Scriptie

“Hoe kan een grote hoeveelheid transactie data verzameld, opgeslagen en beschikbaar gesteld worden waarbij voldaan wordt aan de voorwaarden en wensen van Extendas?”

Voorwoord

Deze scriptie is geschreven als eindproduct van het afstuderen bij Extendas voor de opleiding HBO-ICT Software Engineering aan het Saxion. In deze scriptie is vastgelegd hoe het project is verlopen en tot welk eindresultaat dit heeft geleid.

Bij deze wil ik mijn begeleiders vanuit Extendas, Maarten Sprakel en Guus Vaanholt, bedanken voor hun ondersteuning en begeleiding tijdens het project. Ook wil ik Timothy Sealy bedanken voor de ondersteuning en begeleiding vanuit het Saxion.

Ik wens u veel leesplezier toe.

Wouter den Ouden

Inhoudsopgave

1. Inleiding

Dit verslag beschrijft de afstudeeropdracht die ik heb uitgevoerd in opdracht van Extendas. De opdracht draait om het ontwerpen van een oplossing om een grote hoeveelheid data te verzamelen, op te slaan en beschikbaar te stellen aan verscheidene applicaties. De hoofdvraag van de luidt: ‘Hoe kan een grote hoeveelheid transactie data verzameld, opgeslagen en beschikbaar gesteld worden waarbij voldaan wordt aan de voorwaarden en wensen van Extendas?’.

De aanleiding voor deze opdracht is de ontwikkeling van het project DPS. DPS (Domain Promotion System) is een webapplicatie waarmee gebruikers onder andere geaggregeerde verkoopgegevens kunnen gebruiken om promoties te bepalen.

De eerste hoofdstukken beschrijven de achtergrond en de opdracht, waarbij dieper wordt ingegaan op de aanleiding voor de opdracht. Daarna wordt het onderzoek beschreven, door de deelvragen te beantwoorden met behulp van het onderzoeksverslag [1].

Vervolgens wordt het proces beschreven en gekeken naar de tools die gebruikt zijn voor het onderzoek. Het ontwerp van het data warehouse wordt beschreven en aan de hand daarvan ook het gerealiseerde prototype en resultaten.

Als laatst wordt een conclusie gegeven op de hoofdvraag en het gehele afstudeertraject. Hierbij worden ook eventuele aanbevelingen beschreven en gekeken naar wat kan worden verbeterd en wat de discussiepunten zijn.

2. Achtergrond

De afstudeeropdracht is uitgevoerd bij Extendas. Extendas is een bedrijf dat software ontwikkelt dat organisaties in de petrolbranche helpt bij hun administratieve, logistieke en financiële processen. Deze software wordt voornamelijk gebruikt door tankstations en hun klanten.

Producten

Een aantal van de producten van Extendas die ook van belang zijn voor de opdracht.

SPIN

SPIN (afgekort van Social Petrol INtelligence) is een SaaS oplossing bestaande uit meerdere modules ter ondersteuning van organisaties in de petrolbranche. SPIN bevat de volgende modules:

- **POS** - Een module voor het afhandelen en inzicht geven van transacties. Ook het beheren van de instellingen per POS (Point of Sale) kan vanuit deze module.
- **FOP** - Een module voor digitale facturatie. Deze module geeft klanten van de organisatie toegang om facturen digitaal in te zien, e-mail voorkeuren te beheren en transacties in te zien.
- **Loyalty** - Een module om loyaliteit acties op maat te maken, klanten te laten registreren en klanten mails te sturen. De module maakt het onder andere mogelijk om klanten te laten sparen voor gratis producten of korting door middel van het doen van transacties.

SPIN is ontwikkeld in PHP met het framework Symfony.

FuelOffice

FuelOffice is het backoffice systeem dat Extendas zijn klanten aanbiedt voor het administratieve, financiële en logistieke beheer van zijn/haar onderneming. FuelOffice heeft een koppeling met SPIN voor diverse modules. Een aantal voorbeelden van modules van FuelOffice zijn onder andere:

- Transacties uitlezen van veel gebruikte kassasystemen
- Transacties inlezen via verschillende bestandsformaten
- Facturatie in combinatie met SPIN
- Incasso aan de hand van facturatie
- Relatiebeheer
- Pas beheer
- Voorraadbeheer
- Financiële administratie van in- en verkoop

DPS

SPIN Domain Promotion System is de webapplicatie waarin SWO de Splinter onder andere de dagelijkse werkzaamheden zoals het managen van leden, maken van promotie kalenders, monitoren van promoties, afsluiten van promoties en het versturen van een overzicht van de verkochte promotie artikelen kunnen doen.

DPS is nog in ontwikkeling en zal data ontvangen vanuit het data warehouse en vanuit de medewerkers van SWO de Splinter middels verschillende user interfaces.

FuelPosConnector

De transactiebestanden van Tokheim kassa's (Fuel POS) van leden van SWO de Splinter worden elke nacht op een SFTP server geplaatst van Extendas om vervolgens door de FuelPosConnector omgezet en opgeslagen te worden.

SPINpos

De SPINpos is een applicatie ontwikkeld voor iOS en bedoeld om een regulier kassa systeem te vervangen. Transacties van de SPINpos worden verstuurd naar SPIN. En door middel van API's haalt de SPINpos product en promotie data uit SPIN.

SWO de Splinter

Samenwerkingsorganisatie (SWO) de Splinter is in 2008 opgericht met als doel rendementsverbeteringen te bewerkstelligen voor zelfstandige tankstation ondernemers. Op moment van schrijven telt de Splinter 180 deelnemende leden.

Promotiekalender

In samenwerking met de fabrikanten maakt de Splinter promotie kalenders voor de deelnemende locaties. Aan de hand van de gemaakte transacties wordt bijgehouden wat de verschillende locaties verdienen aan deze promoties.

Begeleiding

Tijdens de afstudeerperiode heeft de afstudeerder begeleiding gehad vanuit Extendas en vanuit Saxion.

Begeleiding Extendas

Vanuit Extendas zijn er twee begeleiders. Maarten Sprakel is de lead developer van het back-end team en beschikbaar voor technische ondersteuning. Guus Vaanholt is de projectmanager en beschikbaar voor projectmatige ondersteuning. Daarnaast zal de begeleiding bestaan uit het bespreken van de voortgang en het ontvangen van feedback op de documenten.

Begeleiding Saxion

Vanuit Saxion is de afstudeerder begeleid door Timothy Sealy. De begeleiding bestond uit een maandelijks voortgangsgesprek, feedback op de documenten en ondersteuning van het proces.

3. Opdracht

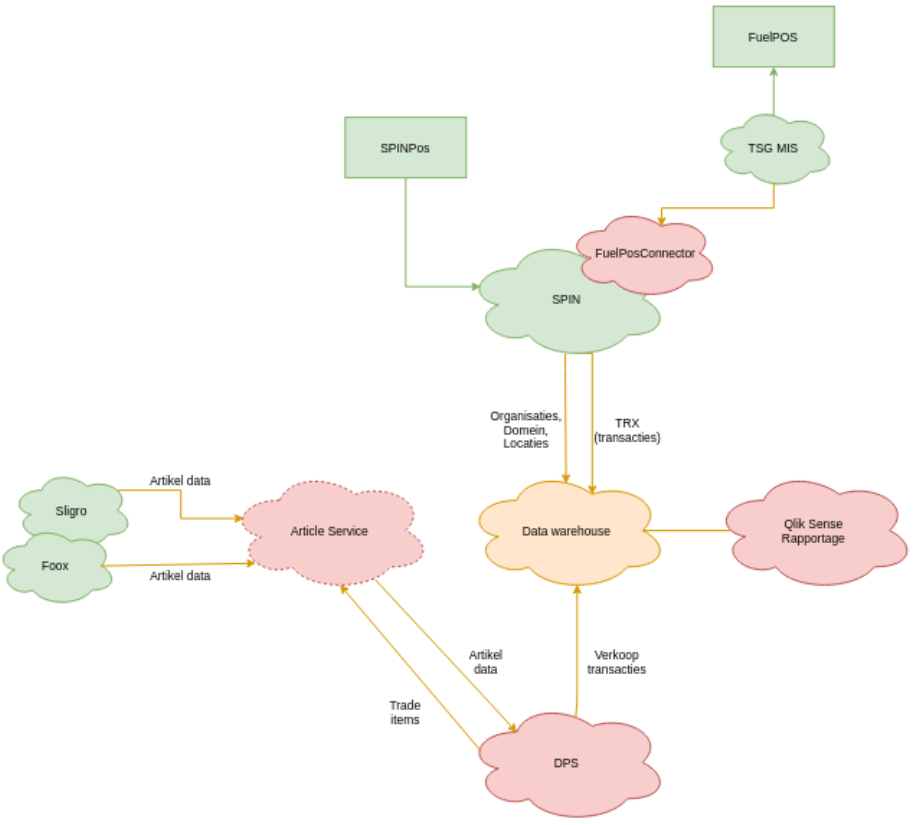
In dit hoofdstuk wordt de opdracht beschreven en onder andere informatie gegeven over de aanleiding van de opdracht.

Aanleiding

SWO de Splinter heeft aan Extendas gevraagd om te onderzoeken of Extendas de verwerking van transactie data die op moment door Orange Peak gedaan wordt over te kunnen nemen en waar mogelijk te verbeteren.

Het huidige proces vergt veel handwerk en is daardoor erg arbeidsintensief. Ook het is het huidige proces fraudegevoelig en lastig om verder uit te breiden met nieuwe functionaliteiten. Extendas is van mening dat door automatisering dit proces kan worden verbeterd.

Om dit proces te verbeteren zullen er een aantal nieuwe platformen gerealiseerd moeten worden waarin de verschillende functionaliteiten worden ondergebracht. Onderstaande afbeelding is opgesteld door Extendas om inzicht te geven in de architectuur. In deze afbeelding staan de betrokken applicaties beschreven, waarbij de groene blokken bestaande applicaties zijn en rode blokken applicaties die nog in ontwikkeling zijn en in de loop van de afstudeerperiode in productie gaan. Het data warehouse (oranje) zou volgens Extendas een mogelijke oplossing kunnen zijn als centrale plek om de data vanuit deze verschillende onderdelen bij elkaar te krijgen.



Figuur 1

Opdrachtomschrijving

Zoals hierboven benoemd, is een van de platformen mogelijk een data warehouse. Het data warehouse moet een centrale plek worden voor het bewaren van transacties. Deze transacties moeten beschikbaar zijn voor applicaties binnen Extendas. Het data warehouse wordt opgezet in het kader van DPS, maar zal in de toekomst ook ingezet kunnen worden voor andere toepassingen.

Data warehouse

Het data warehouse zal het middelpunt zijn voor alle betrokken data uit de omliggende applicaties. Een groot deel van de opdracht bestaat uit het onderzoeken van de verschillende technieken voor het realiseren van het data warehouse. Daarnaast zal moeten worden uitgezocht welke data er verzameld moet worden en hoe deze data verkregen zal worden.

Een data warehouse is een heel ruim begrip. Maar in essentie is het doel van een data warehouse grote hoeveelheden (historische) data op te slaan uit verschillende bronnen om vervolgens te kunnen gebruiken voor analyses en rapportages.

Een van de grondleggers van datawarehousing is Bill Inmon. Hij omschrijft een datawarehouse als volgt: "A (data) warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process." [2] Volgens zijn definitie zijn dit de kenmerken van een data warehouse:

- **Subject-oriented**. Data is gegroepeerd per onderwerp en niet per activiteit, zoals dit bij operationele systemen wel het geval is.
- **Integrated**. Data komt uit verschillende bronnen en zal dus eerst moeten worden afgestemd op elkaar, voor het in het data warehouse wordt opgeslagen.
- **Time-variant**. Een data warehouse bevat historische data. In tegenstelling tot een operationeel systeem, wordt de data weergegeven zoals het op een bepaald tijdstip bekend was en dit zal niet veranderen.
- **Non-volatile**. De opgeslagen data is statisch en zal niet meer veranderen wanneer het eenmaal is opgeslagen in het data warehouse.

Data warehouse of database

In het onderzoeksverslag [1] is gekeken naar de verschillen tussen een database in een operationeel systeem (zoals SPIN) en een data warehouse.

Een database in een operationeel systeem slaat real time data op van een bepaald onderdeel van het bedrijf. Bijvoorbeeld de database van SPIN. Zo'n database is bedoeld om dagelijkse handelingen snel te kunnen verwerken, zoals het toevoegen, verwijderen of aanpassen van data. Deze databases zijn goed in het uitvoeren van een grote hoeveelheden eenvoudige query's.

Zowel een data warehouse als database binnen een operationeel systeem zijn bedoeld om data in op te slaan. Het grote verschil in dit geval is dat het toevoegen of bewerken van data niet constant hoeft te gebeuren. Het toevoegen en bewerken van data gaat vaak niet realtime maar bijvoorbeeld door middel van een dagelijks import script. Het doel van het data warehouse is het uitvoeren van

complexe query's om data op te halen.

Webapplicatie

Ook zal er een webapplicatie worden gerealiseerd om onder andere het aantal transacties en product verkopen weer te geven. De webapplicatie is alleen voor intern gebruik binnen Extendas en bedoeld om een beter inzicht te krijgen van de transactie data in het data warehouse. Werknemers van Extendas kunnen op de webapplicatie inloggen en vervolgens verkoopgegevens inzien en zien hoe de query's in het data warehouse presteren.

Functionele requirements

Hieronder staan de functionele requirements beschreven per onderdeel van het data warehouse. Hierbij is onderscheid gemaakt tussen het opslaan van data in het data warehouse, het ophalen van data uit het data warehouse vanuit externe applicaties en de webapplicatie binnen het data warehouse.

Data warehouse

Het data warehouse wordt gevuld met transactiedata vanuit SPIN.

DW1	De transacties in SPIN moeten dagelijks naar het data warehouse kunnen worden gestuurd en opgeslagen in het gewenste formaat.

Externe applicaties

Op dit moment zijn er twee externe applicaties (DPS en Qlik Sense) die beide op hun eigen manier gebruik moeten kunnen maken van het data warehouse.

E1	Het data warehouse kan door middel van een API het aantal verkochte producten van een barcode per locatie per dag aan DPS leveren.
E2	De nieuwe transacties moeten dagelijks naar Qlik Sense gestuurd worden omgezet naar csv bestanden.

Webapplicatie

Om een beter inzicht te krijgen van de data in het data warehouse, moet er een webapplicatie worden gerealiseerd waarin medewerkers van Extendas informatie kunnen bekijken.

W1	Werknemers van Extendas kunnen inloggen op de webapplicatie.
W2	De ingelogde gebruiker kan per maand bekijken hoeveel transacties er zijn verwerkt.
W3	De ingelogde gebruiker kan het totaal aantal transacties bekijken.
W4	De ingelogde gebruiker kan het totaal aantal transacties per POS bekijken.
W5	De ingelogde gebruiker kan het totaal aantal transacties per organisatie bekijken.
W6	De ingelogde gebruiker kan bekijken hoe lang uitgevoerde query's duren.

Non-functionele requirements

Om een juiste keuze te kunnen maken voor de te gebruiken database, is een aantal voorwaarden opgesteld door Extendas om rekening mee te houden. Hierbij zijn vooral schaalbaarheid en compatibiliteit belangrijk voor Extendas.

	Requirement	**Beschrijving**
1	Schaalbaarheid	Het data warehouse is in staat de grote hoeveelheid transactie data op te slaan. (100.000 nieuwe transacties per dag). Voor zowel de webapplicatie als DPS moet de juiste data binnen een paar seconden opgehaald kunnen worden.
2	Compatibiliteit	Het data warehouse moet te gebruiken zijn in combinatie met de producten die Extendas al in gebruik heeft. Het backend team maakt gebruik van Symfony (PHP framework) / Doctrine (Object-relational mapper). Zie technische tools in het hoofdstuk proces.
3	Kennis	Het data warehouse kan worden gebruikt en/ of uitgebreidt worden met de huidige kennis van de andere teamleden of zonder al te veel nieuwe kennis op te moeten doen.
4	Betrouwbaarheid	De database die wordt gekozen voor het data warehouse is betrouwbaar. Vaak is er over nieuwere technologieën minder informatie beschikbaar en zijn bepaalde problemen nog niet opgelost of gevonden.
5	Documentatie	De gekozen database heeft documentatie beschikbaar waar gebruik van gemaakt kan worden.

4. Onderzoek

In dit hoofdstuk zal antwoord worden gegeven op de hoofdvraag en de daarbij horende deelvragen. Ook zullen de resultaten van het onderzoek kort worden belicht. Het hele onderzoek en de gebruikte bronnen zijn te vinden in het onderzoeksverslag [1].

Hoofdvraag

Om antwoord te kunnen geven op de hoofdvraag is voornamelijk onderzoek gedaan naar verschillende soorten databases. De hoofdvraag is als volgt:

"Hoe kan een grote hoeveelheid transactie data verzameld, opgeslagen en beschikbaar gesteld worden waarbij voldaan wordt aan de voorwaarden en wensen van Extendas?"

Deelvragen

Omdat de hoofdvraag vrij breed is, zijn er deelvragen opgesteld om te helpen bij het beantwoorden van de hoofdvraag. Dit zijn de deelvragen:

1. Welke data moet er worden verzameld?
2. Welke data mag worden verwerkt volgens de AVG?
3. Hoe wordt de verschillende data verkregen?
4. Voor welke externe applicaties moet er data beschikbaar worden gesteld?
5. Welke technieken en frameworks kunnen worden gebruikt om een data warehouse te realiseren?

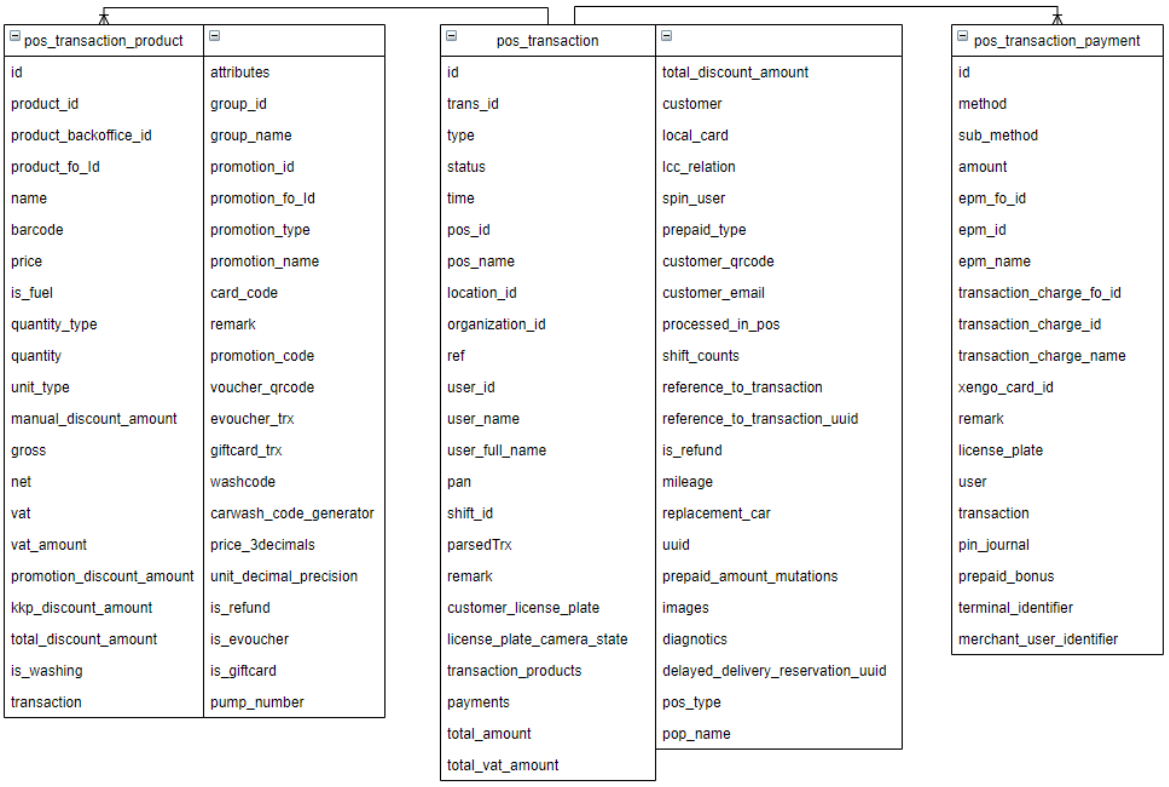
Welke data moet er worden verzameld?

Om te kunnen bepalen welke database geschikt is voor het data warehouse, is het allereerst belangrijk om te weten welke data opgeslagen moet worden. Er zijn veel verschillende databases, allemaal bedacht en uitgewerkt voor bepaalde use cases. Daarom is het belangrijk om te weten welke data opgeslagen moet worden alvorens het kiezen van de juiste database.

Transacties

Het grootste gedeelte van de data in het data warehouse zal bestaan uit transactie data. Transacties worden verwerkt en opgeslagen in SPIN. De deelnemende locaties van SWO de Splinter maken gebruik van de SPINpos en Fuel POS. De transacties van de SPINpos komen rechtstreeks in SPIN terecht en de transacties van de Fuel POS worden ook verwerkt en op dezelfde manier in SPIN opgeslagen.

Deze transacties bestaan uit 3 tabellen die van belang zijn voor het data warehouse. Een transactie bevat de gekochte producten en de betalingen die zijn gedaan voor deze transactie.



SWO de Splinter

Bovenstaande transactie data is nodig om inzichten te kunnen geven in de verkoopdata van DPS. De Splinter wil per dag per barcode het aantal verkopen kunnen bekijken voor de deelnemende locaties. Met deze informatie kunnen ze de promoties beheren en inzien hoe de promoties verlopen bij de deelnemende locaties.

Data warehouse

Op basis van de tabellen met transactiedata uit SPIN wordt een nieuwe tabel opgesteld voor het data warehouse. De data wordt zo plat mogelijk opgeslagen om het aantal joins te beperken. Het uitvoeren van joins om tabellen samen te voegen heeft invloed op de snelheid bij grote hoeveelheden data. Voor elke regel moet namelijk opnieuw gezocht worden naar de data die hierbij hoort in andere tabellen. In het onderzoeksverslag [1] in bijlage E is een aantal tests uitgevoerd om te kijken hoeveel invloed dit heeft op de snelheid van het ophalen van data.

transaction_product		
id	vat_amount	pos_id
transaction_product_id	promotion_discount_amount	pos_name
product_id	kkp_discount_amount	location_id
name	manual_discount_amount	organization_id
barcode	total_discount_amount	ref
price	transaction_id	user_id
quantity	trans_id	user_name
gross	type	product_backoffice_id
net	status	is_refund
vat	time	

Niet alle data uit de vorige tabellen is terug te vinden in de nieuwe tabel. In overleg met Extendas is gekeken naar welke kolommen van belang zijn voor het data warehouse. Een aantal kolommen is weggelaten uit de tabel voor het prototype, omdat deze in de huidige situatie niet benodigd zijn. Voor de webapplicatie zijn het id, transaction_id, location_id, pos_id en datum (time) van belang. Voor DPS komt daar nog de barcode bij.

Conclusie

Door middel van het denormaliseren van de bestaande tabellen is bovenstaande tabel ontstaan. Deze tabel bevat alle data die nodig is voor de web app en voor het ophalen van het aantal verkopen per barcode per dag voor DPS.

Welke data mag worden verwerkt volgens de AVG?

Sinds 25 mei 2018 geldt de Algemene Verordening Gegevensbescherming (AVG) voor de hele Europese Unie. Internationaal heet de wet General Data Protection Regulation (GDPR). Privacyrechten zijn door deze wet uitgebreid, waardoor je als verantwoordelijke en verwerker aan meer verplichtingen dient te voldoen bij het verwerken van persoonsgegevens.

Persoonsgegevens zijn alle gegevens die over iemand gaan of waar iemand aan te herleiden is. Denk

bijvoorbeeld aan een naam, adres of telefoonnummer. Daarnaast zijn er ook bijzondere persoonsgegevens. Deze gegevens gaan bijvoorbeeld over iemands gezondheid of politieke voorkeur. Alleen met een wettelijke uitzondering mag je deze gegevens ook verwerken. [5]

Aan de hand van de gegevens die zijn verzameld in de eerste deelvraag kan gekeken worden welke van deze gegevens persoons- of gevoelige gegevens zijn. De meeste persoonsgegevens komen niet terug in het data warehouse. De user_name en pos_name zijn beide persoonsgegevens, omdat ze te herleiden zijn naar een natuurlijk persoon.

Voor het prototype is afgesproken deze data te pseudonimiseren. Door de user_name en pos_name te hashen blijft de data lokaal wel enigszins representatief, maar is het niet gelijk te herleiden naar het origineel. Helemaal anoniem is het niet, want met de originele user_name of pos_name en het hashing algoritme is het nog steeds te achterhalen. Voor deze use case is het veilig genoeg, omdat lokaal alleen de gepseudonimiseerde data wordt opgeslagen en er dus geen toegang is tot de originele data.

De kolommen die worden aangepast voordat ze worden opgeslagen:

price (multiply)	vat_amount (multiply)
quantity (multiply)	user_name (hash)
gross (multiply)	pos_name (hash)
net (multiply)	

Conclusie

Voor het bouwen van het prototype is gebruik gemaakt van de productiedatabase. Om te zorgen dat deze data lokaal gebruikt kan worden in het prototype worden alle gevoelige gegevens aangepast opgeslagen in de database.

Hoe wordt de verschillende data verkregen?

Per applicatie zal hier worden beschreven hoe de benodigde data verkregen wordt.

SPIN

Alle transacties komen in SPIN terecht. De SPINpos verstuurt de transactie rechtstreeks naar SPIN en de Fuel POS stuurt de transacties naar de FuelPosConnector. Hier worden deze transacties omgezet naar een beter bruikbaar formaat (JSON) zodat SPIN deze vervolgens kan ophalen door middel van een API.

DPS

In DPS worden de verkoop gegevens opgehaald uit SPIN. Uiteindelijk zullen deze gegevens uit het data warehouse worden gehaald met behulp van een REST API. DPS stuurt verrijkingen (DPS trade items) terug naar de Article Service.

Data warehouse

Het data warehouse zal eenmaal per dag de nieuwe transacties uit SPIN krijgen door middel van twee csv bestanden. (Een voor transacties en een voor de bijbehorende producten) Deze worden binnen het data warehouse uitgelezen, samengevoegd, bewerkt en opgeslagen in het data warehouse.

Webapplicatie

De webapplicatie zal de data vanuit het data warehouse kunnen aanroepen met behulp van de REST API.

Conclusie

Voor de opdracht is het alleen van belang hoe de data vanuit SPIN in het data warehouse terecht komt en hoe het data warehouse deze data weer kan aanleveren aan DPS en de webapplicatie. Dit zal dagelijks gebeuren door middel van twee csv bestanden met de nieuwe transactie data.

Voor welke externe applicaties moet er data beschikbaar worden gesteld?

In de eerste fase zal het data warehouse data beschikbaar stellen voor het dashboard van de Splinter (DPS) en voor de BI tool Qlik Sense.

DPS

Voor de Splinter wordt een webapplicatie gebouwd waarmee o.a. de dagelijkse werkzaamheden gedaan kunnen worden zoals:

- Beheren van leden
- Aanmaken van promotie kalenders
- Verrijken van (nieuwe) artikelen
- Monitoren van promoties
- Een gepersonaliseerd overzicht van de verkochte promotie artikelen per deelnemende organisatie sturen

Qlik Sense

Qlik Sense is een data analyse platform waarmee gebruikers zelf eenvoudig en snel dynamische dashboards en statische rapporten kunnen maken. Met de data discovery tool is het mogelijk om alle mogelijke verbanden te vinden in de datasets.

Door middel van connectors is het mogelijk de data snel te koppelen aan Qlik Sense. Er zijn heel veel verschillende connectors beschikbaar voor bekende databases, cloud services en applicaties. Ook worden veel bestandstypes ondersteund en kan Qlik Sense deze bestanden automatisch uitlezen en verwerken.

Extendas maakt al gebruik van een private server voor Qlik Sense. Vanuit FuelOffice wordt een groot aantal csv bestanden gegenereerd die via een (S)FTP verbinding naar de Qlik Sense server worden verstuurd. Vanuit Qlik Sense worden deze csv bestanden vervolgens naar QVD (QlikView Data) bestanden omgezet en opgeslagen. Vanuit het dashboard kunnen deze QVD bestanden worden opgehaald en uitgelezen.

Conclusie

DPS is de belangrijkste applicatie die in eerste instantie gebruik zal gaan maken van het data warehouse. De focus voor het realiseren en het ontwerpen van het prototype zal dan ook voornamelijk liggen op DPS en niet op Qlik Sense.

Welke technieken en frameworks kunnen worden gebruikt om een data warehouse te realiseren?

Het grootste deel van het onderzoek bestond uit het uitzoeken van de juiste database voor het data warehouse. Om een keuze te kunnen maken voor een goede database wordt er eerst gekeken naar welke type databases er beschikbaar zijn.

De databases kunnen op verschillende manieren worden onderscheiden. Allereerst is gekeken naar de manier van dataverwerking. Een database maakt vaak gebruik van OLTP (Online Transaction Processing) om een kleine hoeveelheid data snel te kunnen bewerken, verwijderen en toevoegen. Een data warehouse maakt vaak gebruik van OLAP {OnLine Analytical Processing). OLAP maakt het mogelijk om data uit verschillende bronnen te analyseren . [8]

Daarnaast kan worden gekeken naar ACID vs non-ACID eigenschappen. Veel relationele databases houden zich aan de ACID eigenschappen (atomic, consistent, isolated, durable). Deze eigenschappen zijn er voor bedoeld dat bijvoorbeeld rijen in de database niet tegelijkertijd gewijzigd kunnen worden. In een datawarehouse worden deze regels vaak minder strikt genomen onder andere omdat voornamelijk het lezen van data belangrijk is. Bovendien komt het bewerken veel minder tot niet voor in een data warehouse. [8]

In het onderzoeksverslag [1] is meer informatie te vinden over de verschillende manieren waarop onderscheid gemaakt kan worden tussen de verschillende soorten databases. Hierin wordt verder ingegaan op bovenstaande onderdelen.

Voor het kiezen van de juiste database voor het data warehouse is vooral gekeken naar een aantal punten die belangrijk zijn voor de opdracht en naar de non-functionele requirements:

- Wel of geen relationeel data model
- Gebruik van SQL
- Mate van schaalbaarheid
- ACID vs non-ACID

Met behulp van bovenstaande onderdelen zijn de type databases onderverdeeld in de volgende drie categorieën: Relationele databases, NoSQL databases en NewSQL databases.

Relationele database

De bekendste en meest gebruikte database soort is de relationele database. Veel operationele systemen maken gebruik van een relationele database. De database bestaat uit een collectie van tabellen die met elkaar verbonden zijn. Zo kan een rij in de ene tabel verbonden zijn met een rij in de andere tabel door middel van een overeenkomende waarde in een kolom. Bijvoorbeeld bij een tabel met toetscijfers heeft elke regel een studentnummer, dat overeenkomt met het studentnummer in de student tabel.

Een relationele database is gestructureerd en daardoor is van te voren het schema van de toe te voegen data bekend. Alle rijen in een tabel voldoen aan deze structuur. Bij het ontwerpen van dit schema wordt vaak gebruik gemaakt van normalisatie. In het onderzoeksverslag [1] worden de verschillende normaalvormen beschreven en een voorbeeld gegeven van hoe de het normaliseren van een database werkt.

NoSQL

Een NoSQL (Not only SQL) database heeft vaak geen vaste structuur voor rijen. Veel van deze NoSQL databases maken gebruik van documenten (bijvoorbeeld JSON), die verschillende structuren kunnen hebben. Een groot voordeel van NoSQL databases is dat ze gelijk horizontaal schaalbaar zijn. Dat wil zeggen dat het uitbreiden van de database gebeurt door nieuwe servers toe te voegen. De data is opgesplitst over meerdere servers. Dit kan in sommige gevallen goedkoper zijn dan het toevoegen van resources aan een bestaande server (verticaal schalen). Afhankelijk van de hoeveelheid toegevoegde servers, neemt de complexiteit voor het beheren van het systeem ook toe. Dit brengt uiteindelijk ook extra kosten met zich mee.

NewSQL

De meest recente soort database is de NewSQL database. NewSQL is een verzamelnaam voor databases die als eigenschap hebben dat ze de horizontale schaalbaarheid van NoSQL databases bereiken en toch zoveel mogelijk volgens de ACID regels te werken zoals bij dat gebeurd bij een relationele database.

Al deze NewSQL databases verschillen in de manier waarop ze deze eigenschappen combineren. Ze hebben allemaal gemeen dat ze het relationele model ondersteunen en de taal SQL ondersteunen. Sommige van deze databases ondersteunen compatibiliteit met al bestaande relationele databases zoals PostgreSQL.

Databases

Met behulp van bovenstaande drie categorieën databases en met de voorwaarden van Extendas worden in het onderzoeksverslag [1] de bekendere databases vergeleken. Bij deze non-functionele requirements staat hieronder een score van 1 (laagste) tot 5 (hoogste). Schaalbaarheid en compatibiliteit zijn hierbij voor Extendas het belangrijkst en deze tellen dan ook twee keer ten opzichte van de rest.

Database	Schaalbaarheid	Compatibiliteit	Kennis	Betrouwbaarheid	Documentatie
MySQL (Relationeel)	3 (6)	5 (10)	5	4	5
MariaDB (Relationeel)	3 (6)	5 (10)	5	4	5
PostgreSQL (Relationeel)	4 (8)	5 (10)	4	4	5
MongoDB (NoSQL)	5 (10)	4 (8)	3	4	5
Elasticsearch (NoSQL)	5 (10)	2 (4)	2	3	4
Cassandra (NoSQL)	5 (10)	2 (4)	3	3	4
CockroachDB (NewSQL)	5 (10)	2 (4)	3	3	2

Verdere uitleg voor de database met bovenstaande voorwaarden is terug te vinden in het onderzoeksverslag. [1]

Conclusie

Er zijn heel veel soorten databases om uit te kiezen. Zo is elke database bedacht vanuit een bepaalde use case en daar goed in. Bovenstaande databases hebben dan ook alle hun eigen voor- en nadelen. Om een keuze te kunnen maken voor de juiste database, moet worden gekeken naar welke van de voorwaarden het belangrijkst zijn en welke database aan al deze voorwaarden het beste kan voldoen.

Zoals hierboven in de tabel is te zien, komen veel databases dicht bij elkaar in de buurt of hebben dezelfde score (30). PostgreSQL komt hier met 31 punten nog maar net overheen. De keuze voor PostgreSQL heeft dan ook vooral te maken met de extra mogelijkheden die het biedt voor het bouwen van een data warehouse ten opzichte van de andere relationele databases.

Uit mijn onderzoek blijkt dat PostgreSQL aan al deze voorwaarden voldoet en in bijvoorbeeld schaalbaarheid en snelheid beter presteert dan de andere relationele databases. Compatibiliteit met Symfony / Doctrine is voor Extendas een belangrijke voorwaarde, omdat veel projecten en het hele backend team hier gebruik van maken en bekend mee is. PostgreSQL wordt goed ondersteund door

Doctrine en qua programmeren werkt dit hetzelfde als met de huidige MariaDB.

PostgreSQL maakt gebruik van SQL en houdt zich daarbij goed aan de standaard. Dit biedt Extendas veel opties om gebruik te kunnen maken van de bestaande tools van SQL om de database aan te spreken. Zo kunnen er mogelijk sneller nieuwe functionaliteiten worden ontwikkeld.

PostgreSQL biedt ook de mogelijkheid voor extensies. Dit maakt het bijvoorbeeld mogelijk om Elasticsearch realtime te syncen met de PostgreSQL database. Mochten er in de toekomst meer en complexere analyses moeten worden gemaakt en is de PostgreSQL database daar niet snel genoeg voor, dan is het eenvoudig te koppelen aan Elasticsearch om dit mogelijk te maken. Daarnaast maakt ook CockroachDB gebruik van het wire protocol van PostgreSQL, wat ervoor zorgt dat deze grotendeels dezelfde query's aan kan. Naar eigen zeggen maakt CockroachDB hier gebruik van omdat deze beter gedocumenteerd is dan het MySQL protocol. PostgreSQL gebruikt het Open Source license in tegenstelling tot het GNU license van MySQL. [9]

5. Proces

In dit hoofdstuk wordt het proces en de globale planning voor de afstudeerperiode beschreven. Ook de gebruikte tools worden hier omschreven.

Ontwikkelmethodiek

Voor de afstudeerstage wordt gebruik gemaakt van een combinatie van Scrum en Kanban, ook wel Scrumban [3] genoemd. Alleen de benodigde functionaliteiten van beide methodieken worden hierbij gebruikt. In tegenstelling tot scrum worden er geen strikte sprints gehouden. Elke week wordt er besproken wat er die week gaat gebeuren en wat er is gedaan de afgelopen week. Wanneer het meer werk is, zal het over twee weken verspreid worden. Tijdens het realiseren van het prototype, wordt in de wekelijkse bespreking ook een korte demo gegeven van de nieuwe ontwikkelingen.

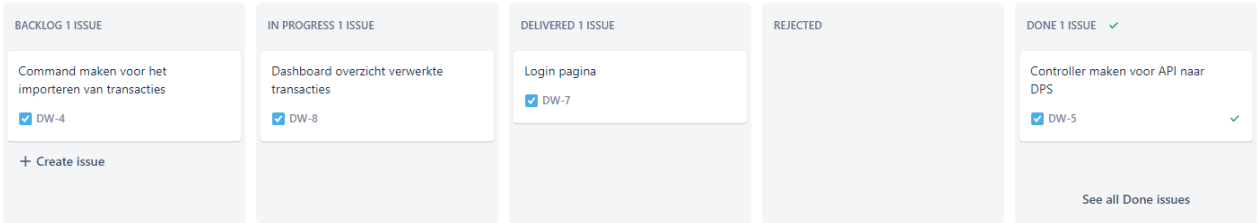
Daily standup

Dagelijks wordt er een standup gehouden met het backend team. De afstudeerder zal hier ook aan deelnemen. Tijdens de standup wordt besproken wat er de dag ervoor is gedaan, wat er die dag gaat gebeuren en eventueel om extra feedback te ontvangen.

Kanban board

Voor het bouwen van het prototype wordt gebruik gemaakt van een kanban board. Dit is een bord waar de taken op komen te staan. Voor het kanban board is gebruik gemaakt van Jira. Het kanban board bevat de volgende stappen:

- **Backlog**. Een lijst met alle taken die uitgevoerd moeten worden.
- **In progress.** Taken die op dat moment worden uitgevoerd. Hier kunnen maximaal 3 taken tegelijk komen te staan om het overzicht te bewaren.
- **Delivered.** Wanneer een taak is afgerond door de afstudeerder, komt deze in delivered. De taak kan dan worden nagekeken.
- **Rejected.** Wanneer een taak in delivered nog niet voldoet aan de eisen of nog kan worden verbeterd, komt deze in rejected te staan.
- **Done.** Taken die zijn afgerond en aan de eisen voldoen, komen in done terecht. Dit is de laatste stap.



Globale planning

Week	1	2	3	4	5	6	7	8	9	10
Voorbereidingsfase										
Schrijven Plan van Aanpak										
Inleveren concept Plan van Aanpak										
Inleveren definitief Plan van Aanpak										
Onderzoeksverslag										
Scriptie										

Week	11	12	13	14	15	16	17	18	19	20
Onderzoeksverslag										
Prototype data warehouse realiseren										
Scriptie										
Functioneel / technisch ontwerp										

Naar aanleiding van de uitbreiding op de afstudeeropdracht is bovenstaand functioneel / technisch ontwerp erbij gekomen voor de webapplicatie.

Hieronder de planning vanaf de verlenging van de afstudeerperiode (Van 1 februari t/m 6 april).

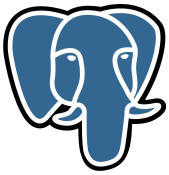
Week	21	22	23	24	25	26	27	28	29	30
Scriptie										
Prototype data warehouse realiseren										
Adviesrapport										
Concept afstudeerdossier inleveren										
Definitief afstudeerdossier inleveren										
Presentatie inleveren										
Afronding afstuderen										

Projectmatige tools

Voor het bouwen van het prototype en het uitvoeren van de afstudeeropdracht zijn verschillende tools gebruikt. Hieronder worden deze kort omschreven ondergedeeld in projectmatige en technische tools.

	**Google Drive** Google Drive werd gebruikt om de verschillende documenten (scriptie, adviesrapport, onderzoeksverslag etc.) op te slaan en bewerken met behulp van Google Docs.
	**Slack** Slack werd gebruikt voor de communicatie met collega's binnen Extendas. Vrijwel alle communicatie is gedaan via Slack.
	**Google Meet** De dagelijkse stand up vond plaats via Google Meet. Google Meet maakt het mogelijk met elkaar te communiceren door middel van video.
	**Microsoft Outlook** Outlook werd gebruikt voor de communicatie met de afstudeerbegeleider van het Saxion.
	**Microsoft Teams** Teams werd gebruikt voor het tussentijds bespreken van de status van het afstuderen met de afstudeerbegeleider van het Saxion.
	**Confluence** In Confluence is de opdracht voor het Splinter dashboard beschreven. Hierin staat ook kort beschreven wat er in het data warehouse moest komen.
	**Lucidchart** Lucidchart is een tool om diagrammen mee te maken. De diagrammen voor de opdracht voor de Splinter zijn hierin gemaakt. Aan de hand van deze diagrammen zijn de modellen voor het data warehouse uitgewerkt.
	**Jira** Met de software van Jira kun je te realiseren software plannen, volgen en uitrollen. Binnen Extendas wordt Jira gebruikt om in teamverband overzicht te houden binnen de projecten en taken te verdelen.

Technische tools

	**PhpStorm** PhpStorm is een IDE die het makkelijk maakt om in PHP te ontwikkelen. Ook biedt PhpStorm ondersteuning voor Symfony en bijvoorbeeld Git. PhpStorm werd gebruikt voor het bouwen van het prototype met Symfony.
	**PostgreSQL** PostgreSQL is gekozen als database oplossing voor het data warehouse.
	**Datagrip** Handige tool om databases mee uit te lezen. Deze is voornamelijk gebruikt voor het testen en bekijken van de verschillende databases.
	**Git** Systeem voor versiebeheer. Gebruikt voor het bouwen van het prototype.
	**GitHub** GitHub is een online platform waarop software opgeslagen kan worden. Dit is gebouwd rond Git. De projecten bij Extendas worden op GitHub opgeslagen en bijgehouden, zo ook het gebouwde prototype.
	**GitKraken** GitKraken is een gratis tool die het gebruik van Git een stuk overzichtelijker maakt. Deze tool voornamelijk gebruikt voor het bekijken van de status van het project in Git en bijvoorbeeld het oplossen van merge conflicten.
	**PHP** PHP is de programmeertaal die voornamelijk wordt gebruikt binnen het back end team van Extendas. PHP is ook gebruikt als een van de voorwaarden voor de database. Het prototype is gebouwd met behulp van PHP.
	**Symfony** Symfony is een framework voor PHP. De meeste producten van het back end team binnen Extendas maken gebruik van dit framework. Het prototype is ook gebouwd met behulp van dit framework.

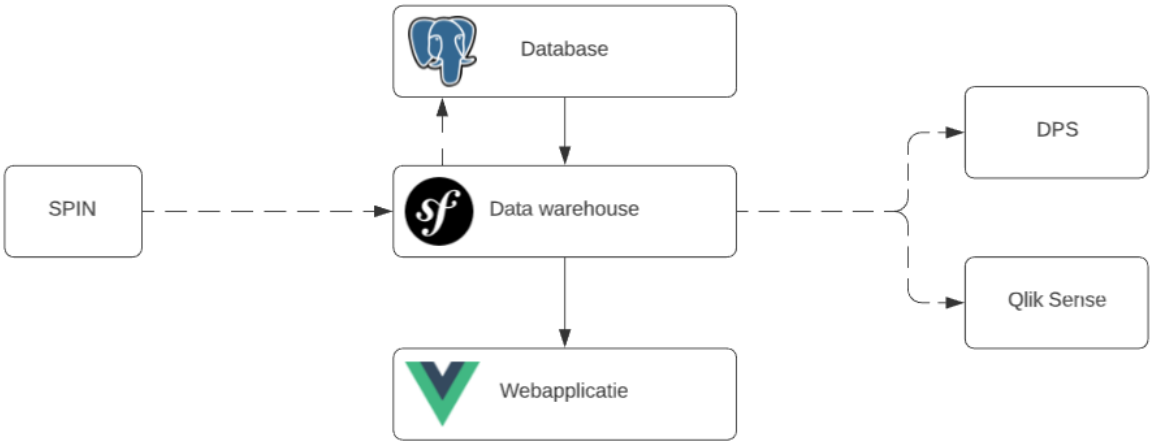
	**Doctrine** Doctrine is een set aan PHP libraries om met de database te werken. Denk hierbij aan een ORM (Object-relational mapper) en DQL (Doctrine Query Language). Deze tools ondersteunen relationele databases zoals MySQL, MariaDB en PostgreSQL, maar ook NoSQL databases zoals MongoDB.
	**Composer** Composer is een dependency manager voor PHP. Deze wordt gebruikt om verschillende bundels en pakketten te downloaden voor het Symfony project.
	**Docker** Docker wordt gebruikt om verschillende software in zogenoemde containers uit te voeren. Deze containers zijn geïsoleerd, wat er voor zorgt dat bijvoorbeeld de verschillende databases geen invloed op elkaar hebben. Het prototype draait met behulp van Docker containers voor bijvoorbeeld de database en de server.
	**Postman** Postman is een tool om API's mee te testen. Voornamelijk gebruikt voor het ophalen van data uit de bestaande API's en voor het testen van de API calls in het prototype.
	**NGINX** NGINX is een webserver voor HTTP. Om het symfony project lokaal te draaien in Docker is gebruik gemaakt van NGINX.
	Vue.js Vue.js is een JavaScript framework voor het bouwen van webapplicaties en gebruikers interfaces. De webapplicatie wordt gebouwd in Vue.js.
	**npm** Npm is een pakketbeheerder voor JavaScript. Npm maakt het mogelijk om de verschillende libraries in het Vue.js project te installeren via de terminal.

6. Ontwerp

In dit hoofdstuk wordt het ontwerp van het data warehouse en de webapplicatie beschreven.

Data warehouse

In figuur 1 (pagina 10) is het data warehouse het onderdeel dat gerealiseerd moet worden. Dit data warehouse bestaat uit een database om de transacties in op te slaan, een webapplicatie om Extendas meer inzicht te geven in de beschikbare data en de bijbehorende backend voor het verwerken van nieuwe transacties en de REST API naar externe applicaties / webapplicatie. Door het beantwoorden van de deelvragen is duidelijk geworden welke onderdelen er in het data warehouse moeten komen. De keuzes voor de implementatie zijn daarbij gebaseerd op de functionele en non-functionele requirements van Extendas.



Database

Het data warehouse maakt gebruik van een PostgreSQL database met een tabel om alle transacties met daarbij horende producten in op te slaan. De tabel is een combinatie van de bestaande transactie en product tabellen uit SPIN. In tegenstelling tot de operationele database van SPIN, wordt in een data warehouse de data gedenormaliseerd opgeslagen. Het normaliseren van data zorgt voor betere integriteit en maakt het eenvoudiger om data te bewerken. Voor een data warehouse is het bewerken van data minder belangrijk, omdat dit eigenlijk niet voorkomt. Voor het ophalen van data zorgt het denormaliseren van de tabellen ervoor dat er geen JOIN statements benodigd zijn en dat heeft invloed op de performance. Het uitvoeren van JOIN statements kost veel tijd omdat voor elke regel in een tabel, de juiste regels uit andere tabellen bij elkaar moet zoeken. In het onderzoeksverslag [1] (bijlage E) is een aantal tests uitgevoerd met verschillende datasets om te kijken hoeveel invloed JOIN statements kunnen hebben op de snelheid van het ophalen. Hieruit blijkt dat voornamelijk bij een grotere hoeveelheid data (5 miljoen regels) dit wel degelijk invloed heeft op de snelheid van het ophalen (Ongeveer 40% sneller).

De gedenormaliseerde tabel in het data warehouse ziet er als volgt uit:

transaction_product	
id (integer) (primary key)	transaction_id (integer)
transaction_product_id (integer)	trans_id (bigint)
product_id (integer)	type (integer)
name (varchar)	status (integer)
barcode (varchar)	time (timestamp)
price (integer)	pos_id (integer)
quantity (numeric)	pos_name (varchar)
gross (integer)	location_id (integer)
net (integer)	organization_id (integer)
vat (double)	ref (integer)
vat_amount (integer)	user_id (integer)
promotion_discount_amount (integer)	user_name (varchar)
kkp_discount_amount (integer)	product_backoffice_id (varchar)
manual_discount_amount (integer)	is_refund (smallint)
total_discount_amount (integer)	

Om snel door deze tabel te kunnen zoeken, wordt een aantal indices toegevoegd. Een index zorgt ervoor dat er minder vergelijkingen nodig zijn om een of meerdere regels te vinden. Een groot deel van de informatie in de web app is gebaseerd op tijd. Dit is dan ook de belangrijkste kolom om toe te voegen aan een index.

index	
time_transaction_idx	transaction_id (ASC), time (DESC)

In veel gevallen maakt het bij een index niet uit welke volgorde je meegeeft. Bij een index met meerdere kolommen kan dit soms wel effect hebben. Na dit zelf geprobeerd te hebben met de tijd zowel op descending als op ascending, leek dit inderdaad invloed te hebben op de snelheid van de query's. Vandaar dat de order van time hier op DESC (aflopend) wordt gezet.

index	
pos_transaction_idx	transaction_id (ASC), pos_id (ASC)

Voor het ophalen van het aantal transacties voor een POS is een index toegevoegd op het pos_id in combinatie met het transaction_id.

index	
organization_transaction_idx	transaction_id (ASC), organization_id (ASC)

Voor het ophalen van het aantal transacties voor een organisatie is een index toegevoegd op het organization_id in combinatie met het transaction_id.

index	
barcode_transaction_idx	transaction_id (ASC), barcode(ASC)

Voor DPS moet er gezocht kunnen worden aan de hand van barcodes. Met de huidige hoeveelheid data verandert de snelheid van het zoeken van het aantal transacties voor een barcode van een minuut naar een paar seconden door deze index.

In het onderzoeksverslag [1] (bijlage G) staat een aantal tests beschreven op query's die gebruik maken van de indices. Hieruit blijkt dat het gebruiken van bovenstaande indices veel invloed heeft op de snelheid van het ophalen van data. In beide tests is de snelheid gemiddeld met 98% versneld door het gebruiken van een index.

Indices brengen ook een aantal nadelen met zich mee. Een klein nadeel is dat elke index extra ruimte nodig heeft in de database. De transaction_product tabel die lokaal voor het prototype is gebruikt is zo'n 12 GB en gebruikt voor bovenstaande indices ongeveer 4 GB.

Maar een groter nadeel is dat indices invloed hebben op de snelheid van het toevoegen, bewerken of verwijderen van regels in de database. Bij elke INSERT, UPDATE of DELETE statement moet ook de bijbehorende indices worden bewerkt. UPDATE en DELETE komt niet / nauwelijks voor in een data warehouse. INSERT wel, daarom is het een mogelijkheid om de indices te verwijderen alvorens het importeren en vervolgens weer toe te voegen na het importeren.

Naast de transaction_product tabel is er een tabel voor de verschillende gebruikers. Hierin worden de gebruikersnaam, email en wachtwoord (gehasht) opgeslagen. Ook de rollen van de gebruiker zijn hier in te vinden. Deze rollen worden opgeslagen in json formaat, zodat Symfony hier automatisch gebruik van kan maken en het biedt de mogelijkheid om meerdere rollen aan een gebruiker toe te kennen.

User	
id (integer) (primary key)	email (varchar)

username (varchar)	roles (json)
password (varchar)	

Backend

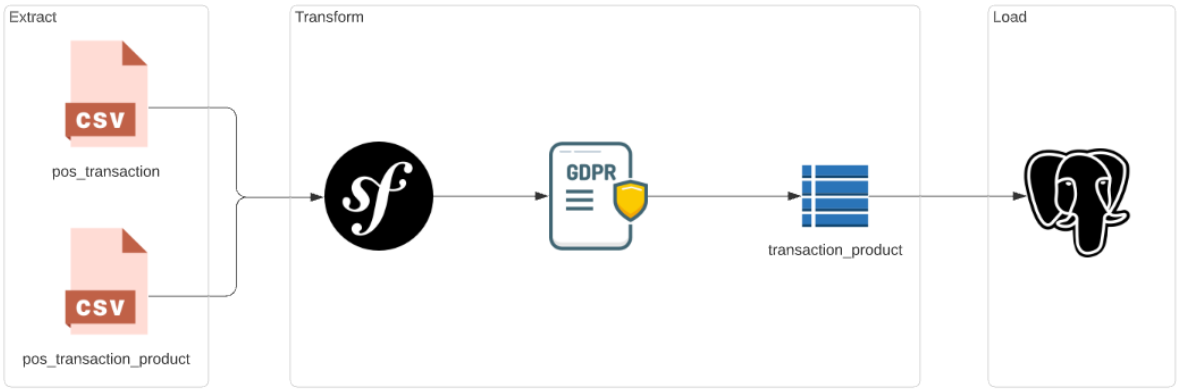
Om het data warehouse te benaderen en beschikbaar te stellen voor de webapplicatie en externe applicaties wordt een project in Symfony opgezet. Dit is het framework dat wordt gebruikt voor de meeste projecten binnen Extendas en hier is dan ook veel kennis over binnen het backend team. Naast de kennis binnen Extendas, bevat dit framework alle onderdelen die nodig zijn om de verschillende stappen van het data warehouse te ontwikkelen.

De backend bestaat uit het importeren van de transactie data in het datawarehouse en de REST API voor de webapplicatie en DPS.

Importeren

Een belangrijke term voor een data warehouse is ETL. De afkorting staat voor Extract, Transform en Load. ETL beschrijft het proces van extraheren van data uit de bron, vervolgens het transformeren naar het juiste formaat en als slot de getransformeerde data laden in het data warehouse. [10]

Het ETL proces voor het importeren van transactie data in het data warehouse ziet er als volgt uit:



Extendas moet de mogelijkheid hebben om de transacties en producten vanuit SPIN te kunnen importeren in het data warehouse. Hiervoor is bepaald dat er dagelijks twee csv bestanden beschikbaar worden gesteld door Extendas met de nieuwe transacties en producten bij deze transacties.

In het prototype komt daar nog bij dat bepaalde kolommen moeten worden gepseudonimiseerd. Dit gebeurt gelijk bij het importeren. Met behulp van een command kan Extendas het importeren starten. De twee csv bestanden worden vervolgens samengevoegd tot een csv bestand met alle regels die in het data warehouse moeten komen. Het nieuwe csv bestand wordt vervolgens in de database geïmporteerd met behulp van het COPY statement. COPY is een statement binnen PostgreSQL dat snel data uit een csv bestand naar de database kan kopiëren. Dit ziet er als volgt uit:

```
\COPY transaction_product_test(transaction_product_id, product_id, name, barcode, price, quantity,
gross, net, vat, promotion_discount_amount, kkp_discount_amount, manual_discount_amount, vat_amount,
total_discount_amount, product_backoffice_id, transaction_id, trans_id, type, time, pos_id,
location_id, ref, user_id, status, user_name, pos_name, organization_id, is_refund)
\FROM 'path\to\file\enriched_transactions.csv' (format csv, null "NULL", DELIMITER ',', HEADER)
```

Zoals eerder benoemd, hebben indices invloed op de snelheid van het toevoegen van nieuwe regels aan de database. De manier om toch zo snel mogelijk nieuwe transacties te importeren is door de indices voor het importeren te verwijderen en na het importeren weer toe te voegen. Het opnieuw toevoegen van de indices duurt even door de grote van de database, maar neemt minder tijd in beslag dan wanneer de indices blijven bestaan tijdens het importeren.

REST API

Door middel van een REST API kan data worden opgehaald uit het data warehouse. Voor zowel DPS als de webapplicatie wordt de data geserveerd vanuit de API in JSON formaat. Elke API call geeft het resultaat terug, de SQL query die is uitgevoerd en de tijd die nodig was om deze SQL query uit te voeren. Om de performance te verbeteren wordt hierbij zo weinig mogelijk gebruik gemaakt van de functionaliteiten binnen Doctrine. Omdat veel query's gebruik maken van de datum, is ook hiervoor gekeken naar de snelste manier om op de datum / periode te sorteren. Bijvoorbeeld door het gebruik van de index op datum en de functies in de query om te zoeken tussen twee verschillende datums.

De volledige uitwerking van onderstaande API endpoints met daarbij horende JSON responses zijn te vinden in bijlage A.

Methode	**Url**
POST	/api/login_check
Import	
GET	/api/transaction/last_transaction_id
Webapplicatie	
GET	/api/transaction/total_yesterday
GET	/api/transaction/total_last_month
GET	/api/transaction/total_count
GET	/api/transaction/total_per_month/{year}
GET	/api/product/total_yesterday
GET	/api/product/total_last_month
GET	/api/product/total_count
GET	/api/product/total_per_month/{year}

GET	/api/pos/total_yesterday/{pos_id}
GET	/api/pos/total_last_month/{pos_id}
GET	/api/pos/total_count/{pos_id}
GET	/api/pos/total_per_month/{year}/{pos_id}
GET	/api/organization/total_yesterday/{organization_id}
GET	/api/organization/total_last_month/{organizations_id}
GET	/api/organization/total_count/{organization_id}
GET	/api/organization/total_per_month/{year}/organization_id}
DPS	
GET	/api/dps/sales

Beveiliging

De REST API wordt beveiligd met JWT (JSON Web Token) authenticatie. Het voordeel hiervan is dat de token die mee wordt gegeven aan een API request, informatie bevat over de gebruiker. Hierdoor weet de server of de gebruiker bevoegd is om bepaalde requests aan te roepen.

De gebruiker vult zijn gebruikersnaam en wachtwoord in de webapplicatie in. Deze roept vervolgens de login call in de REST API aan om een JWT token te ontvangen. De token wordt in de browser opgeslagen, zodat de webapplicatie deze kan gebruiken voor de andere API calls afhankelijk van onderstaande rollen. Als de gebruiker de rol in zijn JWT token heeft, weet de REST API of de gebruiker bevoegd is om de betreffende data op te halen.

Rollen

Om onderscheid te maken in de verschillende API calls, worden deze gekoppeld aan een gebruikersrol.

Rol	**Bevoegdheid**
Admin	Heeft bevoegdheid tot alle API calls, waaronder ook het ophalen van het laatste transactie id.
App user	Kan alle API calls die gebruikt worden in de webapplicatie aanroepen.
DPS	Kan de API calls voor DPS aanroepen.

Testen

Voor het testen in Symfony wordt gebruik gemaakt van PHPUnit. PHPUnit maakt het mogelijk om unit en functional tests uit te voeren. Voor elke API call wordt een aparte test geschreven om de URL aan te roepen met en zonder authenticatie en te testen op de verschillende statuscodes.

Webapplicatie

Voor intern gebruik wordt er een webapplicatie ontwikkeld waarmee Extendas verkoopgegevens kan bekijken uit het data warehouse. Denk hier aan het aantal transacties en producten verdeeld over de maanden. Ook kunnen de uitgevoerde query's hierin worden bekeken om te bepalen of er ergens nog verdere optimalisaties benodigd zijn.

De webapplicatie wordt volledig in Vue.js gebouwd en maakt gebruik van de API uit de backend om data op te halen en weer te geven. De gehele applicatie staat los van het data warehouse.

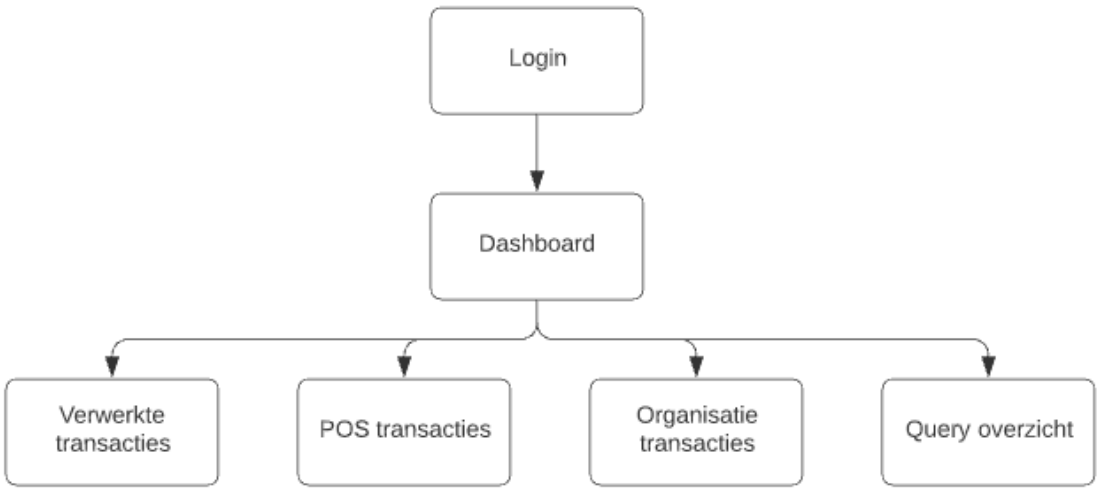
Functioneel ontwerp

Zoals in de opdracht beschreven zijn dit de functionele eisen waaraan de webapplicatie moet voldoen.

W1	Werknemers van Extendas kunnen inloggen op de webapplicatie.
W2	De ingelogde gebruiker kan per maand bekijken hoeveel transacties er zijn verwerkt.
W3	De ingelogde gebruiker kan het totaal aantal transacties bekijken.
W4	De ingelogde gebruiker kan het totaal aantal transacties per POS bekijken.
W5	De ingelogde gebruiker kan het totaal aantal transacties per organisatie bekijken.
W6	De ingelogde gebruiker kan bekijken hoe lang uitgevoerde query's duren.

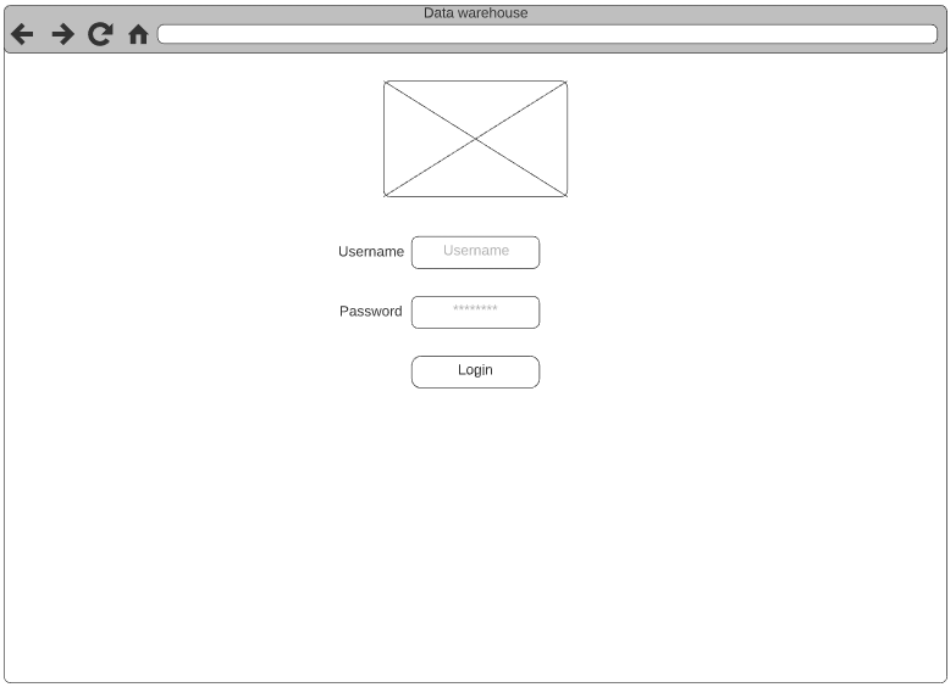
Navigatie

Navigatie binnen de webapplicatie kan in het menu aan de linkerkant. Dit menu komt beschikbaar wanneer de gebruiker is ingelogd. Vanuit dit menu kan genavigeerd worden naar de verschillende overzichten.



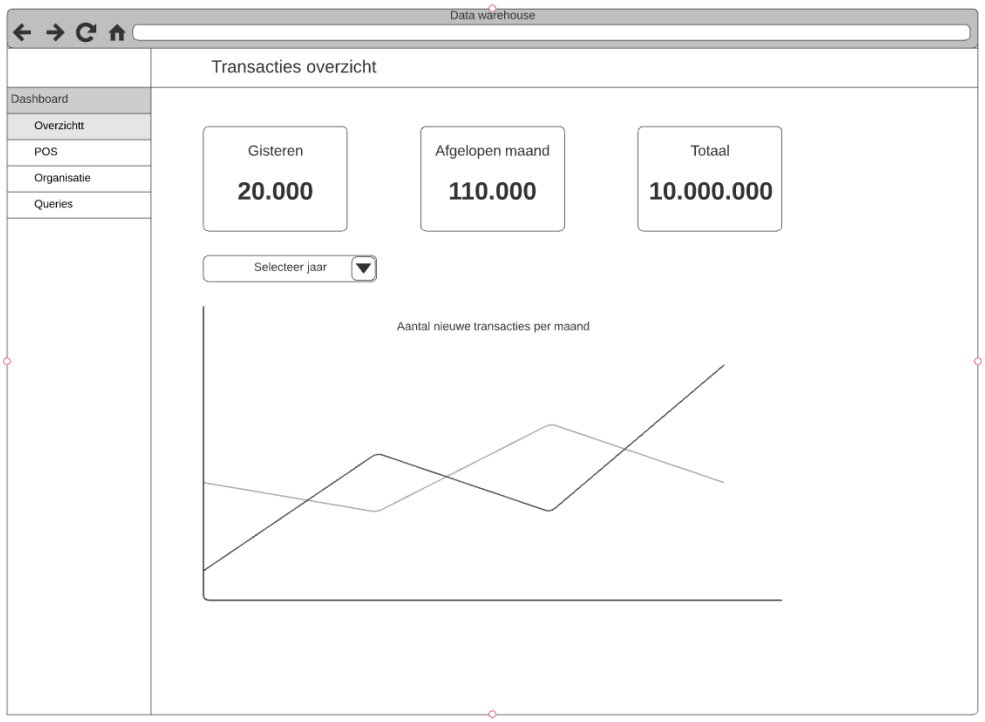
Login pagina

Bij het openen van de webapplicatie komt de gebruiker eerst op de login pagina terecht. Deze pagina zorgt ervoor dat alleen mensen met de juiste gegevens in kunnen loggen en gebruik kunnen maken van de webapplicatie. (functionele requirement 1)



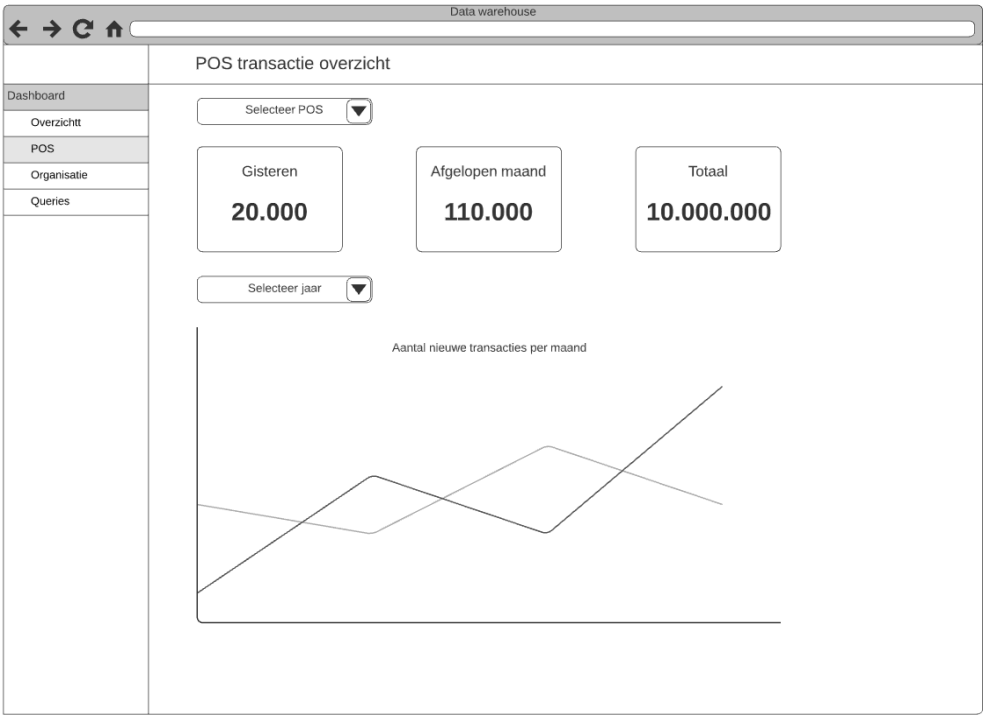
Dashboard overzicht pagina

Wanneer de gebruiker is ingelogd, krijgt deze een overzicht te zien met daarin informatie over het aantal transacties / producten. (functionele requirement 2 en 3)



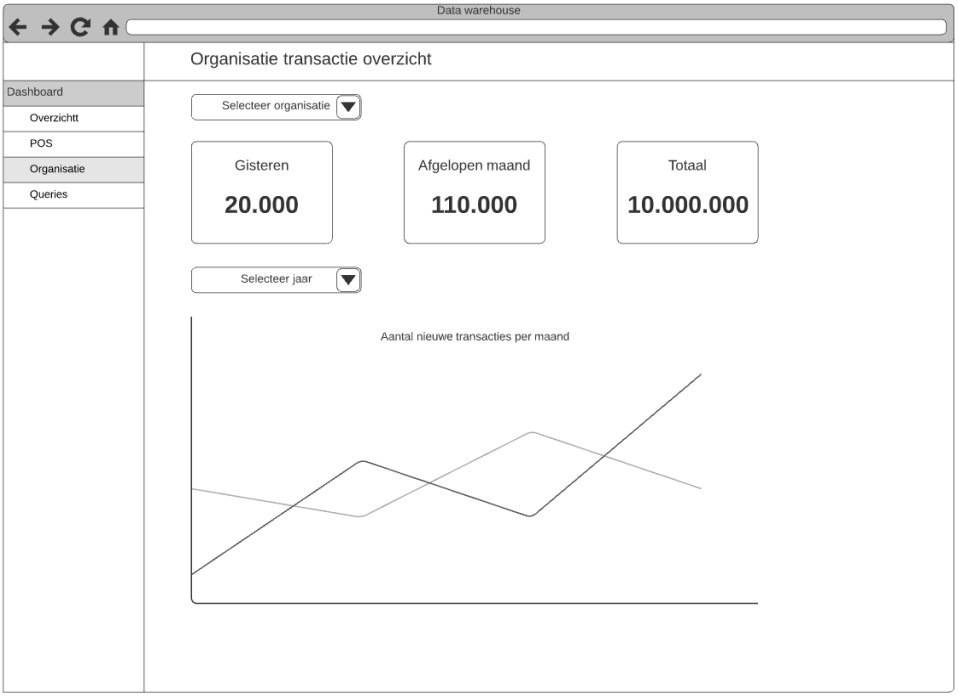
POS transactie pagina

Op de POS pagina kan de gebruiker bekijken hoeveel transacties er per maand per POS zijn. (functionele requirement 4)



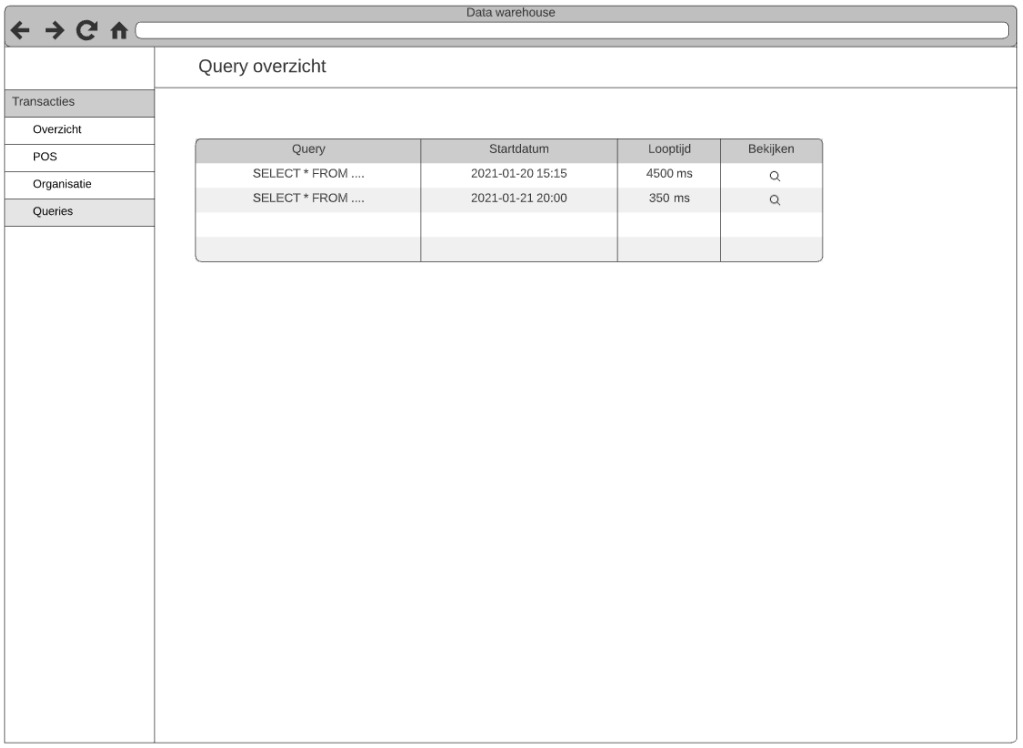
Organisatie transactie pagina

Op de organisatie pagina kan de gebruiker bekijken hoeveel transacties er per maand per organisatie zijn. (functionele requirement 5)



Query pagina

Op de query pagina kan de gebruiker de query's zien die worden uitgevoerd wanneer de webapplicatie data uit het data warehouse ophaalt. (functionele requirement 6)

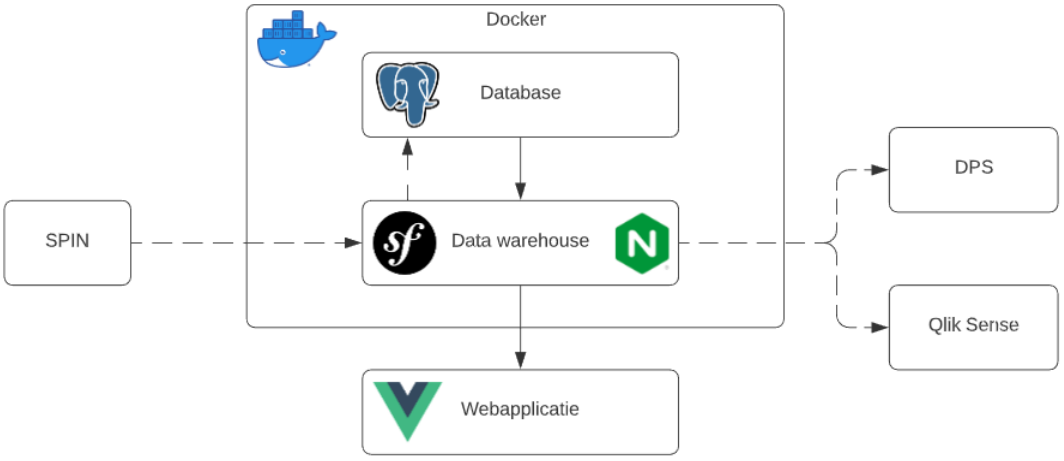


7. Prototype

In dit hoofdstuk wordt het prototype van het data warehouse beschreven. Het project is gebouwd met behulp van Symfony / Doctrine voor de backend en maakt gebruik van PostgreSQL als database oplossing. De web app is gemaakt in Vue.js en draait voor het ontwikkelen lokaal met behulp van de Vue CLI server.

Backend

De backend is gebouwd in de nieuwste Symfony versie (5.2) en maakt gebruik van Doctrine voor het benaderen van de database. De backend maakt het mogelijk om data te importeren in het data warehouse en data beschikbaar te stellen voor externe applicaties. Er is gekozen voor Doctrine en Symfony voor de REST API omdat binnen het backend team hier de meeste kennis is. In de toekomst zou wel gekeken kunnen worden of de performance voor het benaderen van het data warehouse verbeterd kan worden door bijvoorbeeld de database rechtstreeks aan te roepen in plaats van met Doctrine. Vooralsnog is de performance voor het prototype snel genoeg voor Extendas.
Lokaal ziet de architectuur er als volgt uit:



Hoe dit er in productie uit gaat zien is niet onderzocht en valt buiten de scope van de opdracht.

Importeren transacties

Het importeren van de transacties gebeurt door middel van een Symfony commando. Dit commando zorgt er voor dat de transacties en transactie producten vanuit twee csv bestanden worden samengevoegd tot een csv bestand. De kolommen die niet van belang zijn en / of persoonsgegevens bevatten, worden gepseudonimiseerd opgeslagen in het nieuwe csv bestand. Dit bestand wordt vervolgens geïmporteerd in het data warehouse.

Voor het uitlezen van de csv bestanden in PHP is gebruik gemaakt van Spout [11]. Spout is een PHP library die op een snelle en schaalbare manier door grote csv bestanden kan lezen, zonder al te veel geheugen te gebruiken. Ideaal dus voor het uitlezen van deze 2 grote csv bestanden.

Met de Spout PHP library wordt regel voor regel door de transaction csv gelezen. Bij elke transactie worden de bijbehorende producten uit de transaction_product csv gelezen, gecombineerd en waar nodig gehasht toegevoegd aan een nieuwe csv. Deze csv kan vervolgens met het eerder benoemde COPY statement snel worden toegevoegd aan de database. Het volledige ETL proces voor het importeren van de transactie data vindt plaats wanneer het commando wordt uitgevoerd.

REST API

In Symfony zijn de verschillende API calls gegroepeerd in een aantal controllers. Alle API URLs beginnen met /api zodat deze binnen Symfony kunnen worden afgevangen door de authenticator. Elke API call krijgt een rol mee zodat de authenticator weet of de gebruiker bij deze data mag komen. Een groot deel van de afhandeling van de REST API call bevindt zich in de annotation. Hierin staat de URL met de juiste parameters gedefinieerd. Daarnaast wordt hier bepaald welke HTTP methodes beschikbaar zijn en welke rollen bevoegd zijn om bij de call te komen.

```php
/**
 * @Route("/total_product_per_month/{year}", name="total_product_per_month", methods={"GET"}, requirements={"year"="\d+"})
 * @Security("is_granted('ROLE_WEBAPP')")
 * @param $year
 * @return JsonResponse
 */
public function total_per_month($year): JsonResponse
{
    $result = $this->transactionProductRepo->getCountPerMonth($year);
    $counts = [];
    foreach($result['result'] as $r)
    {
        $counts[] = $r['count'];
    }
    $result['result'] = $counts;
    return new JsonResponse($result);
}
```

Met behulp van de query builder van Doctrine worden de query's opgebouwd. Hierbij is wel rekening gehouden met welke functies in de query invloed hebben op de snelheid van het ophalen. Zo worden de kleiner en groter dan operator gebruikt om tussen 2 verschillende datums te zoeken in plaats van bijvoorbeeld de between functie. Between maakt geen gebruik van de index en is daarom veel langzamer.

```php
public function getCountPerMonth($year, $distinct = false)
{
    if($distinct)
        $stmt = 'date_trunc(\'month\', q.time) AS date, COUNT(DISTINCT q.transaction_id) AS count';
    else
        $stmt = 'date_trunc(\'month\', q.time) AS date, COUNT(q.transaction_id) AS count';

    $query = $this->em->createQueryBuilder('q')
        ->select($stmt)
        ->from( from: TransactionProduct::class, alias: 'q')
        ->andWhere('q.time >= \'' . $year . '-01-01 00:00:00\' AND q.time < \'' . ($year+1) . '-01-01 00:00:00\'')
        ->groupBy( groupBy: 'date')
        ->getQuery();
    $result = $query->getArrayResult();
    $duration = $this->stack->queries[1]["executionMS"];
    $sql = $query->getDQL();

    return [
        'result' => $result,
        'query' => $sql,
        'duration' => $duration
    ];
}
```

Beveiliging

Zoals in het ontwerp benoemd, wordt voor de beveiliging van de REST API gebruik gemaakt van een JWT token. Binnen het Symfony project is gebruik gemaakt van de User entity en de LexikJWTAuthenticationBundle. Deze bundel maakt het mogelijk om de API calls te beveiligen met een JWT token. Deze token wordt gegenereerd door de juiste gebruikersnaam en wachtwoord naar de login call te sturen. In de firewall van het Symfony project wordt bepaald wat de login url is en dat de API gebruik maakt van de JWT authenticatie.

```
firewalls:
    login:
        pattern: ^/api/login
        stateless: true
        anonymous: true
        json_login:
            check_path: /api/login_check
            success_handler: lexik_jwt_authentication.handler.authentication_success
            failure_handler: lexik_jwt_authentication.handler.authentication_failure
    api:
        pattern: ^/api
        stateless: true
        provider: app_user_provider
        guard:
            authenticators:
                - lexik_jwt_authentication.jwt_token_authenticator
```

Omdat de JWT token met behulp van de User entity is gegenereerd, bevat deze ook de rollen van de ingelogde gebruiker. Aan de hand van deze rollen kan worden bepaald of de gebruiker de juiste bevoegdheden heeft om data op te halen. Zo kan een gebruiker met de rol voor DPS alleen bij de API endpoints van DPS en een gebruiker met de rol voor de webapplicatie, bij de endpoints die hierbij horen.

Testen

Voor de controllers voor de REST API zijn webtestcases geschreven die elke API call testen op de verschillende foutcodes. De cliënt roept eerst de login call aan om een JWT token te genereren. Vervolgens wordt deze gebruikt om de API calls mee aan te roepen.

Elke controller heeft 4 test cases (Voor elke API call een test case). In totaal zijn er 16 test cases met 52 assertions. Doordat de tests gebruik maken van de lokale omgeving met de complete database, duren de tests in totaal 3 minuten.

Webapplicatie

Binnen Extendas wordt Vue.js gebruikt als het framework voor de meeste frontend projecten. Naar aanleiding van een eerder onderzoek naar de bekendere JavaScript frameworks is Vue.js er als best uitgekomen voor Extendas. [4] Voor deze opdracht is wederom voor Vue.js gekozen, omdat hierin de meeste kennis is binnen het team en het biedt een goed gestructureerde basis voor de webapplicatie. De webapplicatie is een SPA (Single Page Application), wat wil zeggen dat de gehele applicatie op een webpagina draait. De pagina laadt in een keer de hele applicatie. Wanneer er iets veranderd in een component op de pagina, wordt alleen het component herladen. Dit zorgt onder andere voor een betere gebruikerservaring. Ook zorgt het voor minder verbruik op de server doordat componenten die reeds zijn geladen niet opnieuw van de server worden gehaald als ze niet veranderen.

NPM

Npm is een pakketbeheerder voor JavaScript. Hiermee zijn zowel de Vue CLI Server als onderstaande JavaScript libraries geïnstalleerd.

Vuetify

Vuetify is een UI framework bovenop Vue.js. Vuetify maakt het mogelijk om eenvoudig material design toe te passen op de webapplicatie en bevat veel material componenten. Extendas maakt gebruik van een template dat ook gebaseerd is op Vuetify. Het is dus eenvoudig om later over te stappen op het template. Alternatieven voor de UI zijn Vue Material of Buefy. Beide ook goede UI frameworks, maar omdat het template dat Extendas gebruikt voor andere webapplicaties, is gekozen voor Vuetify.

Vuex

Vuex is een zogeheten state management pattern. Vuex dient als een centrale plek in de applicatie om data in op te slaan voor alle componenten. In de webapplicatie wordt alle data uit de REST API opgeslagen in deze store, zodat de hele applicatie hierbij kan. Dit zorgt er ook voor dat wanneer je van pagina's heen en terug navigeert, niet opnieuw alle data hoeft worden geladen. Daarnaast maakt de centrale store het mogelijk om vanuit de query's pagina een overzicht op te maken met de query's uit de rest van de webapplicatie. De componenten kunnen de data uit de store in de hele applicatie inzien maar niet rechtstreeks aanpassen.

Vue Router

Vue Router maakt het mogelijk om een SPA te bouwen. Met behulp van de Vue Router kun je door de app navigeren zonder de verschillende pagina's te herladen in de browser. Een voordeel daarvan is ook dat een gedeelte van de webapplicatie, zoals het menu, niet opnieuw hoeft te laden. Ook kan Vue Router gebruikt worden voor paginering.

Axios

Axios is een JS HTTP client om de REST API aan te roepen.

Chart.js

Voor het weergeven van grafieken is gebruikgemaakt van Chart.js. Dit is een veelgebruikte en

flexibele JavaScript library. De grafieken zijn responsive en worden geladen op een HTML5 canvas. Om Chart.js te gebruiken binnen het Vue.js project, is gebruik gemaakt van vue-chartjs. Dit is een wrapper die het mogelijk maakt om de charts als Vue componenten te gebruiken. Naast Chart.js zijn er veel andere alternatieven (ApexChart, JSCharting) die net zo goed of misschien zelfs beter werken. Maar omdat er veel informatie beschikbaar is over Chart.js en goed te integreren is in Vue.js, is hiervoor gekozen.

Beveiliging

Omdat alle API calls zijn beveiligd met een JWT token, moet de gebruiker eerst inloggen voor deze bij de applicatie kan komen. De gebruiker logt in met behulp van zijn gebruikersnaam en wachtwoord. Wanneer de gegevens correct zijn wordt een JWT token teruggestuurd en deze wordt vervolgens opgeslagen in de store. Alle componenten kunnen vanuit de store checken of de gebruiker is ingelogd door te kijken of er een token beschikbaar is in de local storage. Bij het aanroepen van de andere API calls moet vervolgens de JWT token worden meegegeven in de header.

```
export default function authHeader() {
    let jwt_token = JSON.parse(localStorage.getItem( key: 'jwt_token'));
    if(jwt_token) {
        return { Authorization: 'Bearer ' + jwt_token };
    } else {
        return {};
    }
}
```

Resultaten

In dit hoofdstuk worden de behaalde resultaten en de kwaliteit van het prototype beschreven en gekeken naar eventuele verbeteringen.

Het doel van het data warehouse voor Extendas is dat het zowel als toegankelijke plek voor alle transacties dient, als dat het intern ingezien kan worden vanuit de webapplicatie. Voor DPS betekent dit dat alle transactie data toegankelijk is vanuit het data warehouse en gebruikers van DPS niet meer zelf verkoopgegevens invullen. Dit laatste proces is geautomatiseerd, doordat nieuwe transacties nu rechtstreeks uit beide POS systemen (Fuel POS en SPINpos) in het data warehouse worden geladen. Van hieruit kan DPS de verkoopgegevens door middel van de API ophalen in plaats van dat de gebruikers deze gegevens zelf invullen. Dit maakt het minder arbeidsintensief en fraudegevoelig.

Importeren

Wanneer Extendas de twee csv bestanden klaar heeft staan voor het data warehouse, kan het volgende commando worden aangeroepen:

- App:import-transactions-csv

De twee csv bestanden worden uitgelezen en waar nodig gehasht samengevoegd in een nieuw csv bestand. Vervolgens worden de indices uit de transactie tabel verwijderd, de nieuwe csv geïmporteerd en na het importeren de indices weer toegevoegd aan de database. Het gehele ETL proces wordt doorlopen bij het uitvoeren van dit commando.

REST API

De REST API is gestructureerd opgebouwd met Symfony om ervoor te zorgen dat eventueel nieuwe API calls eenvoudig toe te voegen zijn. Hiervoor is geen bundel gebruikt (zoals FOSRestBundle), omdat de extra functionaliteiten die dat met zich meebrengt niet nodig waren voor het prototype.

Symfony zorgt ervoor dat de authenticatie voor de gehele API goed gaat en gebruik maakt van de JWT authenticatie. Afhankelijk van het doel van de API call zijn deze ondergebracht in verschillende controllers om zo het overzicht te bewaren. Alle API calls geven het resultaat in JSON formaat terug zodat deze goed te gebruiken zijn voor meerdere toepassingen.

De JWT token bevat informatie over de gebruiker die de REST API probeert te benaderen. In de JWT token staat ook de rol van de gebruiker. Met behulp van deze rol wordt bepaald of de gebruiker toegang heeft tot het API endpoint.

Daarnaast zijn er tests geschreven die makkelijk uitgevoerd kunnen worden met PHPUnit. De tests zijn ondergebracht aan de hand van de bijbehorende controller en testen de API op de verschillende statuscodes en resultaten die in bijlage A staan beschreven. In elke test wordt door de cliënt eerst de login API call aangeroepen om een JWT token te generen. Met behulp van deze token worden vervolgens de URLs zowel met als zonder token aangeroepen om zo te kunnen controleren of de authenticatie werkt en of er een goed resultaat terug komt.

```php
protected function setUp(): void
{
    parent::setUp();
    $this->client = static::createClient();
    $this->client->request(
        method: 'POST',
        uri: '/api/login_check',
        [], [],
        ['CONTENT_TYPE' => 'application/json'],
        content: '{"username":"","password":""}'
    );
    $this->assertEquals( expected: 200, $this->client->getResponse()->getStatusCode());
    $response = json_decode($this->client->getResponse()->getContent(), assoc: true);
    $this->jwt_token = $response['token'];
}

public function testTransactionTotalYesterday()
{
    // Without authentication
    $this->client->request(
        method: 'GET',
        uri: '/api/organization/total_organization_yesterday/' . $this->organization_id
    );
    $this->assertEquals( expected: 401, $this->client->getResponse()->getStatusCode());

    $this->client->request(
        method: 'GET',
        uri: '/api/organization/total_organization_yesterday/' . $this->organization_id,
        [], [],
        ['HTTP_Authorization' => 'Bearer ' . $this->jwt_token]
    );
    $this->assertEquals( expected: 200, $this->client->getResponse()->getStatusCode());
}
```

Webapplicatie

Aan de hand van de eisen uit het functioneel ontwerp worden hier de verschillende pagina's weergegeven en beschreven die zijn gerealiseerd. In bijlage B zijn de schermafbeeldingen van de verschillende pagina's terug te vinden die horen bij onderstaande functionele requirements.

Werknemers kunnen inloggen op de webapplicatie (W1)
Bij het openen van de webapplicatie komt de gebruiker bij het login scherm, als deze niet al is ingelogd. Alle pagina's die niet bevoegd zijn voor de gebruiker, verwijzen door naar het login scherm. (scherm 1)

De ingelogde gebruiker kan per maand bekijken hoeveel transacties er zijn verwerkt (W2)
De ingelogde gebruiker komt op het dashboard scherm waar deze een jaartal kan kiezen om de transacties te bekijken per maand. (scherm 2)

De ingelogde gebruiker kan het totaal aantal transacties bekijken (W3)
Op het dashboard scherm worden het aantal nieuwe transacties van de dag ervoor, het aantal transacties van de afgelopen maand en het totaal aantal transacties getoond. (scherm 2)

De ingelogde gebruiker kan het totaal aantal transacties per POS bekijken (W4)
Op het POS scherm kan de ingelogde gebruiker een pos_id invullen om net als bij het dashboard scherm het aantal transacties van gisteren, afgelopen maand en het totaal te zien. (scherm 3)

De ingelogde gebruiker kan het totaal aantal transacties per organisatie bekijken (W5)
Op het organisatie scherm kan de ingelogde gebruiker een organisatie_id invullen. Net als op de andere pagina's worden ook hiervan het aantal transacties van gisteren, de afgelopen maand en het totaal weergegeven. (scherm 4)

De ingelogde gebruiker kan bekijken hoe lang uitgevoerde query's duren (W6)
Op het query scherm wordt een tabel weergegeven met de query's die zijn uitgevoerd tijdens het gebruik van de webapplicatie. Alle resultaten van de query's worden opgeslagen in de Vuex store, dus kunnen vanaf het query scherm eenvoudig worden opgehaald, zonder eerst alle query's opnieuw uit te voeren. (scherm 5)

Aanbevelingen

In dit hoofdstuk staan de aanbevelingen beschreven voor Extendas voor eventuele verdere ontwikkeling van het data warehouse.

Data warehouse

Vooral optimalisatie van de database voor het ophalen en analyseren van data is een belangrijk onderdeel om verder naar te kijken. Binnen PostgreSQL valt nog veel te optimaliseren op het gebied van bijvoorbeeld indexering of het aanpassen van de standaard configuratie afhankelijk van het systeem en de grootte van de database. Aan de hand van de huidige eisen en voor het prototype is PostgreSQL een goede keuze.

PostgreSQL

De wiki van PostgreSQL bevat veel informatie over het optimaliseren van de performance het analyseren van langzame query's. (https://wiki.postgresql.org/wiki/Performance_Optimization en https://wiki.postgresql.org/wiki/Slow_Query_Questions). Daarnaast zijn er voor PostgreSQL een aantal dashboards beschikbaar om de database mee te monitoren. Een voorbeeld hiervan is PgHero. In dit dashboard kunnen live de query's worden bekeken die uitgevoerd worden op de database. Ook kan vanuit hier worden gekeken of er dingen fout gaan in de database en of er winst te behalen valt bij het aanpassen van de configuratie van de database. Deze tool is gratis en eenvoudig te integreren. Andere voorbeelden zijn PgDash (geen gratis versie) en ClusterControl. De laatste is ook voor andere databases beschikbaar zoals MySQL, MariaDB en MongoDB.

Elasticsearch

Een goed alternatief voor voor het analyseren van data (voornamelijk zoeken in teksten) is Elasticsearch. Het wordt afgeraden om Elasticsearch als main database te gebruiken, voornamelijk omdat nodes uit kunnen vallen. Dat zorgt ervoor dat data niet meer beschikbaar is en vaak is het dan niet meer recht te trekken. Het is ook niet gebouwd om als database te dienen, maar als search engine. Wel zijn de makers van Elasticsearch constant bezig met het verbeteren van de veerkracht.

Dat gezegd te hebben, is het wel goed te gebruiken om analyses mee te maken in een grote hoeveelheid tekstuele data.

Elasticsearch kan gebruikt worden in combinatie met PostgreSQL. De PostgreSQL database dient dan als relationele database, maar kan door middel van verschillende extensies data syncen met Elasticsearch.

Een voorbeeld van zo'n extensie is ZomboDB. [6] Deze extensie maakt het mogelijk om binnen PostgreSQL een index te creëren die gebruik maakt van de Elasticsearch cluster. De op Elasticsearch gebaseerde index kan vervolgens gebruikt worden door PostgreSQL om query's uit te voeren, zonder dat de ontwikkelaar hier verder zelf iets voor hoeft te doen. Een groot voordeel van deze extensie is dat onder andere CREATE, UPDATE en DELETE query's ook worden gesynchroniseerd met Elasticsearch.

Vervolgens zou gebruik kunnen worden gemaakt van Kibana. [7] Kibana is een krachtige dashboard tool voor het visualiseren en navigeren in de Elasticsearch data.

Webapplicatie

De webapplicatie is in Vue.js geschreven met behulp van Vuetify en kan dus makkelijk overgezet worden naar het template dat Extendas gebruikt voor de andere webapplicaties. Zoals hierboven al is benoemd zou ook Kibana gebruikt kunnen worden om de analyses te maken. Ook binnen Vue.js zouden bepaalde libraries makkelijk vervangen kunnen worden, zoals Chart.js. De alternatieven zijn namelijk minstens zo goed en gaat dus voornamelijk om persoonlijke voorkeur.

Importeren

Bij het importeren valt er nog te optimaliseren in de snelheid van het uitlezen van de twee csv bestanden. Dit gebeurt nu vanuit het Symfony project met PHP. Regel voor regel wordt door beide bestanden heen gelopen om ze samen te voegen. Bij elke volgende transactie moet weer vooraan door de producten gelopen worden. Een oplossing voor het versnellen van dit proces zou kunnen zijn om de regels die reeds zijn toegevoegd uit de producten csv te verwijderen, zodat deze niet telkens opnieuw hoeven worden doorlopen. Vooral bij deze hele grote csv bestanden heeft dat veel invloed op de snelheid.

Ook zou kunnen worden gekeken naar andere oplossingen om het ETL proces te doorlopen. Hiervoor zijn meerdere tools beschikbaar die gemaakt zijn specifiek voor dit proces. Een van deze tools is Blendo. Deze biedt een goede ondersteuning voor PostgreSQL. Een nadeel van veel van deze tools is dat ze erg prijzig zijn. Een voorbeeld van een open source ETL tool is Singer. De kracht van Singer is dat je veel verschillende bronnen als input kan gebruiken en ook veel verschillende eindbestemmingen als output. Singer is modulair en toe te passen op heel veel verschillende situaties.

Query's

In de huidige implementatie worden de query's uitgevoerd op het moment dat de API calls worden aangeroepen. Voor nu voldoet dit aan de wensen van Extendas. Wanneer er meer query's bij komen en eventueel te veel indices toegevoegd moeten worden, is het een mogelijkheid om in plaats daarvan resultaten in een aparte tabel op te slaan na het importeren. Door middel van jobs kunnen resultaten worden berekend en opgeslagen in een tabel. Hiervoor kunnen nieuwe API endpoints beschikbaar gesteld worden om deze resultaten terug te geven. Hierdoor heb je meer controle over wanneer de query's draaien. Nadeel is dat de resultaten niet geheel realtime zijn, maar dat is niet altijd een probleem in een data warehouse.

Conclusie

Tijdens het onderzoek is gezocht naar het antwoord op de vraag: “Hoe kan een grote hoeveelheid transactie data verzameld, opgeslagen en beschikbaar gesteld worden waarbij voldaan wordt aan de voorwaarden en wensen van Extendas?”

De opdracht beschreef dat er een data warehouse moet komen. Een data warehouse is een ruim begrip. Daarom moest er gekeken worden naar wat Extendas echt nodig heeft om de verschillende applicaties aan elkaar te koppelen en de data op te slaan op een centrale plek.

Om een antwoord te geven op de hoofdvraag en een beter beeld te krijgen van de wensen en eisen van Extendas, zijn een aantal deelvragen beantwoord.

Allereerst is er onderzoek gedaan naar de gegevens die opgeslagen moeten worden in het data warehouse. Het gaat hierbij om een grote hoeveelheid transacties. Dagelijks komen er zo’n 100.000 transacties bij en in totaal zijn dit al rond de 28 miljoen transacties.

Vervolgens is gekeken naar wat een data warehouse is en wat het in het geval van Extendas betekent voor deze opdracht. Omdat er heel veel verschillende databases zijn om een data warehouse mee in te richten zijn er een aantal non-functionele requirements opgesteld vanuit Extendas om een juiste keuze te kunnen maken. Het aantal beschikbare databases is zo groot, dat er eerst een lijstje van meest populaire databases is opgesteld om verder te onderzoeken. Nadat deze lijst van databases is onderzocht is er bepaald welke het best aansluit bij de eisen van Extendas. Dit is gedaan door het wegen van de verschillende eisen en een score toe te kennen aan de onderzochte databases. Databases zoals MySQL, MariaDB, PostgreSQL en MongoDB kwamen hierbij uit op een hoge score. PostgreSQL sloot het beste aan op de eisen van Extendas en is daarom gekozen als de database voor het data warehouse.

Prototype

Omdat het backend team gebruik maakt van Symfony / Doctrine is er gekozen om hier gebruik van te maken voor het realiseren van het prototype van het data warehouse. Het importeren van data en de REST API gaan allemaal vanuit één Symfony project. Zo zijn alle belangrijke onderdelen van het data warehouse bij elkaar ondergebracht in het Symfony project. Dat maakt het ook makkelijker om te onderhouden en overzicht te bewaren.

Een belangrijk onderdeel van het data warehouse is het importeren van nieuwe transacties. Voor de implementatie hiervan is samen met Extendas besloten dat er dagelijks twee csv bestanden klaar staan met de nieuwe transacties. Deze worden vervolgens vanuit het Symfony project met PHP uitgelezen, samengevoegd en gehasht opgeslagen in de PostgreSQL database.

Als laatst is er nog de webapplicatie die dient om intern een beter inzicht te krijgen in het data warehouse. Hiervoor is gekozen voor Vue.js, omdat Extendas dit JavaScript framework voor meerdere projecten gebruikt en het zich daarmee heeft bewezen als goed frontend framework. Dit project staat los van het data warehouse, zodat het eenvoudig is aan te passen en niet afhankelijk is van het data

warehouse. De gewenste functionele eisen voor de webapplicatie zijn gerealiseerd in het prototype en werken naar behoren.

In de huidige implementatie worden de query's uitgevoerd op het moment dat de API wordt aangeroepen. De snelheid hiervan voldoet aan de eisen voor de webapplicatie voor Extendas. Ook zorgt dit ervoor dat gelijk gekeken kan worden hoelang de uitgevoerde query's duren. Dit is tenslotte een belangrijk onderdeel van de webapplicatie.

Reflectie

In dit hoofdstuk wordt gereflecteerd op de afgelopen afstudeerperiode. Hierbij wordt gekeken naar zowel de afstudeerperiode tot de eerste concept deadline en naar de verbeteringen in de verlenging van de afstudeerperiode.

Persoonlijke reflectie

In de afstudeerperiode heb ik heel veel geleerd, zowel over de onderwerpen die in de opdracht zijn behandeld, als in persoonlijke ontwikkeling. Het was zeker niet altijd makkelijk om vanuit huis motivatie te vinden voor zo'n groot project. Wat ik vooral erg moeilijk vond was het overzicht behouden en het totaal in kleine stappen te brengen.

Bij het onderzoek naar de databases kwam ik er al snel achter dat er heel veel goede oplossingen zijn. Daardoor was het af en toe lastig om een goed begin te krijgen en niet te veel informatie te verwerken, maar datgene vinden wat echt nodig is voor het onderzoek.

Ook heb ik geleerd dat het belangrijk is om de lijntjes kort te houden en veel contact te blijven hebben met de verschillende partijen (school- en bedrijfsbegeleiders) om ervoor te zorgen dat niet te veel wordt afgeweken van datgene dat gevraagd wordt.

In de eerste periode heeft de hoeveelheid werk ervoor gezorgd dat ik redelijk snel vast liep en mijn motivatie verloor. Dit heb ik in de verlenging verbeterd door veel contact te blijven houden en alles op te delen in kleinere stappen. Hierbij heb ik ook veel meer gekeken om eerst alles uit te werken en goed uit te denken, voordat ik vanalles ging proberen te bouwen. Toch vind ik het nog steeds erg lastig om keuzes te maken over waar de prioriteiten liggen als er veel onderwerpen behandeld moeten worden.

Uiteindelijk ben ik tevreden met de stappen die ik heb gezet en zou in het vervolg waarschijnlijk een stuk beter om kunnen gaan met het uitwerken van zo'n groot project. Vooral het plannen is enorm verbeterd ten opzichte van het begin van de afstudeerperiode. Het grootste minpunt vind ik het feit dat ik een aantal onderwerpen heb moeten laten liggen vanwege tijdgebrek en daarom ook niet alles wat ik in het prototype had willen hebben, heb kunnen realiseren.

Wat ik vooral anders zou doen in de toekomst, is het project eerder afbakenen en goed uitzoeken wat gedaan moet worden. Zodat ik daar de focus op kan leggen en niet vast loop door de grote van het project.

Bronnen

1. W. den Ouden, Onderzoeksverslag “Hoe kan een grote hoeveelheid transactie data verzameld, opgeslagen en beschikbaar gesteld worden waarbij voldaan wordt aan de voorwaarden en wensen van Extendas?”, ongepubliceerd, Saxion Hogeschool, Enschede, 2020
2. W. H. Inmon, “Building the Data Warehouse”, Wiley, 1990
3. K. Gevel, “Scrumban: concept, werkwijze, effect (+Kanban Template)”, Agile Scrum Group, 24 december 2020 [Online], beschikbaar: https://agilescrumgroup.nl/scrumban/, [Geraadpleegd op 4 maart 2021]
4. W. den Ouden, Stageverslag “JS Frameworks”, ongepubliceerd, Saxion Hogeschool, Enschede, 2018
5. “Wat zijn persoonsgegevens?”, Autoriteit Persoonsgegevens, beschikbaar: https://autoriteitpersoonsgegevens.nl/nl/over-privacy/persoonsgegevens/wat-zijn-persoonsgegevens
6. “Making Postgres and Elasticsearch work together like it's 2021”, ZomboDB, beschikbaar: https://www.zombodb.com/
7. Kibana, beschikbaar: https://www.elastic.co/kibana
8. “The Difference Between a Data Warehouse and a Database”, Panoply, beschikbaar: https://panoply.io/data-warehouse-guide/the-difference-between-a-database-and-a-data-warehouse/
9. CockroachDB Frequently Asked Questions, beschikbaar: https://www.cockroachlabs.com/docs/stable/frequently-asked-questions.html
10. A. Nordeen, “Learn Data Warehousing in 24 Hours”, Guru99, 2020
11. Spout, beschikbaar: https://github.com/box/spout

Bijlagen

Bijlage A

POST		**Url**:	/api/login_check
Beschrijving:		Controleer login gegevens en krijg een JWT token om de API mee te benaderen.	
Body:		{ “username”: “user123”, “password”: “pass123” }	
Statuscode		**Response**	
200	OK	{ “token”: “eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2lzX2FkbWluIjp0cnVlLCJjb21wYW55IjoiVG9wdGFsIiwiYXdlc29tZSI6dHJ1ZX0. yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw” }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid credentials” }	

Import

GET		**Url**:	/api/transaction/last_transaction_id
Beschrijving:		Het ID van de laatst toegevoegde transactie, zodat Extendas weet vanaf welke transactie nieuwe transacties moeten worden geïmporteerd	
Statuscode		**Response**	
200	OK	{ “id”: 123456 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

Webapplicatie

GET		**Url**:	/api/transaction/total_yesterday
Beschrijving:		Het totaal aantal transacties van gisteren	
Statuscode		**Response**	
200	OK	{ “transactions”: 10000, “duration”: 300 }	

400	Bad request	{ “error”: “Something went wrong” }
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }

GET		**Url**:	/api/transaction/total_last_month
Beschrijving:		Het totaal aantal transacties van de afgelopen maand	
Statuscode		**Response**	
200	OK	{ “transactions”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/transaction/total_count
Beschrijving:		Het totaal aantal transacties	
Statuscode		**Response**	
200	OK	{ “transactions”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/transaction/total_per_month/{year}
Parameter:		Year (integer): Het jaartal	
Beschrijving:		Het totaal aantal transacties per maand in het aangegeven jaar	
Statuscode		**Response**	
200	OK	{ “transactions”: [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000], “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/product/total_yesterday
Beschrijving:		Het totaal aantal producten van gisteren	

Statuscode		**Response**
200	OK	{ “products”: 10000, “duration”: 300 }
400	Bad request	{ “error”: “Something went wrong” }
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }

GET		**Url**:	/api/product/total_last_month
Beschrijving:		Het totaal aantal producten van de afgelopen maand	
Statuscode		**Response**	
200	OK	{ “products”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/product/total_count
Beschrijving:		Het totaal aantal producten	
Statuscode		**Response**	
200	OK	{ “products”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/product/total_per_month/{year}
Parameter:		year (integer)	
Beschrijving:		Het totaal aantal producten per maand in het aangegeven jaar	
Statuscode		**Response**	
200	OK	{ “products”: [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000], “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/pos/total_yesterday/{pos_id}
Parameter:		pos_id (integer)	
Beschrijving:		Het totaal aantal transacties van gisteren voor de aangegeven POS	
Statuscode		**Response**	
200	OK	{ “transactions”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/pos/total_last_month/{pos_id}
Parameter:		pos_id (integer)	
Beschrijving:		Het totaal aantal transacties van de afgelopen maand voor de aangegeven POS	
Statuscode		**Response**	
200	OK	{ “transactions”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/pos/total_count/{pos_id}
Parameter:		pos_id (integer)	
Beschrijving:		Het totaal aantal transacties voor de aangegeven POS	
Statuscode		**Response**	
200	OK	{ “transactions”: 10000, “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

GET		**Url**:	/api/pos/total_per_month/{year}/{pos_id}
Parameter:		year (integer), pos_id (integer)	
Beschrijving:		Het totaal aantal transacties per maand voor de aangegeven POS en	

		jaar
Statuscode		**Response**
200	OK	{ “transactions”: [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000], “duration”: 300 }
400	Bad request	{ “error”: “Something went wrong” }
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }

GET	**Url**:	/api/organization/total_yesterday/{organization_id}
Parameter:		organization_id (integer)
Beschrijving:		Het totaal aantal transacties van gisteren voor de aangegeven organisatie
Statuscode		**Response**
200	OK	{ “transactions”: 10000, “duration”: 300 }
400	Bad request	{ “error”: “Something went wrong” }
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }

GET	**Url**:	/api/organization/total_last_month/{organizations_id}
Parameter:		organization_id(integer)
Beschrijving:		Het totaal aantal transacties van de afgelopen maand voor de aangegeven organisatie
Statuscode		**Response**
200	OK	{ “transactions”: 10000, “duration”: 300 }
400	Bad request	{ “error”: “Something went wrong” }
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }

GET	**Url**:	/api/organization/total_count/{organization_id}
Parameter:		organization_id (integer)
Beschrijving:		Het totaal aantal transacties voor de aangegeven organisatie
Statuscode		**Response**

200	OK	{ “transactions”: 10000, “duration”: 300 }
400	Bad request	{ “error”: “Something went wrong” }
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }

GET		**Url**:	/api/organization/total_per_month/{year}/organization_id}
Parameter:		year (integer), organization_id (integer)	
Beschrijving:		Het totaal aantal transacties per maand voor de aangegeven organisatie en jaar	
Statuscode		**Response**	
200	OK	{ “transactions”: [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000], “duration”: 300 }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

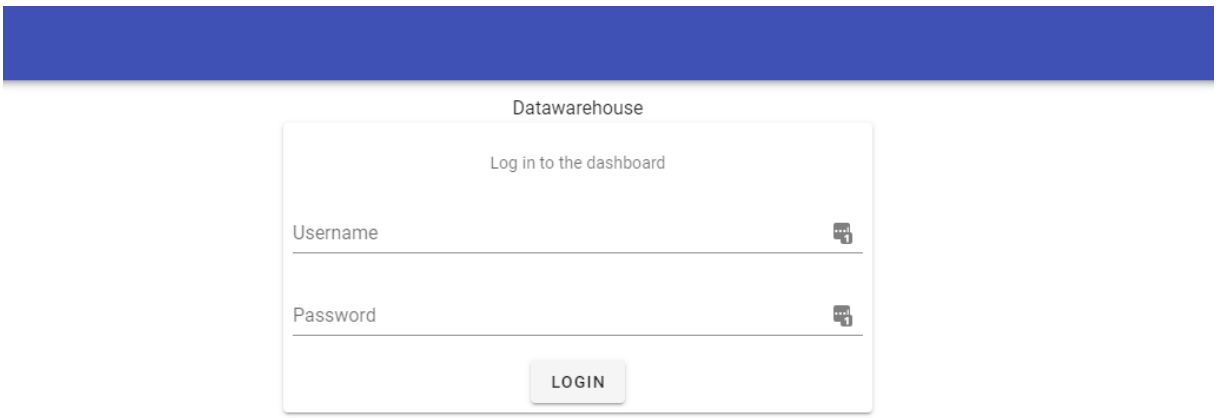
DPS

POST		**Url**:	/api/dps/sales
Body:		{ “barcode”: 12345, “date”: “2021-02-01”, “location_id”: 123 }	
Beschrijving:		Het totaal aantal verkopen op een locatie van een barcode voor een dag	
Statuscode		**Response**	
200	OK	{ “barcode” : 12345, “location_id”: 123, “total”: 10000, “date”: “2021-02-01” }	
400	Bad request	{ “error”: “Something went wrong” }	
401	Unauthorized	{ “unauthorized”: “Invalid authentication token” }	

Bijlage B

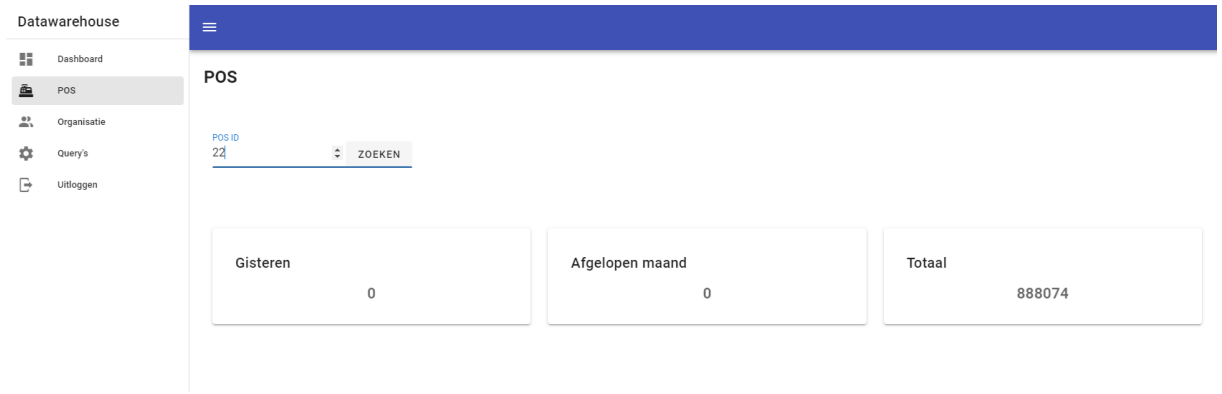
In deze bijlage de schermafbeeldingen waarnaar wordt verwezen in het hoofdstuk resultaten.

Scherm 1 login

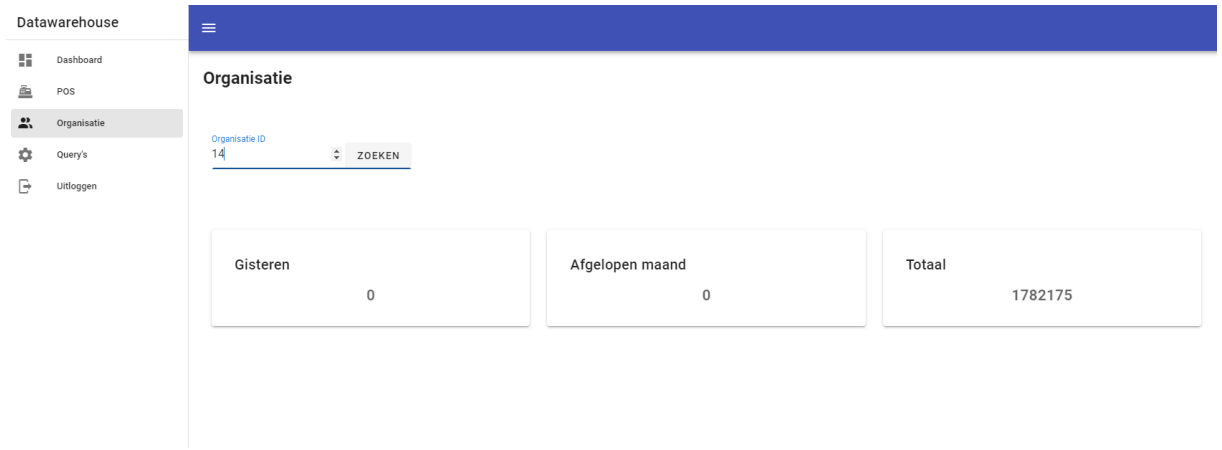


Scherm 2 dashboard

Scherm 3 POS transacties



Scherm 4 Organisatie transacties



Scherm 5 Query overzicht