



## **Smart office** Realisatiedocument

**Bachelor in de toegepaste informatica  
keuzerichting applicatieontwikkeling**

Academiejaar 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

**Mathias Alen**



# 1 Inhoudsopgave

2	Inleiding .....	3
2.1	Het stagebedrijf .....	3
2.2	De opdracht .....	4
3	Planning .....	5
4	Kennis verwerven .....	5
4.1	Anypoint Platform Development: Fundamentals .....	5
4.2	Raml contract .....	6
4.3	Anypoint studio .....	6
4.4	Dataweave .....	7
5	Realisaties .....	8
5.1	System API .....	8
5.1.1	Weather API .....	8
5.1.2	Raml contract .....	10
5.1.3	Anypoint studio .....	12
5.2	Process API .....	17
5.2.1	Database .....	17
5.2.2	Raml contract .....	18
5.2.3	S3-bucket .....	20
5.2.4	Anypoint studio .....	20
5.3	Experience API .....	27
5.3.1	Back office API .....	27
5.3.2	Smart tv experience API .....	29
6	Resultaat .....	31
7	Communicatie & opvolging .....	31
8	Besluit .....	31

## 2 Inleiding

Een stage binnen de IT biedt werktraject studenten de mogelijkheid om ervaring op te doen binnen een andere sector dan hun huidige werksector. Binnen deze stage werken de studenten aan een project dat helpt om hun vaardigheden te ontwikkelen en versterken. Deze vaardigheden kunnen variëren van programmeervaardigheden tot communicatievaardigheden. Daarnaast biedt de stage ook een mogelijkheid tot netwerken en contacten te leggen. Dit voor eventuele carrièremogelijkheden.

### 2.1 Het stagebedrijf

De afgelopen 18 weken heb ik gewerkt bij Dots & Arrows, gelegen te Herentals. Hier werd ik aangenomen als Mulesoft developer.

Het is een consultancy bureau die Mulesoft als officiële partner heeft. Mulesoft is een softwarebedrijf dat een integratieplatform genaamd Anypoint platform aanbiedt. Het stelt bedrijven/organisaties in staat om verschillende applicaties, gegevens en apparaten te verbinden via API's.

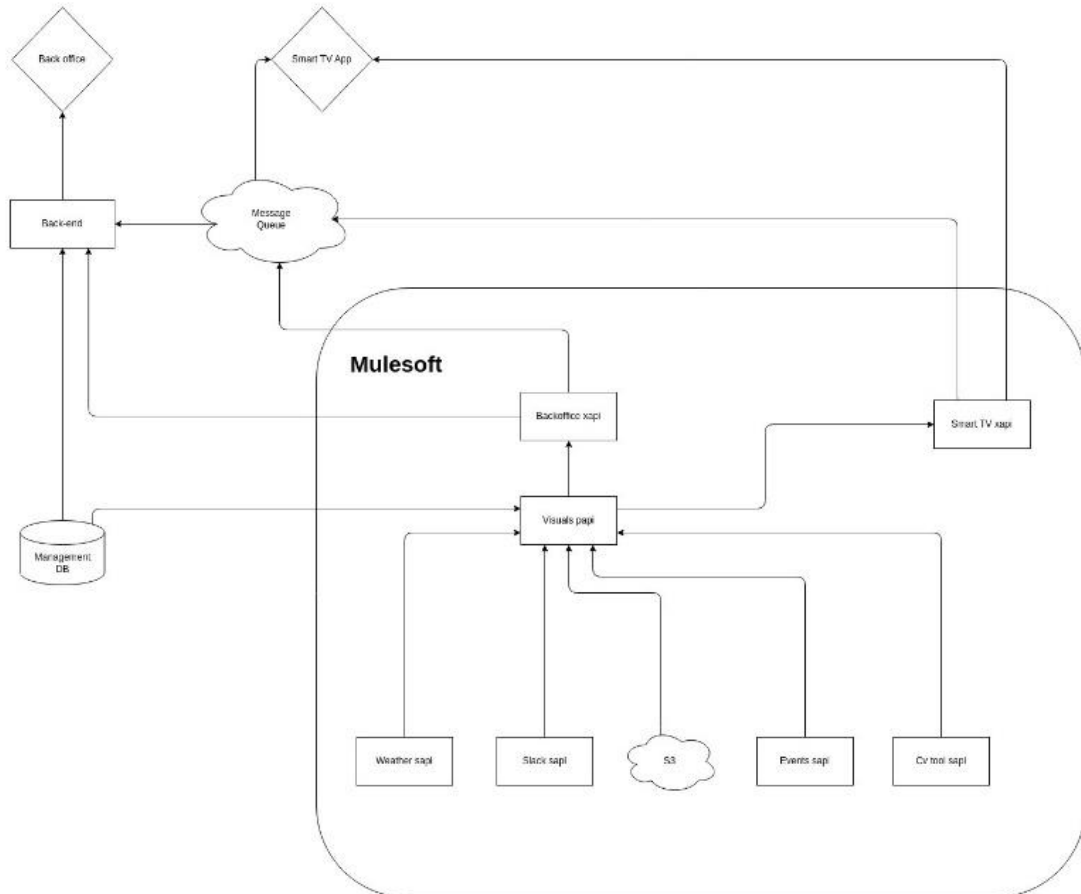
Daarnaast staat er een team van Mulesoft developers klaar die helpen bij de implementatie en ontwikkeling van een API architectuur. Alsook het beoordelen van bestaande installaties. Daarnaast hebben ze gecertificeerde Mulesoft trainers die *on site* opleidingen geven.

Voor extra informatie verwijs ik je graag door naar mijn plan van aanpak.



## 2.2 De opdracht

Ik kreeg als opdracht om de tv schermen op kantoor “slim” te maken. Hierop werd data getoond zoals: verjaardagen, evenementen en het weer. Dit werd manueel ingesteld zonder enige automatisatie inbegrepen. Het was mijn taak om via de API-led connectivity de weather system API, visuals process API en de 2 experience API's op te stellen (zie afbeelding). Zodanig dat het weer automatisch wordt opgehaald en als template getoond kan worden op de tv's. Het doel en de betekenis van elke API wordt in dit document nog uitgebreid besproken.



### 3 Planning

Bij het maken van de planning heb ik mij de eerste 3 weken gegeven om mij in te werken in de nieuwe technologie. Als eerste opdracht had ik de weather system API. Daarnaast was de visual proces API de grootste blok in mijn opdracht. Hiervoor heb ik dan ook 9 weken gerekend. De laatste 6 weken zijn vooral overgebleven voor de experience API's en voor eventuele aanpassingen aan de visual proces API.

#### Week 1-3

- Opstart + aanmaak [weather system API](#)

#### Week 4-6

- POST [method visuals process API](#) (aanspreken s3 bucket)

#### Week 7-9

- GET [method visuals process API](#) + aanmaken database + aanpassingen maken aan de reeds bestaande system [API's](#)

#### Week 10-12

- Afmaken GET [method visuals process API](#)

#### Week 13-15

- Backoffice [experience API](#)

#### Week 16-18

- Smart-tv [experience API](#)

### 4 Kennis verwerven

Om aan de opdracht te kunnen beginnen was er een zekere kennis van Mulesoft vereist. Omdat dit volledig nieuw was, werd mij aangeraden om de gratis cursus van Mulesoft te volgen.

#### 4.1 Anypoint Platform Development: Fundamentals

Als eerste heb ik de gehele cursus gevolgd. Deze bestond uit 2 grote delen: *getting started* en *building Mule applications*. Respectievelijk hadden deze nog eens 5 en 8 delen. Uiteindelijk zijn hier toch rond de 5 dagen naartoe gegaan. In deze cursussen wordt een volledige cyclus doorlopen van de API-led connectivity. Het maken van Raml contracten tot het implementeren binnen anypoint studio.

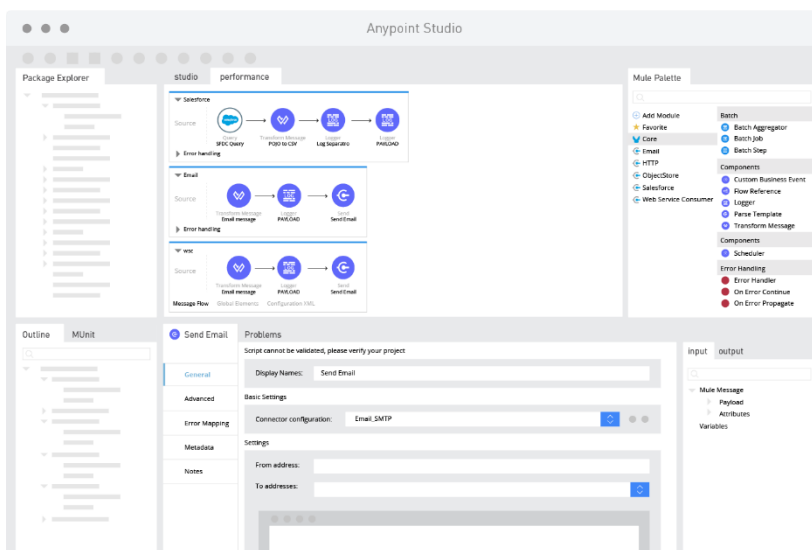
## 4.2 Raml contract

Tijdens de uitleg van de cursus werd er ook aandacht besteed aan het opstellen van Raml contracten. (Raml = Restful API Modeling Language, Rest = Representational State Transfer) Het biedt een gestructureerde manier om de endpoints, resources, parameters, ... te beschrijven. Met een RAML-contract kan een API-ontwikkelaar een duidelijke en gestandaardiseerde beschrijving maken van de functionaliteit en de verwachte gedragingen van de API. Het wordt gezien als de blauwdruk van de API.

```
5  /employees:
6  get:
7    responses:
8      200:
9        body:
10         application/json:
11           example: { "name" : "Jon Smith" }
12  post:
13    body:
14      application/json:
15        examples:
16          ex1: !include examples/employee-example-1.yaml
17          ex2: !include examples/employee-example-2.json
18    responses:
19      200:
20        body:
21          application/json:
22            examples: !include examples/employee-example-3.raml
23
24  delete:
25    responses:
26      200:
27        body:
28          application/json:
29            example: "Data deleted"
```

## 4.3 Anypoint studio

Naast de theorie van de cursus werden ook oefeningen aangebracht om te leren werken met anypoint studio. Dit is de geïntegreerde ontwikkelomgeving (IDE) van Mulesoft. Met Anypoint Studio kunnen ontwikkelaars op een grafische manier integraties ontwerpen door gebruik te maken van bouwstenen zoals connectors en transformaties. Het biedt ook mogelijkheden voor het configureren van gegevenstransformaties, het implementeren van beveiligingsmechanismen en het bewaken van de prestaties van integraties.



#### 4.4 Dataweave

Dataweave is een gegevenstransformatietaal die wordt gebruikt binnen het Anypoint Platform van Mulesoft. Het is ontworpen om gegevens uit verschillende bronnen zoals JSON, XML en andere formaten, te integreren en te transformeren in het gewenste formaat van integratie. Het biedt een makkelijke syntax om gegevens te transformeren aan de hand van functies, operators en variabelen. Het ondersteunt ook zeker complexe transformaties, denk aan: aggregatie, filtering en sorteren, ... Dataweave zit geïntegreerd in Anypoint Studio.

Hiervoor heb ik de tutorial op de dataweave website zelf gevolgd.

Ook heb ik elke keer als ik een datatransformatie moet doen, gebruik gemaakt van de Dataweave playground.



DataWeave



## 5 Realisaties

Hier ga ik al mijn realisaties bespreken die ik heb gemaakt. Ik ga ze opdelen in drie delen: system API, process API en experience API.

**Opmerking:** wanneer je volgende syntax tegenkomt: `${}` betekent dat dit bepaald wordt in de config file die geëncrypteerd wordt. (privacy)

### 5.1 System API

Mijn allereerste opdracht was om een weather system API te maken. Ik ben beginnen zoeken op het internet naar een gratis weather API zodat ik deze kan implementeren in anypoint platform. Uiteindelijk ben ik gestoten op WeatherAPI.

We zoeken een API die we willen implementeren (weatherAPI **5.1.1**). We maken het RAML contract (**5.1.2**) aan en gebruiken dit voor de implementatie in anypoint studio (**5.1.3**).

Weather flow → getForecast flow (transformaties) → weatherAPI flow (API call)

Er wordt een API call gedaan naar de weatherAPI. Deze gegevens worden daarna omgevormd en teruggegeven. We kunnen de call testen via Postman (**5.1.3.3**). Wanneer de postman werkt wordt het project geëxporteerd en op runtime manager gezet zodat we er niet enkel lokaal aankunnen (**5.1.3.4**).

Hieronder wordt dit proces gedetailleerd besproken.

#### 5.1.1 Weather API

Hier kan ik een gratis account maken waar ik 1 miljoen calls per maand kan doen. Dit is zeker genoeg voor wat de opdracht inhoudt. Na het aanmaken van mijn account krijg ik een API key die ik elke keer moet gebruiken om een call te doen. Daarnaast is er ook een query parameter voor de locatie. Dit is zeker gewenst omdat ik voor twee locaties (Hasselt en Herentals) het weer kan nodig hebben.

Voorbeeld:

Parameter	Value	Type	Location	Description
q	London	string	query	Pass US Zipcode, UK Postcode, Canada Postalcode, IP address, Latitude/Longitude (decimal degree) or city name. Visit <a href="#">request parameter</a> section to learn more.

Figuur 1 Website weather API

Als laatste kan ik ook nog kiezen voor de endpoint (= URL die specifieke functies uitvoert) forecast, hierbij kan ik ook nog meegeven voor hoeveel dagen ik het weer wil krijgen.

Voorbeeld response body:

Er wordt zeer veel informatie gegeven in verband met het weer. Deze ga ik niet allemaal kunnen gebruiken, maar het is zeker nuttig om deze mee te nemen voor uitbreidbaarheid.

```
{
  "location": {
    "name": "Herentals",
    "region": "",
    "country": "Belgium",
    "lat": 51.18,
    "lon": 4.83,
    "tz_id": "Europe/Brussels",
    "localtime_epoch": 1683884959,
    "localtime": "2023-05-12 11:49"
  },
  "current": {
    "last_updated_epoch": 1683884700,
    "last_updated": "2023-05-12 11:45",
    "temp_c": 14.0,
    "temp_f": 57.2,
    "is_day": 1,
    "condition": {
      "text": "Partly cloudy",
      "icon": "///cdn.weatherapi.com/weather/64x64/day/116.png",
      "code": 1003
    }
  },
}
```

```
    "wind_mph": 11.9,
    "wind_kph": 19.1,
    "wind_degree": 70,
    "wind_dir": "ENE",
    "pressure_mb": 1016.0,
    "pressure_in": 30.0,
    "precip_mm": 0.2,
    "precip_in": 0.01,
    "humidity": 88,
    "cloud": 75,
    "feelslike_c": 12.8,
    "feelslike_f": 55.0,
    "vis_km": 10.0,
    "vis_miles": 6.0,
    "uv": 3.0,
    "gust_mph": 13.0,
    "gust_kph": 20.9
  },
  "forecast": {
```

```
"forecastday": [
  {
    "date": "2023-05-12",
    "date_epoch": 1683849600,
    "day": {
      "maxtemp_c": 18.8,
      "maxtemp_f": 65.8,
      "mintemp_c": 10.4,
      "mintemp_f": 50.7,
      "avgtemp_c": 13.6,
      "avgtemp_f": 56.4,
      "maxwind_mph": 13.2,
      "maxwind_kph": 21.2,
      "totalprecip_mm": 22.6,
      "totalprecip_in": 0.89,
      "totalsnow_cm": 0.0,
      "avgvis_km": 8.4,
      "avgvis_miles": 5.0,
      "avghumidity": 87.0,
      "daily_will_it_rain": 1,
      "daily_chance_of_rain": 98,
      "daily_will_it_snow": 0,
      "daily_chance_of_snow": 0,
      "condition": {
        "text": "Heavy rain",
        "icon": "///cdn.weatherapi.com/weather/64x64/day/308.png",
        "code": 1195
      },
      "uv": 3.0
    },
    "astro": {
      "sunrise": "05:56 AM",
      "sunset": "09:21 PM",
      "moonrise": "03:34 AM",
      "moonset": "12:06 PM",
      "moon_phase": "Last Quarter",
      "moon_illumination": "57",
      "is_moon_up": 0,
      "is_sun_up": 0
    },
    "hour": [
      {
        "time_epoch": 1683842400,
        "time": "2023-05-12 00:00",
        "temp_c": 11.4,
        "temp_f": 52.5,
        "is_day": 0,
        "condition": {
          "text": "Light rain shower",
          "icon": "///cdn.weatherapi.com/weather/64x64/night/353.png",
          "code": 1240
        }
      },

```

Figur 2 response body weather api

### 5.1.2 Raml contract

Na het vinden van de API ben ik begonnen met het opstellen van mijn eerste raml contract.

```
##RAML 1.0
title: weather-sapi

/forecast:
  get:
    queryParameters:
      key:
        type: string
        required: true
        example: 
      location:
        type: string
        required: true
        example: "Hasselt"
      days:
        type: integer
        required: true
        minimum: 1
        maximum: 3
        example: 1
    responses:
      200:
        body:
          application/json:
            type: !include types/forecast.raml
```

Figuur 3 raml contract weather system API

Hier zien we dat ik een get call naar forecast aanmaak en ik mijn: key, locatie en dagen meegeef als query parameters. Belangrijk bij dagen is dat ik zeker een maximum en minimum meegeef. Bij het gratis account kan ik tot maximum 3 dagen opvragen.

Daarnaast moet ik in de *body* bepalen wat voor type ik terugkrijg van de API call:

```
##RAML 1.0 DataType
type: object
properties:
  location:
    type: object
    required: true
    properties:
      name:
        type: string
        required: true
        example: "Hasselt"
      region:
        type: string
        required: false
        example: "Limburg"
      country:
        type: string
        required: true
        example: "Belgium"
      lat:
        type: string
        required: true
        example: "50.93"
      lon:
        type: string
        required: true
        example: "5.33"
```

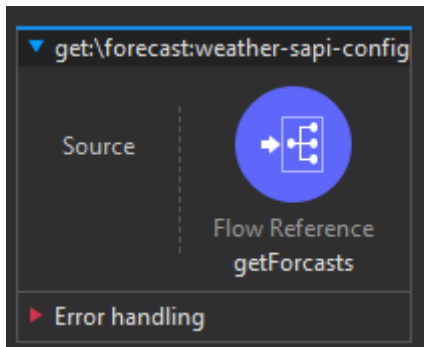
*Figuur 4 Datatype weather API raml contract*

Hier zie je een voorbeeld hoe een datatype eruit ziet. Ik heb dus bijvoorbeeld een naam nodig, dit zal een string zijn met als voorbeeld "Hasselt".

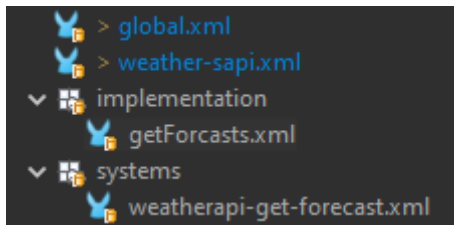
Wanneer alles klopt kan je het RAML contract publiceren.

### 5.1.3 Anypoint studio

Nadat het contract klaar en gepubliceerd is, maak ik een project aan in anypoint studio. Hier beslis ik dat het project gebruik moet maken van mijn RAML contract. Zo worden er automatisch al het aantal calls aangemaakt. In dit voorbeeldje zal er al 1 get call worden aangemaakt.



Figuur 5 Get flow in anypoint studio

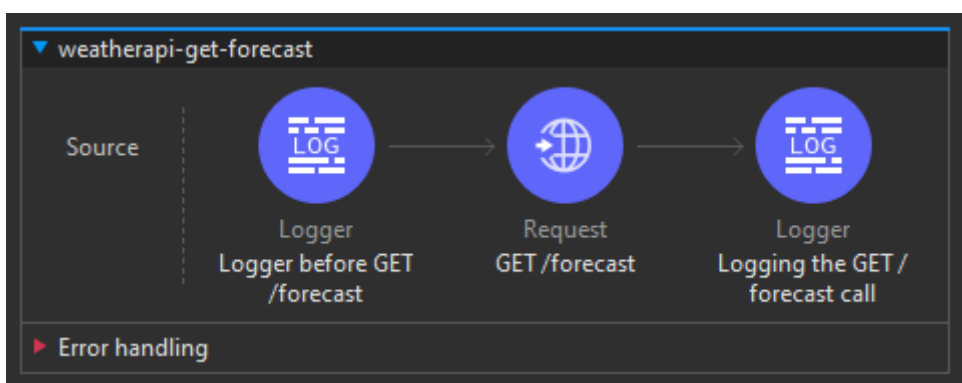


Figuur 6 Mappenstructuur weather System API in anypoint studio

Hier zie je de mappenstructuur in anypoint studio. Binnen *system* valt de call naar de API zelf. In *implementation* komt de omvorming van wat we binnenkrijgen van de API.

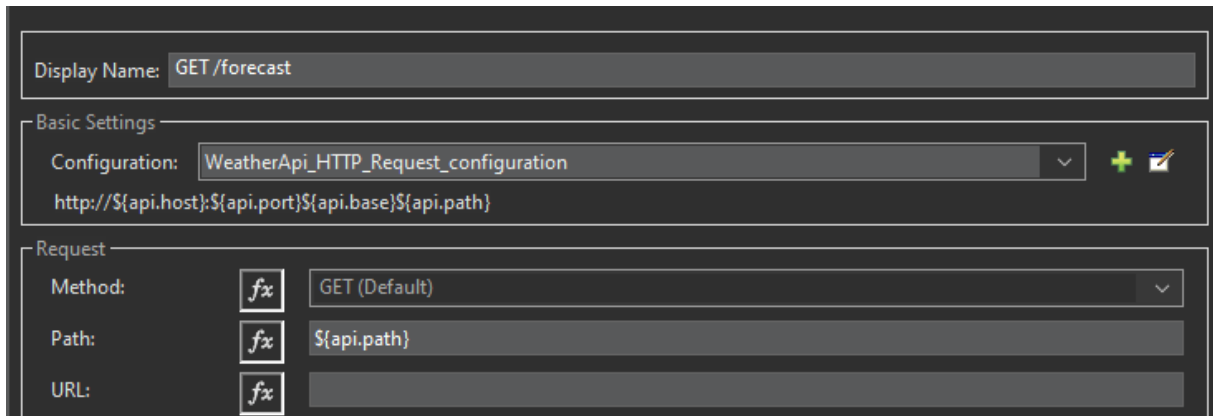
#### 5.1.3.1 Systems

Dit is de flow van de API call. Wanneer ik dus een call maak naar `/forecast` zal ik eerst *loggen* dat ik een get request ga uitvoeren. Daarna wordt de get request uitgevoerd en als laatste zal ik nog een keer *loggen*. Dit *loggen* wordt vooral gedaan om eventuele problemen op te sporen.



Figuur 7 get forecast flow in anypoint studio

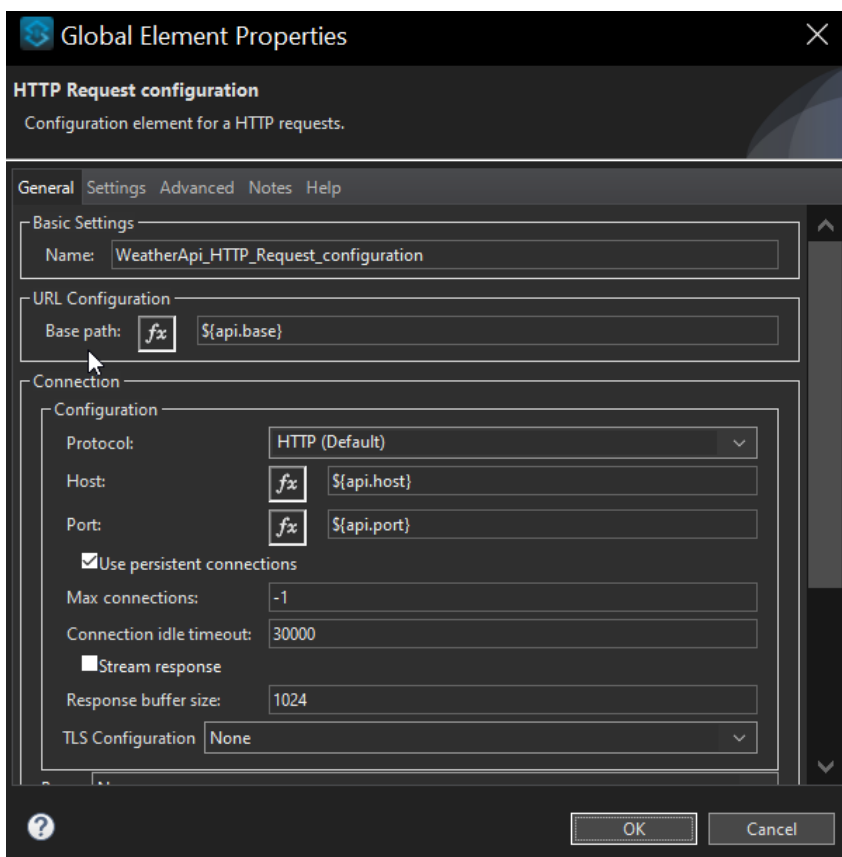
De bouwstenen van de weather system API zijn er, maar nu moeten we nog verbinding kunnen maken met de API. Dit doen we via de eigenschappen van de *Request* bouwsteen. Hierin wordt bepaald welke configuratie gebruikt moet worden, de methode en het pad waar we naartoe moeten gaan.



The screenshot shows a configuration window for a GET request. At the top, the 'Display Name' is 'GET/forecast'. Below this is a 'Basic Settings' section with a 'Configuration' dropdown set to 'WeatherApi\_HTTP\_Request\_configuration' and a URL template 'http://{api.host}:{api.port}\${api.base}\${api.path}'. The 'Request' section contains three fields: 'Method' (a dropdown with 'GET (Default)' selected), 'Path' (a text field with '\${api.path}' and a function icon), and 'URL' (a text field with a function icon).

Figuur 8 De properties van GET forecast

Om wat dieper in te gaan op een configuratie van een *request*. Er zal een basis-URL (base path), domein (host) en poort (port) moeten worden bepaald. Zo zullen we verbinding kunnen maken met de weather API.



The screenshot shows the 'Global Element Properties' dialog for 'HTTP Request configuration'. It has tabs for 'General', 'Settings', 'Advanced', 'Notes', and 'Help'. The 'General' tab is active, showing 'Basic Settings' with 'Name: WeatherApi\_HTTP\_Request\_configuration'. Below is 'URL Configuration' with 'Base path: \${api.base}' and a function icon. The 'Connection' section includes 'Configuration' with 'Protocol: HTTP (Default)', 'Host: \${api.host}', 'Port: \${api.port}', and checkboxes for 'Use persistent connections' (checked) and 'Stream response' (unchecked). Other fields include 'Max connections: -1', 'Connection idle timeout: 30000', 'Response buffer size: 1024', and 'TLS Configuration: None'. 'OK' and 'Cancel' buttons are at the bottom.

Figuur 9 Configuratie get call

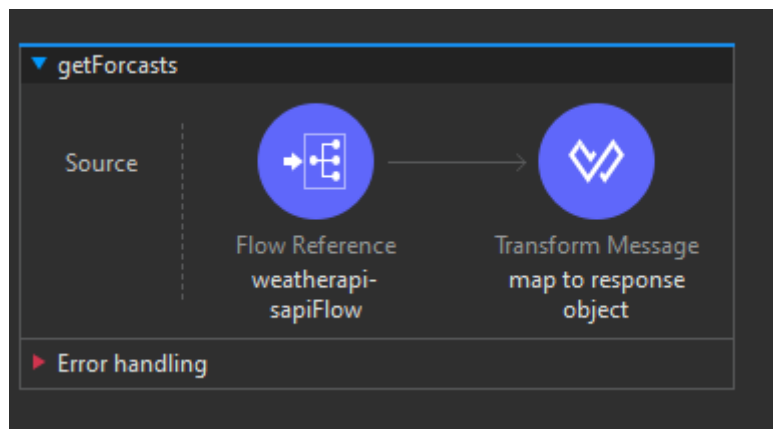
Daarstraks hadden we al besproken dat we bij de weather API moeten meegeven welke locatie en voor hoeveel dagen in de toekomst we het weer willen opvragen. Dit wordt bepaald in de zoekparameter (query parameter).

```
output application/java
---
{
  "q" : attributes.queryParams.location,
  "days" : attributes.queryParams.days
}
```

Figuur 10 query parameters van de GET forecast call

#### 5.1.3.2 Implementatie

Daarnet hebben we alle bouwstenen gemaakt voor het opvragen van het weer via een API request. Bij de implementatie zorgen we ervoor dat de info die we binnenkrijgen wordt omgevormd. Zo zie je dat de eerste bouwsteen verwijst naar *systems* (5.1.3.1). Het omvormen van info gebeurt in de *Transform Message* bouwsteen.



Figuur 9 getForecast Flow

Dit is de inhoud van de *transform message* bouwsteen:

```
%dw 2.0
output application/json skipNullOn="everywhere"
---
{
  "location": {
    "name": payload.location.name,
    "region": payload.location.region,
    "country": payload.location.country,
    "lat": payload.location.lat as String,
    "lon": payload.location.lon as String,
    "tzid": payload.location.tz_id,
    "localtimeEpoch": payload.location.localtime_epoch as String,
    "localtime": payload.location.localtime
  },
  "current": {
    "lastUpdatedEpoch": payload.current.last_updated_epoch as String,
    "lastUpdated": payload.current.last_updated,
    "tempC": payload.current.temp_c,
    "tempF": payload.current.temp_f,
    "isDay": payload.current.is_day,
    "conditionText": payload.current.condition.text,
    "conditionIcon": payload.current.condition.icon,
    "conditionCode": payload.current.condition.code as String,
    "windMph": payload.current.wind_mph,
    "windKph": payload.current.wind_kph,
    "windDegree": payload.current.wind_degree,
    "windDir": payload.current.wind_dir,
    "pressureMb": payload.current.pressure_mb,
    "pressureIn": payload.current.pressure_in,
    "precipMm": payload.current.precip_mm,
    "precipIn": payload.current.precip_in,
    "humidity": payload.current.humidity,
    "cloud": payload.current.cloud,
    "feelslikeC": payload.current.feelslike_c,
    "feelslikeF": payload.current.feelslike_f,
    "visKm": payload.current.vis_km,
    "visMiles": payload.current.vis_miles,
    "uv": payload.current.uv,
    "gustMph": payload.current.gust_mph,
    "gustKph": payload.current.gust_kph
  },
}
```

```
"forecast": payload.forecast.forecastday map {
  "date": $.date,
  "dateEpoch": $.date_epoch as String,
  "maxtempC": $.day.maxtemp_c,
  "maxtempF": $.day.maxtemp_f,
  "mintempC": $.day.mintemp_c,
  "mintempF": $.day.mintemp_f,
  "avgtempC": $.day.avgtemp_c,
  "avgtempF": $.day.avgtemp_f,
  "maxwindMph": $.day.maxwind_mph,
  "maxwindKph": $.day.maxwind_kph,
  "totalprecipMm": $.day.totalprecip_mm,
  "totalprecipIn": $.day.totalprecip_in,
  "totalsnowCm": $.day.totalsnow_cm,
  "avgvisKm": $.day.avgvis_km,
  "avgvisMiles": $.day.avgvis_miles,
  "avghumidity": $.day.avghumidity,
  "dailyWillItRain": $.day.daily_will_it_rain,
  "dailyChangeOfRain": $.day.daily_chance_of_rain,
  "dailyWillItSnow": $.day.daily_will_it_snow,
  "dailyChanceOfSnow": $.day.daily_chance_of_snow,
  "conditionText": $.day.condition.text,
  "conditionIcon": $.day.condition.icon,
  "conditionCode": $.day.condition.code,
  "uv": $.day.uv,
  "sunrise": $.astro.sunrise,
  "sunset": $.astro.sunset,
  "moonrise": $.astro.moonrise,
  "moonset": $.astro.moonset,
  "moonPhase": $.astro.moon_phase,
  "moonIllumination": $.astro.moon_illumination,
  "isMoonUp": $.astro.is_moon_up,
  "isSunUp": $.astro.is_sun_up,
  hours: $.hour map{
    "timeEpoch": $.time_epoch as String,
    "time": $.time,
    "tempC": $.temp_c,
    "tempF": $.temp_f,
    "isDay": $.is_day,
    "conditionText": $.condition.text,
```

Figuur 10 example transform message



### 5.1.3.3 Postman

Nadat de implementatie klaar is heb je een API gemaakt. Deze kan je vervolgens gaan testen in Postman. Postman is een programma dat wordt gebruikt bij het testen van API's. Het biedt een gebruiksvriendelijke interface waarmee ontwikkelaars API-verzoeken kunnen maken, verzenden en ontvangen, en de reacties kunnen controleren en analyseren.

```
{
  "location": {
    "name": "Hasselt",
    "region": "",
    "country": "Belgium",
    "lat": "50.93",
    "lon": "5.33",
    "tzId": "Europe/Brussels",
    "localtimeEpoch": "1683892396",
    "localtime": "2023-05-12 13:53"
  },
  "current": {
    "lastUpdatedEpoch": "1683891900",
    "lastUpdated": "2023-05-12 13:45",
    "tempC": 15.0,
    "tempF": 59.0,
    "isDay": 1,
    "conditionText": "Overcast",
    "conditionIcon": "https://cdn.weatherapi.com/weather/64x64/day/122.png",
    "conditionCode": "1009",
    "windMph": 8.1,
    "windKph": 13.0,
    "windDegree": 80,
    "windDir": "E",
    "pressureMb": 1016.0,
    "pressureIn": 30.0,
    "precipMm": 0.0,
    "precipIn": 0.0,
    "humidity": 77,
    "cloud": 100,
    "feelslikeC": 14.0,
    "feelslikeF": 57.1,
    "visKm": 10.0,
    "visMiles": 6.0,
    "uv": 4.0,
    "gustMph": 11.9,
    "gustKph": 19.1
  }
}
```

Figuur 11 Postman output na GET call

### 5.1.3.4 Runtime manager

Wanneer de call werkt en correct wordt uitgevoerd, wordt het project op de runtime manager gezet. Hierdoor wordt het project online gezet en kunnen we via een URL het weer ophalen.

● bewire-weather-sapi-sandbox

Domain: [bewire-weather-sapi-sandbox.de-c1.cloudhub.io](https://bewire-weather-sapi-sandbox.de-c1.cloudhub.io) · Last Updated 2023-05-12 12:36:33PM · 1 micro worker, using 4.4.0

## 5.2 Process API

De volgende stap in de opdracht is om alle system API's te combineren in één grote process API genaamd visual process API. Een visual is een scherm op de tv. Bijvoorbeeld: visual bevat het weer van de komende drie dagen alsook de verjaardagen van die maand.

Een visual heeft verschillende eigenschappen zoals positie, actief, begin datum, eind datum, ... Deze gegevens moeten ook worden bijgehouden, daarom wordt er een database aangemaakt. Een visual is een HTML bestand. Deze moet ook worden bijgehouden, daarom wordt er gekozen voor een s3 bucket. Dit is een cloudopslagsservice waarin verschillende soorten gegevens bewaard kunnen worden, in ons geval een HTML document. Ook is het makkelijk om via anypoint studio een connectie te leggen naar de s3 bucket. Dit bepaald ook meteen wat de *POST method* zal moeten doen, uploaden van het HTML document en de eigenschappen binnen de database aanmaken (In ons geval gaat het enkel om de documentnaam.). Naast het uploaden moeten we ook alle visuals kunnen opvragen die in de database reeds aanwezig zijn. Dit wordt gedaan met de *GET method*. Belangrijk hierbij is dat elk HTML document een tag bevat die verwijst naar 1 van de system API's bijvoorbeeld weather system API. Deze info wordt samengevoegd en getoond aan de gebruiker. Als laatste is er de *refresh method*. Deze zorgt ervoor dat de visuals op een *message queue* terechtkomen. Dit is een wachtrij waarin berichten worden opgeslagen totdat ze worden verwerkt door de ontvangende component. Deze methode gaat ook worden aangesproken door een *scheduler*. Dit is een manier om de methode op een automatische manier te laten uitvoeren.

Hieronder wordt dit uitgebreid besproken.

### 5.2.1 Database

Allereerst moet er een plaats zijn waar we de eigenschappen van de visual gaan bijhouden. Dit zal een MySQL database worden. Samen met mijn stagebegeleider zijn we op volgend MySQL script geëindigd:

```
CREATE TABLE tools.Visuals (
  ID BIGINT NOT NULL auto_increment,
  TemplateFileName varchar(255) NOT NULL,
  TransitionType ENUM('FADE', 'SLIDE', 'ZOOM'),
  Duration INT,
  PRIMARY KEY (ID)
);

CREATE TABLE tools.VisualsSchedules (
  VisualID BIGINT NOT NULL,
  Position JSON NOT NULL,
  IsActive BOOL NOT NULL,
  FromDate DATETIME NOT NULL,
  UntilDate DATETIME,
  IsFullDay BOOL NOT NULL,
  IsRecurring BOOL NOT NULL,
  VisibilityDuration INT,
  RecurringInterval INT,
  Location ENUM('Hasselt', 'Herentals'),
  FOREIGN KEY (VisualID) REFERENCES tools.Visuals(ID)
);

CREATE TABLE tools.Tags (
  ID BIGINT auto_increment NOT NULL,
  Value varchar(255) NOT NULL,
  PRIMARY KEY (ID)
);

CREATE TABLE tools.VisualsTags(
  VisualID BIGINT NOT NULL,
  TagID BIGINT NOT NULL,
  FOREIGN KEY (VisualID) REFERENCES tools.Visuals(ID),
  FOREIGN KEY (TagID) REFERENCES tools.Tags(ID)
);
```

Figuur 12 mysql script voor de database

### 5.2.2 Raml contract

Daarna wordt het Raml contract opgesteld.

```
8  /visuals:
9    post:
10     body:
11       multipart/form-data:
12         type: visual
13         examples:
14           visual: !include examples/visual.raml
15     responses:
16       200:
17         body:
18           application/json:
19             example: !include examples/visualOK.raml
20       404:
21         body:
22           application/json:
23             example: !include examples/visualFAIL.raml
24     get:
25       responses:
26         200:
27           body:
28             application/json:
29               type: visuals[]
30  /refresh:
31    get:
32      queryParameters:
33        visualId:
34          required: false
35          type: string
36      responses:
37        200:
38          body:
39            application/json:
40              example: {"message": "Visual(s) refreshed successfully."}
41        404:
42          description: The specified visual ID could not be found.
```

Figuur 13 RAML contract visuals process API

Hierbij hebben we een *POST*, *GET* en *refresh method*. *POST* wordt gebruikt om een visual toe te voegen aan de s3-bucket (Simple Storage Service = een opslagcontainer die wordt aangeboden door AWS) alsook de eigenschappen van de visual aan de database. *GET* wordt gebruikt om een *array* aan visuals te tonen. *Refresh* wordt gebruikt om de visuals op een *message queue* te zetten.

Het type van de *POST method*, hiervoor hebben we een file, pubDate, lastUpdated, filename, description en extension nodig.

```
1  #%RAML 1.0 DataType
2
3  properties:
4    visualId: number
5    file:
6      type: file
7      fileTypes: ['*/*']
8      maxLength: 10485760
9    pubDate: string
10   lastUpdated:
11     type: string
12     required: false
13   filename: string
14   description:
15     type: string
16     required: false
17   extension: string
```

Figuur 14 Datatype *POST method*

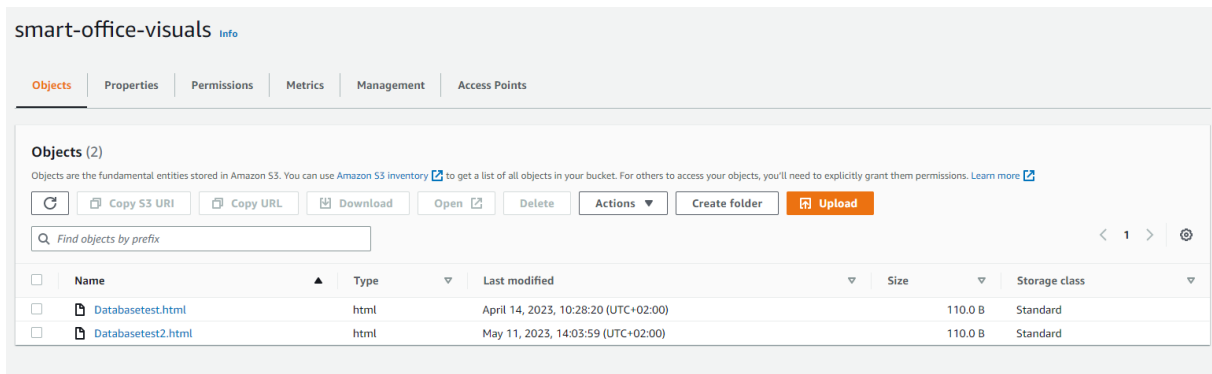
Het type van de *GET method* is zeer uitgebreid en ga ik niet helemaal in detail bespreken, maar het is een *array* die verschillende items teruggeeft in een object. Zoals bijvoorbeeld de *duration* en *transition* van een visual. Zo zie je dat *transition* een object is in de array die op zijn beurt weer verschillende eigenschappen bevat.

```
1  #%RAML 1.0 DataType
2   type: array
3   items:
4     type: object
5     properties:
6       identifier:
7         type: string
8         example: "51038924-5567-4180-95a5-03e834d3925b"
9       positions:
10        type: array
11        example: [1,5]
12      duration:
13        type: number
14        example: 1234
15      transition:
16        type: object
17        properties:
18          mode:
19            type: string
20            example: "FADE"
21            enum: [FADE, SLIDE, ZOOM]
22          duration:
23            type: number
24            example: 1234
25      schedules:
26        type: array
27        items:
28          type: object
29          properties:
30            type:
31              type: string
32              example: "ACTIVE"
33              enum: [ACTIVE, INACTIVE]
34            from:
35              type: datetime-only
36              example: "2023-02-03T10:03:00"
37            until:
38              type: datetime-only
```

Figuur 15 Datatype *GET method*

### 5.2.3 S3-bucket

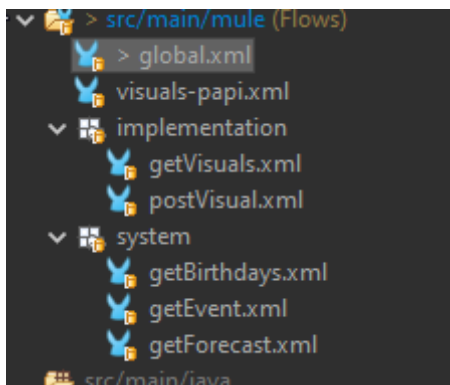
Deze heb ik zelf niet moeten aanmaken, maar enkel moeten aanspreken in anypoint studio. Hier zal mijn visual worden opgeslagen na de *POST method*.



Figuur 16 s3-bucket

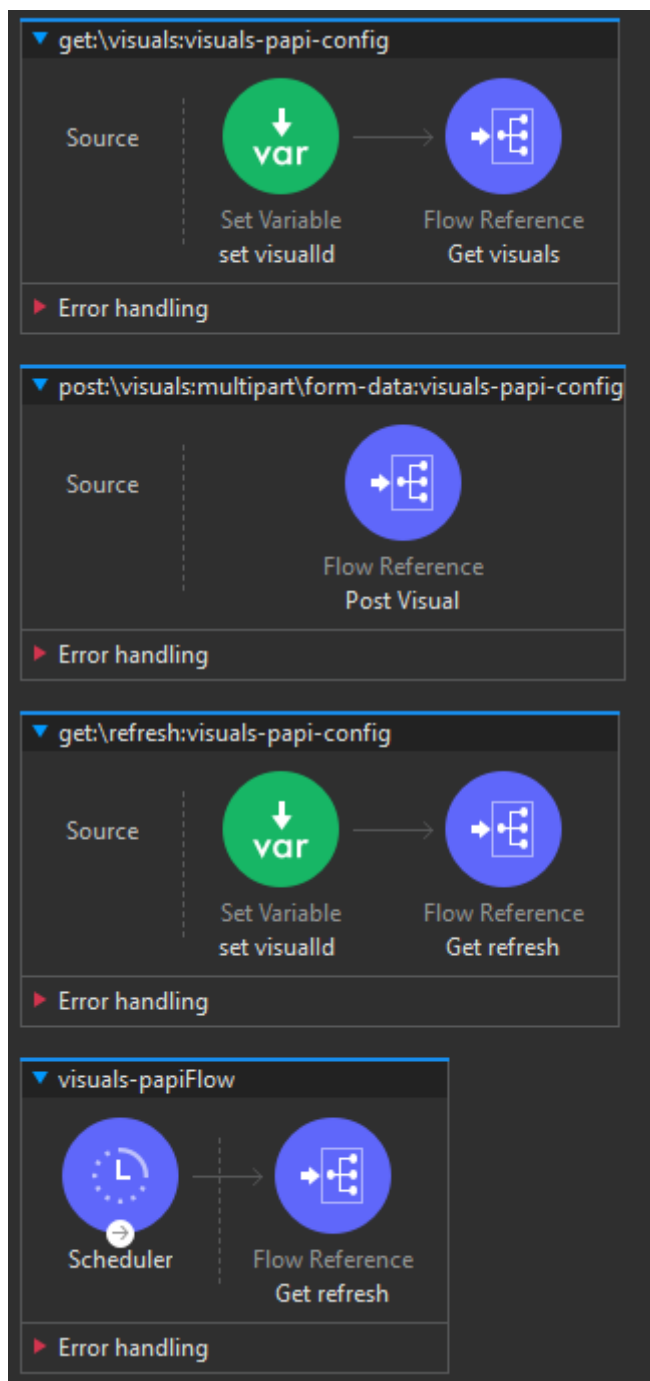
### 5.2.4 Anypoint studio

Hier volgt weer dezelfde werkwijze als bij mijn weather system API. Hieronder volgt de mappenstructuur die we aanhouden.



Figuur 17 mappenstructuur visual process API

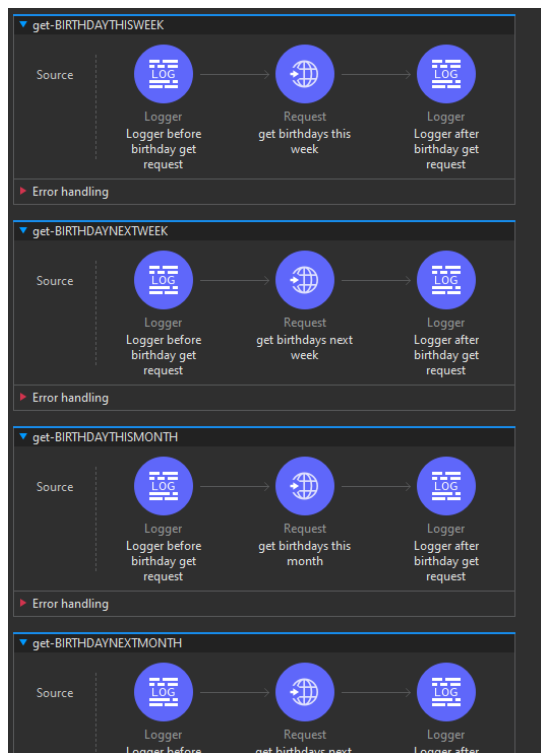
Hier zien we de 4 *main flows* van de visual process API. We hebben de GET visuals, POST visual, GET refresh en een *scheduler* (= deze zal een proces uitvoeren elke nacht om 3 uur).



Figuur 18 4 main flows van visual proces API

#### 5.2.4.1 Systems

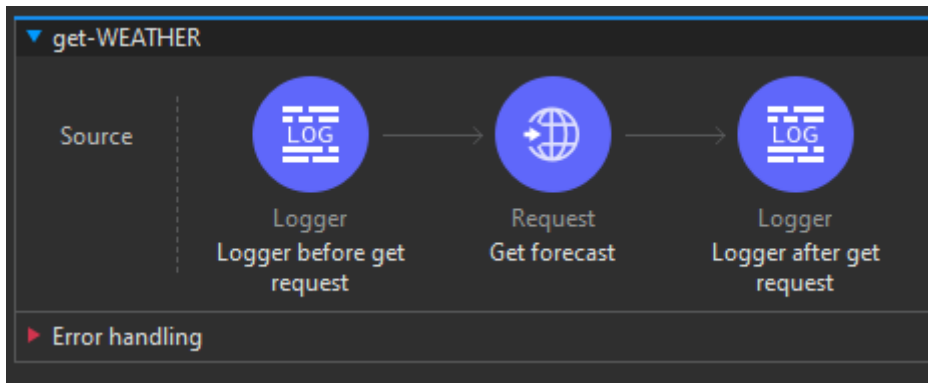
Hier worden alle calls gedaan naar de verschillende system API's. De birthday (figuur 21) en event (figuur 22) system API's waren al aangemaakt door een vorige stagestudent. Ik heb deze enkel aangepast zodat ik een from- en untildate kan meegeven zodat we per week een maand kunnen kijken, alsook op deze dag. Als laatste zien we op figuur 23 de net aangemaakte weather system API. enkel GET visuals en GET refresh zal deze system API's nodig hebben.



Figuur 19 GET birthday system API



Figuur 20 GET event system API

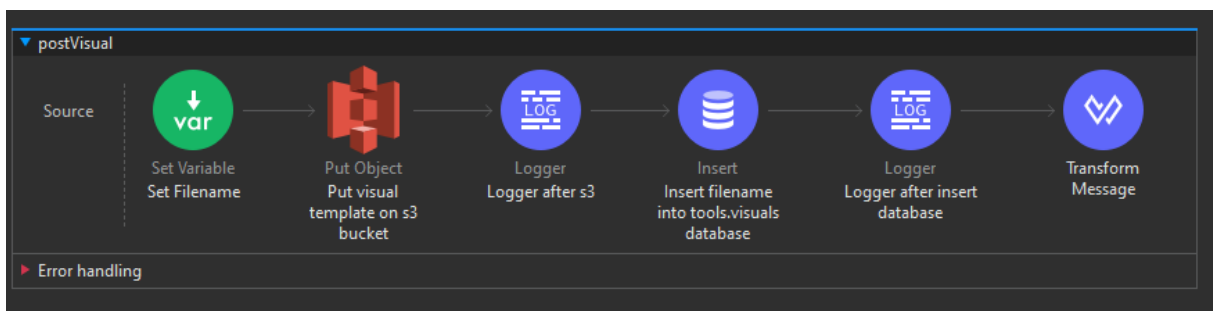


Figuur 21 weather system API

#### 5.2.4.2 implementatie

##### 5.2.4.2.1 POST method

De post method zorgt ervoor dat er een visual template wordt toegevoegd aan de s3 bucket alsook in de database. Hier zien we de POST flow in anypoint studio. De *filename* wordt als variabelen bewaard. Daarna wordt de Visual op de s3-bucket geplaatst en wordt er gelogd om te checken of hier geen error is. Vervolgens wordt de *filename* ook toegevoegd aan de database en wordt er nog eens gelogd. Als laatste geven we de *filename* en de message: "file uploaded" terug naar de gebruiker.



Figuur 22 POST visual flow

#### 5.2.4.3 postman

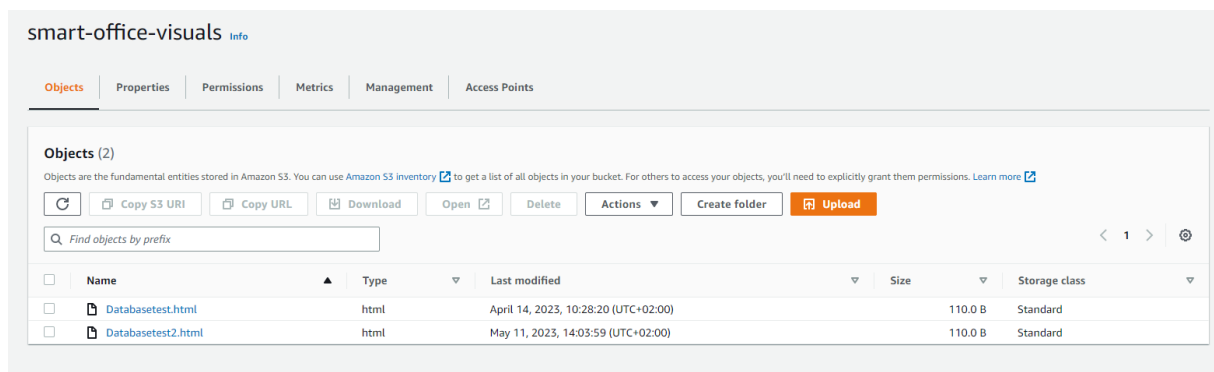
Om deze POST method te testen gaan we in postman een POST call doen naar visual met de juiste body die we in het RAML contract hebben opgesteld. Zo wordt er een visualId, file, pubDate, filename en extension meegegeven.

KEY	VALUE
<input checked="" type="checkbox"/> visualId	4
<input checked="" type="checkbox"/> file	dummyHTML.html
<input checked="" type="checkbox"/> pubDate	"20"
<input type="checkbox"/> lastUpdated	"20"
<input checked="" type="checkbox"/> filename	"Databasetest2"
<input checked="" type="checkbox"/> extension	"html"
Key	Value

Figuur 23 postman POST visual



Na het uitvoeren zien we deze ook verschijnen op de s3 bucket:



Figuur 24 s3-bucket na het toevoegen visual

Alsook in de database:

ID	ABC TemplateFileName	ABC TransitionType	123 Duration
1	Databasetest.html	FADE	123
2	Databasetest.html	ZOOM	1,234
3	Databasetest.html	ZOOM	1,500
4	Databasetest.html	SLIDE	1,350
5	Databasetest.html	[NULL]	[NULL]
6	Databasetest2.html	[NULL]	[NULL]
7	Databasetest2.html	[NULL]	[NULL]

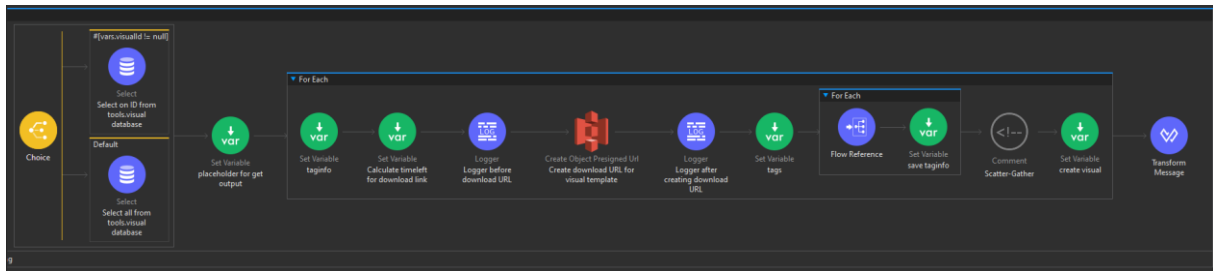
Figuur 25 database na POST method

Dit betekent dat onze API correct gemaakt is.

#### 5.2.4.3.1 GET method

Dan komt de *GET method*, deze is uitgebreid doordat die veel dingen samenbrengt. Allereerst moeten we alles uit de database ophalen, hier wordt een opsplitsing gemaakt op het feit of er een visualId is meegegeven of niet. Dan volgt een *for each* die per visual kijkt. We houden de taginfo apart bij voor de volgende for each. We maken ook een download link van de visual op de s3 bucket zodat we de inhoud kunnen downloaden. Dan gaan we per tag per visual de info ophalen bij de verschillende system API's. Als laatste wordt alle info samengezet en omgevormd.

Hier zien we de GET visual flow die ervoor gaat zorgen dat we alle visuals te zien krijgen.



Figuur 26 GET visual flow

#### 5.2.4.4 Postman

Ook deze gaan we weer testen via postman.

Voorbeeld na de GET visual call. We zien hieronder alle informatie mooi gebundeld per visual. Alle eigenschappen van de database, de downloadlink en de data van de verschillende system API's worden getoond.

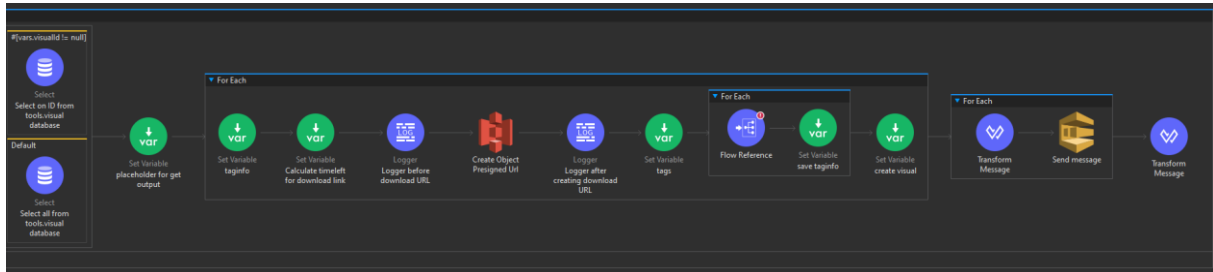
```
1 {
2   "identifier": 1,
3   "positions": "[1, 5]",
4   "type": "TEMPLATE",
5   "source": "https://smart-office-visuals.s3.eu-west-1.amazonaws.com/DatabaseTest.html?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230512T123444&X-Amz-SignedHeaders=host&X-Amz-Expires=600000",
6   "mode": "FADER",
7   "duration": 123
8 }
9
10
11 "schedules": [
12   {
13     "type": true,
14     "from": "1000-01-01T01:00:00",
15     "until": "10000-01-01T00:59:59",
16     "isFullDay": true,
17     "isRecurring": true,
18     "visibilityDuration": 500,
19     "recurringInterval": 5
20   }
21 ],
22 "location": "Herentals",
23 "data": [
24   {
25     "type": "EVENTTHISMONTH",
26     "items": [
27       {
28         "title": "Bewire Afterwork 8-9",
29         "link": "https://events.bewire.services/calendar/bewire-afterwork-8/",
30         "pubDate": "Mon, 08 May 2023 12:31:32 +0000",
31         "description": "17 mei 2023 - 17:30 <br /> Gezellig bijpraten met je collega's, herinneringen ophalen van voorbije events vergezeld van lekkere hapjes, biertjes, wijntjes",
32         "category": "Bewire Events",
33         "eventDate": "17 mei 2023 - 17:30"
34       }
35     ]
36   },
37   {
38     "type": "EVENTNEXTWEEK",
39     "items": [
40       {
41         "title": "Bewire Afterwork 8-9",
42         "link": "https://events.bewire.services/calendar/bewire-afterwork-8/",
43         "pubDate": "Mon, 08 May 2023 12:31:32 +0000",
44         "description": "17 mei 2023 - 17:30 <br /> Gezellig bijpraten met je collega's, herinneringen ophalen van voorbije events vergezeld van lekkere hapjes, biertjes, wijntjes",
45         "category": "Bewire Events",
46         "eventDate": "17 mei 2023 - 17:30"
47       }
48     ]
49   }
50 ]
51 }
```

Figuur 27 voorbeeld GET visual call

#### 5.2.4.4.1 Refresh method

Daarnaast hebben we ook nog de *refresh method*. Deze zal zoals de *GET method* ook alle info nodig hebben. Daarna zal hij de visuals op een *message queue* zetten. Je kan kiezen of je 1 visual of alle visuals op de *message queue* wilt zetten. Dit door middel van een queryparameter visualId.

Eerst wordt er gekeken of er een visualId wordt meegegeven. Zo kan een andere *SELECT query* gedaan worden voor de database.



Figuur 28 Refresh flow

#### 5.2.4.4.2 Scheduler

De *scheduler* zal ook de refresh flow uitvoeren, maar dan elke keer om drie uur 's nachts en enkel de flow om alle visuals op de *message queue* te zetten.

#### 5.2.4.5 Runtime manager

Als laatste wordt het anypoint project op de runtime manager gezet zodat deze weer niet enkel lokaal beschikbaar is.

### ● bewire-visual-papi

Domain: [bewire-visual-papi.de-c1.cloudhub.io](https://bewire-visual-papi.de-c1.cloudhub.io) · Last Updated 2023-04-28 10:15:00AM · 1 micro worker, using 4.4.0

Figuur 29 Runtime manager visual proces API

### 5.3 Experience API

We zijn begonnen met het aanmaken van een system API voor het weer. Daarna hebben we ervoor gezorgd via de proces API dat we gegevens op een database en s3 kunnen zetten, de info van alle visuals opvragen en dat we de visuals op een *message queue* kunnen zetten. Nu zijn we aangekomen bij het laatste gedeelte en dat zijn de experience API's. Deze moeten er voor zorgen dat specifieke methodes beschikbaar worden gesteld aan gebruikers of externe systemen. De back-office experience API zal 2 methodes beschikbaar stellen. De *GET method* en de *refresh method*, die werden aangemaakt binnen de visual proces API. Daarnaast zal de smart tv experience API ook 2 methodes beschikbaar stellen, namelijk de *refresh method* en de *GET method*, de tweede methode mag maar 1 visual tonen en dit doen door een visual ID mee te geven.

#### 5.3.1 Back office API

##### 5.3.1.1 RAML contract

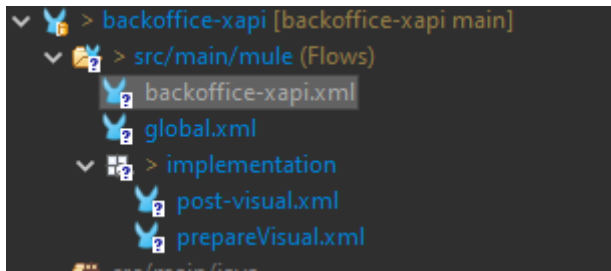
Hieronder zien we het RAML contract. Het datatype van de *POST method* kennen we al van de visual proces API. De *GET call* van de *refresh* zal dus kunnen kiezen of er een visualId wordt meegegeven of niet doordat *required: false* is.

```
1  #%RAML 1.0
2  title: backoffice-xapi
3
4  types:
5    visual: !include datatypes/visual.raml
6
7  /visual:
8    post:
9      body:
10       multipart/form-data:
11         type: visual
12         examples:
13           visual: !include examples/visual.raml
14       responses:
15         200:
16           body:
17             application/json:
18               example: !include examples/visualOK.raml
19     /refresh:
20       get:
21         queryParameters:
22           visualId:
23             required: false
24             type: string
25         responses:
26           200:
27             body:
28               application/json:
29                 example: {"message": "Visual(s) refreshed successfully."}
30           404:
31             description: The specified visual ID could not be found.
32
```

Figuur 30 RAML contract backoffice experience API

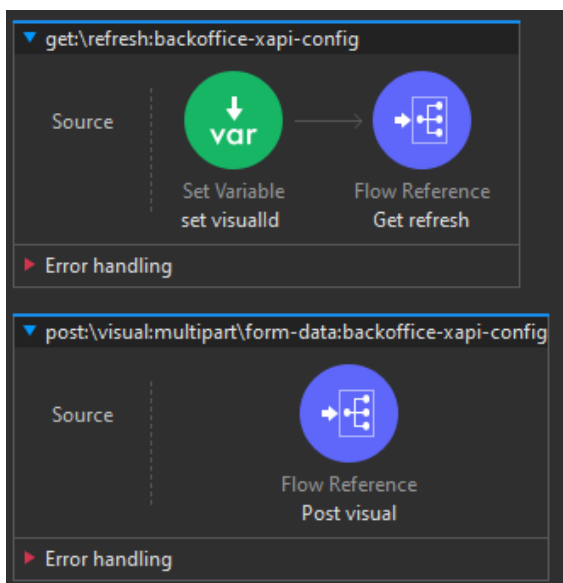
### 5.3.1.2 Anypoint studio

Hier zien we de mappenstructuur van de back office experience API. Zo zien we ook dat de experience API's kleiner zijn dan de proces API's.



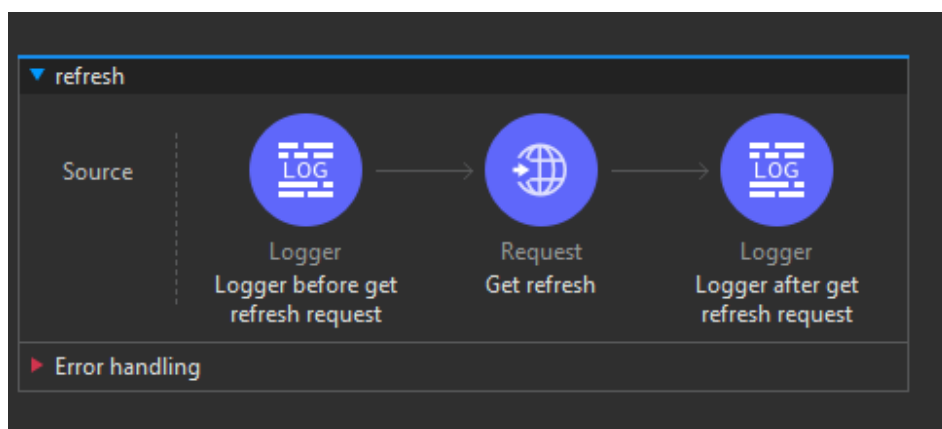
Figuur 31 mappenstructuur experience API

Hier zien we de twee *main flows* van de experience API, de *refresh* en *POST method*.



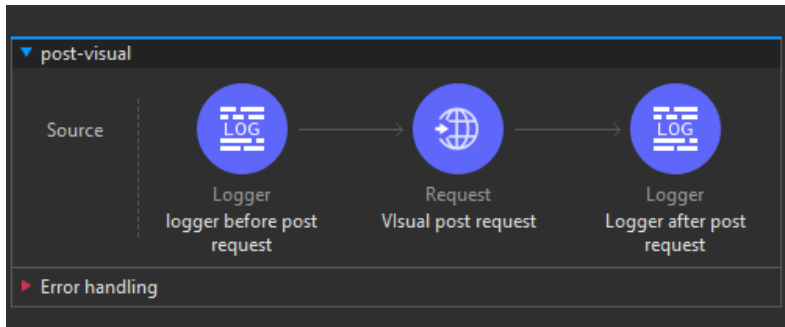
Figuur 32 main flows experience API

Bij de *refresh method* doen we een request naar de visual proces API. Dit kan met een visualId of zonder. Zo zal er ofwel 1 visual gerefreshed worden ofwel allemaal.



Figuur 33 refresh GET flow

Dit is de *POST* visual flow. Hier wordt een request gedaan naar de visual proces API om zo een visual te uploaden naar de s3 bucket.



Figuur 34 post visual flow

### 5.3.2 Smart tv experience API

#### 5.3.2.1 RAML contract

De smart-tv-experience API zal twee dingen moeten doen. Deze zal net als de backoffice experience API de *refresh method* moeten kunnen. Daarnaast zal deze ook een visual moeten kunnen opvragen aan de hand van een visualId.

```

1  #%RAML 1.0
2  title: smart-tv-xapi
3
4  types:
5    visuals: !include datatypes/visuals.raml
6
7  /refresh:
8    get:
9      queryParameters:
10        visualId:
11          required: false
12          type: string
13      responses:
14        200:
15          body:
16            application/json:
17              example: {"message": "Visual(s) refreshed successfully."}
18        404:
19          description: The specified visual ID could not be found.
20  /visual:
21    get:
22      queryParameters:
23        visualId:
24          type: string
25      responses:
26        200:
27          body:
28            application/json:
29              type: visuals[]
  
```

Figure 37 RAML contract smart-tv-experience API

#### 5.3.2.2 Anypoint studio

Hier zien we weer de mappen structuur van een experience API.

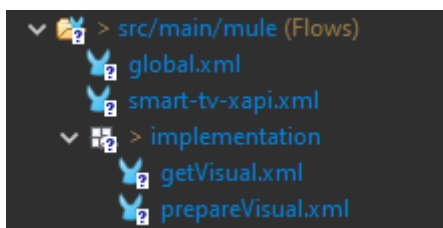


Figure 38 mappenstructuur smart-tv-experience API

Er zijn weer twee main *flows* toegevoegd aan het project, om zo de GET calls te kunnen doen naar de refresh of de visual *endpoints*. De *flow reference* verwijst weer naar de implementatie.

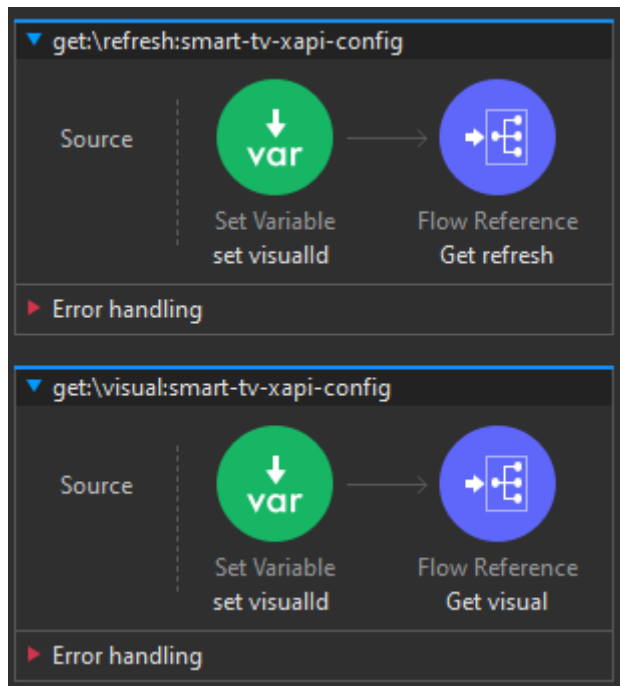


Figure 39 main flows smart-tv-experience API

De *refresh method* is exact hetzelfde als die van de bakcoffice experience API, dus deze ga ik niet meer tonen. Dan is het nog de GET visual implementatie. Deze ziet er ook hetzelfde uit maar doet een request naar de visual *endpoint* in de plaats van de refresh *endpoint*.

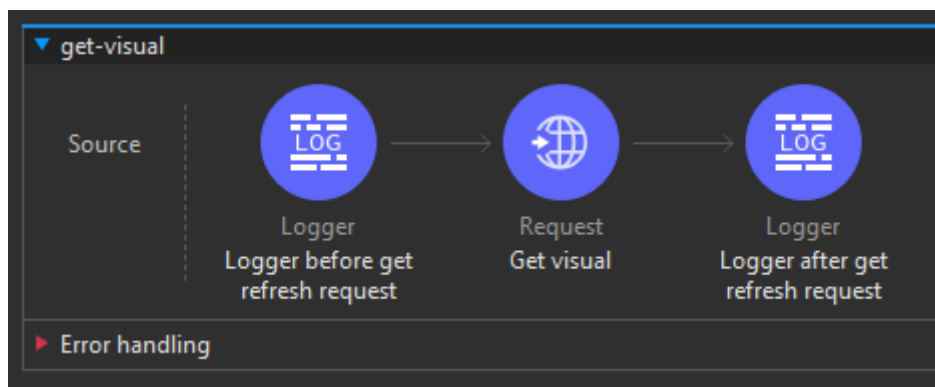


Figure 40 visual GET flow

Na het testen van de *flows* van beiden experience API's via postman zullen ook deze projecten online worden gezet via de runtime manager van *anypoint platform*. Zodat deze *endpoints* kunnen gebruikt worden voor verdere ontwikkeling.

## 6 Resultaat

We zijn begonnen met een system weather API, deze kan van een externe weather API alle gegevens opvragen en omvormen. Met de visual proces API willen we verschillende system API's samenbrengen in 1 geheel alsook alle info van de visuals tonen. Daarnaast is er ook een methode om nieuwe visuals te uploaden naar de database en s3 bucket. Als laatste zijn er 2 experience API's deze hebben verschillende methodes van de visual proces API. Deze zullen gebruikt worden voor verdere uitwerking van deze opdracht. Zo zal er nog een back office applicatie gemaakt moeten worden om er voor te zorgen dat alle eigenschappen van een visual worden toegevoegd aan de database. Daarnaast zal er ook nog een smart tv applicatie moeten gemaakt worden om effectief de visual op de tv te kunnen tonen.

## 7 Communicatie & opvolging

Het project werd opgevolgd aan de hand van wekelijkse *stand-ups*. Hier kon ik vragen stellen over het project die onduidelijk waren. Ook werd mijn code nagekeken aan de hand van *code reviews*. Hieruit kon ik veel leren. Ik heb veel gebruik gemaakt van de cursus *Mulesoft fundamentals* alsook de dataweave playground. Op 16 juni zal ik een presentatie geven voor mijn stagementor over het gerealiseerde project.

## 8 Besluit

De opdracht was om de implementatie te voorzien voor een *smart-office* systeem. Hierdoor heb ik een eerste ervaring opgedaan binnen de Mulesoft omgeving. Het was een parcours, van vallen en opstaan waar ik in mijn reflectieverslag dieper op zal ingaan. Naar mijn gevoel heb ik gerealiseerd wat er verwacht werd. Ik zou ook graag mijn stagementor Jorne van Helvert en mijn stagebegeleider Sarah Moeremans willen bedanken voor alle steun en hulp die ik heb gekregen van hun.