Stage Acerta

Realisaties

Bachelor in de toegepaste informatica
keuzerichting application development

Raf Bergs

Academiejaar 2022-2023

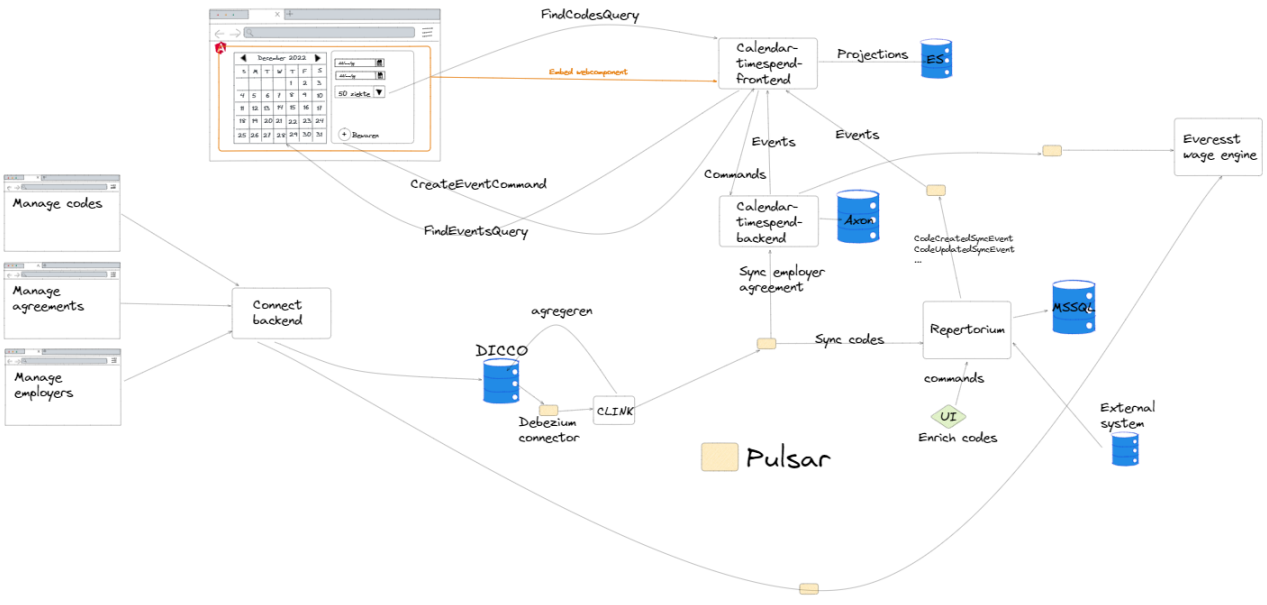
Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

Contents

1 STAGE OPDRACHT

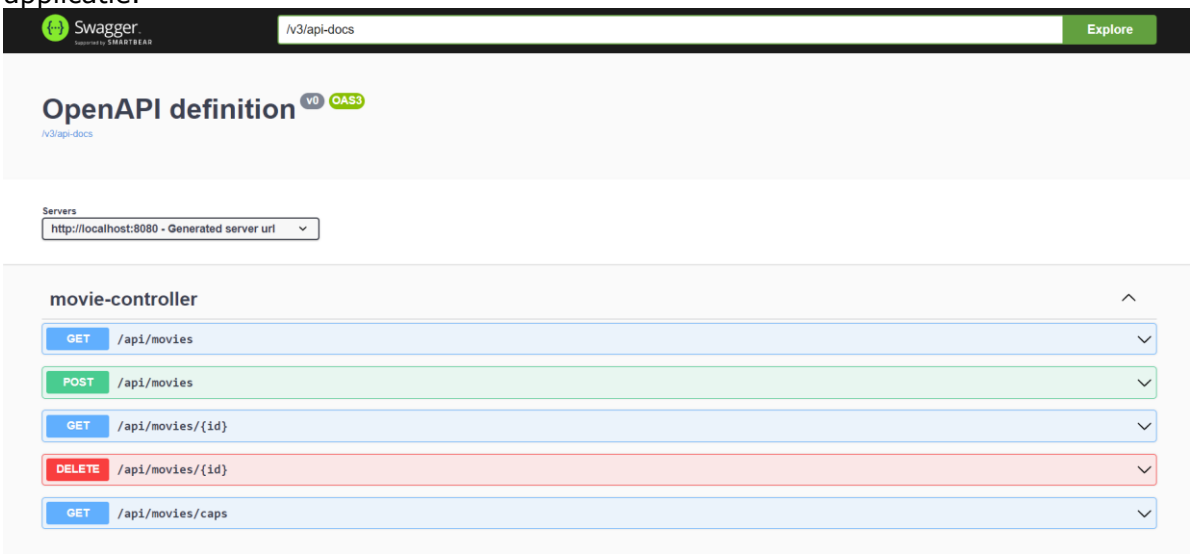
Acerta is een HR-bedrijf en biedt verschillende HR-tools aan. Deze stage opdracht was voor het connect evolution team binnen Acerta. Dit team is bezig aan de nieuwe versie van een kalender applicatie. In deze kalender kunnen werknemers aanduiden wanneer ze werken of iets anders doen, denk aan bijvoorbeeld aangeven wanneer iemand ziek is, en aan de hand van deze informatie kunnen de lonen berekend worden. Om aan te kunnen geven of er gewerkt werd of iets anders aan de hand was worden verschillende prestatie codes gebruikt. Al deze codes moeten natuurlijk beheert kunnen worden en hiervoor dient het repertorium dat in deze stageopdracht gemaakt werd. In deze repertorium applicatie moeten alle codes vanuit een database kunnen opgehaald worden, de codes moeten automatisch up-to-date blijven met deze andere database, de codes moeten aangepast kunnen worden binnen het repertorium zelf en tot slot zijn er een aantal zaken die niet in de masterdatabase staan en deze moeten toegevoegd kunnen worden aan deze codes via het repertorium. Deze stageopdracht focust op de backend van deze repertorium applicatie. Binnen Acerta kunnen klanten ook zelf nog bepaalde zaken aanpassen aan codes en deze klant specifieke codes worden apart opgeslagen, maar deze klant specifieke codes vallen buiten de scope van deze stageopdracht.

Op onderstaande afbeelding wordt een schema getoond van de omgeving waarin de repertorium applicatie zal werken. Het repertorium bevindt zich rechtsonder in dit schema. Aan dit repertorium hangen verschillende zaken namelijk: een UI, databasen en een verbinding met Pulsar. Eén van deze databasen zal dienen om de prestatie codes in het repertorium op te slaan en de andere database dient om eventueel data van ergens anders op te halen als dit nodig zou zijn. De UI zal dienen om het beheren van de codes in het repertorium te vergemakkelijken en om extra data toe te voegen aan codes. De verbinding met pulsar dient om events binnen te halen in het repertorium wanneer er een verandering gebeurt in de database die aanzien wordt als master van de data en dan kan via dit event de nodige aanpassingen gebeuren in het repertorium. Deze oude database (DICCO) zien we linksonder in het schema staan. Dit is de database die gebruikt wordt door de oudere applicatie (connect). Rechtsboven in het schema zien we dan de nieuwe applicatie, in de vorm van de geschetste kalender. Als we dit schema volgen, vertrekkende vanuit de nieuwe kalender, zien we dat we uiteindelijk in het repertorium terecht komen wanneer we informatie over codes gaan opvragen.



2 STUDIES

Voor het begin van de stage opdracht was er een korte periode waarin nagekeken werd of alle kennis die nodig was voor de stageopdracht aanwezig was. Dit begon met een korte test waarbij een CRUD-applicatie gemaakt moest worden om een lijst van filmen te beheren. Op onderstaande afbeelding zien we de resulterende swagger van deze test applicatie.



Nadat deze test applicatie gemaakt was werd er tijd voorzien om een aantal cursussen rond spring(boot), maven en git door te nemen.

3 DATABASE ONDERZOEK

Bij dit onderdeel moest er onderzocht worden hoe de database van de huidige applicaties zich gedraagt en dit gedrag moest dan neergeschreven worden in een document. Bij dit onderzoek zijn verschillende scenario's overlopen om heel wat gedrag van de database gedocumenteerd te krijgen. Dit onderzoek ging niet direct goed en er waren dus twee mindere pogingen voordat de derde poging de goede was. Om dit aan te tonen wordt hieronder drie keer hetzelfde voorbeeld, een kleur aanpassen in een klant specifieke code, besproken, maar dan wel één keer per poging om de vooruitgang weer te geven.

In de eerste poging werd eerst de query zelf getoond gevolgd door screenshots van de resultaten van deze query. In deze eerste poging missen een aantal zaken, namelijk: een schets van een event dat gemaakt moet worden, meer context rond welke aanpassing er juist gebeurd is en het resultaat in de tabellen van het repertorium. In deze versie zit op zich alle nodige informatie van de scenario's, maar deze versie is niet duidelijk genoeg.

YETI kleurknop -> paars

```
select top 10 * from GBTSSR01 where RPT_DEF_CD = '0031' ORDER BY MUT_TS DESC;
```

	RPT_DEF_NO	RPT_DEF_CD	SRC_NO	PTT_TYPE_IC	SLR_TYPE_IC	RST_TYPE_IC	STT_DT
1	00001111684	0031	00800002836	1	1	0	<null>
2	90000000007	0031	00000000000	1	1	0	1800-01-01
3	00001111033	0031	00800003167	0	1	0	<null>
4	00000008344	0031	00000000370	1	1	0	<null>
5	00000008336	0031	00000000651	1	1	0	<null>
6	00000008279	0031	00000000445	1	1	0	<null>
7	00000004693	0031	00000000624	1	1	0	<null>
8	00000004689	0031	00000000623	1	1	0	<null>
9	00000004646	0031	00000000618	1	1	0	<null>
10	00000004376	0031	00000000763	1	1	0	<null>

END_DT	PTY_NO	CPN_ORG_NO	REG_USER_CD	REG_TS	MUT_USER_CD	MUT_TS
<null>	00001604368	589277	GA07949	2023-03-17 16:03:28.600820	GA07949	2023-03-17 16:03:28.600820
9999-12-31	00000000000	0000000	upload	2014-05-02 00:00:00.000000	GA07949	2023-03-17 13:56:10.821622
<null>	00000160522	101311	JD42042	2021-11-18 14:52:22.295240	JD42042	2021-11-18 14:58:59.360514
<null>	00011476555	52236	JD42042	2016-11-25 14:22:38.796922	JD42042	2016-11-25 14:22:38.796922
<null>	00011833246	55726	JD42042	2016-11-25 14:10:30.632142	JD42042	2016-11-25 14:10:30.632142
<null>	00011345702	340874	JD42042	2016-11-25 13:58:32.004031	JD42042	2016-11-25 13:58:32.004031
<null>	00012679519	758108	JD42042	2016-11-23 14:51:28.173262	JD42042	2016-11-23 14:51:28.173262
<null>	00012149212	758108	JD42042	2016-11-23 14:51:07.370216	JD42042	2016-11-23 14:51:07.370216
<null>	00011553239	758108	JD42042	2016-11-23 14:48:57.644701	JD42042	2016-11-23 14:48:57.644701
<null>	00012294751	52812	JD42042	2016-11-23 14:36:41.894683	JD42042	2016-11-23 14:36:41.894683

```
select top 10 * from GBTSSR02 where RPT_DEF_NO= '90000000007'ORDER BY MUT_TS DESC;
```

	RPT_DEF_NO	STS_CD	BTN_CLD_PST_NO	BTN_CLR_CD	FONT_CLR_CD	GRP_CD	WRK_SDL_USE_CD	CLT_AJT_USE_CD	IDV_AJT_USE_CD
1	90000000007	01	0	8CFA3B	000000	A	0	0	0

	PSD_USE_CD	DEF_TLT_NO	FSC_RPT_DEF_NO	REG_USER_CD	REG_TS	MUT_USER_CD	MUT_TS	SLR_EGN_U
1	0	00000035390	<null>	GA07949	2023-03-17 13:56:10.822764	GA07949	2023-03-17 15:47:36.512876	3

```
select top 10 * from GBTSSR10 where DEF_TLT_NO = '00000035390' ORDER BY MUT_TS DESC;
```

	DEF_TLT_NO	LGG_CD	SHT_DSB_TX	LNG_DSB_TX	BTN_LBL_TX	CRT_LBL_TX	ESS_PTT_DSB_TX	ESS_SLR_DSB_TX
1	00000035390	EN			vacation			
2	00000035390	FR						
3	00000035390	NL			vakanties	V		

REG_USER_CD	REG_TS	MUT_USER_CD	MUT_TS
GA07949	2023-03-17 13:56:10.827068	GA07949	2023-03-17 15:47:36.515269
GA07949	2023-03-17 13:56:10.826414	GA07949	2023-03-17 15:47:36.514637
GA07949	2023-03-17 13:56:10.825646	GA07949	2023-03-17 15:47:36.513833

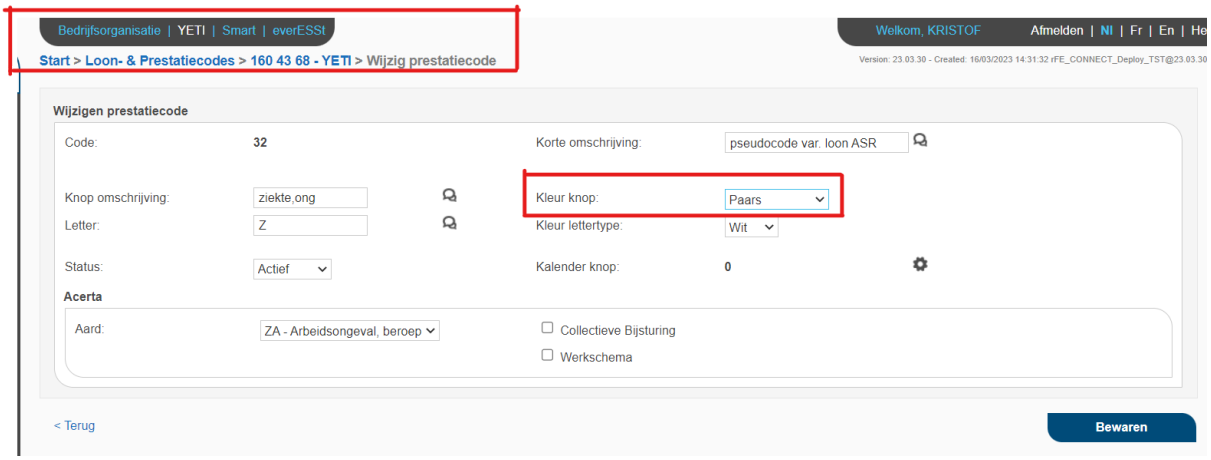
```
select top 10 * from GBTSSR02 where RPT_DEF_NO= '00001111684'ORDER BY MUT_TS DESC;
```

	RPT_DEF_NO	STS_CD	BTN_CLD_PST_NO	BTN_CLR_CD	FONT_CLR_CD	GRP_CD	WRK_SDL_USE_CD	CLT_AJT_USE_CD	IDV_AJT_USE_CD
1	00001111684		<null>	9736F9					

	PSD_USE_CD	DEF_TLT_NO	FSC_RPT_DEF_NO	REG_USER_CD	REG_TS	MUT_USER_CD	MUT_TS	SLR_EGN_US
1			<null>	GA07949	2023-03-17 16:03:28.604045	GA07949	2023-03-17 16:03:28.620974	

In de tweede versie werd er al meer context gegeven rond wat er juist werd aangepast in de oude applicatie door middel van een screenshot. Hierbuiten lijkt deze versie nog steeds sterk op de eerste versie en zijn er dus nog altijd bepaalde zaken die ontbreken.

Klant (YETI) specifieke code 0032 één aanpassing



Query 11:

```
select top 10 * from GBTSSR01 where RPT_DEF_CD = '0032' ORDER BY MUT_TS DESC;
```

	RPT_DEF_NO	RPT_DEF_CD	SRC_NO	PTT_TYPE_IC	SLR_TYPE_IC	RST_TYPE_IC	STT_DT
1	00001111685	0032	00800002836	1	1	0	<null>
2	90000000661	0032	00000000000	1	1	0	1800-01-01
3	00000034627	0032	00000000552	1	1	0	<null>
4	00000004673	0032	00000000621	1	1	0	<null>
5	00000004599	0032	00000000612	1	1	0	<null>

END_DT	PTY_NO	CPN_ORG_NO	REG_USER_CD	REG_TS	MUT_USER_CD	MUT_TS
<null>	00001604368	589277	GA07949	2023-03-20 11:36:29.120351	GA07949	2023-03-20 11:36:29.120351
9999-12-31	00000000000	0000000	upload	2014-09-01 00:00:00.000000	GA07949	2023-03-20 10:22:20.967354
<null>	00011466391	370221	JD33078	2018-11-30 15:10:03.706374	JD33078	2018-11-30 15:10:03.706374
<null>	00012146314	758108	JD42042	2016-11-23 14:50:28.047250	JD42042	2016-11-23 14:50:28.047250
<null>	00011681697	753418	JD42042	2016-11-23 14:40:54.084469	JD42042	2016-11-23 14:40:54.084469

Nieuwe rij voor YETI code.

In de laatste versie zijn er verschillende dingen verandert. Om te beginnen is er een voor situatie aanwezig waarin aangetoond wordt wat de situatie van alle tabellen was voor dat de aanpassing aangebracht werd. Dit wordt nu ook weergegeven in een tabel in plaats van een screenshot en dit helpt met de leesbaarheid. De screenshot die context biedt van in de tweede versie is ook terug aanwezig. Iets dat afwezig is zijn de query's om de resultaten te tonen, omdat deze geen meerwaarde bieden in dit soort document. Dit document dient om de resultaten van verschillende aanpassing aan te tonen en hiervoor is de query op zich niet nodig. Een schets van een mogelijk event is nu ook toegevoegd aan elke aanpassing, deze schets kan later gebruikt worden om het daadwerkelijke event op te baseren. We zien nu ook een reeks tabellen voor de situatie na de aanpassing en in deze tabellen werd alles dat verandert is aangeduid met een kleur om deze verandering makkelijk op te merken. Zaken die veranderd waren of toegevoegd werden als resultaat van de aanpassing in de oude applicatie werden met groen gemarkeerd en zaken die verwijderd waren als resultaat van de aanpassing werden gemarkeerd met rood. In onderstaand voorbeeld werden er enkel zaken aangepast dus zien we enkel de groene kleur. Tot slot zien we nog twee tabellen, deze twee geven weer wat er juist opgeslagen zou worden in het repertorium aan de hand van het event dat werd geschetste.

Kleur aanpassen voor klantspecifieke(manus)

Before:

SSR01

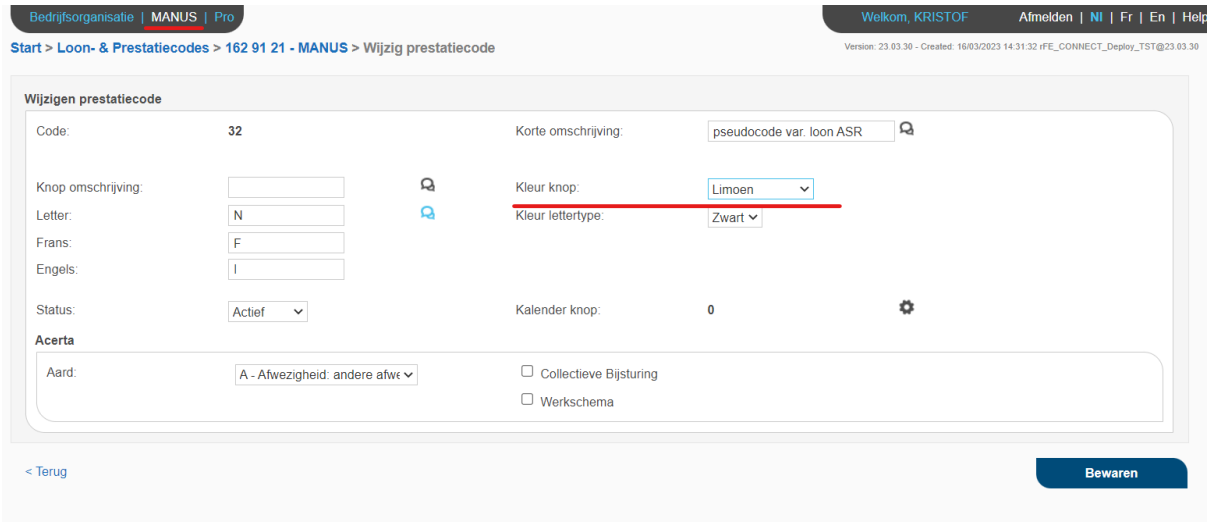
RPT_DEF_NO	RPT_DEF_ CD	STT_DT	END_DT	MUT_TS	PTT_IC	SLR	partyId
00001111689	0032	null	null	2023-03-24 14:42:11.108373	1	1	00001629121

SSR02

ID	Status	Color	Kind	Work	Col	Indv	labelId	Engine	Mut_ts
00001111689							00000035396		2023-03-24 14:42:11.110865

SSR10

ID	languageCode	Short	letter	MUT_TS
00000035396	NL		N	2023-03-24 14:42:11.111404



After:

SSR01

RPT_DEF_NO	RPT_DEF_CD	STT_DT	END_DT	MUT_TS	PTT_IC	SLR	partyId
00001111689	0032	null	null	2023-03-24 15:52:43.143946	1	1	00001629121

SSR02

ID	Status	Color	Kind	Work	Col	Indv	labelId	Engine	Mut_ts
00001111689		8CFA3B					00000035396		2023-03-24 15:52:43.139364

SSR10

ID	languageCode	Short	letter	MUT_TS
00000035396	NL		N	2023-03-24 15:52:43.141522

Event:


```
{
  "@type": "ConnectPrestationCodeSynchronizationEvent",
  "code": "0032",
  "codeType": "CONTEXT",
  "employerId": "00001629121",
  "kind": "ABSENCE",
  "fromDate": null,
  "untilDate": null,
  "color": "8CFA3B",
  "isIndividualAdjustment": null,
  "isCollectiveAdjustment": null,
  "isWorkschedule": null,
  "salaryEngines": null,
  "labels": [
    {
      "languageCode": "NL",
      " mediumDescription ": null,
      "letter": "N"
    }
```

```
    ]
}


}
```

RepertoriumService:

Prestation_code

ID	Code	Type	Engine	Context	employerId	Kind	From	Until	Color	allowHours	allowPercentages	Indv	Col	Work	Icon	iconColor	iconFam
900000006611	0032	GENERAL	null	null	null	ABSENCE	1800-01-01	9999-12-31	FC251B	false	false	false	false	false	null	null	null
900000006611-EVERESST	0032	WAGE_ENGINE	EVERESST	CONNECT	null	null	null	null	null	null	null	null	null	null	null	null	null
900000006611-CONNECT-EVERESST	0032	CONTEXT	EVERESST	CONNECT	null	null	null	null	null	null	null	null	null	null	null	null	null
900000006611-EVERESST-manus	0032	CONTEXT	EVERESST	CONNECT	00001629121	null	null	null	8CFA3B	null	null	null	null	null	null	null	null

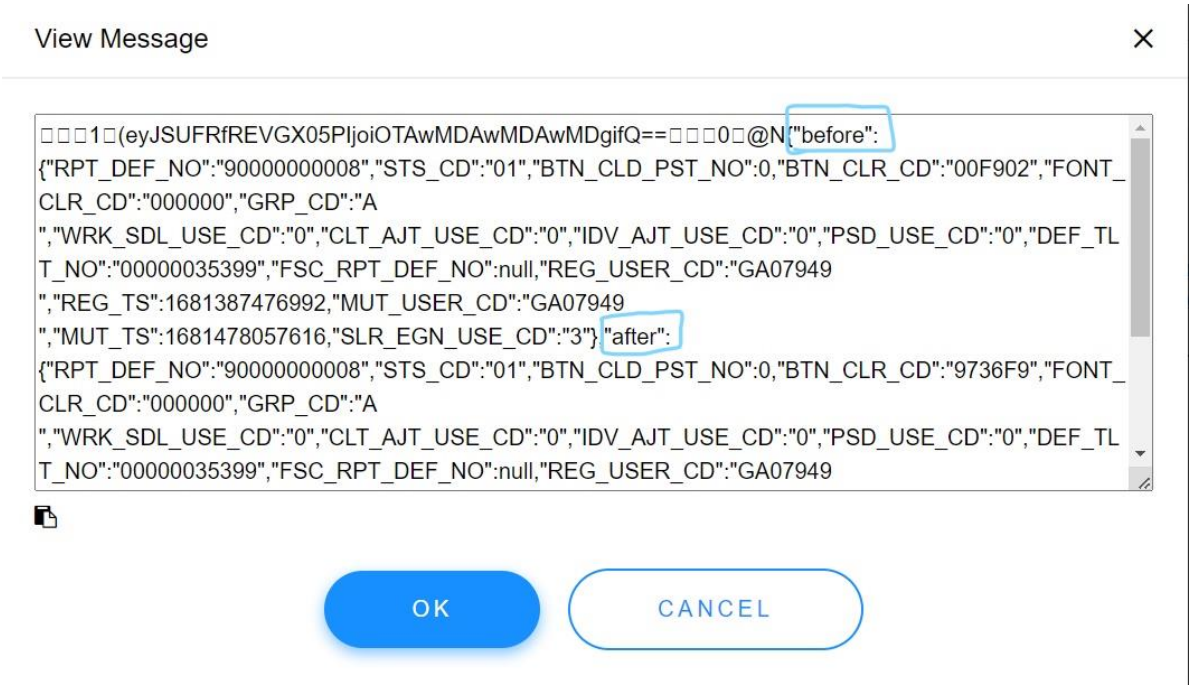
Prestation_code_label

Id	PrestationCodeId	languageCode	Short	Medium	Long	letter
1	90000000661	EN	null	pseudocode var salary ASR	null	I
2	90000000661	FR	null	pseudocode sal. Var. DRS	null	F
3	90000000661	NL	null	pseudocode var. loon ASR	null	null
4	90000000661	DE	null	Pseudokode var Gehalt MSR	null	null
5	90000000661-manus	NL	null	null	null	N

4 CLINK

Clink is de applicatie die gebruikt wordt om events te maken en deze op pulsar te zetten wanneer er een aanpassing in de database met prestatie codes gedetecteerd wordt. Deze applicatie bestond al voor het begin van de stageopdracht, maar aangezien voor de stageopdracht de prestatie codes nodig zijn moest clink uitgebreid worden met de mogelijkheid om events te maken voor deze prestatie codes.

Hieronder zien we een voorbeeld van een change data capture bericht op pulsar. In de twee blauwe vakjes zien we before en after staan. Dit is dus de status van een bepaalde tabel voor en na een aanpassing. Eens dat we dit bericht dan in clink terecht komt weet clink dat er iets veranderd is in deze kolom en dat clink een query moet uitvoeren om alle nodige data op te halen om een nieuw event aan te maken.



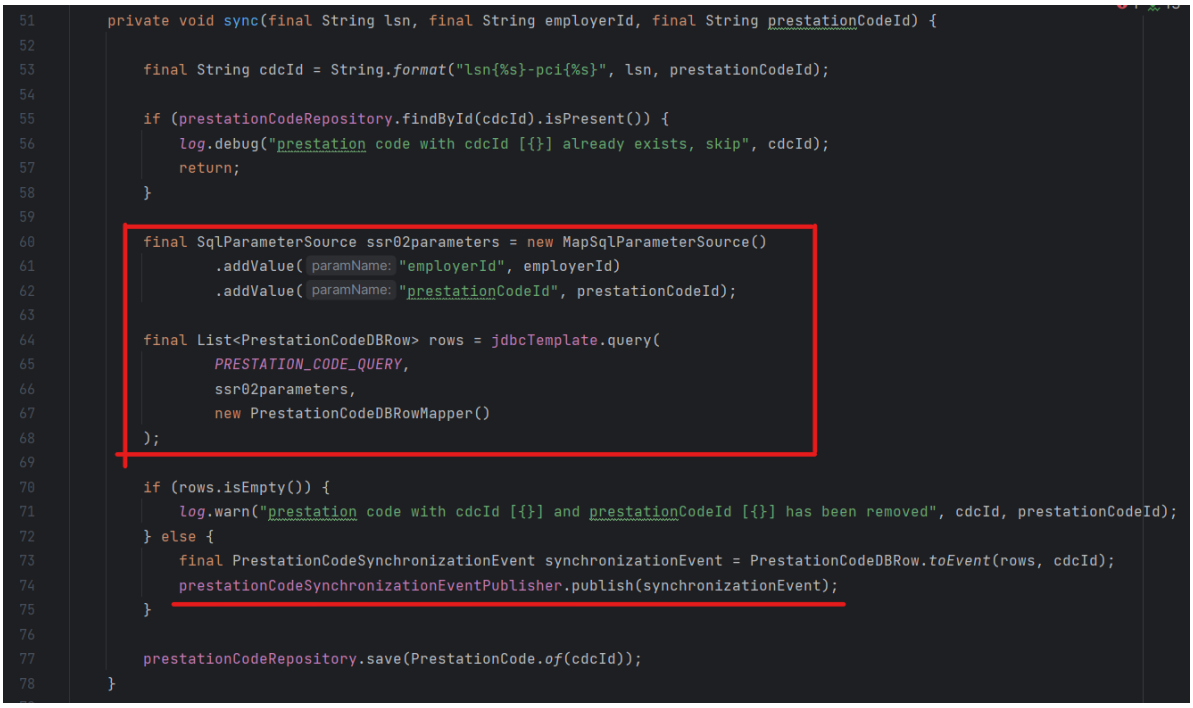
In de code van clink kunnen we hier zien dat een pulsar listener gebruikt wordt om dit bericht binnen te halen. Eens dat het bericht binnen gehaald is in clink worden er een paar checks uitgevoerd om na te gaan of het een generieke code of een klant specifieke is en om te zien of het event ook daadwerkelijk nog bestaat. Na deze checks wordt één van de twee synchronisatie methodes aangeroepen, de generieke methode voor een generieke code en de klant specifieke methode voor de klant specifieke code, en in deze methodes wordt de query uitgevoerd en wordt het nodige event gemaakt.

```java
@PulsarListener(consumerName = "Ssr02MessageHandler",
        topic = "${pulsar.consumers.Ssr02MessageHandler.subscription.topic}",
        schemaType = CdcEvent.class)
public void on(final CdcEvent event) {
    log.info("Handle Ssr02 event [{}]", event.toString());

    if (event.isDeleted()) {
        log.error("code with id [{}] has been deleted, skip", event.getBefore().get(PRESTATION_CODE_ID));
        return;
    }
    final String prestationCodeId = event.getAfter().get(PRESTATION_CODE_ID);
    final String employerId = lookupService.findEmployerIdByPrestationCodeId(prestationCodeId);

    if (ACERTA_EMPLOYER_ID.equals(employerId)) {
        prestationCodeSynchronizationService.syncGeneralCode(event.getStartLsn(), prestationCodeId);
    } else {
        prestationCodeSynchronizationService.syncEmployerCode(event.getStartLsn(), employerId, prestationCodeId);
    }
}
}
```

Als we dan gaan kijken naar wat het verschil tussen deze twee synchronisatie methodes is kunnen we zien dat bij de generieke code een Acerta employer id wordt meegegeven, aangezien alle generieke codes deze zelfde employer id hebben, en bij de klant specifieke code wordt dan de employer id van deze specifieke klant meegegeven.

```java
public synchronized void syncGeneralCode(final String lsn, final String prestationCodeId) {
    sync(lsn, ACERTA_EMPLOYER_ID, prestationCodeId);
}

1 usage  Raf Bergs +1
public synchronized void syncEmployerCode(final String lsn, final String employerId, final String prestationCodeId) {
    if (!syncFilter.isEmployerAllowed(employerId)) {
        log.debug("Employer with id [{}] not allowed to sync, skip", employerId);
        return;
    }
    sync(lsn, employerId, prestationCodeId);
}
```

In beide methodes wordt dezelfde sync methode opgeroepen en deze methode dient dan om het event aan te maken en vervolgens dit event op pulsar te zetten zodat andere applicaties, in het geval van de prestatie codes is dit de repertorium applicatie, hiervan gebruik kunnen maken. In deze sync methode zien we dat er eerst een cdcId gedefinieerd wordt om zo elk event te kunnen identificeren. Hierna wordt weer nagegaan of dit event al zou bestaan of niet, als het event al bestaat zouden we niets meer moeten doen en wordt dit event geskipt. Hierna gebruiken we de informatie die we van pulsar binnen hebben gekregen om een query op te stellen, deze query is nodig aangezien het bericht op pulsar over één tabel gaat waarvan we in het database onderzoek gezien hebben dat deze bij eender welke aanpassing ook aangepast wordt, maar we hebben uiteindelijk informatie uit drie verschillende tabellen nodig. Deze query dient dus om alle nodige informatie uit de drie tabellen te halen en dan kunnen we al deze informatie samen in één event steken. Eens dat de parameter waarden van de query zijn ingevuld wordt deze uitgevoerd met deze parameters. Nadat deze query is uitgevoerd wordt er nog eens nagekeken of deze query daadwerkelijk iets terug gegeven heeft en hierna wordt deze data omgezet in een event en op pulsar gezet.



```
51      private void sync(final String lsn, final String employerId, final String prestationCodeId) {
52
53          final String cdcId = String.format("lsn{%s}-pci{%s}", lsn, prestationCodeId);
54
55          if (prestationCodeRepository.findById(cdcId).isPresent()) {
56              log.debug("prestation code with cdcId [{}] already exists, skip", cdcId);
57              return;
58          }
59
60          final SqlParameterSource ssr02parameters = new MapSqlParameterSource()
61                  .addValue( paramName: "employerId", employerId)
62                  .addValue( paramName: "prestationCodeId", prestationCodeId);
63
64          final List<PrestationCodeDBRow> rows = jdbcTemplate.query(
65                  PRESTATION_CODE_QUERY,
66                  ssr02parameters,
67                  new PrestationCodeDBRowMapper()
68          );
69
70          if (rows.isEmpty()) {
71              log.warn("prestation code with cdcId [{}] and prestationCodeId [{}] has been removed", cdcId, prestationCodeId);
72          } else {
73              final PrestationCodeSynchronizationEvent synchronizationEvent = PrestationCodeDBRow.toEvent(rows, cdcId);
74              prestationCodeSynchronizationEventPublisher.publish(synchronizationEvent);
75          }
76
77          prestationCodeRepository.save(PrestationCode.of(cdcId));
78      }
```

Hieronder zien we de query. In deze query zien we op het einde in het WHERE-gedeelte ook nog twee parameters staan, we kunnen deze herkennen aan het dubbel punt dat voor hen staat, en dit zijn de parameters die ingevuld worde in onze code.

```
WITH COD$ as (SELECT COD.CODE_WD, COD2.TAAL_KD, COD2.KORT_OMS_TK
              FROM GBTACD11 COD11
                       JOIN GBTACD01 COD on COD.CODE_DEF_IDEN_NR =
COD11.CODE_DEF_IDEN_NR
                       LEFT JOIN GBVACD02 COD2 on COD.CODE_IDEN_NR =
COD2.CODE_IDEN_NR
              WHERE RTRIM(COD11.CODE_DEF_NM) =
'BV7_RepertoryDefinitionCode')
select CODE.RPT_DEF_NO
PRESTATION_CODE_ID,
       CODE.STT_DT
FROM_DATE,
       CODE.END_DT
UNTIL_DATE,
       CODE.PTY_NO
EMPLOYER_ID,
       RTRIM(CODE.RPT_DEF_CD)
CODE,
       RTRIM(PRESTATION.GRP_CD)
KIND,
       NULLIF(RTRIM(PRESTATION.BTN_CLR_CD), '')
COLOR,
       PRESTATION.IDV_AJT_USE_CD
IS_INDIVIDUAL_ADJUSTMENT,
       PRESTATION.CLT_AJT_USE_CD
IS_COLLECTIVE_ADJUSTMENT,
       PRESTATION.WRK_SDL_USE_CD
IS_WORKSCHEDULE,
       PRESTATION.SLR_EGN_USE_CD
ENGINE,
       COD.TAAL_KD
LANGUAGE_CODE,
       coalesce(RTRIM(NULLIF(LABEL.SHT_DSB_TX, '')),
RTRIM(COD.KORT_OMS_TK)) SHORT_DESCRIPTION,
       nullif(RTRIM(LABEL.CRT_LBL_TX), '') LETTER
```

```
from GBVSSR01 CODE
         JOIN COD$ COD on CODE.RPT_DEF_CD = COD.CODE_WD
         LEFT JOIN GBVSSR02 PRESTATION on CODE.RPT_DEF_NO =
PRESTATION.RPT_DEF_NO
         LEFT JOIN GBVSSR10 LABEL on PRESTATION.DEF_TLT_NO =
LABEL.DEF_TLT_NO and COD.TAAL_KD = LABEL.LGG_CD
where CODE.PTY_NO = :employerId
  and CODE.RPT_DEF_NO = :prestationCodeId;
```

Aangezien we verschillende talen hebben voor de omschrijvingen van de codes krijgen we met onze query verschillende rijen terug als resultaat, maar deze verschillende rijen hebben dezelfde basisinformatie en het enige wat verschilt zijn dus deze omschrijvingen. Om dit op te lossen gebruiken we de toEvent methode. In deze methode wordt eerst de basisinformatie uit de eerste rij die we terugkrijgen gehaald en daarna lopen we over alle rijen om telkens hun omschrijvingen met de nodige informatie zoals hun taal code en letter in een lijst te steken en deze lijst komt dan in het event terecht.

```
static PrestationCodeSynchronizationEvent toEvent(List<PrestationCodeDBRow> rows, String cdcId) {
    PrestationCodeDBRow prestationCode = rows.get(0);
    return PrestationCodeSynchronizationEvent.of(
            prestationCode.getId(),
            cdcId,
            prestationCode.getCode(),
            prestationCode.getEmployerId(),
            PrestationCodeDBRow.mapToKind(prestationCode.getKind()),
            prestationCode.getFromDate(),
            prestationCode.getUntilDate(),
            prestationCode.getColor(),
            prestationCode.isIndividualAdjustment(),
            prestationCode.isCollectiveAdjustment(),
            prestationCode.isWorkSchedule(),
            PrestationCodeDBRow.mapToEngines(prestationCode.getWageEngines()),
            rows.stream() Stream<PrestationCodeDBRow>
                    .map(row -> PrestationCodeDescription.of(
                            LanguageCode.valueOf(row.getLanguageCode()),
                            row.getShortDescription(),
                            row.getLetter()
                    )) Stream<PrestationCodeDescription>
                    .toList()
    );
}
```

Tot slot zien we hieronder dan een voorbeeld van een event dat op pulsar gepubliceerd is. Hier kunnen we dan zien dat alle basisinformatie er maar één keer in staat en dat alle omschrijvingen in één lijst staan als deel van het event. Onderaan het event kunnen we ook nog zien dat er @Type staat en dit dient om aan te geven wat soort event het is aan de applicaties die dit event ontvangen zodat zij weten wat er precies in zit. En de nodige omzettingen kunnen doen tussen Json en Java.

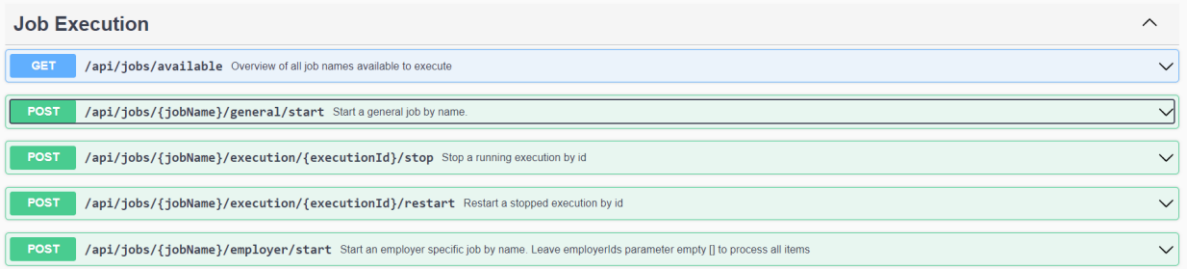

```
□□□1
□
□publishedBy□□clink□□00001111696□□□(□□□□□00□@□{
 "prestationCodeId" : "00001111696",
 "cdcId" : "lsn{000154140001DD240022}-pci{00001111696}",
 "code" : "0033",
 "employerId" : "00001629121",
 "color" : "00F902",
 "isIndividualAdjustment" : false,
 "isCollectiveAdjustment" : false,
 "isWorkSchedule" : false,
 "descriptions" : [ {
  "languageCode" : "DE",
  "shortDescription" : "Trigger tweede berek. vak.geld",
  "letter" : null
 }, {
  "languageCode" : "EN",
  "shortDescription" : "Trigger tweede berek. vak.geld",
  "letter" : "E"
 }, {
  "languageCode" : "FR",
  "shortDescription" : "Trigger deuxième cal. péc.vac.",
  "letter" : "F"
 }, {
  "languageCode" : "NL",
  "shortDescription" : "Trigger tweede berek. vak.geld",
  "letter" : null
 } ],
 "@type" : "PrestationCodeSynchronizationEvent"
}
```


Dus nu kunnen we zien dat als er iets verandert in de oude applicatie (connect) dat deze verandering automatisch opgemerkt zal worden en dan zal clink hiervoor een event maken dat dan weer gebruikt kan worden door het repertorium en zo zal het repertorium gesynchroniseerd blijven met DICCO, de oude database, aangezien deze de master van de data blijft. Er zijn ook nog default waarden voor de beschrijvingen van codes, maar deze staan in een andere tabel en moeten we dus een tweede change data capture gebruiken om deze op te halen. De code voor dit allemaal te regelen wordt hier niet besproken aangezien deze hetzelfde is als de code voor de prestatie codes, maar dan met andere data in het event.

5 INITIAL LOAD

De initial load applicatie lijkt heel hard op de clink applicatie. Beide applicaties dienen om events te maken. Het grote verschil tussen beide applicaties is dat de inital load applicatie dient om een groot deel aan codes tegelijkertijd op te vragen met de query, dit is dezelfde query als in clink gebruikt werd, en dan een hele reeks aan events op pulsar te zetten. Deze applicatie dient zoals de naam zegt om een groot aantal data initieel te laden voor een andere applicaties. Voor deze stageopdracht werd deze applicatie dus uitgebreid met de mogelijk om generieke codes op te kunnen vragen. De klant specifieke codes vallen buiten scope voor deze stageopdracht dus hiervoor werd een start gemaakt, maar is niet volledig afgewerkt.

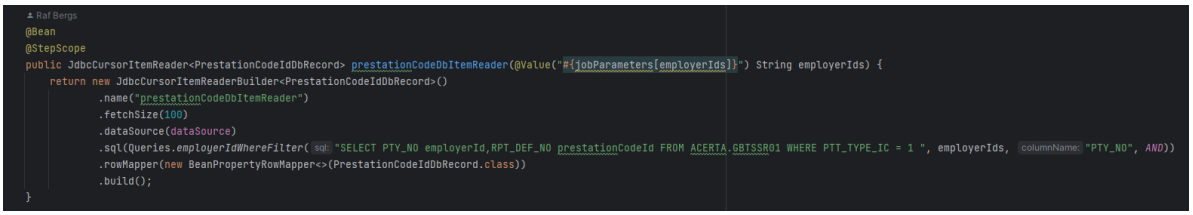
De initial load applicatie heeft geen front-end en om deze te gebruiken wordt er gebruik gemaakt van een swagger UI. In deze UI zien we drie controllers staan, voor deze stageopdracht moest één van deze drie aangepast worden om met de prestatiecodes om te kunnen gaan. In deze swagger kunnen we vijf methodes zien. Al deze vijf methodes gaan over jobs, een job gaat altijd over welke data precies geladen moet worden met de initial load applicatie in het geval van het repertorium zullen dit dus prestatie codes zijn. Voor het repertorium bestond waren al deze jobs klant specifiek, maar omdat het repertorium ook generieke codes heeft en niet enkel klant specifieke codes zijn deze jobs opgesplitst in general en employer jobs.



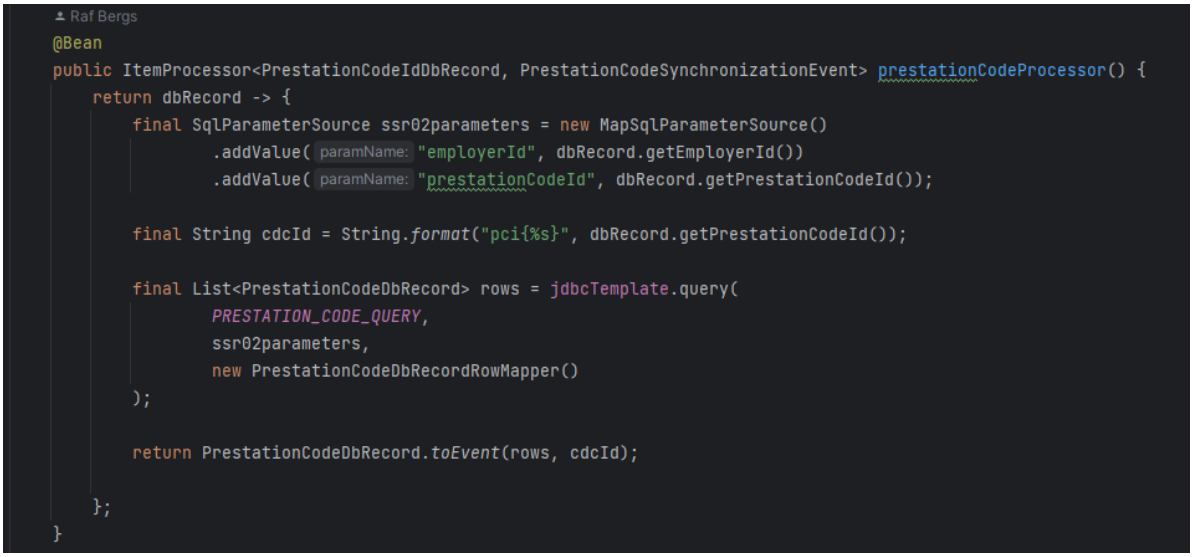
Wanneer we naar de code gaan kijken die uitgevoerd wordt wanneer de general job van prestatie codes gestart wordt kunnen we direct de gelijkenis met clink zien. In onderstaande screenshot kunnen we de verschillende stappen die uitgevoerd worden bij het opstarten van de job. Als we naar de code van deze stappen gaan kijken zien we al snel de gelijkenis tussen de clink applicatie en de initial load applicatie. Aangezien de intial load grote aantallen aan data tegelijkertijd moet opvragen wordt er in de inital load gebruik gemaakt van spring batch.

```
Raf Bergs +1
@Bean
public Step syncPrestationCode() {
    return new StepBuilder( name: "syncPrestationcode", jobRepository)
            .<~>chunk( chunkSize: 100, transactionManager)
            .reader(prestationCodeDbItemReader(OVERRIDDEN_BY_EXPRESSION))
            .processor(prestationCodeProcessor())
            .writer(prestationCodePulsarWriter())
            .listener(new ChunkCountListener())
            .build();
}
```

Wanneer we gaan kijken naar de reader van de start van deze job kunnen we zien dat deze gebruikt wordt om data op te halen die verder verwerkt wordt in de rest van de job. In dit geval gaan we data ophalen in stappen van 100.



Hierna kunnen we naar de processor kijken van deze job en hier herkennen we dan direct de query van clink alsook het omzetten van deze query data naar een event.



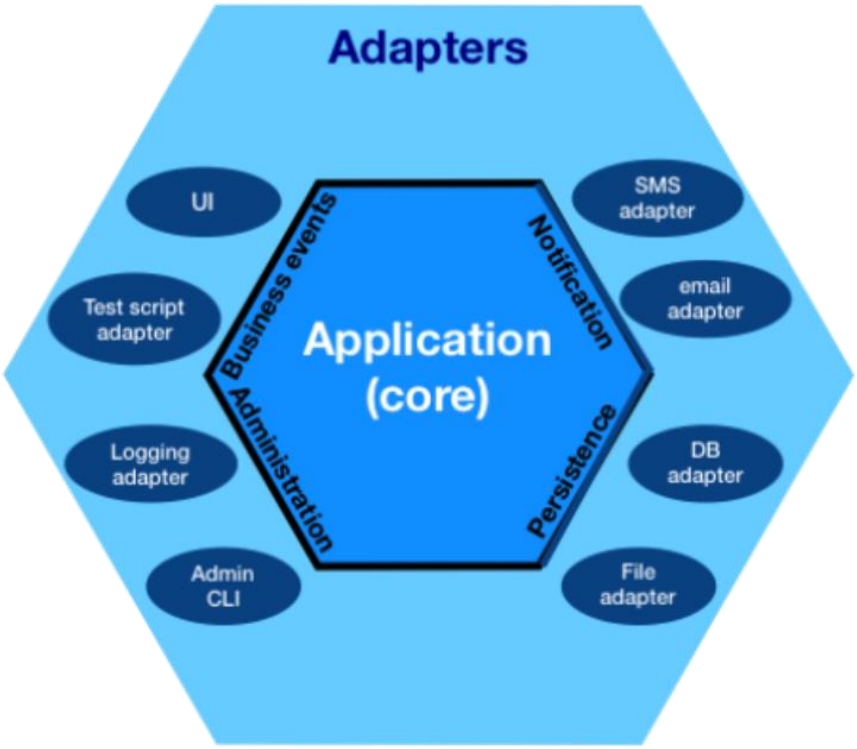
Tot slot hebben we dan nog de writer. Deze writer zorgt er ook voor dat deze grote hoeveelheden van events op pulsar worden gezet zodat het repertorium hier aan kan.

We kunnen in deze initial load dus zien dat de code sterk op de code van clink lijkt maar dan werkt op veel codes tegelijkertijd en dat deze applicatie ook manueel moet geactiveerd worden om te gebruiken in plaats van dat deze automatisch wordt aangeroepen door middel van change data capture.

6 REPERTORIUM

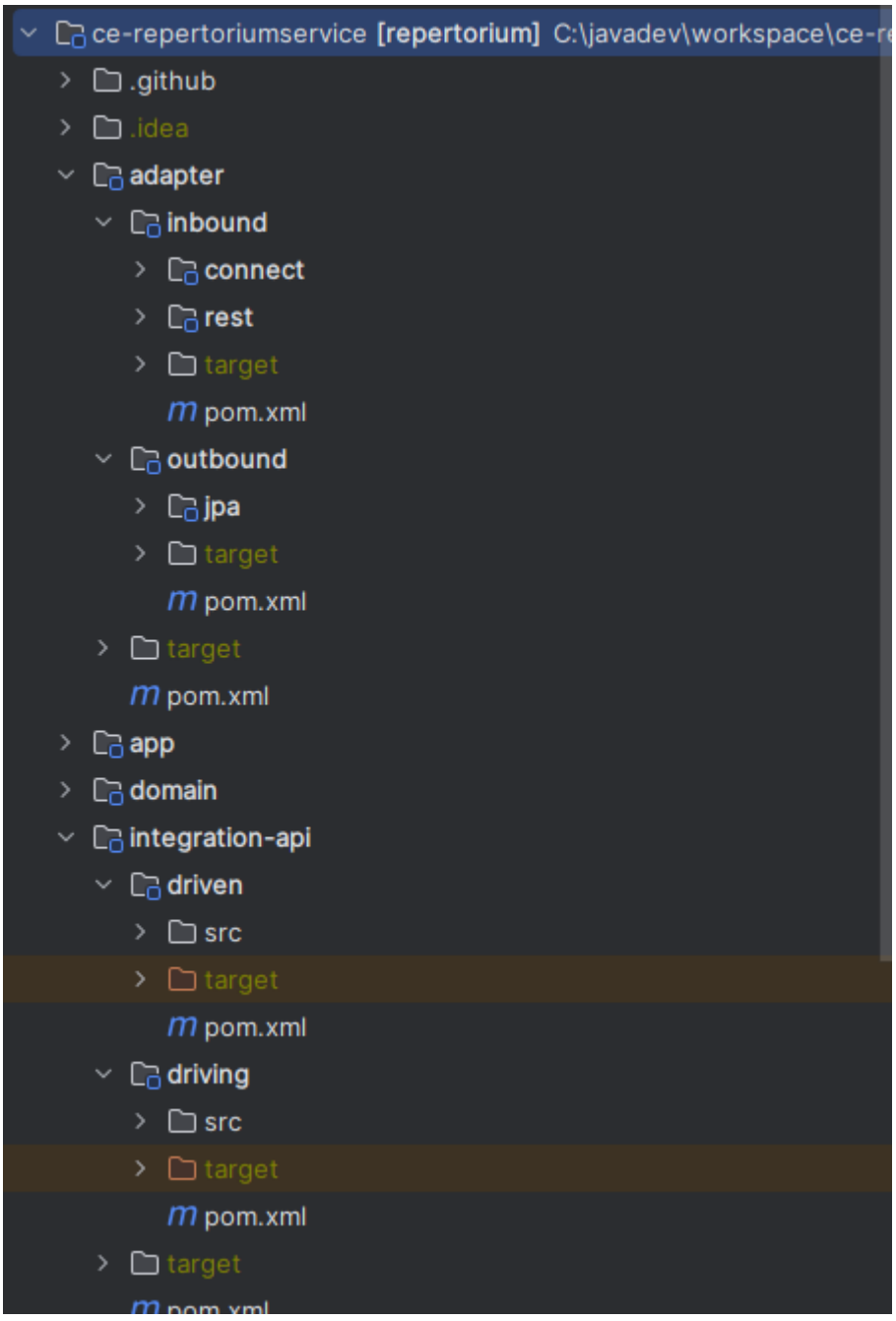
6.1 Structuur

De repertorium applicatie heeft een hexagonale architectuur deze dient ervoor om een scheiding te maken tussen de kernlogica en de omliggende omgeving. In de kern van de zeshoek zien we de kern van de applicatie, in deze kern vinden we het domeinmodel en kernlogica. Rond deze kern zit een laag van adapters die via interfaces interageren met de kern van de applicatie. Deze adapters zetten binnenkomende verzoeken om in een vorm die bruikbaar is voor de kern applicatie. Door deze architectuur toe te passen wordt de applicatie zelf onafhankelijk van de omgeving waarin deze draait.



Wanneer we de structuur van de repertorium applicatie bekijken kunnen we deze architectuur ook zien terugkomen. De structuur van de modules is als volgt:

- App
- Domain
- Integration-api
    - Driven
    - Driving
- Adapter
    - Inbound
        - Connect
        - Rest
    - Outbound
        - Jpa



In de app module staat niet veel. Dit is de plaats waar de springboot applicatie staat en ook een aantal bestanden die te maken hebben met configuratie, de configuratie die hier gebeurt is vooral Liquibase. Liquibase is een tool die gebruikt wordt om de database te beheren en te controleren, dit zorgt er bijvoorbeeld voor dat de database wordt aangemaakt moest deze nog niet bestaan. Buiten Liquibase staat hier ook nog de configuratie van Jackson. Jackson wordt gebruikt om omzettingen te doen tussen Json data en Java code.

Vervolgens hebben we de domain module. Deze module zorgt voor de domeinmodellen. De meeste klassen en enums die aangemaakt werden komen hier dus terecht. We zien hier ook enkele interfaces in staan, deze zijn nodig om bepaalde omzettingen mogelijk te maken zonder dat dit domain afhankelijk wordt van andere modules.

In de integration-api module zien we vooral zaken die andere applicaties nodig hebben. Een goed voorbeeld hiervan is bijvoorbeeld het PrestationCodeSynchronizationEvent.

Dit is het event dat in het repertorium gebruikt wordt om aanpassingen aan de database aan te brengen, maar als we terugdenken naar clink en de initial-load weten we ook dat dit event daar wordt aangemaakt. Dus deze twee andere applicaties hebben een dependecy op deze integration-api module, meer specifiek de driving module onder deze integration-api module, om zo aan dit event aan te kunnen.

```
7 usages   Raf Bergs
@Value(staticConstructor = "of")
public class PrestationCodeSynchronizationEvent implements IntegrationEvent {

    String prestationCodeId;
    String cdcId;
    String code;
    String employerId;
    Kind kind;
    LocalDate fromDate;
    LocalDate untilDate;
    String color;
    boolean individualAdjustment;
    boolean collectiveAdjustment;
    boolean workSchedule;
    List<WageEngine> wageEngines;
    List<PrestationCodeDescription> descriptions;
}
```

Buiten dit event en een gelijkaardig event voor de omschrijvingen van de prestatie codes zien we in deze module ook nog een kopie van een aantal enums aangezien deze ook gebruikt worden in het event en de andere twee applicaties hier natuurlijk ook aan moeten kunnen zonder rechtstreeks de domeinmodellen aan te spreken. Een voorbeeld van één van deze enums is de lijst van loonmotors (WageEngine) in bovenstaand event.

Tot slot hebben we de adapters module. In deze module zien we de inbound en outbound modules. Dit is de module waarin we de adapters terugvinden, zoals eerder gezegd is dit dus de laag rond de kernapplicatie die ervoor zorgt dat binnenkomende berichten omgezet worden in objecten waar onze kernapplicatie mee kan werken. In de outbound module zien we de jpa module en zoals de naam van deze module al zegt vinden we hier de bestanden terug die met jpa te maken hebben en dus nodig zijn om al de prestatie codes in de database van het repertorium op te slaan en ook om data op te vragen van deze database.

In de inbound module vinden we eerst de connect module terug. In deze module staan de Pulsarlisteners voor het repertorium dit is dus de plek waar we de events van pulsar binnen nemen in het repertorium. Naast de connect module hebben we hier ook nog de rest module. In deze module vinden we de restcontrollers terug die gebruikt worden in het repertorium alsook de response objecten die teruggegeven worden wanneer we zaken opvragen via rest calls. Naast de response objecten vinden we ook de request objecten terug deze worden gebruikt om binnenkomende events om te zetten in bruikbare events voor onze applicatie.

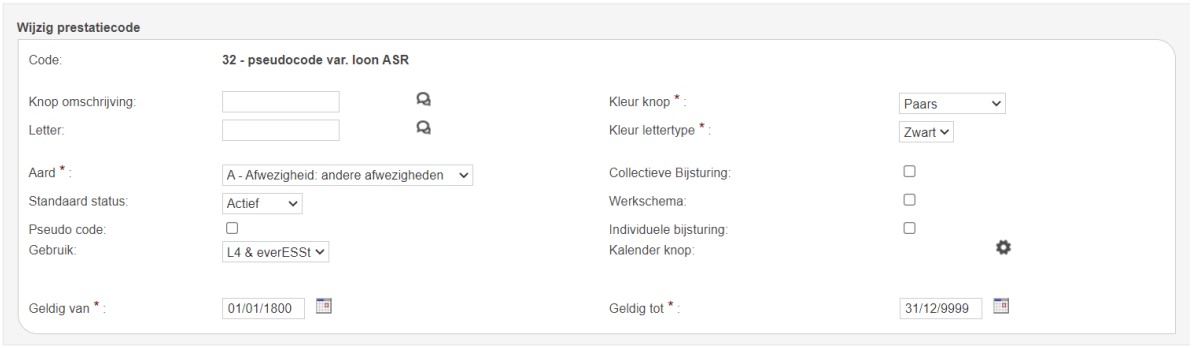
6.2 Database configuratie

Eerder werd al vermeld dat de setup van de database gedaan werd met Liquibase. Om dit voor elkaar te krijgen roept Liquibase een sql bestand aan dat geschreven werd voor deze stage opdracht. In dit sql bestand worden de tabellen aangemaakt en we kunnen hier ook zien hoe de prestationCodeId samengesteld wordt uit verschillende kolommen. Dit sql bestand kunnen we hieronder zien. Deze samengestelde id wordt samengesteld uit anderen kolommen en er wordt ook nagekeken of de waarden in de kolom wel bestaan zodat we geen streepjes hebben staan zonder dat er iets achter staat.

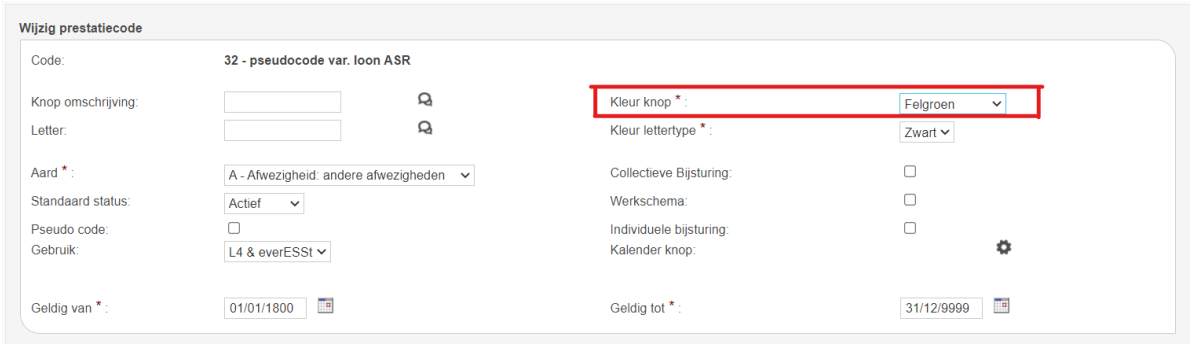
```
CREATE TABLE PRESTATION_CODE
(
    ID                    BIGINT IDENTITY NOT NULL,
    PRESTATION_CODE_ID AS concat(CODE, iif(WAGE_ENGINE IS NULL, '',
concat('-', WAGE_ENGINE)),
        iif(CONTEXT IS NULL, '', concat('-', CONTEXT)),
        iif(EMPLOYER_ID IS NULL, '', concat('-', EMPLOYER_ID))),
    CODE                  VARCHAR(4)  NOT NULL,
    CODE_TYPE             VARCHAR(32) NOT NULL,
    WAGE_ENGINE           VARCHAR(32) NULL,
    CONTEXT               VARCHAR(32) NULL,
    EMPLOYER_ID           VARCHAR(11) NULL,
    KIND                  VARCHAR(255) NULL,
    ICON_FAMILY           VARCHAR(32) NULL,
    ICON_COLOR            VARCHAR(32) NULL,
    ICON                  VARCHAR(32) NULL,
    WORK_SCHEDULE         BIT NULL,
    COLLECTIVE_ADJUSTMENT BIT NULL,
    INDIVIDUAL_ADJUSTMENT BIT NULL,
    ALLOW_PERCENTAGES     BIT NULL,
    ALLOW_HOURS           BIT NULL,
    COLOR                 VARCHAR(32) NULL,
    UNTIL_DATE            DATE NULL,
    FROM_DATE             DATE NULL,
    CONSTRAINT PK_PRESTATION_CODE PRIMARY KEY (ID),
);

CREATE TABLE PRESTATION_CODE_DESCRIPTION
(
    PRESTATION_CODE_ID BIGINT NOT NULL,
    LANGUAGE_CODE      VARCHAR(2),
    SHORT_DESCRIPTION  VARCHAR(10),
    MEDIUM_DESCRIPTION VARCHAR(50),
    LONG_DESCRIPTION   VARCHAR(200),
    LETTER             CHAR(1),
    CONSTRAINT FK_PRESTATION_CODE_DESCRIPTION FOREIGN KEY
(PRESTATION_CODE_ID) REFERENCES PRESTATION_CODE (ID)
);
```

6.3 Prestatie code intergration handler en rest controller

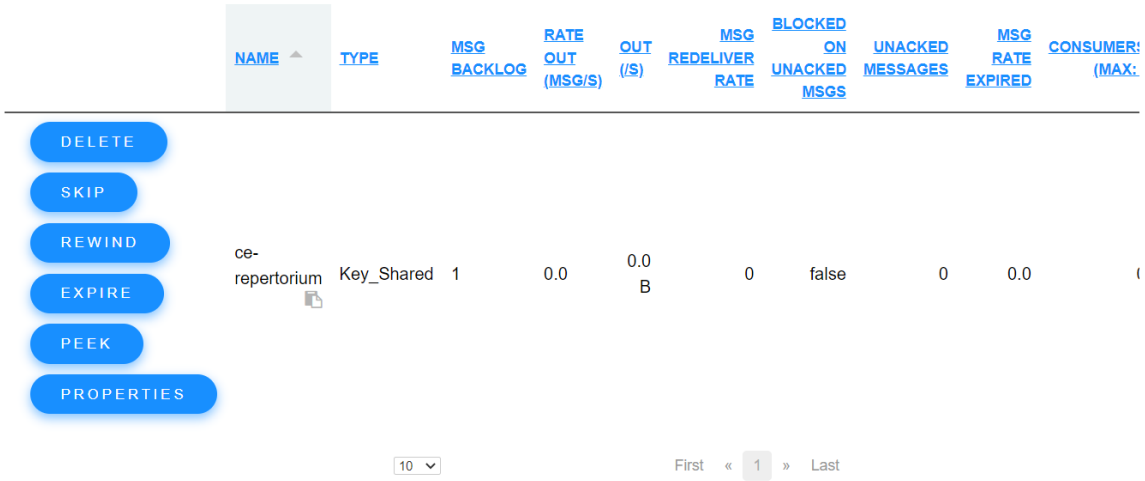
Om de werking van deze applicatie te bespreken gaan we de weg volgen die een code zou afleggen wanneer deze wordt aangepast in de oude applicatie. We beginnen dus bij de oude connect applicatie aangezien deze de master van de data is. In dit scenario gaan we verschillende zaken aanpassen aan code 0032 om de volledige functionaliteit van het repertorium te doorlopen. In onderstaande afbeelding zien we initiële situatie van code 0032.



Als we kijken naar de database van het repertorium is deze momenteel leeg. De eerste aanpassing die gebeurt is het aanpassen van de kleur code van paars naar felgroen.



Wanneer we deze aanpassing opslaan zal in clink een wijziging gedetecteerd worden, door dat change data capture een aanpassing ziet in de oude database van connect. Dan zal clink als eerder besproken de nieuwe status van de code omzetten in een event en dit event op pulsar zetten. Op onderstaande afbeelding zien we het topic op pulsar waar dit event geplaatst werd en waar het repertorium eraan kan om dit event binnen te halen.



In normale omstandigheden zouden we enkel een backlog zien bij grote aantallen aan events, maar om de werking beter te demonstreren werd de repertorium applicatie tijdelijk uitgezet om deze backlog weer te kunnen geven. Als we dan kijken naar het event in deze backlog kunnen we het volgende zien.


```
{
   "prestationCodeId": "90000000661",
   "cdcId": "lsn{0001636F0001DA2C0022}-pci{90000000661}",
   "code": "0032",
   "employerId": "00000000000",
   "kind": "ABSENCE_OTHER_ABSENCES",
   "fromDate": "1800-01-01",
   "untilDate": "9999-12-31",
   "color": "#00F902",
   "individualAdjustment": false,
   "collectiveAdjustment": false,
   "workSchedule": false,
   "wageEngines": [
      "L4",
      "EVERESST"
   ],
   "descriptions": [
      {
         "languageCode": "DE",
         "shortDescription": "Pseudokode var Gehalt MSR",
         "letter": null
      },
      {
         "languageCode": "EN",
         "shortDescription": "pseudocode var salary ASR",
         "letter": "I"
      },
      {
         "languageCode": "FR",
         "shortDescription": "pseudocode sal. Var. DRS",
         "letter": "F"
      },
      {
         "languageCode": "NL",
         "shortDescription": "pseudocode var. loon ASR",
         "letter": null
      }
   ],
   "@type": "PrestationCodeSynchronizationEvent"
}
```

We zien hier dus alle informatie van de prestatie code na de aanpassing van de kleurcode. Nu dat dit event op pulsar staat kan het repertorium dit event binnen nemen via een pulsarlistener zoals hieronder afgebeeld staat. We zien ook dat eens dat dit event binnen gekomen is in het repertorium dat het omgezet wordt in een command en meegegeven wordt aan de handleSync methode.

```
24      @PulsarListener(
25              consumerName = "PrestationCodeIntegrationEventHandler",
26              topic = "${pulsar.consumers.PrestationCodeIntegrationEventHandler.subscription.topic}",
27              schemaType = PrestationCodeSynchronizationEvent.class,
28              deadLetterStrategy = PULSAR)
29      public void on(final PrestationCodeSynchronizationEvent event) {
30          log.info("Handling PrestationCodeSynchronizationEvent [{}]", event);
31          prestationCodeService.handleSync(toCommand(event));
32      }
```

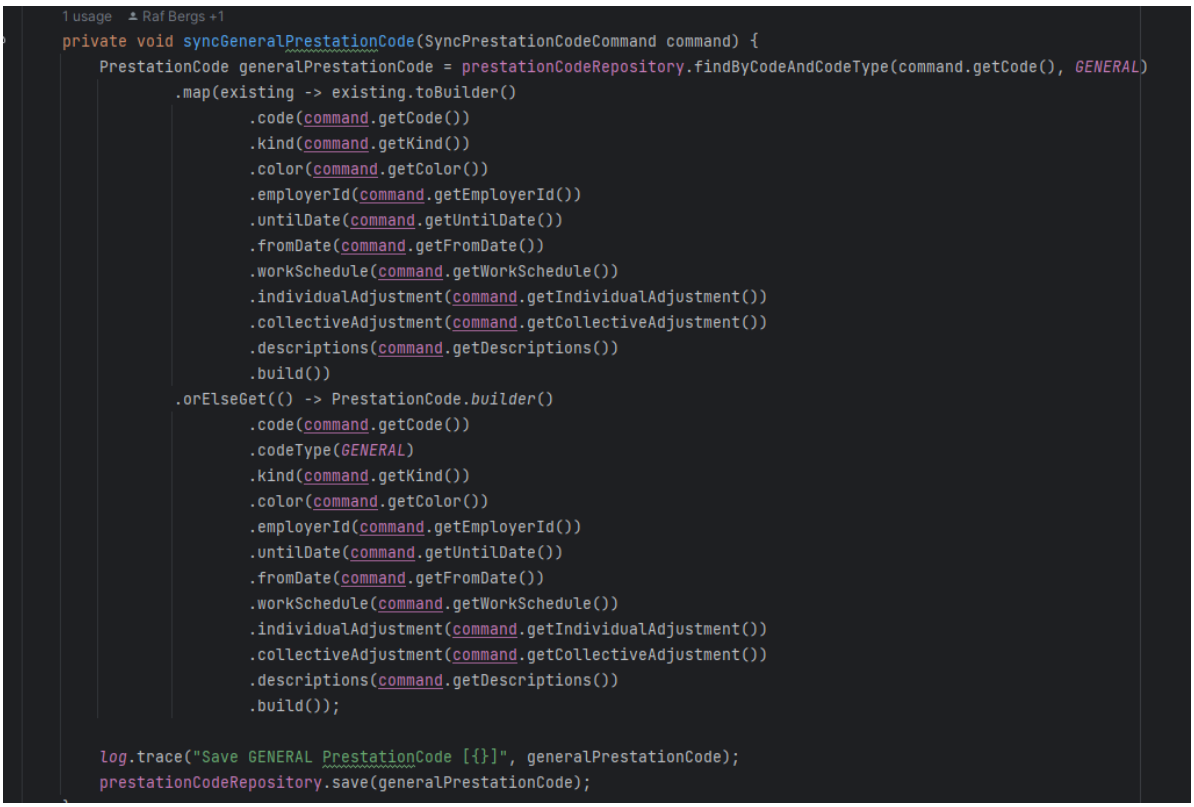
Op onderstaande afbeelding zien we hoe de toCommand methode werkt. In deze methode wordt gebruik gemaakt van een builder. De data die meegegeven wordt aan deze builder komt uit het event dat clink op pulsar had gezet.

```
public SyncPrestationCodeCommand toCommand(PrestationCodeSynchronizationEvent event) {
    return SyncPrestationCodeCommand.builder()
            .code(event.getCode())
            .kind(event.getKind() != null ? Kind.valueOf(event.getKind().name()) : null)
            .workSchedule(event.isWorkSchedule())
            .collectiveAdjustment(event.isCollectiveAdjustment())
            .individualAdjustment(event.isIndividualAdjustment())
            .wageEngines(event.getWageEngines() != null ? event.getWageEngines().stream() Stream<WageEngine>
                    .map(engine -> WageEngine.valueOf(engine.name())) Stream<WageEngine>
                    .toList() : null)
            .color(event.getColor())
            .untilDate(event.getUntilDate())
            .fromDate(event.getFromDate())
            .descriptions(event.getDescriptions() List<PrestationCodeDescription>
                    .stream() Stream<PrestationCodeDescription>
                    .map(description -> PrestationCodeDescription.builder()
                            .languageCode(LanguageCode.valueOf(description.getLanguageCode().name()))
                            .letter(description.getLetter())
                            .mediumDescription(description.getShortDescription())
                            .build()) Stream<PrestationCodeDescription>
                    .toList())
            .build();
}
```

Wanneer we verder gaan kijken naar de handleSync methode zien we daar als eerste de methode syncGeneralPrestationCode. In het repertorium moeten prestatie codes op verschillende niveaus aangepast kunnen worden. Het hoogste niveau hiervan is het general niveau en als dit niveau nog niet bestaat dan wordt dit aangemaakt in dit eerste stuk van de handleSync methode.

```
@Override
public void handleSync(SyncPrestationCodeCommand command) {
    log.info("Handling SyncPrestationCodeCommand [{}]", command);

    List<PrestationCode> existingWageEngines = prestationCodeRepository.findAllByCodeAndCodeType(command.getCode(), WAGE_ENGINE);
    // Save or update the general code
    syncGeneralPrestationCode(command);
```

Wanneer we naar de syncGeneralPrestationCode methode zelf gaan kijken zien we dat hier een prestatie code wordt aangemaakt door middel van builders die de nodige data uit het command halen en daarna opgeslagen wordt. Als deze code al moest bestaan zien we dat we deze terug in een builder steken en deze up-to-date wordt gebracht.



```
1 usage   Raf Bergs +1
private void syncGeneralPrestationCode(SyncPrestationCodeCommand command) {
    PrestationCode generalPrestationCode = prestationCodeRepository.findByCodeAndCodeType(command.getCode(), GENERAL)
            .map(existing -> existing.toBuilder()
                    .code(command.getCode())
                    .kind(command.getKind())
                    .color(command.getColor())
                    .employerId(command.getEmployerId())
                    .untilDate(command.getUntilDate())
                    .fromDate(command.getFromDate())
                    .workSchedule(command.getWorkSchedule())
                    .individualAdjustment(command.getIndividualAdjustment())
                    .collectiveAdjustment(command.getCollectiveAdjustment())
                    .descriptions(command.getDescriptions())
                    .build())
            .orElseGet(() -> PrestationCode.builder()
                    .code(command.getCode())
                    .codeType(GENERAL)
                    .kind(command.getKind())
                    .color(command.getColor())
                    .employerId(command.getEmployerId())
                    .untilDate(command.getUntilDate())
                    .fromDate(command.getFromDate())
                    .workSchedule(command.getWorkSchedule())
                    .individualAdjustment(command.getIndividualAdjustment())
                    .collectiveAdjustment(command.getCollectiveAdjustment())
                    .descriptions(command.getDescriptions())
                    .build());

    log.trace("Save GENERAL PrestationCode [{}]", generalPrestationCode);
    prestationCodeRepository.save(generalPrestationCode);
}
```

Het volgende deel van de handleSync methode gaat over het volgende niveau. Dit is het niveau op basis van de loon motors. Er zijn hier ook drie mogelijkheden, om te beginnen is er zoals bij dit voorbeeld nog geen loonmotor aanwezig in de database en zal deze dus aangemaakt moeten worden. We kunnen nu in onderstaande code zien dat er eerst nagekeken wordt of er meer loon motors aanwezig zijn in het command dat we binnen krijgen of in de records die al in de database staan voor die bepaalde code. Als er meer loonmotors in het command staan zullen de ontbrekende loonmotor records worden aangevuld. Voor de loonmotors zijn er drie opties namelijk: L4, EVERESST of L4 en EVERESST samen. Om te weten welke loonmotors nu juist toegevoegd moeten worden kunnen we de lijst met loonmotors in het command vergelijken met de loonmotors in de database en dan kijken welke er verschillen en dan deze verschillende toe te voegen. In ons voorbeeld zijn er nog geen loonmotors aanwezig in de databank en staat deze code op connect ingesteld om L4 en EVERESST beiden te gebruiken, dus zullen er twee nieuwe records aangemaakt worden. In onze database zal er dus een record voor L4 bijkomen en een record voor EVERESST.

```
46          // Add new wage engines
47          if (command.getWageEngines().size() > existingWageEngines.size()) {
48              log.trace("Add new wage engine(s)");
49              syncNewWageEngines(
50                      command.getCode(),
51                      command.getWageEngines() List<WageEngine>
52                              .stream() Stream<WageEngine>
53                              .filter(newEngine -> !existingWageEngines.stream().map(PrestationCode::getWageEngine).toList().contains(newEngine))
54                              .toList()
55              );
56          }
```

Om te weten hoe deze records juist worden aangemaakt moeten we kijken naar de syncNewWageEngines methode gaan kijken. In deze code, die je onder deze tekst kan zien, wordt dus een nieuw record gemaakt per loonmotor die ontbreekt in de database. Enkel de code, het code type en de loonmotor worden hier meegegeven dit betekend ook dat alle andere velden null zullen zijn. Dit is gedaan omdat we geen data dubbel willen bijhouden, alle nodige data zal in het record van de general prestatie code staan en in de front-end zullen de velden die null zijn overschreven worden met de overeenkomende waarde van het general record. De data die hier wel ingevuld wordt is dus nodig om dit loonmotor record te onderscheiden van het general record.

```
    1 usage   Raf Bergs +1
115 @   private void syncNewWageEngines(String code, List<WageEngine> newEngines) {
116         newEngines.stream()  Stream<WageEngine>
117                 .map(engine -> PrestationCode.builder()
118                         .code(code)
119                         .codeType(WAGE_ENGINE)
120                         .wageEngine(engine)
121                         .build())  Stream<PrestationCode>
122                 .peek(e -> log.trace("Add new wage engine [{}]", e))
123                 .forEach(prestationCodeRepository::save);
124     }
```

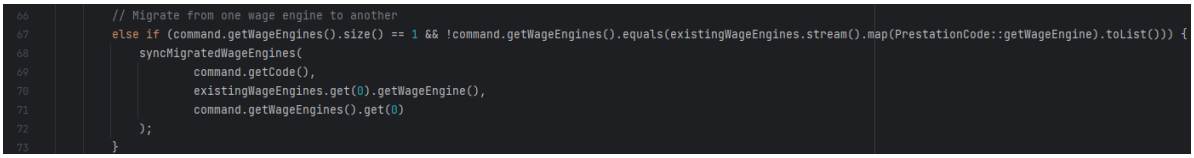
In het geval dat er meer loon motors in de database zitten dan in het command wil dit dus zeggen dat er een loonmotor verwijdert moet worden. Als we bijvoorbeeld zouden wisselen van beide loonmotors naar enkel EVERESST dan moet het record van L4 natuurlijk verwijdert worden. Dit wordt dus nagekeken in onderstaande code en dan wordt de loonmotor die verwijdert moet worden meegegeven aan de syncDeleteWageEngines methode.

```
// Remove deleted wage engines
else if (command.getWageEngines().size() < existingWageEngines.size()) {
    log.trace("Delete existing wage engine(s)");
    syncDeletedWageEngines(
            existingWageEngines.stream()
                    .filter(existingEngine -> !command.getWageEngines().contains(existingEngine.getWageEngine()))
                    .toList()
    );
}
```

In deze methode wordt de code die we meegaven opgezocht aan de hand van de code en de loonmotor en eens dat deze code gevonden is wordt deze uit de database verwijdert.

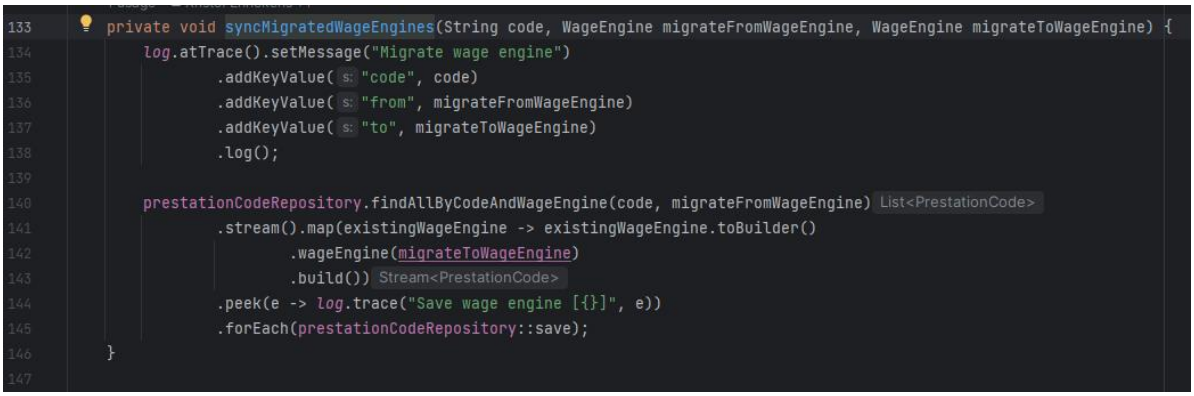
```
    1 usage   Raf Bergs +1
126 @   private void syncDeletedWageEngines(List<PrestationCode> enginesToDelete) {
127         enginesToDelete.forEach(engine -> {
128             log.trace("Delete existing wage engine [{}]", engine);
129             prestationCodeRepository.deleteAllByCodeAndWageEngine(engine.getCode(), engine.getWageEngine());
130         });
131     }
```

Het kan natuurlijk ook zijn dat we van loonmotor willen wisselen. Bijvoorbeeld als we L4 gebruiken en we willen wisselen naar enkel EVERESST. Om dit op te merken moet dus het aantal loonmotors hetzelfde zijn, maar de loonmotor in het command moet dan wel verschillen van de loonmotor in de database. Dit wordt nagekeken in onderstaand stuk van de handleSync methode.



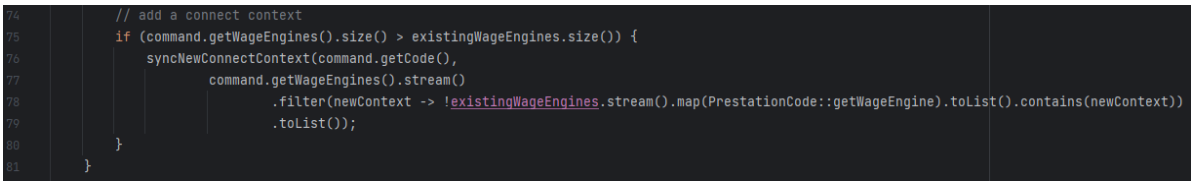
```
// Migrate from one wage engine to another
else if (command.getWageEngines().size() == 1 && !command.getWageEngines().equals(existingWageEngines.stream().map(PrestationCode::getWageEngine).toList())) {
    syncMigratedWageEngines(
            command.getCode(),
            existingWageEngines.get(0).getWageEngine(),
            command.getWageEngines().get(0)
    );
}
```

Hier zien we dat we aan de SyncMigrateWageEngines methode de huidige en de nieuwe loonmotor meegeven. Als we dan gaan kijken wat er gebeurt in deze methode krijgen we onderstaande code te zien. In deze code zien we dat er eerst duidelijk gelogd wordt van welke loonmotor we komen en naar welke loonmotor we moeten migreren. Hierna zien we dat het oude loonmotor record terug in een builder gestoken wordt en dan enkel de loonmotor aangepast wordt en daarna wordt dit terug opgeslagen.



```
private void syncMigratedWageEngines(String code, WageEngine migrateFromWageEngine, WageEngine migrateToWageEngine) {
    log.atTrace().setMessage("Migrate wage engine")
            .addKeyValue( s: "code", code)
            .addKeyValue( s: "from", migrateFromWageEngine)
            .addKeyValue( s: "to", migrateToWageEngine)
            .log();

    prestationCodeRepository.findAllByCodeAndWageEngine(code, migrateFromWageEngine) List<PrestationCode>
            .stream().map(existingWageEngine -> existingWageEngine.toBuilder()
                    .wageEngine(migrateToWageEngine)
                    .build()) Stream<PrestationCode>
            .peek(e -> log.trace("Save wage engine [{}]", e))
            .forEach(prestationCodeRepository::save);
}
```

Tot slot is er nog het niveau per context. We hebben het al over connect gehad en dit is de oude applicatie die gebruikt werd, maar er zijn nog andere applicaties die deze prestatie codes nodig hebben en deze applicaties worden contexten genoemd in het repertorium. Buiten connect zijn er ook nog connect easy en connect HR office (CHRO). Aangezien connect nog altijd de master va de data blijft hebben we altijd een niveau voor de connect context nodig en dit zien we in het laatste stuk van de handleSync methode. Het niveau van de context komt onder het niveau van de wage engines en deze wage engines vallen dan weer onder het general niveau. Dus we hebben per wage engine een connect context nodig. In dit onderstaande stuk code zoeken we dus eerst welke loonmotors nodig zijn en dan worden deze meegegeven aan de syncNewConnectContext methode.



```
// add a connect context
if (command.getWageEngines().size() > existingWageEngines.size()) {
    syncNewConnectContext(command.getCode(),
            command.getWageEngines().stream()
                    .filter(newContext -> !existingWageEngines.stream().map(PrestationCode::getWageEngine).toList().contains(newContext))
                    .toList());
}
}
```

Wanneer we in deze syncNewConnectContext methode gaan kijken zien we dat hier weer een builder gebruikt wordt om een nieuw record te maken op het context niveau per loonmotor die aanwezig is.

```
1 usage  Raf Bergs
148 @  private void syncNewConnectContext(String code, List<WageEngine> wageEngines) {
149        log.trace("adding connect context");
150        wageEngines.stream() Stream<WageEngine>
151                .map(newContextEngine -> PrestationCode.builder()
152                        .code(code)
153                        .codeType(CONTEXT)
154                        .context(CONNECT)
155                        .wageEngine(newContextEngine)
156                        .build()) Stream<PrestationCode>
157                .forEach(prestationCodeRepository::save);
158    }
```

Aangezien we de default waarde van de omschrijvingen van de codes uit een andere tabel moeten halen die niets met de prestatie code tabellen te maken heeft moeten we deze ook apart ophalen met een andere listener als deze default waarde ooit aangepast zou worden moeten alle codes die deze default waarde gebruiken dus ook aangepast worden. Deze code lijkt heel sterk op de code om de prestatie code zelf op te halen dus deze wordt hier niet herhaald we kijken hier enkel wat er in de handleSync voor deze omschrijvingen gebeurt. We zien hieronder dus dat er een andere handleSync methode is die in plaats van een gewoon SyncPrestationCodeCommand gebruikt wordt hier een SyncPrestationCodeDescriptionCommand gebruikt. In deze code zien we dat er eerst naar het general niveau van de code waarbij deze omschrijvingen horen gezocht wordt en wanneer deze gevonden wordt, wordt er in de tabel van de omschrijvingen een record gemaakt en dan wordt hier een id, die automatisch gegenereerd werd voor de prestatie codes in hun tabel, meegegeven wordt om naar de juiste prestatie code te kunnen verwijzen. Als we na het zoeken zien dat deze beschrijving al bestaat wordt deze ook up-to-date gebracht in plaats van een nieuwe te maken.

```
160    @Override
161    public void handleSync(SyncPrestationCodeDescriptionCommand command) {
162        log.info("Handling SyncPrestationCodeDescriptionCommand [{}]", command);
163
164        prestationCodeRepository.findByCodeAndCodeType(command.getCode(), GENERAL)
165                .ifPresent(existingCode -> {
166                    List<PrestationCodeDescription> updatedDescriptions = existingCode.getDescriptions().stream()
167                            .map(existingDescription -> {
168                                final String newDescription = command.getDescriptions().get(existingDescription.getLanguageCode());
169                                return existingDescription.toBuilder()
170                                        .mediumDescription(newDescription != null ? newDescription : existingDescription.getMediumDescription())
171                                        .build();
172                            })
173                            .toList();
174
175                    PrestationCode updatedCode = existingCode.toBuilder()
176                            .descriptions(updatedDescriptions)
177                            .build();
178
179                    log.trace("Save PrestationCode [{}]", existingCode);
180                    prestationCodeRepository.save(updatedCode);
181                });
182
183    }
```

We hebben nu alle functionaliteit overlopen waar een event van een prestatie code door moet wanneer dit binnenkomt en we kunnen nu dus kijken naar het resultaat voor ons voorbeeld. We zien hieronder dat er vijf records in de prestation code tabel zijn bijgemaakt één op general niveau, twee op het loonmotor niveau en dan nog twee op context niveau. We zien hier een heel aantal nullen staan die dan dienen overschreven te worden in met de nodige informatie uit het record op het general niveau. Dit overschrijven gebeurt in de front-end.

	ID	PRESTATION_CODE_ID	CODE	CODE_TYPE	WAGE_ENGINE	CONTEXT	EMPLOYER_ID	KIND	ICON_FAMILY
1	1	0032	0032	GENERAL	<null>	<null>	<null>	ABSENCE_OTHER_ABSENCES	<null>
2	2	0032-L4	0032	WAGE_ENGINE	L4	<null>	<null>	<null>	<null>
3	3	0032-EVERESST	0032	WAGE_ENGINE	EVERESST	<null>	<null>	<null>	<null>
4	4	0032-L4-CONNECT	0032	CONTEXT	L4	CONNECT	<null>	<null>	<null>
5	5	0032-EVERESST-CONNECT	0032	CONTEXT	EVERESST	CONNECT	<null>	<null>	<null>

Als we dan nog eens gaan kijken naar de andere tabel die is aangemaakt, de tabel van de omschrijvingen, zien we dat hier een record per taal is bijgekomen met een id die naar de id verwijst van het general record.

	PRESTATION_CODE_ID	LANGUAGE_CODE	SHORT_DESCRIPTION	MEDIUM_DESCRIPTION	LONG_DESCRIPTION	LETTER
1	1	DE	<null>	Pseudokode var Gehalt MSR	<null>	<null>
2	1	EN	<null>	pseudocode var salary ASR	<null>	I
3	1	FR	<null>	pseudocode sal. Var. DRS	<null>	F
4	1	NL	<null>	pseudocode var. loon ASR	<null>	<null>

Als er in clink nu nog iets aangepast wordt aan deze code zien we dat we heel wat functionaliteit hebben om met deze veranderingen om te gaan. Als voorbeeld wordt de loonmotor van L4 en EVERESST naar enkel EVERESST gezet en dan krijgen we het volgende resultaat.

WHERE ORDER BY

	ID	PRESTATION_CODE_ID	CODE	CODE_TYPE	WAGE_ENGINE	CONTEXT	EMPLOYER_ID	KIND	ICON_FAMILY
1	1	0032	0032	GENERAL	<null>	<null>	<null>	ABSENCE_OTHER_ABSENCES	<null>
2	3	0032-EVERESST	0032	WAGE_ENGINE	EVERESST	<null>	<null>	<null>	<null>
3	5	0032-EVERESST-CONNECT	0032	CONTEXT	EVERESST	CONNECT	<null>	<null>	<null>

We zien nu dat alle records die onder de loonmotor L4 vielen, en de kinderen van deze loon motor, nu verwijderd zijn.

Wanneer we een prestatie code die al in de database staat willen aanpassen via de UI van het repertorium gebruiken we de put methode in de rest controller van de prestatie codes. Met deze put methode geven we dan een nieuwe versie van een code mee om deze te wijzigen.

```
    Raf Bergs
34  @Operation(summary = "Save or update code")
35  @PutMapping("/prestation-codes")
36  public ResponseEntity<Void> save(@RequestBody SavePrestationCodeRequest request) {
37      prestationCodeService.handleSave(request.toCommand());
38      return ResponseEntity.noContent().build();
39  }
40  }
```

We kunnen nu gaan kijken naar deze handleSave methode die hieronder staat. In deze methode zoeken we eerst naar de bestaande code en gebruiken dan de info die werd meegegeven om deze code te wijzigen met een builder. Deze put methode kan ook gebruikt worden om een nieuw record aan te maken in dit geval wordt er geen bestaande code gevonden dus zal er een nieuwe code aangemaakt worden aan de hand van de informatie die meegegeven werd.

```java
@Override
public void handleSave(SavePrestationCodeCommand command) {
    log.info("Handling SavePrestationCodeCommand [{}]", command);

    PrestationCode updatedPrestationCode = prestationCodeRepository.findByPrestationCodeId(command.getPrestationCodeId())
            .map(existing -> updatePrestationCode(existing.toBuilder(), command))
            .orElseGet(() -> PrestationCode.builder()
                    .codeType(command.getCodeType())
                    .code(command.getCode())
                    .wageEngine(command.getWageEngine())
                    .context(command.getContext())
                    .kind(command.getKind())
                    .color(command.getColor())
                    .employerId(command.getEmployerId())
                    .untilDate(command.getUntilDate())
                    .fromDate(command.getFromDate())
                    .allowPercentages(command.getAllowPercentages())
                    .allowHours(command.getAllowHours())
                    .workSchedule(command.getWorkSchedule())
                    .individualAdjustment(command.getIndividualAdjustment())
                    .collectiveAdjustment(command.getCollectiveAdjustment())
                    .descriptions(command.getDescriptions())
                    .icon(command.getIcon())
                    .iconFamily(command.getIconFamily())
                    .iconColor(command.getIconColor())
                    .build()
            );

    log.trace("Save PrestationCode [{}]", updatedPrestationCode);
    prestationCodeRepository.save(updatedPrestationCode);
```

Er was ook afgesproken als een record op context niveau wordt aangepast dat deze aanpassing voor beide loonmotors moet gebeuren. In onderstaande code zien we dat er gekeken wordt of de code die wordt aangepast een code is op context niveau en als dit zo is worden alle records op context niveau voor deze code gezocht en allemaal aangepast. Als een code bijvoorbeeld L4 en EVERESST gebruikt zal er een context connect niveau zijn voor beide deze loonmotors en we willen dus L4-connect en EVERESST-connect gelijk houden.

```java
        // Keep contexts the same over different wage engines
        if (isContext(command)) {
            prestationCodeRepository.findAllByCodeAndCodeType(command.getCode(), CONTEXT) List<PrestationCode>
                    .stream() Stream<PrestationCode>
                    .filter(context -> context.getWageEngine() != command.getWageEngine())
                    .filter(context -> context.getContext() == command.getContext())
                    .map(existing -> updatePrestationCode(existing.toBuilder(), command))
                    .peek(e -> log.trace("Save context [{}]", e))
                    .forEach(prestationCodeRepository::save);
        }
    }
```

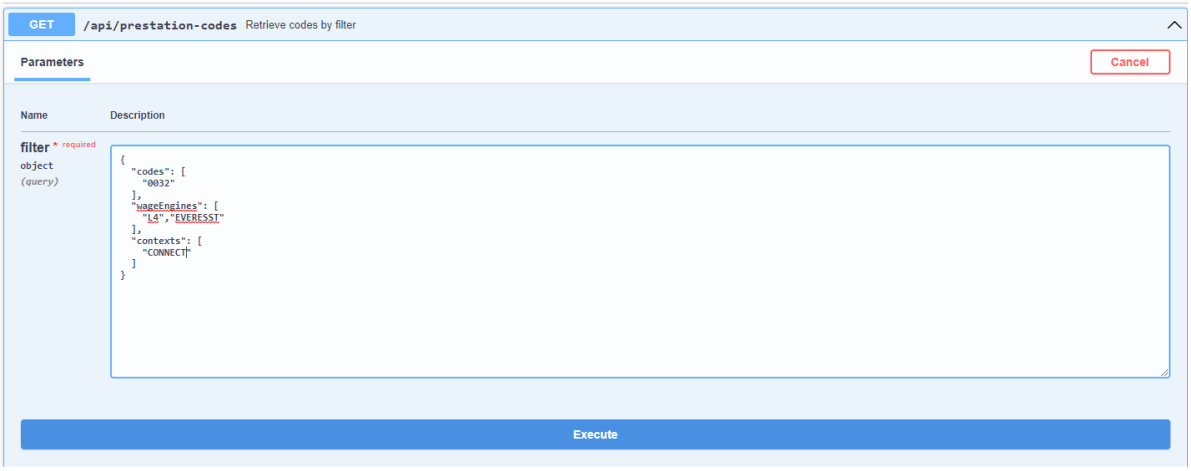
Via deze zelfde rest controller kunnen we ook nog een aantal records opvragen via een get methode. In deze findByFilter methode zien we dat er een filter wordt meegegeven en de findBySelection methode wordt opgeroepen.

```java
    Raf Bergs
    @Operation(summary = "Retrieve codes by filter")
    @GetMapping("/prestation-codes")
    public ResponseEntity<List<PrestationCodeResponse>> findByFilter(PrestationCodeFilterRequest filter) {
        return ResponseEntity.ok(
                PrestationCodeResponse.fromEntities(
                        prestationCodeService.findBySelection(filter.codes(), filter.wageEngines(), filter.contexts())
                )
        );
    }
```

Als we dan gaan kijken naar deze findBySelection methode zien we het onderstaande. In deze code geven we dus drie sets mee en daarna wordt er gekeken welke sets al dan niet leeg zijn. Als enkel de code ingevuld wordt krijgen we enkel het general niveau terug. Als de code en de loonmotors zijn ingevuld krijgen we de general code en elke ingevulde en bestaande loonmotor terug. Als de code, de loon motors en de contexten zijn ingevuld krijgen we elke niveau terug dat terug te vinden is in de database.

```
1 usage   Raf Bergs
@Override
public List<PrestationCode> findBySelection(
        final Set<String> codes,
        final Set<WageEngine> wageEngines,
        final Set<Context> contexts) {
    final Set<CodeType> codeTypes;

    if (!contexts.isEmpty()) {
        codeTypes = Set.of(CodeType.GENERAL, CodeType.WAGE_ENGINE, CodeType.CONTEXT);
    } else if (!wageEngines.isEmpty()) {
        codeTypes = Set.of(CodeType.GENERAL, CodeType.WAGE_ENGINE);
    } else {
        codeTypes = Set.of(CodeType.GENERAL);
    }

    final Specification<PrestationCode> specification = Specification.allOf(
            hasCodeTypes(codeTypes),
            codes.isEmpty() ? null : hasFieldWithValuesOrNull( field: "code", codes),
            wageEngines.isEmpty() ? null : hasFieldWithValuesOrNull( field: "wageEngine", wageEngines),
            contexts.isEmpty() ? null : hasFieldWithValuesOrNull( field: "context", contexts)
    );

    return jpaPrestationCodeRepository.findAll(specification);
}
```

We kunnen beide deze get methode gebruiken om onze aangemaakte code op te vragen. In onderstaand voorbeeld geef ik genoeg informatie mee om alle mogelijke codes op te vragen.



Als we dan gaan kijken wat deze methode teruggeeft is dat het volgende.


```
[
  {
    "id": "0032",
    "code": "0032",
    "codeType": "GENERAL",
    "wageEngine": null,
    "context": null,
    "employerId": null,
    "kind": "ABSENCE_OTHER_ABSENCES",
    "icon": {
      "code": null,
      "color": null,
      "fontFamily": null
    },
    "workSchedule": false,
    "collectiveAdjustment": false,
    "individualAdjustment": false,
    "allowPercentages": null,
    "allowHours": null,
    "color": "#00F902",
    "untilDate": "9999-12-31",
    "fromDate": "1800-01-01",
    "descriptions": [
      {
        "language": "DE",
        "shortDescription": null,
        "mediumDescription": "Pseudokode var Gehalt MSR",
        "longDescription": null,
        "letter": null
      },
      {
        "language": "EN",
        "shortDescription": null,
        "mediumDescription": "pseudocode var salary ASR",
        "longDescription": null,
        "letter": "I"
      },
      {
        "language": "FR",
        "shortDescription": null,
        "mediumDescription": "pseudocode sal. Var. DRS",
        "longDescription": null,
        "letter": "F"
      },
      {
        "language": "NL",
        "shortDescription": null,
        "mediumDescription": "pseudocode var. loon ASR",
        "longDescription": null,
        "letter": null
      }
    ]
  },
  {
    "id": "0032-L4",
    "code": "0032",
    "codeType": "WAGE_ENGINE",
    "wageEngine": "L4",
    "context": null,
    "employerId": null,
    "kind": null,
    "icon": {
      "code": null,
```

```
      "color": null,
      "fontFamily": null
    },
    "workSchedule": null,
    "collectiveAdjustment": null,
    "individualAdjustment": null,
    "allowPercentages": null,
    "allowHours": null,
    "color": null,
    "untilDate": null,
    "fromDate": null,
    "descriptions": []
  },
  {
    "id": "0032-EVERESST",
    "code": "0032",
    "codeType": "WAGE_ENGINE",
    "wageEngine": "EVERESST",
    "context": null,
    "employerId": null,
    "kind": null,
    "icon": {
      "code": null,
      "color": null,
      "fontFamily": null
    },
    "workSchedule": null,
    "collectiveAdjustment": null,
    "individualAdjustment": null,
    "allowPercentages": null,
    "allowHours": null,
    "color": null,
    "untilDate": null,
    "fromDate": null,
    "descriptions": []
  },
  {
    "id": "0032-L4-CONNECT",
    "code": "0032",
    "codeType": "CONTEXT",
    "wageEngine": "L4",
    "context": "CONNECT",
    "employerId": null,
    "kind": null,
    "icon": {
      "code": null,
      "color": null,
      "fontFamily": null
    },
    "workSchedule": null,
    "collectiveAdjustment": null,
    "individualAdjustment": null,
    "allowPercentages": null,
    "allowHours": null,
    "color": null,
    "untilDate": null,
    "fromDate": null,
    "descriptions": []
  },
  {
    "id": "0032-EVERESST-CONNECT",
    "code": "0032",
    "codeType": "CONTEXT",
    "wageEngine": "EVERESST",
    "context": "CONNECT",
    "employerId": null,
    "kind": null,
    "icon": {
```

```
      "code": null,
      "color": null,
      "fontFamily": null
    },
    "workSchedule": null,
    "collectiveAdjustment": null,
    "individualAdjustment": null,
    "allowPercentages": null,
    "allowHours": null,
    "color": null,
    "untilDate": null,
    "fromDate": null,
    "descriptions": []
  }
]
```


We zien hier dus een lijst met alle codes die aangemaakt zijn door het event dat we eerder binnen hebben genomen.

6.4 Configuratie rest controller

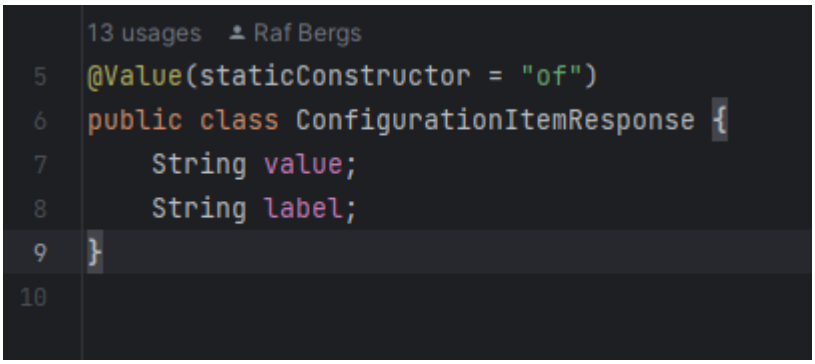
Buiten de rest controller voor de prestatie code is er ook nog de configuratie rest controller. Deze controller dient om data door te geven aan de front-end die ze nodig kunnen hebben voor zaken zoals bijvoorbeeld een dropdown te vullen met alle mogelijke codes. We kunnen in onderstaande code zien dat in deze get methode alle codes, dus enkel de codes zoals bijvoorbeeld 0032, worden opgevraagd en de medium lange beschrijving in het Nederlands hierachter wordt gezet. Deze get methode kan dan aangesproken worden door de front-end om de dropdown te vullen met alle mogelijke codes.

```
Raf Bergs
@Operation(summary = "Retrieve all codes with description.")
@GetMapping("/codes")
public ResponseEntity<List<ConfigurationItemResponse>> findPrestationCodes() {
    return ResponseEntity.ok(
            prestationCodeService.findByCodeType(GENERAL) List<PrestationCode>
                    .stream() Stream<PrestationCode>
                    .map(code -> ConfigurationItemResponse.of(code.getCode(), String.format("%s - %s", code.getCode(),
                            code.getDescriptions() List<PrestationCodeDescription>
                                    .stream() Stream<PrestationCodeDescription>
                                    .filter(d -> d.getLanguageCode().equals(NL))
                                    .findFirst() Optional<PrestationCodeDescription>
                                    .map(PrestationCodeDescription::getMediumDescription) Optional<String>
                                    .orElse( other: null)))) Stream<ConfigurationItemResponse>
                    .toList()
    );
}
```

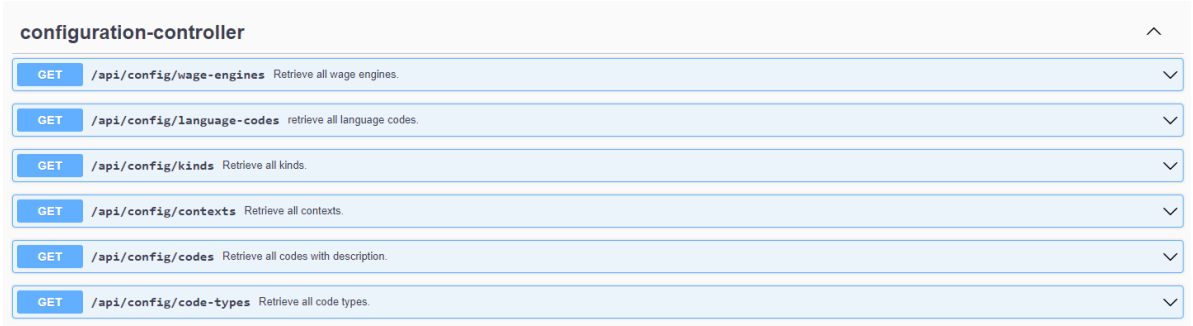
Buiten deze dropdown zijn er nog een aantal anderen, maar dit zijn allemaal methodes waar we alle waardes van bepaalde enums teruggeven dus deze code is heel gelijkaardig voor de rest van deze get methodes dus wordt hieronder eentje besproken ten voorbeeld. Als voorbeeld wordt de get methode om alle loonmotors uit een enum te halen en te laten zien getoond. In deze methode zien we dat we een lijst maken van ConfigurationItemResponses en deze responses worden gevuld met informatie van de WageEngine enum.

```
Raf Bergs
@Operation(summary = "Retrieve all wage engines.")
@GetMapping("/wage-engines")
public ResponseEntity<List<ConfigurationItemResponse>> findWageEngines() {
    return ResponseEntity.ok(
            Arrays.stream(WageEngine.values()) Stream<WageEngine>
                    .map(wageEngine -> ConfigurationItemResponse.of(wageEngine.name(), wageEngine.name())) Stream<ConfigurationItemResponse>
                    .toList()
    );
}
```

Als we dan gaan kijken wat er teruggegeven wordt in zo een response zien we dat hier enkel een value en een label meegeven omdat dit makkelijk gebruikt kan worden in dropdowns.



Voor de andere get methodes gebeurt hetzelfde, maar dan wel met de waardes van de andere enums natuurlijk. Hieronder wordt nog de swagger van deze configuratie controller getoond dat als overzicht van al deze methodes kan dienen.



7 DEMO

https://youtu.be/Ou21p9cxNNo

Met bovenstaande link kan de demo van het repertorium bekeken worden. In deze demo worden eerst de oude applicatie (Connect) en de UI van het repertorium getoond. Hierna kunnen we zien hoe de codes eruitzien in de UI en hoe we deze kunnen aanpassen. Vervolgens wordt de kalender applicatie getoond met een voorbeeld van een werknemer die fulltime werkt. Deze werknemer geeft dan in op een bepaalde dag sollicitatieverlof in. We kunnen nu een extra tijdscode op deze dag zien. Eens dat deze tijdscode zichtbaar is wordt in Connect de kleur van deze code aangepast en kunnen we zien dat deze automatisch mee aangepast wordt in de kalender. Eens dat dit gebeurt is wordt er getoond hoe deze kleur code ook via het repertorium overschreven kan worden.