet hierbij zowel gebeuren op de data en de metadata. De gebruikersinterface is minder belangrijk, gezien het script ingebed hoort te worden in verschillende externe interfaces. Modulariteit, generaliteit en herbruikbaarheid in externe software is dus wel heel erg belangrijk.

2.2 Algemene opstelling

Gezien bovenstaande context wordt de BIM-Validator als een web-gebaseerde service ontwikkeld, met bijhorende architectuur en opstelling. Dit resulteert in de algemene architectuur zoals geschetst in Figuur 1, overgenomen uit de URD.



Figuur 1 - Systeemarchitectuur voor de BIM Validator tool.

De BIM-Validatortool wordt uitgewerkt in een Python-script. Het bestaat uit een aantal delen (Fig. 1).

- Web service in Django:
 - o Een web service wordt ontwikkeld in Python Django. Op deze manier wordt een backend ontwikkeld die in staat is om bestanden (ZIP-folder, IFC-file, JSON-file, PDF-file) te ontvangen en deze te verwerken op een server (in de cloud of lokaal). De backend biedt dus een infrastructuur aan voor bestandscommunicatie (HTTP POST/GET) en rapportage (RETURN).
 - o De functionaliteit van de Django backend bestaat uit een Application Programming Interface (API) die beschikbaar is, en gedocumenteerd is adhv. swagger of vergelijkbaar. De verschillende API-functies laten verschillende validaties toe (metadata-check, data-check, IFC-check, etc.) en is dus modulair opgebouwd.
 - o Er wordt geen databank voorzien in de backend. Logging en rapportage gebeurt aan de hand van lokale bestandopslag op de server (.log).
 - o De Django backend is aanspreekbaar via Postman en de browser. Er wordt een minimale browserinterface voorzien in HTML + Javascript, maar die is enkel nuttig

voor demonstratiedoeleinden. Het doel is dat de backend wordt ingebed of gebruikt in andere tools.

- File validators in losstaande Python-scripts:
 - Metadata-validator: een alleenstaand Pythonscript wordt ontwikkeld die toelaat de verschillende metadata te evalueren en valideren. Dit Pythonscript wordt ingeladen door de Django backend, maar is in principe een alleenstaande softwarebibliotheek.
 - BIM-validator: een alleenstaand Pythonscript wordt ontwikkeld die de IFC-bestanden kan valideren (1) op geldigheid (correctheid), en (2) op conformiteit met een machine-leesbaar ILS.

3. Implementatiedetails

Meer specifieke details van de verschillende componenten worden in dit hoofdstuk indicatief toegelicht. Gezien de software nog niet geïmplementeerd is, zijn dit vooral intenties en algemene structuur. Deze implementatiedetails zullen zeker wijzigen gedurende het eigenlijke implementatietraject.

3.1 Server backend

Het eerste onderdeel in de systeem-architectuur is de server waarop de BIM-Validator draait. Deze wordt opgebouwd als een Django web framework, met zowel een development server (localhost) als een development server voor het uitvoeren van tests (DigitalOcean Ubuntu Virtual Machine). Op de server worden functionaliteiten aangeboden volgens de vastgelegde vereisten (URD), wat betreft autorisatie, opladen van bestanden, metadata-controle, controle IFC-ILS.

3.1.1 Hosting

Voor de web hosting wordt een virtuele machine (VM) aangemaakt op DigitalOcean als web host. Dit is een tijdelijke locatie voor dit project die toelaat om de software uit te testen (development server).

De VM op DigitalOcean heeft volgende specificaties:

- OS: Ubuntu 20.04 LTS x64
- 1 CPU
- 1 GB RAM
- 25 GB SSD storage space
- 1000GB transfer

Duurdere mogelijkheden zijn ook beschikbaar indien nodig, maar naar verwachting niet nodig voor een development server.

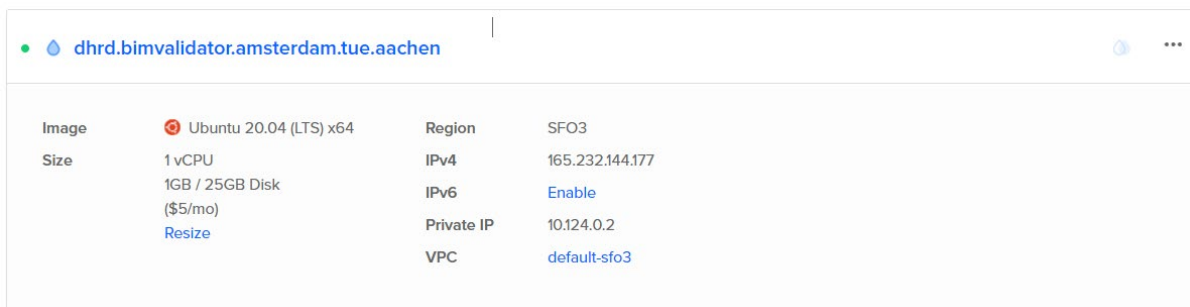
| | | | | | |
|--|---|--|--|---|--|
| CPU options: <input checked="" type="radio"/> Regular Intel with SSD <input type="radio"/> Premium Intel with NVMe SSD NEW <input type="radio"/> Premium AMD with NVMe SSD NEW | | | | | |
| \$5/mo \$0.007/hour | \$10/mo \$0.015/hour | \$15/mo \$0.022/hour | \$20/mo \$0.030/hour | \$40/mo \$0.060/hour | \$80/mo \$0.119/hour |
| 1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer | 2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer | 2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer | 4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer | 8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer | 16 GB / 8 CPUs 320 GB SSD Disk 6 TB transfer |

Figuur 2: Alternatieve VM-mogelijkheden binnen Digital Ocean.

De server is toegankelijk via de nodige SSH-keys. In dit geval wordt 1 SSH key login aangemaakt voor de server-administratie, op naam van Pieter Pauwels, TU Eindhoven.

De server heeft volgende gegevens:

- Servernaam: dhrd.bimvalidator.amsterdam.tue.aachen
- IP-adres: 165.232.144.177



| | | | |
|-------|---|------------|------------------------------|
| Image | Ubuntu 20.04 (LTS) x64 | Region | SFO3 |
| Size | 1 vCPU 1GB / 25GB Disk (\$5/mo) Resize | IPv4 | 165.232.144.177 |
| | | IPv6 | Enable |
| | | Private IP | 10.124.0.2 |
| | | VPC | default-sfo3 |

Figuur 3: Specificaties van DHRD BIMValidator server.

Bijkomend aan deze web hosting worden ook nog volgende zaken geïmplementeerd:

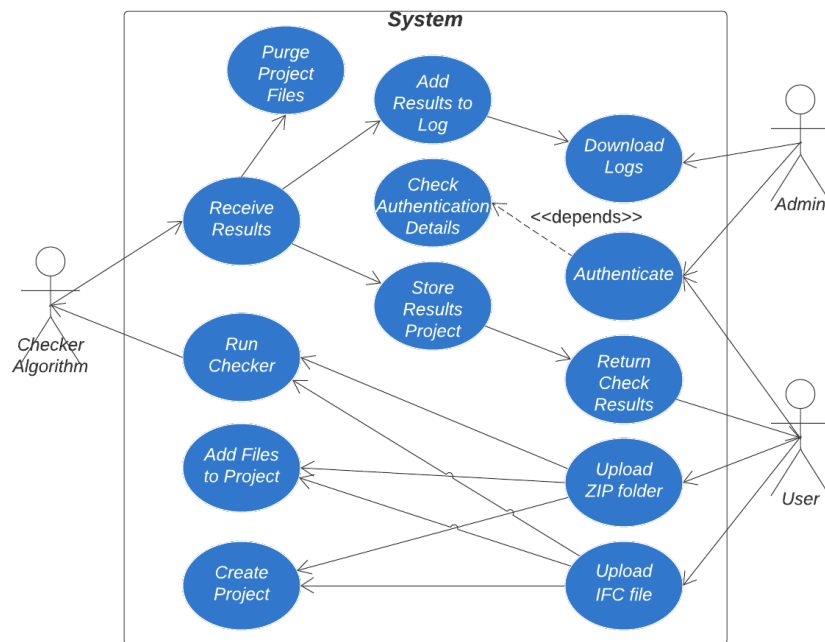
- koppeling van IP-adres aan leesbare domeinnaam:
<https://www.linkedbuildingdata.net/dhrd/BIMValidator/>
- toevoeging van een SSL-certificaat voor beveiligde HTTPS toegang.

3.1.2 Backend code

Op de server wordt een Python Django framework geïnstalleerd. Deze code wordt op GitHub bijgehouden (version control + collaboration). Deze code is enkel bedoeld om de algemene functionaliteit van de server te implementeren (login, nieuw project, opladen van bestanden, controleren van bestanden, rapporteren). De code wordt als een standaard Python project uitgewerkt:

- aanmaak en activering van virtuele omgeving
- inladen van dependencies:
 - o pip install django
 - o pip install djangorestframework
 - o pip install rdflib
 - o pip install pyyaml uritemplate
 - o pip install pywin32
 - o pip install requests
 - o pip install xmlschema
 - o pip install SPARQLWrapper
- manueel installeren van ifcopenshell in Python Virtual Environment
- manueel refereren naar IFctoLBD.jar
- uitwerken code volgens use case diagram

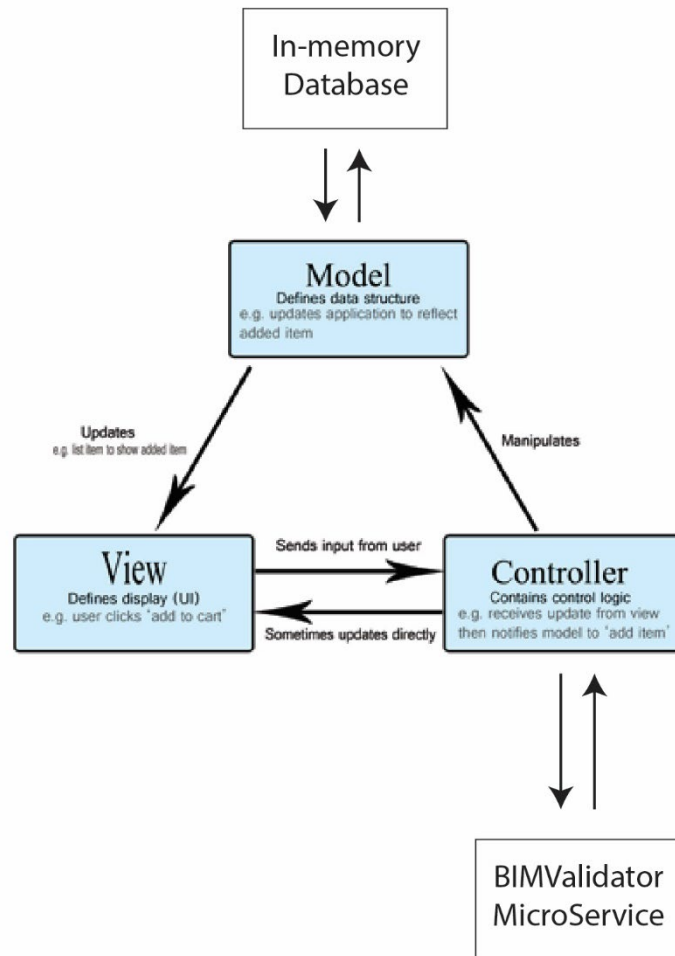
Voor deze code wordt het UML use case diagram gehanteerd uit Figuur 4. De individuele use cases worden beschreven in de URD.



Figuur 4: UML Use Case Diagram for the server backend.

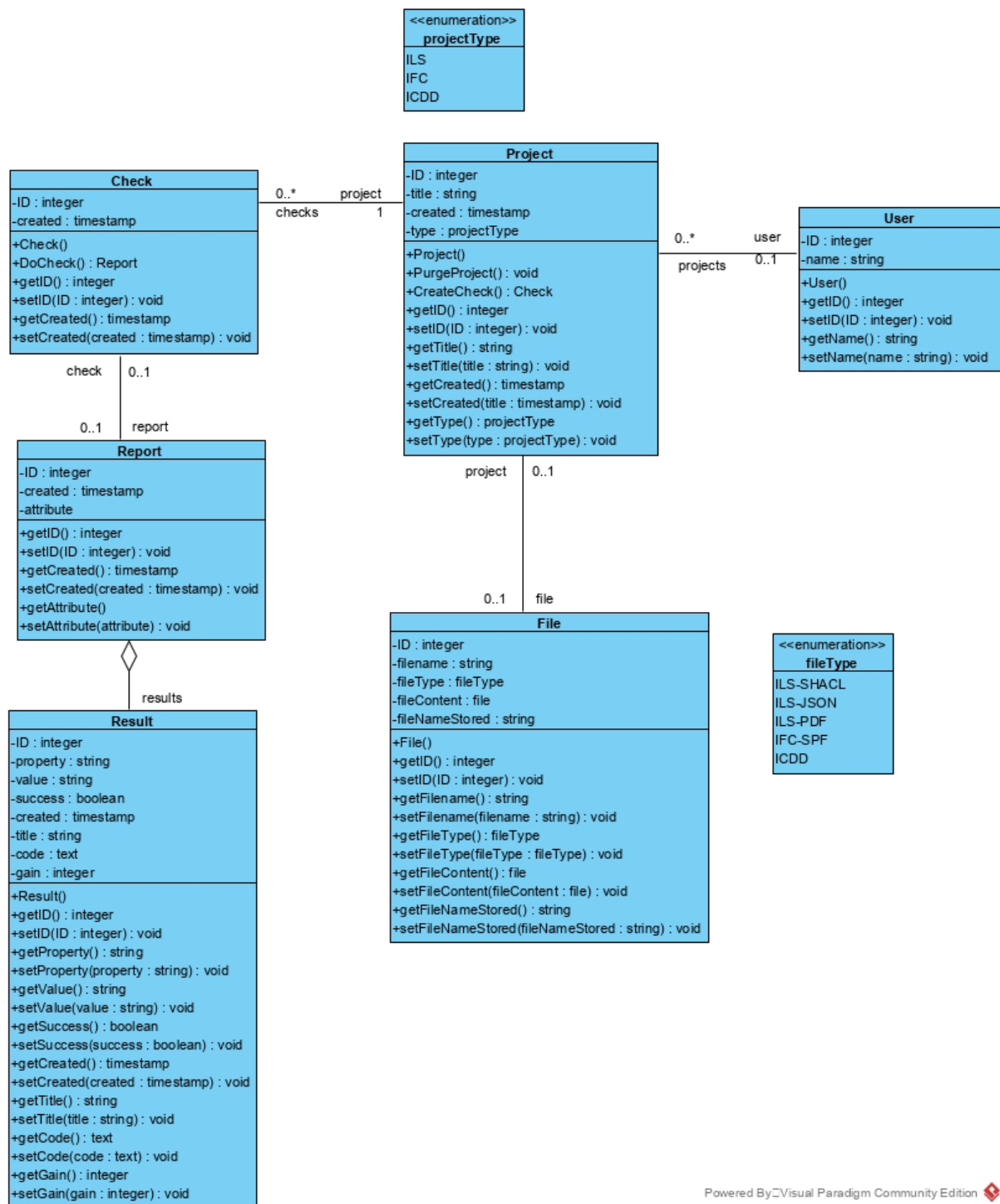
Als onderdeel van de code worden models, views, en tests geïmplementeerd, resulterend in een doorsnee Model-View-Controller (MVC) architectuur (Fig. 5):

- Model: overzicht van de datastructuur van de server (project, gebruiker, validatie, etc.), volgens UML Class Diagram.
- View: collectie aan pagina's die toegang geven tot het model.
- Controller: de router die informatie van View naar model doorgeeft, en omgekeerd. Deze is ook verantwoordelijk voor het aanroepen van de validatiescripts (zie later).



Figuur 5: MVC architectuur, met bovenaan het Model die rechtstreeks in interactie staat met de lokale databank; en rechtsonder de controller die interactie aanstuurt met zowel de interface als met de microservice die de eigenlijke controle of BIMValidatie doet.

Bijkomend aan deze MVC-architectuur wordt een databank in-memory geïmplementeerd door middel van een eenvoudige sqlite databank (relationele databank in SQL). Zowel de databank als het Model volgen de datastructuur zoals weergegeven in het UML Class Diagram in Figuur 6.



Figuur 6: UML Class Diagram for the server backend.

3.1.3 Autorisatie

Hoewel autorisatie niet noodzakelijk is, wordt dit toch geïmplementeerd dmv standaard Django-functionaliteit¹. Echter, de bedoeling van dit project is dat de validator open kan uitgevoerd worden, zonder opslag van informatie (uitgezonderd interne logs en statistieken zonder gebruikersinformatie). Enkel een minimale mogelijkheid tot registratie en inloggen wordt voorzien.

¹ <https://docs.djangoproject.com/en/4.1/topics/auth/default/>

Er zijn twee gevallen waarin autorisatie wenselijk is:

- De use case “Download logs” door de administrator heeft nood aan een authenticatie-mechanisme voor het downloaden van logs van de server. Als voorlopige back-up kan dit ook gerealiseerd worden door rechtstreekse toegang tot de server.
- Extra beveiliging is noodzakelijk om gebruik van de server te beperken tot die personen die expliciet toegang vragen (access key). Zo’n beveiliging beschermt de server van een overload.

In de testfase is enkel één admin-gebruiker handmatig geregistreerd:

```
python manage.py createsuperuser --email admin@dhrd.nl --username admin
```

Listing 1: Aanmaak van username en paswoord.

Het paswoord voor deze *admin*-gebruiker kan zelf ingesteld worden bij opstart en wordt hier ingesteld op ‘dhrdamsterdam’. Registratie van bijkomende gebruikers is niet geïmplementeerd in deze fase.

3.1.4 File storage

De interface laat toe om bestanden op te laden. Verschillende soorten bestanden kunnen opgeladen worden via de interface, namelijk:

- .icdd-bestanden (volgens de ICDD specificatie²)
- .ifc-bestanden
- .shacl-bestanden (ILS)
- .pdf-bestanden (ILS)
- .xml bestanden (ILS)
- .json bestanden (ILS)

De bestanden worden opgeladen via de interface, en komen terecht in de folder ‘uploads’. Hierbij wordt gebruik gemaakt van standaard Django-functionaliteit. Het opladen naar het lokaal bestandssysteem gebeurt in *models.py*, waar aangegeven wordt in welke folder de bestanden moeten komen (Listing 2). De inhoud van opgeladen bestanden, net als de oorspronkelijke bestandsnaam en de opgeslagen bestandsnaam (die wordt automatisch uniek gemaakt), wordt allemaal opgeslagen in het Django model en in de bijbehorende sqlite databank.

```
class File(models.Model):
    fileName = models.CharField(max_length=100, blank=True, default='')
    fileContent = models.FileField(upload_to='uploads', default='')
    fileNameStored = models.CharField(max_length=100, blank=True, default='')
```

Listing 2: Opslag van bestanden in databank en file system.

Om bestanden te kunnen downloaden of te converteren of te kunnen checken, wordt telkens gebruik gemaakt van deze uploads-folder, bv zoals weergegeven in Listing 3:

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
fileIFC = BASE_DIR + '\\..\\uploads\\' + fileName
fileTTL = os.path.splitext(fileIFC)[0]+'\\.ttl'
```

Listing 3: Ophalen van bestanden uit file system.

² <https://www.iso.org/obp/ui/#iso:std:iso:21597:-1:ed-1:v1:en>

3.1.5 Triple databank

De opgeladen IFC-bestanden worden standaard geconverteerd naar RDF triples, waarna deze in een lokale graph databank van Ontotext GraphDB worden opgeladen. De conversie gebeurt door middel van een 'executable JAR' (uitvoerbare JAVA code) die het IFC-bestand omzet in standaard LBD triples:

```
jarpath = "C:\\\" + "IFCtoLBD-0.1-shaded.jar"
os.system("java -jar \"\" + jarpath + "\" \"\" + fileIFC + "\" \"\" + fileTTL +
"\"")
```

Listing 4: Conversie van IFC-bestanden naar RDF triples via IFC-to-LBD convertor.

Deze conversiecode resulteert in een TTL-bestand dat rechtstreeks weggeschreven wordt in de uploads-folder (Fig. 7).

| | | | | | |
|---|--|---|--------------|----------|------|
|  | 7m900_tue_hello_wall_with_door_LKjQ6He.ifc |  | 07/07/202... | IFC File | 9 KB |
|  | 7m900_tue_hello_wall_with_door_LKjQ6He.ttl |  | 07/07/202... | TTL File | 3 KB |
|  | 7m900_tue_hello_wall_with_door_mVJexdd.ifc |  | 08/07/202... | IFC File | 9 KB |
|  | 7m900_tue_hello_wall_with_door_mVJexdd.ttl |  | 08/07/202... | TTL File | 3 KB |
|  | 7m900_tue_hello_wall_with_door_t3zAo1y.ifc |  | 07/07/202... | IFC File | 9 KB |
|  | 7m900_tue_hello_wall_with_door_t3zAo1y.ttl |  | 07/07/202... | TTL File | 3 KB |
|  | 7m900_tue_hello_wall_with_door_UPv19Sn.ifc |  | 07/07/202... | IFC File | 9 KB |
|  | 7m900_tue_hello_wall_with_door_UPv19Sn.ttl |  | 07/07/202... | TTL File | 3 KB |

Figuur 7: UML Class Diagram for the server backend.

Bovendien worden deze triples ook ingevoerd in een lokale graphDB graph databank (Listing 5).

```
## Load into GraphDB
headers = {
    'Content-Type': 'application/x-turtle',
    'Accept': 'application/json'
}
with open(fileTTL, 'rb') as f:
    requests.post("http://localhost:7200/repositories/DHRD/statements",
data=f, headers=headers)
```

Listing 5: Opladen van RDF triples in lokale triple store.

De locatie van deze triple store wordt bijgehouden in settings.py:

```
GRAPHDB_BIN = "C:/Users/20194060/AppData/Local/GraphDB Free/app/bin/"
SPARQL_ENDPOINT_1 = 'http://localhost:7200/repositories/DHRD'
SPARQL_ENDPOINT_2 = 'http://localhost:7200/repositories/DHRD/statements'
DEFAULT_NAMESPACE = 'https://dhrd.nl/amsterdamrepo/'
ORGANIZATION_DEFAULT_NAMESPACE = 'https://dhrd.nl/amsterdamOrg#'
CURRENT_ORG = 'https://dhrd.nl/amsterdamOrg#fictional_test_org'
```

Listing 6: Instellingen van de RDF triple store (GraphDB – settings.py).

Na opladen kunnen de data dus via het Database Management Systeem (DBMS) van GraphDB ook geconsulteerd worden. Een voorbeeld hiervan is hieronder zichtbaar met een van de testbestanden (TUE Vertigo – Fig. 8).

GraphDB

FREE

Import

Explore

Graphs overview

Class hierarchy

Class relationships

Visual graph

Similarity

SPARQL

Monitor

Setup

Help

DHRD

Vertigo Building

Source: http://linkedbuildingdata.net/ifo/resources20220707_231514/building_29

subject

predicate

Explicit only

Show Blank Nodes

Download as

Visual graph

object

context

all

| | subject | predicate | object | context |
|---|----------------------------------|-------------------------------|--|---|
| 1 | inst:building_29 | rdf:type | bot:Building | http://www.ontotext.com/explicit |
| 2 | inst:building_29 | rdfs:comment | TU/e Department of the Built Environment | http://www.ontotext.com/explicit |
| 3 | inst:building_29 | rdfs:label | Vertigo Building | http://www.ontotext.com/explicit |
| 4 | inst:building_29 | bot:hasGuid | 3ca7e585-4e3e-4969-a86f-f049f4fbde52 | http://www.ontotext.com/explicit |
| 5 | inst:building_29 | bot:hasStorey | inst:storey_35 | http://www.ontotext.com/explicit |

Figuur 8: RDF triples voor TUE Vertigo gebouw in OntoText GraphDB DBMS.

3.2 Interfaces

3.2.1 User Interface

De BIMValidator volgt een MVC-architectuur, zoals aangegeven in Figuur 5, waarbij de View gerealiseerd wordt door een HTML-interface. Hier wordt een standaard template gebruikt. Een draft interface wordt ontworpen aan de hand van Figma³. De resulterende interface wordt omgezet in code en beschikbaar gemaakt op dezelfde server. Dit bestaat uit volgende bestanden:

1. Een basistemplate base.html
2. Specifieke HTML-pagina's met bijbehorende code:
 - a. checks.html
 - b. newproject.html
 - c. projects.html
 - d. reports.html
 - e. settings.html

De verschillende HTML-pagina's zijn telkens gebaseerd op de basistemplate. In elke HTML-pagina wordt input vanwege views.py weergegeven. Dit volgt standaard Django-logica, zoals kort weergegeven in Listing 7.

```
{% if user.is_authenticated %}
    <br />
    {% if projectId is None %}
    <p>For project {{ projectId }}, the following checks are available:</p>
    {% if checks %}
        <p>Checks overview here.</p>
    {% else %}
        <p>No checks available in project.</p>
    {% endif %}
    <button type="submit"><a href="{% url 'newcheck' projectId=projectId %}">Create a
new check</a></button>
    <button type="submit"><a href="{% url 'projects' %}">Go to Projects</a></button>

    {% endif %}
{% else %}
    <p>You are not logged in</p>
    <a href="{% url 'login' %}">Log In</a>
{% endif %}
```

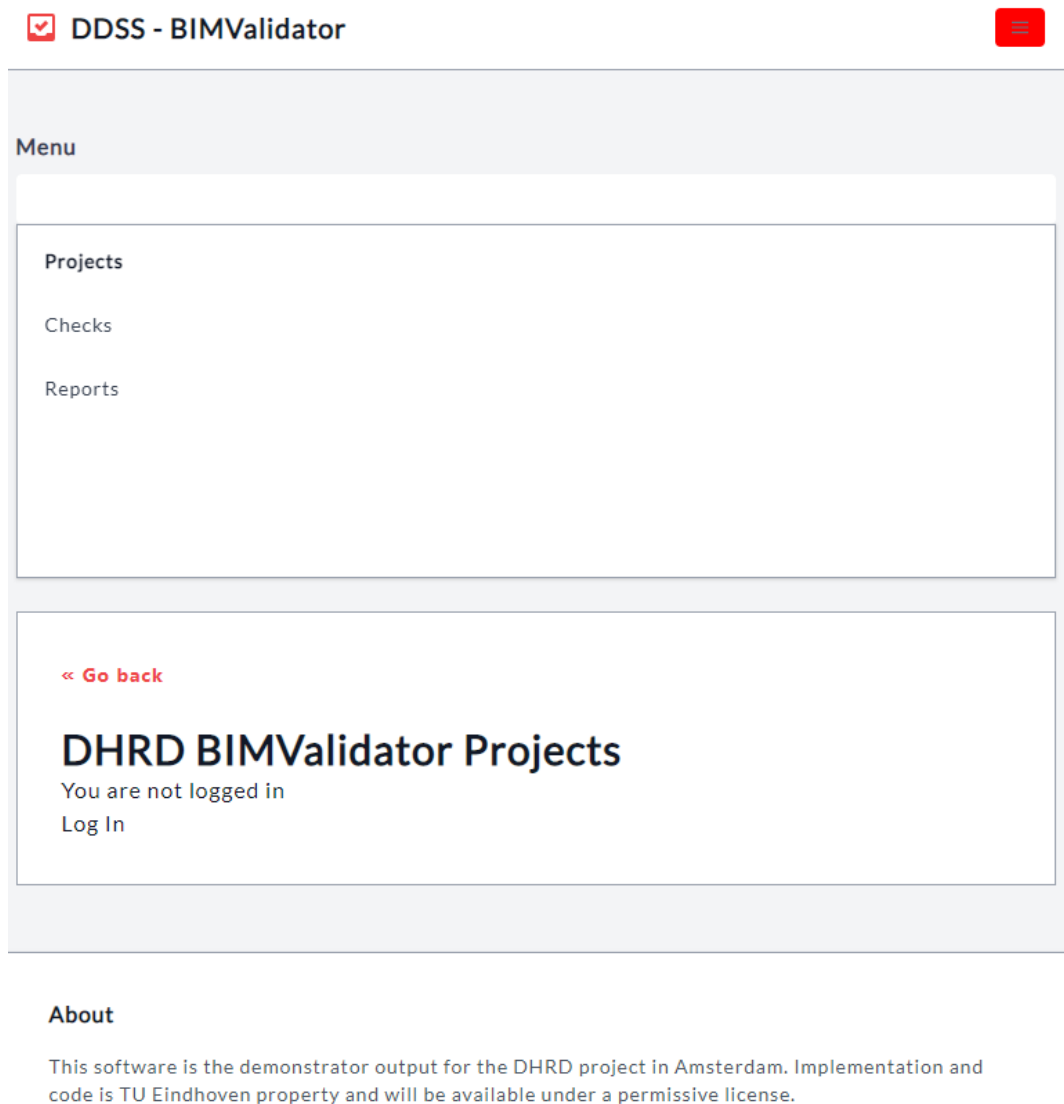
Listing 7: Invoer van model parameters (views.py) in de HTML interface (projects.html).

De logica van de interface volgt de verschillende use cases die in het URD werden voorgesteld en opgelijst. Dit wordt hieronder kort gedocumenteerd.

³ <https://www.figma.com/>

Authenticatie en autorisatie

Bij het opstarten van het platform in de browser, komt de gebruiker terecht op een algemene inlogpagina (Figuur 9). Via de inlogknop kan de gebruiker inloggen met haar paswoord.

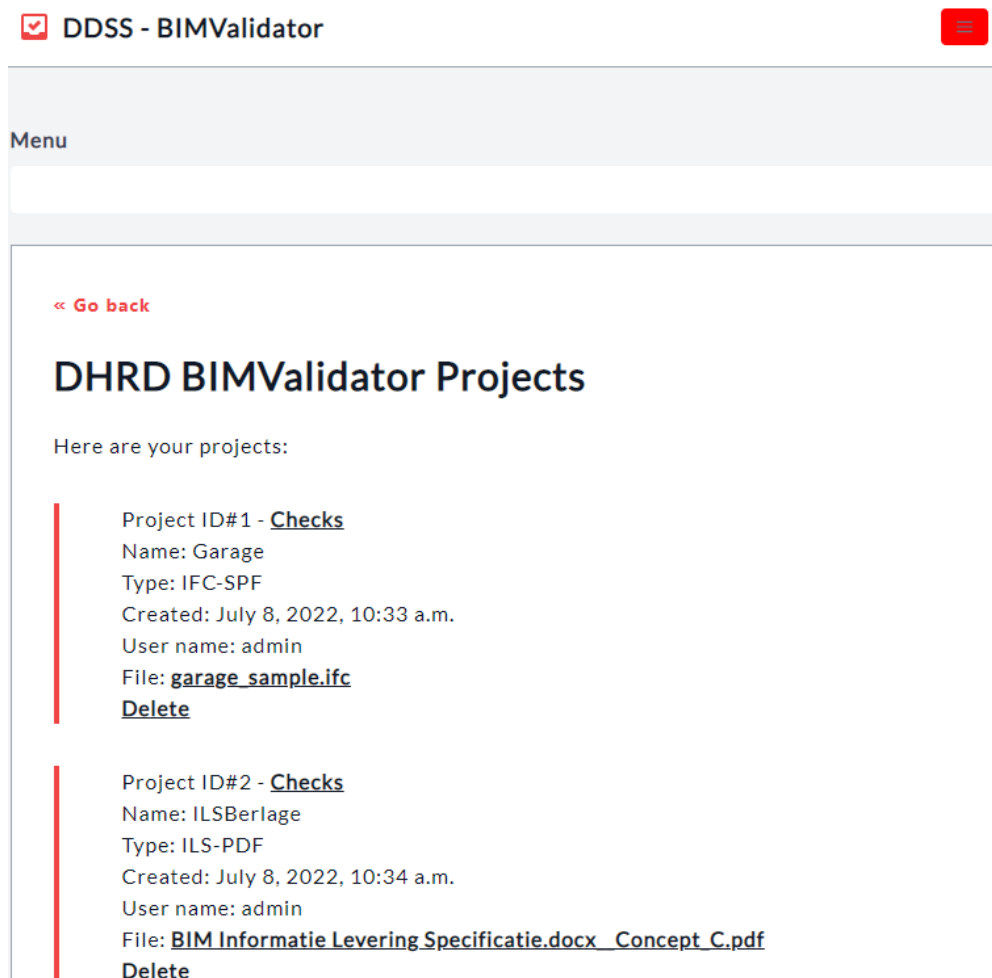


Figuur 9: Startpagina met inlogfunctionaliteit.

Projectenoverzicht

Na succesvolle aanmelding komt de gebruiker terecht op een pagina met projecten die voor haar beschikbaar zijn. In deze projectoverzichtspagina wordt voor elke pagina een hoeveelheid details weergegeven, volgens het model van het project: ID, naam, type, datum, gebruiker, en geassocieerd bestand (Figuur 10). Op deze overzichtspagina kan de gebruiker navigeren naar:

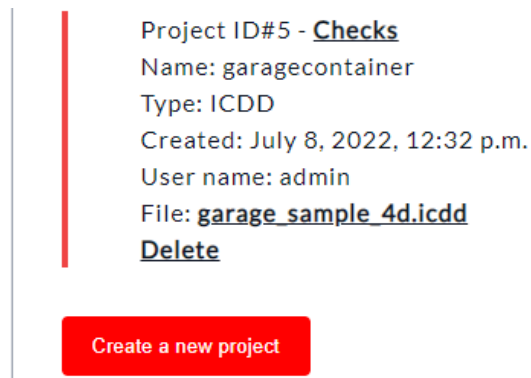
- Checks: dit zijn alle controles die op dit project werden uitgevoerd.
- File: het bestand kan rechtstreeks gedownload worden
- Delete: het project kan uit de databank verwijderd worden, inclusief alle geassocieerde bestanden (CASCADE = True).



Figuur 10: HTML pagina met overzicht van projecten.

Nieuw project

Onderaan de overzichtspagina kan met een knop een Nieuw project aangemaakt worden (Fig. 11).

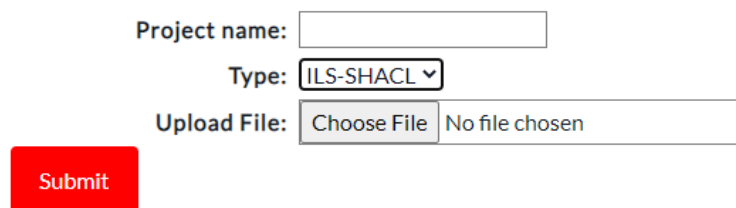


Project ID#5 - Checks
Name: garagecontainer
Type: ICDD
Created: July 8, 2022, 12:32 p.m.
User name: admin
File: garage_sample_4d.icdd
Delete

Create a new project

Figuur 11: Mogelijkheid tot aanmaak van nieuwe projecten.

Het aanmaken van een nieuw project gebeurt met een eenvoudig formulier waarin een naam wordt gegeven, een type-project (keuzes: ILS-SHACL, ILS-JSON, ILS-PDF, IFC-SPF, ICDD), en een bestand (Fig. 12). Na opladen van deze invoer is een nieuw project beschikbaar waarvoor automatisch een controle of check kan gestart worden.



Project name:

Type:

Upload File:

Submit

Figuur 12: Formulier voor opladen van bestanden in een project.

Checks

Per project kan een aantal checks (controles) uitgevoerd worden, afhankelijk van het type bestand. Deze checks worden opgeslagen en blijven consulteerbaar. Checks kunnen ook verwijderd worden uit de databank. Bij elke check zijn een aantal metadata beschikbaar, inclusief één of meerdere rapporten (IFC rapport, ILS rapport, ICDD rapport – zie Figuur 13).

DHRD BIMValidator Checks

For project 1, the following checks are available:

Check ID#1

Created at: July 8, 2022, 10:33 a.m.

Project ID#1

[Delete](#)

[> IFC Report](#)

[> ILS Report](#)

[> ICDD Report](#)

Check ID#4

Created at: July 8, 2022, 11:32 a.m.

Project ID#1

[Delete](#)

[> IFC Report](#)

[> ILS Report](#)

[> ICDD Report](#)

Check ID#5

Created at: July 8, 2022, 11:34 a.m.

Project ID#1

[Delete](#)

[> IFC Report](#)

[> ILS Report](#)

[> ICDD Report](#)

[Create a new check](#)

[Go to Projects](#)

Figuur 13: UML Class Diagram for the server backend.

Elk rapport is apart consulteerbaar. Checks kunnen gewoon verwijderd worden. Het aanmaken van een nieuwe check gebeurt met een druk op de knop en genereert een nieuwe check voor hetzelfde project. Let op, als het bestand identiek is, zal de check ook identiek zijn.

Dit is voor uitbreiding en verbetering vatbaar, maar voorlopig wordt dit beperkt gehouden: 1 bestand per project, en meerdere identieke checks per bestand – project. Indien een nieuw bestand moet gecontroleerd worden, moet een nieuw project aangemaakt worden.

Report

Per check is 1 of meerdere projecten beschikbaar, afhankelijk van het type project.

- Voor een IFC-SPF project kan enkel een IFC rapport geleverd worden.
- Voor een ILS project kan enkel een ILS rapport geleverd worden.
- Voor een ICDD project kunnen IFC, ILS, en ICDD rapporten geleverd worden.

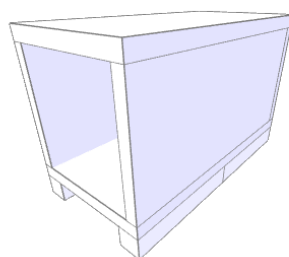
Een rapport bestaat uit:

- Enkele interne metadata: ID, type, datum, project identifier
- Een 3D view in het geval van een IFC of ICDD project
- Een content view (e.g. text viewer voor ILS-SHACL) in het geval van een ILS project
- Een metadata checkrapport:
 - o Hier wordt een lijst aan resultaten (Result) getoond voor specifieke controles (bv. correct bestandstype, aanwezigheid van auteur in metadata, etc.)
 - o Voor elk resultaat wordt een score gegeven die wordt opgeteld en aangeeft hoe hoog het duurzaamheidskarakter van het bestand is.
- Een bestandsinhoud checkrapport
 - o Hier wordt een lijst aan resultaten (Result) getoond voor specifieke controles (bv. correct bestandstype, aanwezigheid van auteur in metadata, etc.)
 - o Voor elk resultaat wordt een score gegeven die wordt opgeteld en aangeeft hoe hoog het duurzaamheidskarakter van het bestand is.

Een voorbeeld rapport wordt getoond in Figuur 14. Elk rapport kan verwijderd of gedownload worden in PDF.

DHRD BIMValidator Reports

Report ID# 1
Type: IFC-SPF
Created at: July 8, 2022, 10:33 a.m.
Project ID# 1
[Delete](#)



Metadata check

Check: Filetype
Value: application/Ifc
Success: True
Score: 0

File Content check

Figuur 14: Rapport voor een check van een IFC-SPF project.

In elk rapport wordt met kleur aangegeven of een controleresultaat een fouten of verbeterpunt (rood, groen – Figuur 15). Ook wordt voor elk resultaat een beschrijving gegeven van de uitgevoerde controle, inclusief het resultaat/waarde van de controle.

| |
|--|
| Check: File name in metadata tags Check: The file name is given inside the IFC file header (FILE_NAME()) Value: 0001 Success: True Score: 1 out of 1 |
| Check: Time Stamp Check: The time of creation of the IFC file is given inside the IFC file header (FILE_NAME()) Value: 2011-09-07T12:28:29 Success: True Score: 2 out of 2 |
| Check: Author Check: The author name is given inside the IFC file header (FILE_NAME()) Value: Success: False Score: 0 out of 2 |
| Check: Authorization Check: The authoriser's name is given inside the IFC file header (FILE_NAME()) Value: Success: False Score: 0 out of 2 |
| Check: Organization Check: The organization name is given inside the IFC file header (FILE_NAME()) Value: Success: False Score: 0 out of 1 |

Figuur 15: Weergave van controleresultaten per resultaat.

Elk resultaat heeft een gewicht die wordt aangegeven met een score ('gain' in models.py). Voor elk rapport wordt een totaalscore gegeven, zowel voor de metadatacontrole als de inhoudelijke controle (zie Fig. 16).

Report ID# 25
Type: IFC-SPF
Created at: Aug. 24, 2022, 9:18 a.m.
Metadata score 17 out of 24
Content score 17 out of 17
Project ID# 11
[Delete](#)

Figuur 16: Weergave van controleresultaten per rapport.

De inhoud van de controle wordt toegelicht in Sectie 3.3.

3.2.2 Programming Interface (API)

In aanvulling tot de Graphical User Interface (GUI) wordt ook een Application Programming Interface (API) opgezet die rechtstreeks toegang geeft tot de meest essentiële functionaliteit van de BIMValidator tool, namelijk de validatie en controle.

De API zelf is deel van de Python code (zie views.py en Sectie 3.1.2). Specifieke methodes uit de backend worden open gesteld en beschikbaar gemaakt voor ontwikkelaars. Dit gebeurt door

annotations die de interface koppelen aan het model van de backend code. Onderstaand voorbeeld toont (Listing) hoe via de API een methode 'getProjects' wordt opengesteld. De API is makkelijk uitbreidbaar en voorlopig enkel uitgewerkt met basisfunctionaliteit.

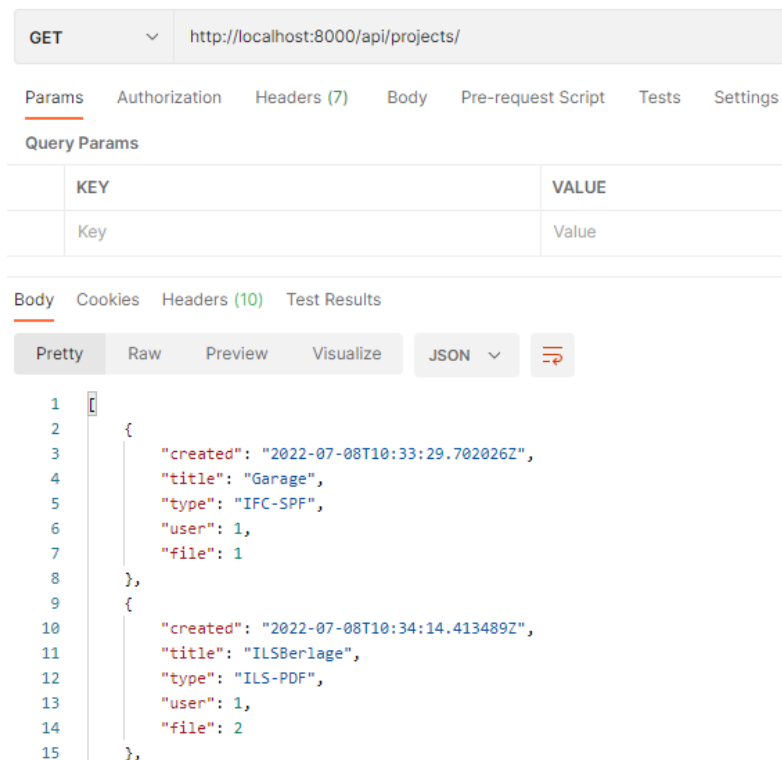
```
class api:

    @api_view(['GET'])
    def getProjects(request):
        """
        List all projects.
        """

        #if request.method == 'GET':
        projects = Project.objects.all()
        serializer = ProjectSerializer(projects, many=True)
        return JsonResponse(serializer.data, safe=False)
```

Listing 8: Definitie van API methodes (views.py).

Op deze manier kan met de software gewerkt worden via GET en POST requests zonder gebruik te moeten maken van de user interface. In Figuur 17 wordt weergegeven hoe de data van beschikbare projecten kunnen opgevraagd worden in JSON-formaat.



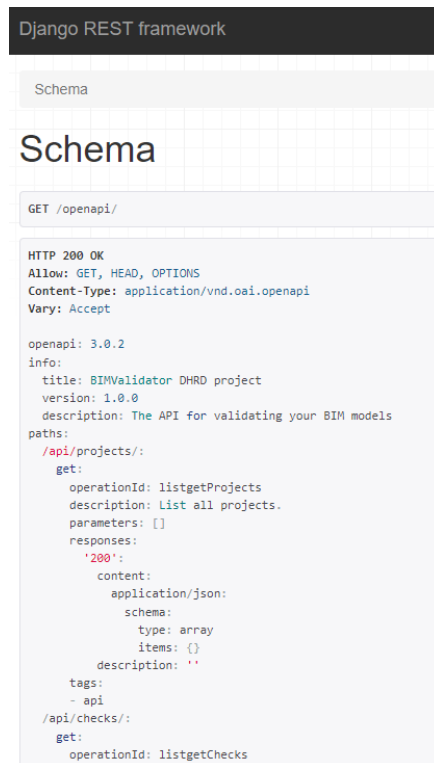
The screenshot shows a web browser interface for testing API requests. The method is set to GET and the URL is http://localhost:8000/api/projects/. The response is displayed in JSON format, showing a list of two projects.

| KEY | VALUE |
|-----|-------|
| Key | Value |

```
1 [
2   {
3     "created": "2022-07-08T10:33:29.702026Z",
4     "title": "Garage",
5     "type": "IFC-SPF",
6     "user": 1,
7     "file": 1
8   },
9   {
10    "created": "2022-07-08T10:34:14.413489Z",
11    "title": "ILSBerlage",
12    "type": "ILS-PDF",
13    "user": 1,
14    "file": 2
15  },
16 ]
```

Figuur 17: Voorbeeld API GET request voor het opvragen van beschikbare projecten.

In aanvulling wordt de documentatie van de API beschikbaar gemaakt via een OpenAPI specificatie (Fig. 18) en een Swagger UI (Fig. 19).



Figuur 18: OpenAPI interface voor de BIMValidator API.



Figuur 19: Swagger interface voor de BIMValidator API (gemaakt op basis van de live OpenAPI spec).

3.3 De validatie-scripts

3.2.1 Types controle

Bovenstaande onderdelen documenteren de algemene structuur van de web server plus de verschillende interfaces en hoe deze functioneren. Via deze tooling is het mogelijk om een project aan te maken van het type ICDD, IFC-SPF, of ILS. Dit zijn drie verschillende controles die worden mogelijk gemaakt met de software, zoals aangegeven in het URD.

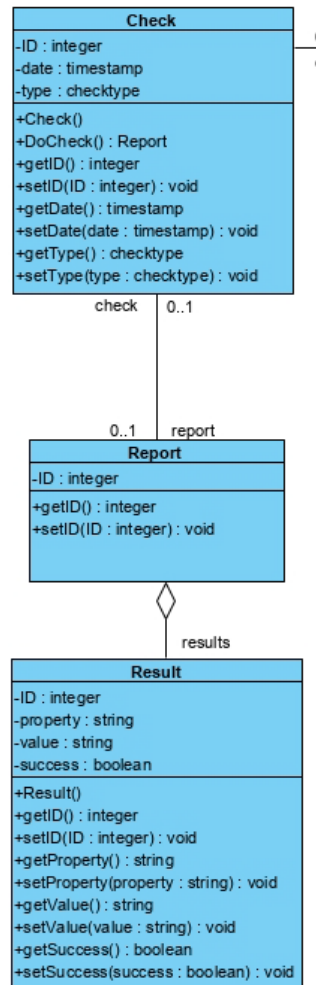
Voor deze 3 types projecten gelden volgende verwachtingen:

- Een IFC-Project:
 - o in een IFC-project wordt enkel een validatie aangeboden van een IFC-bestand
 - o in een IFC-project wordt één enkel bestand opgeladen, beperkt tot bestandstype ifc-spf (.ifc)
 - o als een nieuw IFC-bestand wordt opgeladen in hetzelfde project, dan wordt het voorgaande IFC-bestand overschreven.
 - o in deze controle kan enkel metadata en bestandsvaliditeit gecontroleerd worden.
- Een ILS-Project:
 - o in een ILS-project wordt enkel een validatie aangeboden van een ILS (informatieleveringsspecificatie)
 - o in een IFC-project wordt één enkel bestand opgeladen, van het bestandstype .json, .xml, .pdf, of .shacl
 - o Voor een PDF-file kan weinig gevalideerd worden, behalve heel algemene metadata.
 - o Voor de machine-readable versies van de ILS (JSON, XML, SHACL) kan metadata en bestandsvaliditeit worden gevalideerd.
- Een ICDD-Project:
 - o in een ICDD-project wordt een ZIP-folder verwacht met een .icdd extensie
 - o het bestand met .icdd extensie hoort de ICDD-standaard⁴ te volgen (= controle / check)
 - o in een ICDD-project wordt één enkel bestand opgeladen dat op de server uitgepakt wordt in verschillende bestanden
 - o het resultaat zijn verschillende files die gerelateerd zijn aan één ICDD-Project in de Django backend
 - o Voor een ICDD-project kan ook een ILS-check en een IFC-check uitgevoerd worden. Dit zijn aparte controles die de individuele IFC- en ILS-bestanden in de ICDD-container controleren.
 - o Voor een ICDD-project kan bovendien ook een ICDD-check uitgevoerd worden. In deze controle wordt nagegaan in hoeverre het IFC-bestand inhoudelijk voldoet aan het ILS-bestand (= crosscheck).

Elk van deze controles wordt uitgevoerd als volgt:

1. één Check-object wordt aangemaakt per Project (bovenaan Fig. 20)
2. één Report-object wordt aangemaakt per Check (midden Fig. 20)
3. Bij de aanmaak van het Report worden verschillende validatorscripts uitgevoerd, afhankelijk van het type project (IFC-SPF check, ILS-PDF check, ILS-SHACL check, ILS-JSON check, ILS-XML check, ICDD check).

⁴ <https://www.iso.org/obp/ui/#iso:std:iso:21597:-1:ed-1:v1:en>



Figuur 20: Ketting van Check – Report – Results.

Uitvoering van de check resulteert in een rapport (Report) dat tijdelijk wordt opgeslagen en bestaat uit een aantal resultaten (Results). Een Result object bestaat uit een naam, code, beschrijving, property, value, success, gain, en total.

3.2.2 Metadata-controle

Uitvoering van de metadata-controle kan gebeuren op IFC-files, ILS-files, en ICDD-files (ZIP). De check wordt telkens voor slechts 1 van deze files tegelijk uitgevoerd. Telkens wordt het te controleren bestand geëvalueerd volgens een lijst met voorgedefinieerde metadata-vereisten (zie URD).

Onderstaande lijst geeft een indicatie welke types metadata-controles uitgevoerd moeten worden voor een IFC-bestand. Dit werd opgesteld op basis van de resultaten in WP1.

Eis

- Versie-indicator
- Indicatie langdurige toegang
- Periode digitale beschikbaarheid
- Auteursgegevens
- Vocabularia
- Bestandsformaat
- Beoogd doelpubliek voor hergebruik

Personen met toegang
Definitie toegangs- en/of gebruiksbeperkingen
Markering van privacy-gevoelige bestanden
Indicatie soort data en soort detailniveau
Aanwezigheid ILS
Aanwezigheid leeswijzer
Auteur
Datum gecreeerd
Status van data
Hoe is data gecreërd
Datum laatst gewijzigd
...

Deze controle gebeurt door het parsen van het bestand en het controleren van de waarden in de header (volgens de specificatie voor de header⁵). Ook de metadata van het bestand zelf worden gecontroleerd. Voor de ILS- en ICDD-checks gebeurt een gelijkaardige controle. Het resultaat wordt opgeslagen in een Report-object met geassocieerde Result-objecten die in feite adhv. key-value pairs het resultaat van de controle aangeven (zie Result in Fig. 20).

3.2.3 Inhoudelijke controle

Voor de inhoudelijke controle van ILS en IFC worden de bestanden gedeserialiseerd en ingelezen, waarna een controle van het bestand wordt uitgevoerd. Dit is in eerste instantie een controle van de geldigheid en kwaliteit van het bestand.

- Voor IFC-bestanden: kwaliteitscontrole wordt geïmplementeerd adhv. IfcOpenShell
- Voor ILS-bestanden:
 - o In het geval een PDF wordt opgeladen, kan geen controle uitgevoerd worden.
 - o In het geval een JSON wordt opgeladen, wordt deze JSON-structuur gecontroleerd tov. een bijhorend JSON-schema.

Het resultaat wordt opgeslagen in een Report object met geassocieerde Result-objecten die in feite adhv. key-value pairs het resultaat van de controle aangeven (zie Result in Fig. 20).

3.2.4 ICDD Cross-check

Tot slot is het ook mogelijk om een ICDD cross-check uit te voeren. Daar wordt de inhoud van de ILS vergeleken met de data in het IFC-bestand, om na te gaan in hoeverre het IFC-bestand beantwoordt aan de vereisten van de ILS. Deze controle bestaat uit volgende stappen:

- Metadata-check ICDD: zie Section 3.2.2
- Uitpakken van ZIP-container (=ICDD)
- Metadata-check ILS
- Metadata-check IFC
- Controle in hoeverre de inhoud van het IFC-bestand overeenkomt met wat gevraagd wordt in de ILS

5

https://standards.buildingsmart.org/documents/Implementation/ImplementationGuide_IFCHeaderData_Version_1.0.2.pdf

Het resultaat wordt opgeslagen in een Report object in JSON-formaat, met een reeks key-value properties die het resultaat van de controle aangeven (zie Result in Fig. 20).

4. Validators

In dit hoofdstuk worden meer details opgenomen van de controles die worden uitgevoerd. Dit wordt beschreven per type project.

4.1 IFC-SPF project

Voor een IFC-SPF project wordt een metadata-controle en een inhoudelijke controle uitgevoerd. Dit gebeurt aan de hand van IfcOpenShell. Een project krijgt dus een score op beide aspecten (metadata, inhoud). Een IFC-bestand wordt ingelezen en specifieke karakteristieken worden op deze manier gecontroleerd. Hierbij wordt vooral getracht om de waarden binnen in de metadata-header van het IFC-bestand te controleren (Listing 9).

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [CoordinationView]'),'2;1');
FILE_NAME('0001','2011-09-07T12:28:29',(''),(''),'Autodesk Revit Architecture 2011
- 1.0','20100326_1700','');
FILE_SCHEMA(('IFC2X3'));
ENDSEC;
DATA;
#1=IFCORGANIZATION($,'Autodesk Revit Architecture 2011',$,$,$);
#2=IFCAPPLICATION(#1,'2011','Autodesk Revit Architecture 2011','Revit');
#4=IFCCARTESIANPOINT((0.,0.));
```

Listing 9: Voorbeeld van header in IFC file.

Referentie voor deze controle is de implementatiegids voor deze header⁶ zoals opgegeven door BuildingSMART International (Fig. 21).

2 IFC Header Data Definition Guide

The following example gives an overview on how the header section should be populated:

| FILE DESCRIPTION | Example | Explanation |
|----------------------|--|--|
| description | ViewDefinition[CoordinationView] | formal definition of the underlying view definition |
| implementation_level | 2;1 | see ISO10303-21, currently always 2;1 for IFC files |
| FILE NAME | Example | Explanation |
| name | C:\IAI_Test_Building.ifc | local path and file name of the IFC export file |
| time_stamp | 2007-02-06T10:28:37 | time of creation of the IFC export file |
| author | Andreas Geiger, Andreas.Geiger@iai.fzk.de | user defined field to capture the author/creator of the IFC file |
| organization | Forschungszentrum Karlsruhe | user defined field to capture the organization of the author |
| preprocessor_version | ECCO Toolkit Version V 3.2.1 | name of the toolbox used to create the IFC file |
| originating_system | IfcExplorer Version 2.2a (Build 437) | name of the application (including built number) |
| authorization | Karl-Heinz Haefele | user defined field to capture the authorizer of the IFC file |
| FILE_SCHEMA | Example | Explanation |
| schema_identifiers | IFC2X3 | name of the IFC schema |

Figuur 21: Requirements en verwachtingen voor de IFC-SPF header.

6

https://standards.buildingsmart.org/documents/Implementation/ImplementationGuide_IFCHeaderData_Version_1.0.2.pdf

Dit leidt tot de controles in Tabel 1. Hierin zijn ook de gewichten en beschrijvingen neergeschreven. Dit is geïmplementeerd in views.py. Deze controle wordt geïnitieerd telkens een nieuw rapport wordt opgevraagd. In totaal worden 50 punten gecontroleerd (33 punten metadata; 17 punten inhoud).

| name | code | score | description |
|---|----------|-----------|--|
| Filetype | metadata | 1 | Check if there is a file type available (MIME type, file extension) |
| Filetype Match | metadata | 2 | Check if the file extension matches with the MIME type and therefore is correct |
| IFC schema | metadata | 4 | The IFC schema is included inside the IFC file header |
| Correct IFC schema | metadata | 4 | An existing and published IFC schema is included inside the IFC file header |
| File name in metadata tags | metadata | 1 | The file name is given inside the IFC file header (FILE_NAME()) |
| Correct file name in metadata tags | metadata | 3 | The file name inside the IFC file header matches with the actual file name |
| Time Stamp | metadata | 2 | The time of creation of the IFC file is given inside the IFC file header (FILE_NAME()) |
| Time Stamp Match | metadata | 4 | The time of creation of the IFC file in the IFC file header matches with time of creation of the file (file settings) |
| No modifications to Original | metadata | 2 | The time of creation of the IFC file in the IFC file header matches with time of last modification of the file (file settings) |
| Author | metadata | 2 | The author name is given inside the IFC file header (FILE_NAME()) |
| Authorization | metadata | 2 | The authoriser's name is given inside the IFC file header (FILE_NAME()) |
| Organization | metadata | 1 | The organization name is given inside the IFC file header (FILE_NAME()) |
| Originating system | metadata | 2 | The name of the originating system is given inside the IFC file header (FILE_NAME()) |
| Creator Toolbox Used | metadata | 3 | The name of the toolbox used to create the IFC file are given inside the IFC file header (FILE_NAME()) |
| View Definition | content | 4 | The underlying view definition is named (Coordination View, Reference View, etc.) |
| Implementation Level of View Definition | content | 3 | The implementation level of the view definition is given (e.g. 2;1) |
| Number of BOT sites | content | 4 | BOT sites are present in the file |
| Number of BOT buildings | content | 3 | BOT buildings are present in the file |
| Number of BOT buildingstoreys | content | 1 | BOT building storeys are present in the file |
| Number of BOT elements | content | 2 | BOT elements are present in the file |
| total | | 50 | |

4.2 ILS project

Naast de controle van een IFC-SPF bestand is het mogelijk om een Informatieleveringsspecificatie (ILS) te controleren. In veel gevallen is enkel een PDF-bestand beschikbaar. De controle van zo'n PDF bestand is heel kort en weinig relevant. Behalve een vaststelling dat het inderdaad gaat om een PDF-bestand kan hier weinig automatisch aan gecontroleerd worden.

Daarnaast is in het BIMValidator project een plaats voorzien om machine-readable ILS-specificaties op te laden (XML, JSON, SHACL). Die bestaan helaas niet in een standaard of uitgewerkte versies. Na zoekwerk werden twee types ILS-bestanden gevonden die machineleesbaar zijn, namelijk een ids.XML en een constraints.shacl bestand. De kwaliteit van die bestanden worden in dit project niet gecontroleerd, maar onderstaand worden deze type bestanden wel beschreven. Dit kan dienen voor verdere voorbeelden en ontwikkeling.

4.2.1 ILS-XML check

In Listing 10 wordt een voorbeeld XML-bestand gegeven. Hierin wordt getoond hoe in zo'n XML-bestand meerdere eisen ('requirements') kunnen gebundeld worden in specificaties ('specifications'). De verschillende eisen worden afgetoetst met de relevante entiteiten ('applicable entities') in het IFC-bestand; in dit geval elke 'IfcWallStandardCase'. In het voorbeeld in Listing X wordt vastgelegd dat alle muren in een IFC-bestand de eigenschap 'isExternal' moeten hebben, én dat deze eigenschap de waarde True of False moet hebben.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- IDS (INFORMATION DELIVERY SPECIFICATION) CREATED USING IFCOPENSHELL -->
<ids xmlns="http://standards.buildingsmart.org/IDS"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://standards.buildingsmart.org/IDS/ids_04.xsd">
  <info>
    <title>DUTO BIM DHDR toets voor BIM modellen</title>
    <date>2022-08-18</date>
  </info>
  <specifications>
    <specification ifcVersion="IFC2X3" name="DUTO BIM specifications for
IfcWall">
      <applicability>
        <entity>
          <name>
            <simpleValue>IfcWallStandardcase</simpleValue>
          </name>
        </entity>
      </applicability>
      <requirements>
        <property>
          <propertySet>
            <simpleValue>Pset_WallCommon</simpleValue>
          </propertySet>
          <name>
            <simpleValue>IsExternal</simpleValue>
          </name>
        </property>
      </requirements>
    </specification>
  </specifications>
</ids>
```

```

    </name>
    <value>
      <xs:restriction base="xs:string">
        <xs:enumeration value="T" />
        <xs:enumeration value="F" />
        <xs:enumeration value="IFCBOOLEAN(.F.)" />
        <xs:enumeration value="True" />
        <xs:enumeration value="False" />
        <xs:enumeration value="true" />
        <xs:enumeration value="false" />
      </xs:restriction>
    </value>
  </property>
</requirements>
</specification>
</specifications>
</ids>

```

Listing 10: Voorbeeld van een ILS-XML bestand.

4.2.2 ILS-SHACL check

In Listing 11 wordt een voorbeeld SHACL-bestand gegeven. Hierin wordt één regel (rule:ElementLinkedToMinOneTask) vastgelegd die gecontroleerd kan worden op de inhoud van een ICDD-container. Meer specifiek wordt hier vastgelegd dat elk gebouwelement, hier gecodeerd als 'bot:Element', gekoppeld moet zijn minstens 1 taak, hier gecodeerd als 'cto:Task'.

```

rule:ElementLinkedToMinOneTask
  a sh:NodeShape ;
  sh:targetClass bot:Element ;
  sh:property [
    sh:path [sh:alternativePath (icddl:linkedDirectedBinary
icddl:linkedDirectedBinaryInverse icddl:linkedDirected)];
    sh:minCount 1 ;
    sh:class cto:Task ;
    sh:message "Each building element must be linked to (at least) one
task."
  ]

```

Listing 11: Voorbeeld van een SHACL constraint.

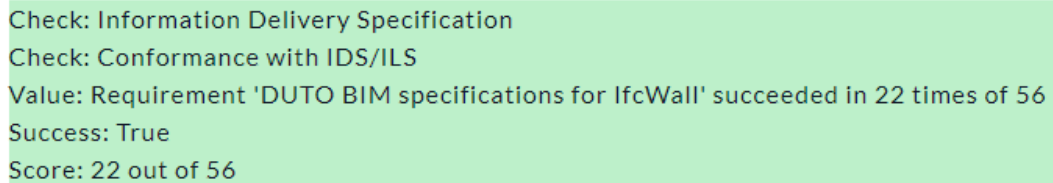
4.3 ICDD project

Het controleren van een individueel IFC-SPF bestand of een individueel ILS-bestand (Sections 4.1 en 4.2) kan nuttig zijn voor het nagaan van de kwaliteit van die individuele bestanden. Het meest interessant is echter de controle waarbij een IFC-bestand tegen een ILS-bestand wordt gecontroleerd. Dit wordt gecontroleerd in het ICDD-project. Het is namelijk zo dat de ICDD-standaard verwacht dat een ICDD-container wordt opgeleverd, waar dan typisch zowel het IFC-bestand als het bijhorende ILS-bestand in kunnen teruggevonden worden. De cross-check van IFC en ILS worden dus in het IFC-project geïmplementeerd.

4.3.1 ILS-XML check

In deze controle wordt een IFC-bestand gevalideerd aan de hand van het ILS-XML-bestand dat weergegeven wordt in Listing 10. In Figuur 22 wordt het resultaat van deze controle weergegeven zoals die in de interface wordt gevisualiseerd. Hierin is zichtbaar dat in het opgeladen IFC-bestand 56 elementen werden gecontroleerd. Er werden met andere woorden 56 elementen gevonden waarop de regel in het ILS-XML bestand toepasbaar was. Dit zijn in dit geval 56 elementen van type IfcWallStandardCase, zoals zichtbaar in Listing 10. Van deze 56 elementen voldoen 22 elementen aan de eis dat zij een eigenschap 'IsExternal' hebben met een waarde 'true' of 'false' (zie Listing 10).

File Content check



Check: Information Delivery Specification
Check: Conformance with IDS/ILS
Value: Requirement 'DUTO BIM specifications for IfcWall' succeeded in 22 times of 56
Success: True
Score: 22 out of 56

Figuur 22: Resultaten van de controle van een IFC-bestand tegen een aantal eisen in ILS-XML.

4.3.2 ILS-SHACL check

De controle van de SHACL constraint met de RDF graph van het gebouw is niet geïmplementeerd in de huidige fase, maar alles is klaargezet om dit mogelijk te maken, namelijk:

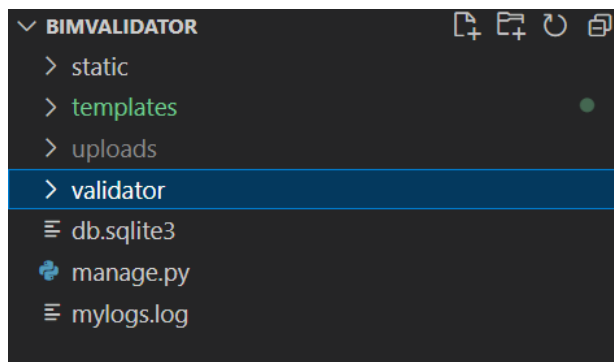
- Beschikbaarheid van GraphDB triplestore
- Conversie van IFC-bestanden naar RDF graphs (triples)
- Opladen van RDF graphs naar GraphDB triple store
- Implementatie van check tussen SHACL en RDF graphs

Wat ontbreekt, is:

- Inbedding in webserver

5. Getting Started

1. Download en unzip de BIMValidator code
2. Open de BIMValidator code in Visual Studio Code. Je ziet volgende folderstructuur in VSCode.



3. Open een nieuwe Terminal bij 'Terminal > New Terminal'
4. Activeer de virtuele omgeving (virtual environment) door volgend commando in te geven in de terminal:

env\Scripts\Activate.ps1

```
PS C:\Users\20194060\OneDrive - TU Eindhoven\Git\tue\ISBE_github\DUTOBIMValidator\BIMValidator> env\Scripts\Activate.ps1
(env) PS C:\Users\20194060\OneDrive - TU Eindhoven\Git\tue\ISBE_github\DUTOBIMValidator\BIMValidator>
```

5. Voer het startcommando voor de server in in de Terminal:
python manage.py runserver
6. Open de web browser en navigeer naar <https://localhost:8000/>
7. Log in met je ingestelde paswoord:
 - a. Admin - paswoord
8. Klaar! Maak je eerste project aan, eerste check, en doe een controle!