Onderzoek & analyse certificatenbeheer

Stage Remmicom

Brent Verlinden, Nicholas Coorevits 3ITF
Verslag in verband met onderzoek & analyse

Inhoudstafel

Inhoud

1 Inleiding

De applicatie die wij zullen ontwerpen tijdens onze stage bij Remmicom zou het werk van de certificaatbeheerders moeten vergemakkelijken. Om goed van start te kunnen gaan hebben wij erg veel onderzoek verricht. Dankzij deze onderzoeken en analyses hebben wij nu een beter beeld van de concrete opdracht en de verschillende technologieën die wij mogelijk zullen moeten gebruiken tijdens onze stage. In dit document vindt u daarom meer informatie terug omtrent deze onderzoeken en analyses.

Het verslag begint met een korte introductie tot het stagebedrijf en de concrete opdracht. Hierin staat vermeld wat het huidige certificatenbeheerproces inhoud, en wat de mogelijke verbeteringen zijn die wij zouden kunnen brengen. Nadien komen we aan bij enkele algemene keuzes die wij gemaakt hebben. Denk onder andere aan welke technologieën we gaan gebruiken. Vervolgens vermelden we ook kort welke project management tools gebruikt worden binnen Remmicom.

Daarna komen we aan bij de wat meer technische onderzoeken. Eerst en vooral worden er enkele termen uitgelegd die van belang zijn wanneer we spreken over certificaten. Daarna komen we terecht bij de eisenanalyse van het systeem dat wij zullen ontwikkelen. Verder vinden we hier ook enkele initiële prototypes terug. Daarnaast kan u ook wat meer informatie terugvinden omtrent het UML-diagram dat we opgesteld hebben. Dit diagram stelt onze databank voor.

Vervolgens komen we aan bij de authenticatie en autorisatie. In dit onderdeel wordt er wat meer informatie gegeven omtrent de rechten van de verschillende personen die in aanraking zullen komen met onze applicatie. Verder wordt er ook wat meer informatie gegeven omtrent de concrete uitwerking hiervan.

Verder vindt u ook nog meer informatie terug over RxJS. Dit wordt gebruikt binnen Angular om data op te halen via een API. Daarnaast staan er ook nog enkele andere technologieën vermeld die we onderzocht hebben. Aangezien we deze voornamelijk onderzocht hebben aan de hand van filmpjes hebben we deze niet in detail beschreven. Tot slot kan u ook nog een conclusie terugvinden aan het einde van dit verslag.

2 Remmicom

Remmicom is een Belgisch bedrijf opgericht in 1988. Ze zijn gevestigd in het bedrijvenpark van Westmeerbeek, La Corbeille. Ze zijn de specialist in software en hardware oplossingen voor lokale overheden, zowel voor gemeente als OCMW. Ze leveren verschillende diensten zoals:

- ✓ IT-oplossingen gebaseerd op de nieuwste technologieën
- ✓ Ondersteuning van de hardware en infrastructuur bij klanten
- ✓ Optimale dienstverlening, met gepersonaliseerde aanpak, gekoppeld aan kwaliteitsproducten
- ✓ Oplossingen op maat

Remmicom is actief in meer dan 100 administraties over het hele Vlaamse landsgedeelte. Zoals u misschien al wel weet zijn certificaten onmiskenbaar voor overheden. Remmicom is verantwoordelijk voor het beheren van certificaten van talloze organisaties. Concreet zullen wij dus aan de slag gaan met het verbeteren van het huidige proces omtrent het beheren van certificaten.

3 Opdrachtbeschrijving

Voor onze stage bij Remmicom moeten wij een applicatie ontwerpen die het huidige systeem voor het werken met certificaten kan vervangen. Het huidige systeem vergt namelijk veel handmatig werk en heeft geen gebruiksvriendelijke gebruikersinterface. We hebben natuurlijk vooral onderzoek gedaan naar de werking van het huidige systeem om zo een beter beeld te kunnen krijgen van wat er van ons verwacht wordt. Hieronder staat kort het huidige systeem beschreven. Verder staat er ook vermeld wat er zoal verbeterd kan worden. Meer informatie omtrent de huidige werking en wat wij willen verbeteren kan u terugvinden in de andere onderdelen van dit document.

3.1 Huidige werking

Het certificatenbeheerproces omvat een interne- en externe kant. De interne kant omvat de applicatie die wij zullen ontwerpen om certificaten eenvoudiger te kunnen beheren. De interne tool is dus de tool die Remmicom zal gebruiken. De externe kant is de kant van de gemeenten. Zij moeten de certificaten die bij Remmicom gecreëerd zijn nadien weer kunnen ophalen. Dit gebeurt op basis van de externe tool.

Interne tool

De interne kant van onze opdracht gaat dus over de interne tool die we moeten ontwerpen voor Remmicom.

De certificaatbeheerders beschikken niet over een concrete centrale tool om alles te beheren. Momenteel beschikken ze over een databank die ze via SQL-statements aanvullen. Dit betekent dus dat er heel veel tijd steekt in het maken van wijzigingen. De informatie die in hun databank staat zetten ze om in een Excel-bestand. Dit is hun enige vorm van gebruikersinterface. Wijzigingen hierin maken heeft natuurlijk geen zin aangezien de databank zelf niet mee veranderd. Wijzigingen gebeuren dus uitsluitend via SQL-statements. Het Excel-bestand kan dus alleen maar gebruikt worden om deze informatie te bekijken.

Verder beschikken de certificaatbeheerders over een paar basic tools. Eén van die tools zorgt voor de generatie van een .p12-bestand in combinatie met een random wachtwoord. Deze generatie gebeurt op basis van .crt- en .key-bestanden. Deze bestanden bevatten de publieke- en privésleutel. Meer informatie hierover vindt u terug in onderdeel 5 van dit verslag, “Certificaten & IDP”. Nadien zal de tool het .p12-bestand in combinatie met het bijhorende random wachtwoord in een kdbx-bestand plaatsen voor de gemeente waarvoor dit certificaat dient. Dit kdbx-bestand wordt dan automatisch op de FTP-server geplaatst via de tool.

Zoals u kan zien vergt dit dus veel handmatig werk en is het ook niet de beste methode. Er is geen duidelijke gebruikersinterface en via hun huidige gebruikersinterface (Excel) kunnen ze dus ook geen wijzigingen maken aan hun databank. Verder moeten ze de tools ook steeds lokaal op hun pc draaien, wij hopen deze dus te kunnen verwerken in onze eigen webapplicatie.

Externe tool

Verder is er ook nog de externe kant die we zullen proberen te regelen. Dit is de kant van de gemeentes.

De certificaten die via de tool op de FTP-server zijn geplaatst, in een kdbx-bestand met alle .p12-bestanden en hun bijhorende random wachtwoord, worden nadien opgehaald vanuit de server van de gemeente. Momenteel is er een tool die ze kunnen opstarten waarna er gezocht wordt op de FTP-server naar een .kdbx-bestand met de naam van hun gemeente. Dit bestand wordt dan opgehaald en de informatie hiervan wordt in een eenvoudige databank geplaatst.

Dit is echter niet ideaal, aangezien ze het script steeds zelf moeten opstarten wanneer ze de certificaten op willen halen. Het verbeteren van dit proces is echter niet de prioriteit van onze stage.

3.2 Mogelijke verbeteringen

Interne tool

We gaan een Angular webapplicatie met een .NET API ontwerpen. Via deze applicatie kunnen de mensen met de juiste bevoegdheid de certificaten raadplegen en beheren. Op deze manier kunnen we een goede gebruikersinterface aanbieden die gemakkelijk is in gebruik. De API zal dan geconnecteerd worden met onze vernieuwde databank om de CRUDs te laten werken. Dankzij deze aanpassing zullen de certificaatbeheerders veel minder gebruik moeten maken van rechtstreekse SQL-statements. Dit betekent dus dat er veel minder handmatig werk uitgevoerd zal moeten worden. Verder willen we in onze applicatie ook de mogelijkheid geven een .crt en .key bestand op te laden, waarna u via de web interface een .pfx-bestand met een bijhorend random wachtwoord kan genereren. Dit bestand komt dan terecht in onze databank. Afhankelijk van de mogelijkheden kunnen we deze informatie eventueel ook doorsturen naar de FTP-server via bijvoorbeeld een kdbx-bestand. De mogelijkheden hierbij zijn gevarieerd. Wij zullen de focus dan ook leggen op de interne applicatie/tool aangezien deze de meest kritieke wijzigingen vereist.

Externe tool

De externe kant is de kant van de overheid. Zij beschikken over een externe tool die gebruikt kan worden om de certificaten weer op te halen vanuit de FTP-server van Remmicom. Deze tool neemt dan het KDBX-bestand van de gemeente of OCMW en plaatst de bijhorende certificaten in een databank. Een uitbreiding zou kunnen zijn om de FTP-server te vervangen door een betere optie die rechtstreeks en centraal is, maar dit is echter niet de prioriteit van onze stageopdracht. Een mogelijke optie zou zijn om de huidige externe tool te vervangen door een API in combinatie met een databank die op de server van de gemeente kan draaien, die zal zorgen voor het ophalen van alle certificaten vanuit de FTP-server van Remmicom. We kunnen er dan voor zorgen dat het ophalen van de certificaten elk uur gebeurt aan de hand van een service. Nadien kan elke afdeling hun eigen certificaat ophalen vanuit de databank van de gemeente.

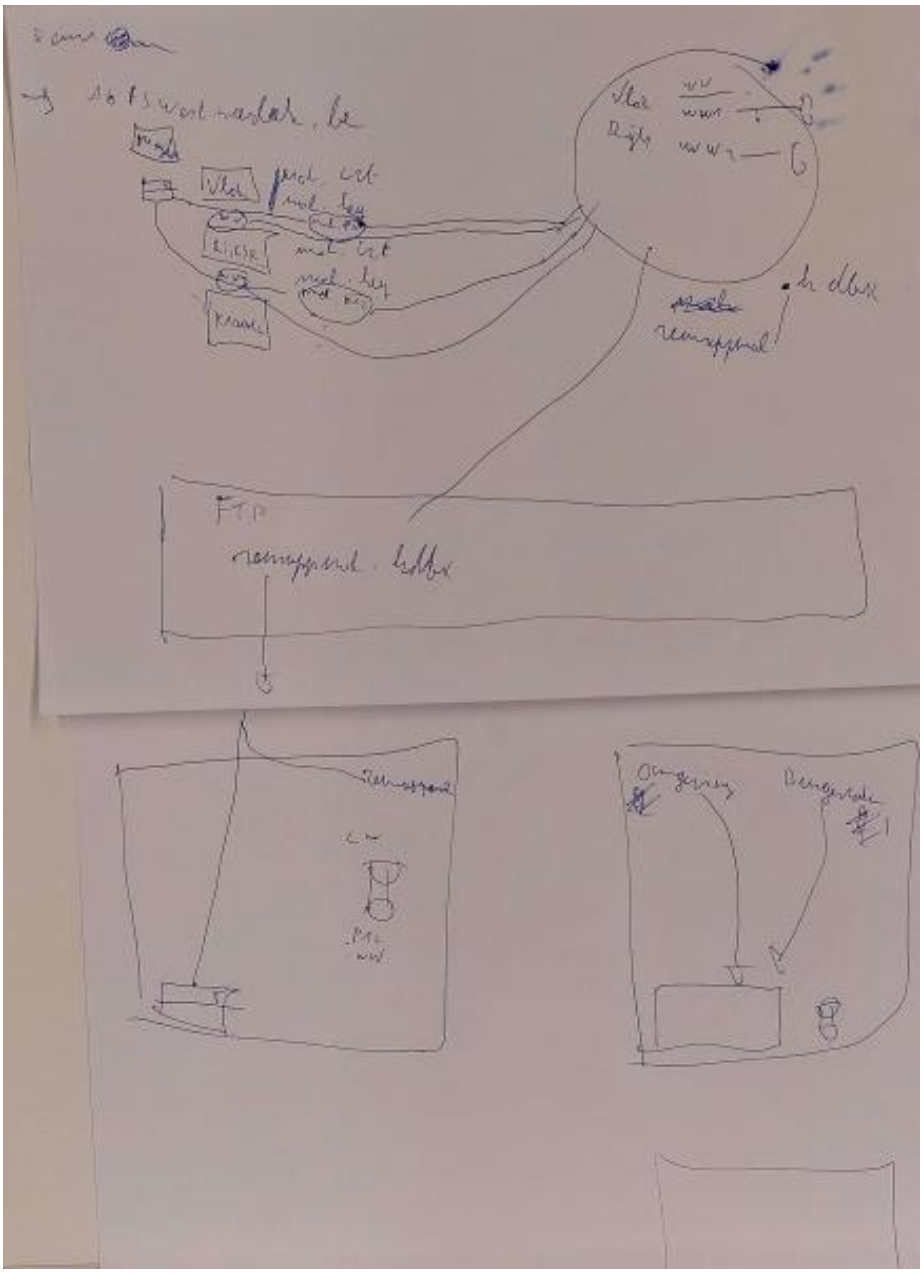
Terminologie (*)

- ✓ CRUD: Dit staat voor create, read, update, delete. Dit zijn de belangrijkste acties die u kan uitvoeren.

3.3 Extra Informatie

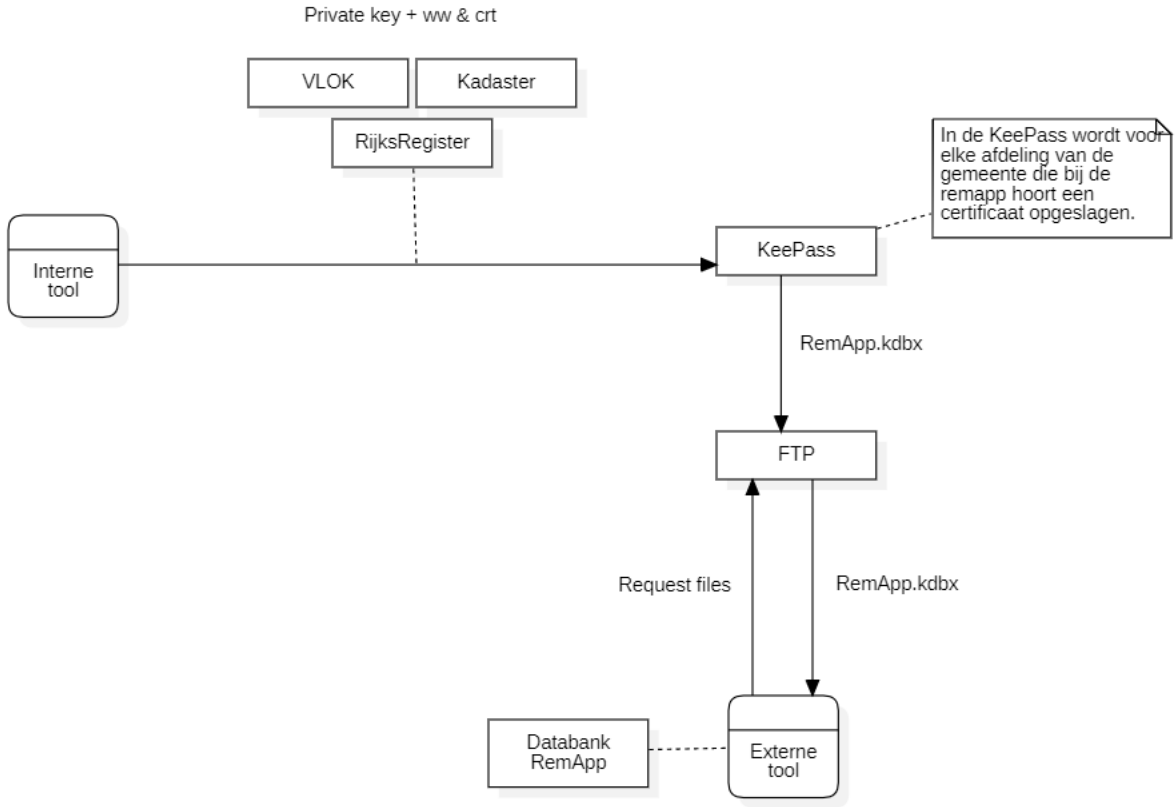
Hieronder staat een algemeen schema over de opdracht die ons geholpen heeft bij het verkrijgen van een beter beeld. Op deze afbeelding staat de huidige werkwijze vermeld waarop wij ons gebaseerd hebben voor het ontwerp van onze applicatie. Linksboven op de afbeelding ziet u dat er via een tool certificaten aangemaakt worden. Deze PFX-certificaten worden nadien in een KDBX-bestand geplaatst, ofwel een keepass vault. Dit is een soort van bestand dat beveiligd is waar u informatie en bestanden in kan opslaan. De tool zet deze PFX-bestanden van de correcte organisatie dan in een KDBX-bestand waarna deze op de FTP-server van Remmicom wordt geplaatst. Dit alles wordt verwezenlijkt aan de interne kant, binnen Remmicom zelf.

De organisatie (meestal een gemeente of OCMW) beschikt dan weer over een externe tool die de informatie van het KDBX-bestand kan ophalen vanuit de FTP-server van Remmicom. Deze bestanden worden dan opgehaald en in een databank geplaatst, klaar voor gebruik.



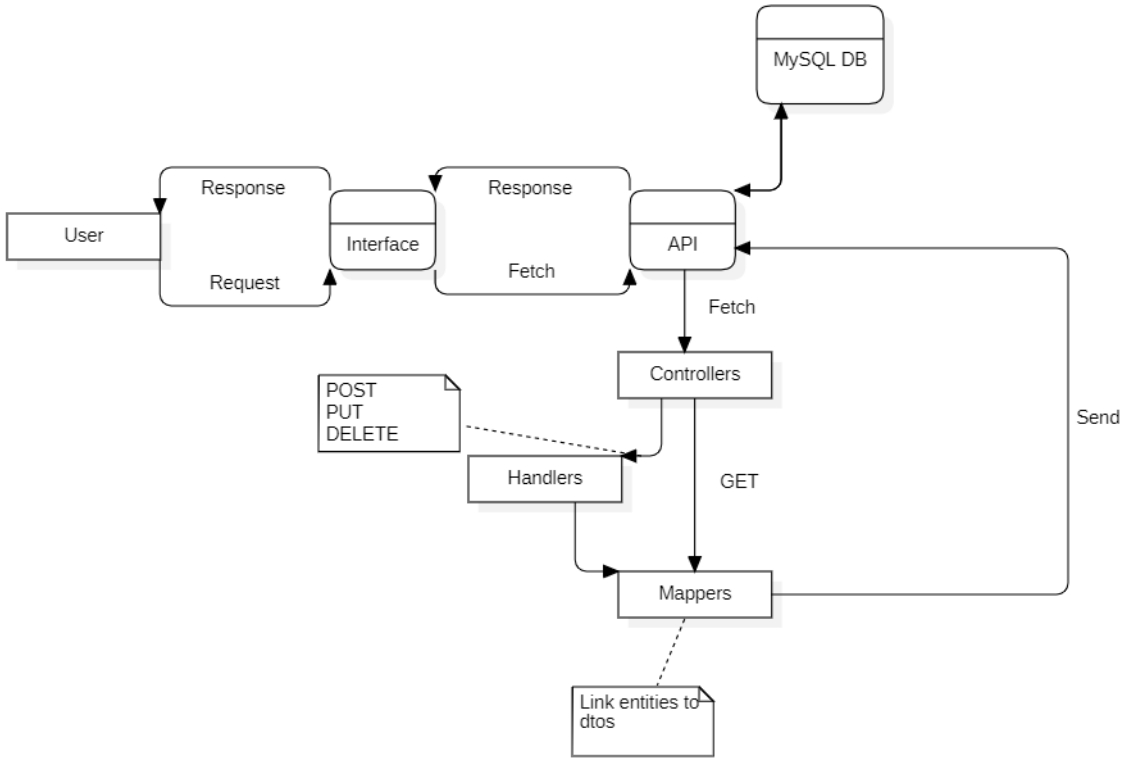
Schema

Om bovenstaande foto te verduidelijken hebben wij deze omgegoten in een schema:



Data flow

In de onderstaande afbeelding hebben wij uitgebeeld hoe onze interne applicatie een request behandelt van wanneer een gebruiker of beheerder een request uitvoert via onze webinterface. De interface gaat de data ophalen in de API die verbonden is met de databank waarna de controller ofwel een get request uitvoert of verder verwijst naar de handlers als het geen get request is. Deze leiden allebei naar de mappers waar de entiteiten hun data aan een "Data Transfer Object" (DTO) gelinkt worden. Dit is de data die getoond mag worden. Daarom wordt deze data verstuurd naar de API, die dan de response kan teruggeven aan de interface. Na dit proces krijgt de gebruiker te zien wat hij heeft opgevraagd.



4 Algemene keuzes

Voor dit project maken we gebruik van enkele technologieën die gewaardeerd worden binnen Remmicom, waaronder:

- ✓ Angular
- ✓ .NET 7
- ✓ SQL Server
- ✓ SCSS
- ✓ Remmicom packages
    - Voor styling, componenten, …

Verder moeten we via onze applicatie connecteren met de gebruikersbeheertool van Remmicom. Deze tool bevat alle gebruikers met hun rechten per applicatie. Onze applicatie staat hierop geregistreerd. Onze stagebegeleider kan dan mensen toevoegen aan onze applicatie zodat zij toegang kunnen krijgen. Er is sprake van lees- en schrijfrechten. De gebruikers met leesrechten zijn dus meestal de werknemers van Remmicom, zij kunnen alles bekijken maar niet wijzigen. De certificaatbeheerders hebben ook schrijfrechten. Zij kunnen dus aanpassingen maken.

5 DevOps

Binnen Remmicom maakt men gebruik van Azure DevOps. Op deze manier beschikken ze dan ook over een centraal systeem waar alles bijgehouden wordt. Eén onderdeel van de DevOps-omgeving is de sprintplanner. Ook wij maken gebruik van deze sprintplanner om een beter overzicht te kunnen krijgen van wat we reeds hebben verwezenlijkt en wat we nog moeten verwezenlijken.

Verder kan je per project ook gebruik maken van de GitHub repository die hierin verwerkt zit. Voor ons stageproject maken wij dus ook gebruik van Git. Als branching strategy maken we gebruik van de feature branching strategy. Dit houdt in dat we per feature van onze applicatie een nieuwe branch zullen aanmaken. Dit zorgt ervoor dat we de volledige feature eerst goed kunnen testen voordat we deze toelaten tot de huidige versie van onze applicatie.

6 Certificaten & IDP

Certificaten

Certificaten worden gebruikt om data te versleutelen en om ervoor te zorgen dat deze data enkel ontsleuteld kan worden door de partij met de juiste sleutel. In dit proces komen er drie belangrijke bestanden voor.

- ✓ .crt: Dit is het certificaat samen met de public key.
- ✓ .key: Dit is de private key, deze hebben we nodig om de data van het certificaat te ontsleutelen.
- ✓ .pfx, .p12: Om een .p12 bestand aan te maken hebben we een .crt en een .key bestand nodig. De combinatie van deze twee vormt dan het .p12/.pfx bestand. Dit bestand bestaat uit een certificaat met een public en private key. Meestal staat er ook een wachtwoord op het bestand zodat niet iedereen zomaar meteen volledige toegang heeft. Dit omdat het bestand alles omvat dat nodig is om de achterliggende data kunnen te bemachtigen.

Certificaten vormen een belangrijk onderdeel van onze applicatie. Aangezien wij niet alleen informatie moeten kunnen beheren van certificaten, maar ook PFX-bestanden moeten kunnen genereren en versturen naar een FTP-server, zullen we dus veel in contact komen met deze terminologie.

Identity provider

Een Identity Provider (IdP) wordt gebruikt om de identiteit van gebruikers te verifiëren. Deze verificatie gebeurt op basis van het wachtwoord dat de gebruiker ingeeft. Als de gebruiker geverifieerd is kan men aan de data waarvoor de IdP ingesteld staat. Dit proces bestaat uit 3 stappen.

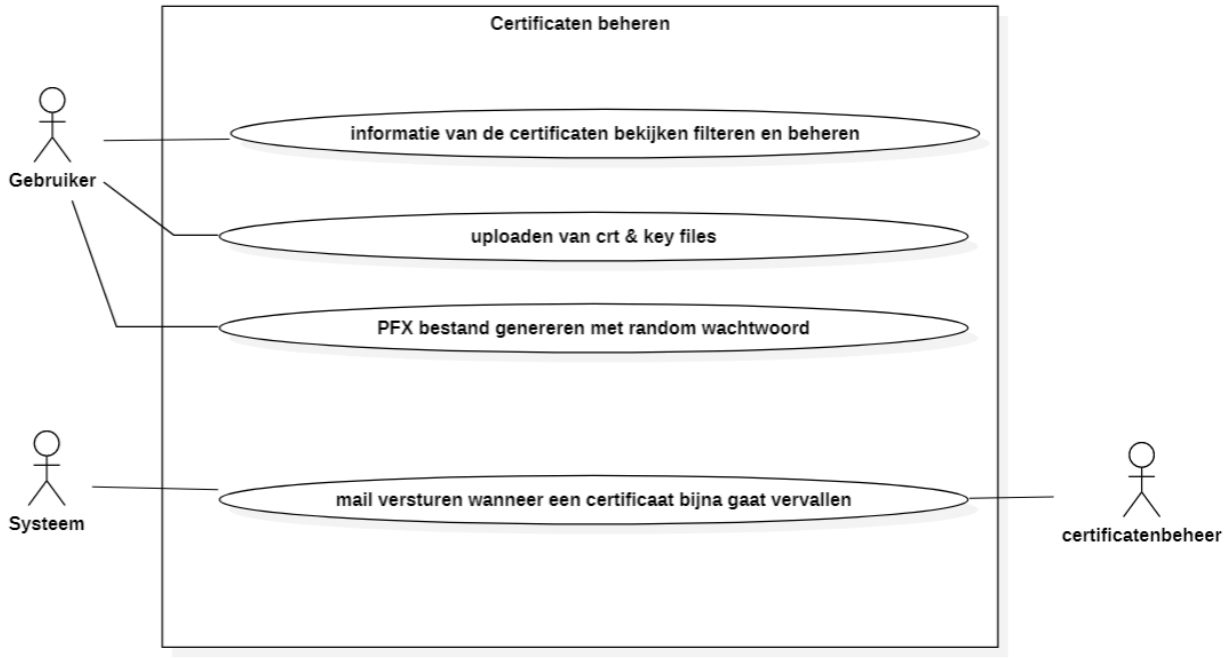
- ✓ Request: De gebruiker wordt gevraagd om een vorm van identiteit op te geven.
- ✓ Verification: De IdP gaat kijken of deze gegevens kloppen en welke informatie deze gebruiker mag zien.
- ✓ Unlocking: De gebruiker krijgt toegang tot de data waarvoor ze geautoriseerd zijn.

Onze applicatie zal connecteren met de gebruikersbeheertool van Remmicom. Dit is een interne tool die gebruikt wordt om mensen toegang te verlenen tot de interne applicaties van Remmicom. Op basis van de geregistreerde gebruikersrechten zal de persoon in kwestie dan wel of niet toegang verkrijgen tot bepaalde onderdelen van de applicatie.

7 Eisenanalyse

In dit onderdeel bekijken we wat de belangrijkste eisen van het systeem zijn dat we moeten ontwikkelen.

7.1 UC-diagram



Nog enkele opmerkingen omtrent de afbeelding hierboven:

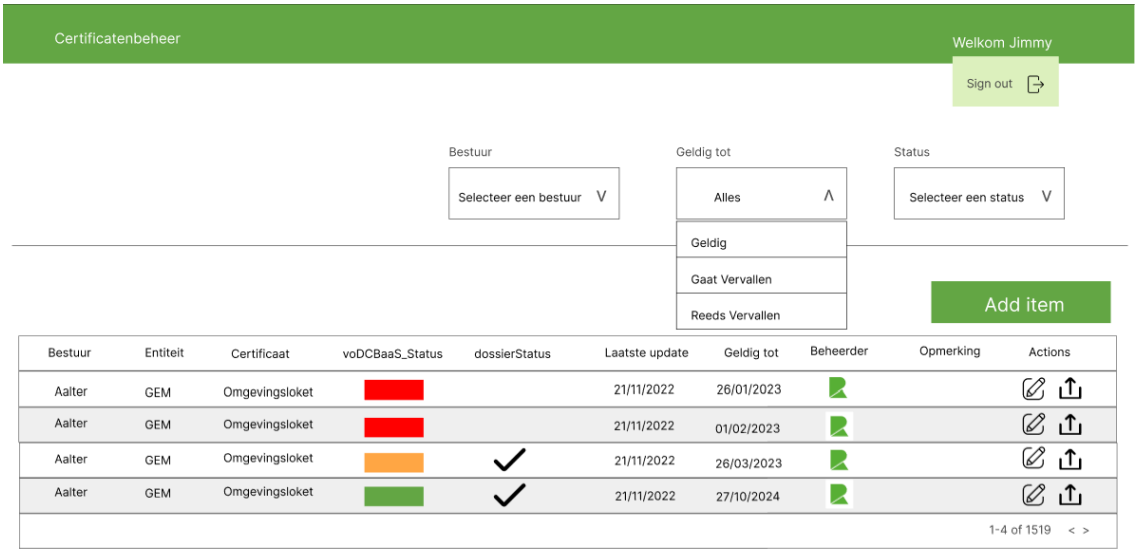
- ✓ Het versturen van een PFX/P12-bestand met hun bijhorende random wachtwoord is ook een functionaliteit. Dit verloopt echter gelijk met het genereren van het certificaat (PFX/P12-bestand) en zit daar dus mee in verwerkt.
- ✓ Het versturen van een e-mail is niet van de hoogste prioriteit. Via filtering kan men steeds kijken welke certificaten bijna gaan vervallen.

7.2 Figma prototype

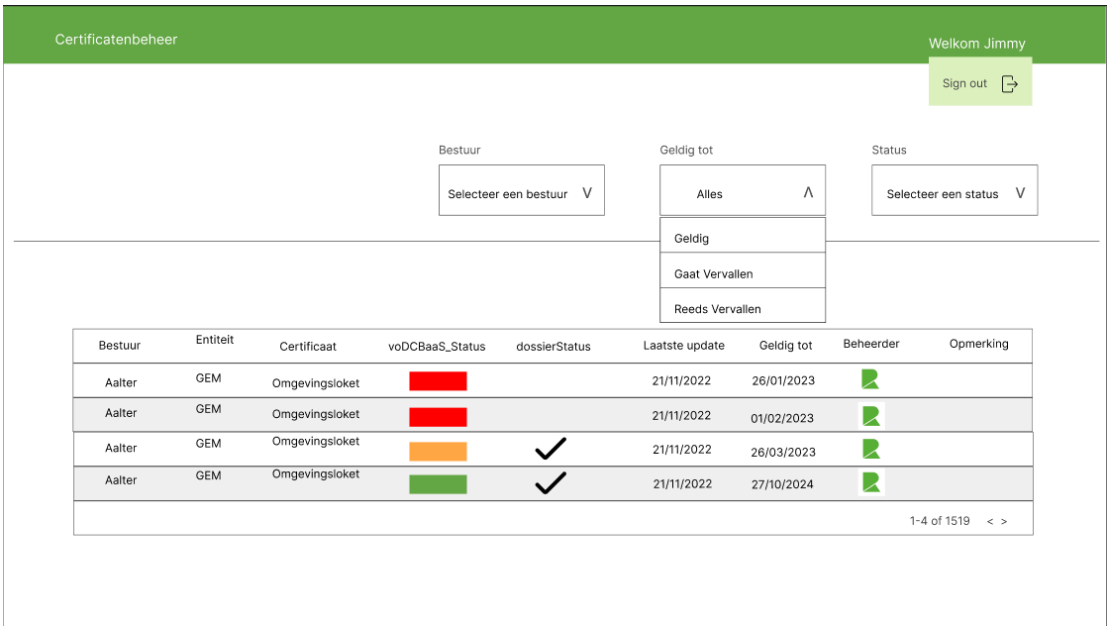
We hebben een prototype ontworpen in Figma zodat we met onze stagebegeleider en de eindgebruikers konden overleggen of het idee voor de applicatie goed was. Onze initiële feedback hierover was goed. Het is echter belangrijk om te weten dat er oorspronkelijk gezien maar één CRUD aanwezig ging zijn in de applicatie. Het leek ons echter absurd om enkel de data van de certificaten te kunnen beheren via onze applicatie. We hebben daarom ook besloten om nog extra CRUDs aan te maken voor de andere onderdelen uit onze databank. Deze staan echter niet bij de prototypes hieronder aangezien we dit pas later beslist hebben. Voor deze CRUDs hebben we het algemene design behouden. Nadien hebben we deze aangepast waar nodig om aan de eisen te kunnen voldoen.

✓ Onze homepagina

Beheerders:

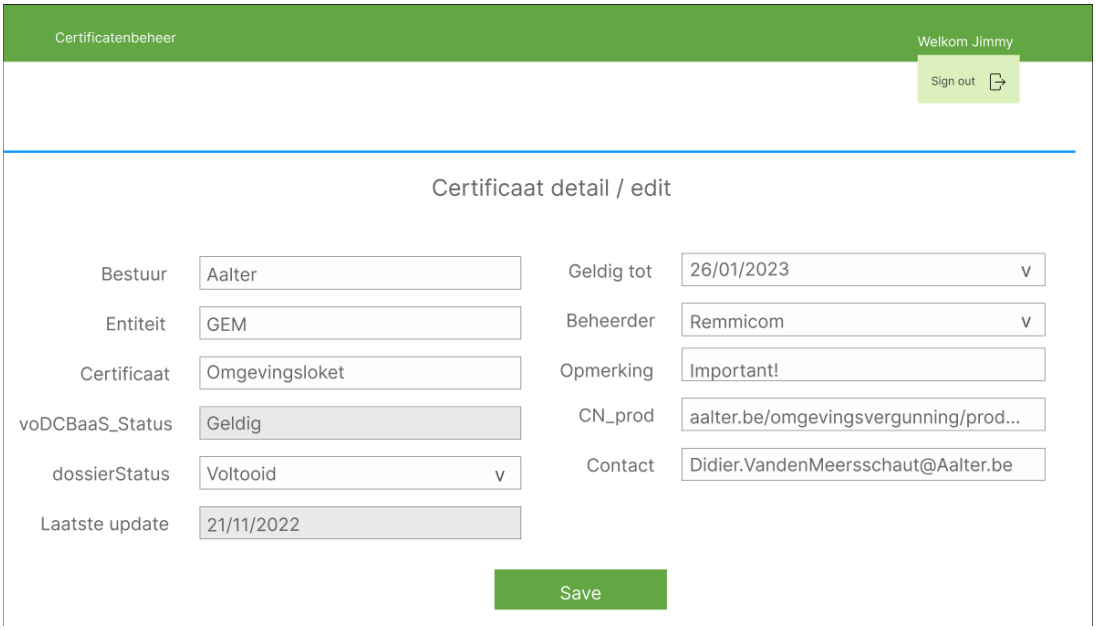


Gebruikers:

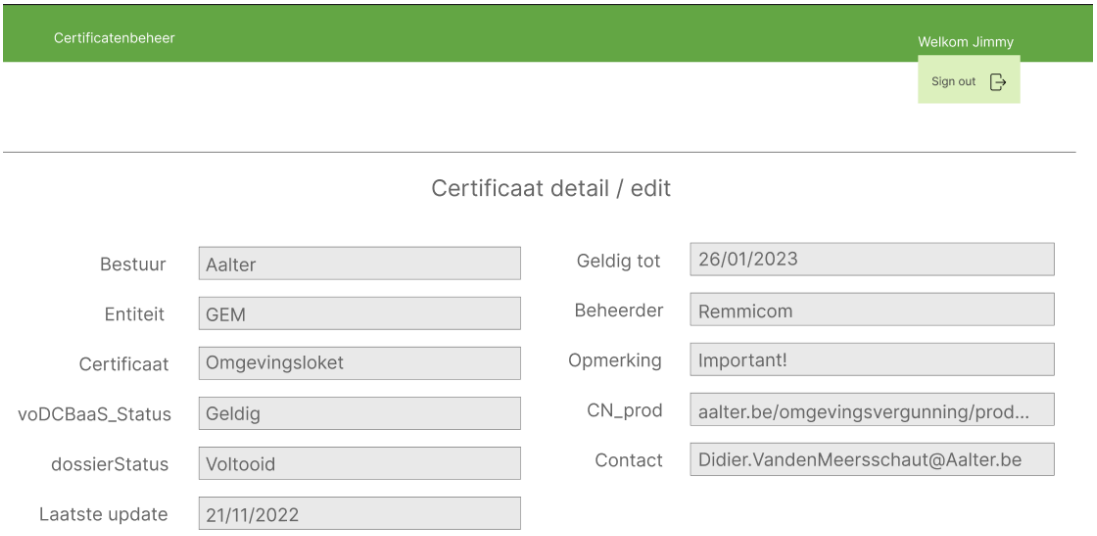


✓ Onze edit & detailpagina

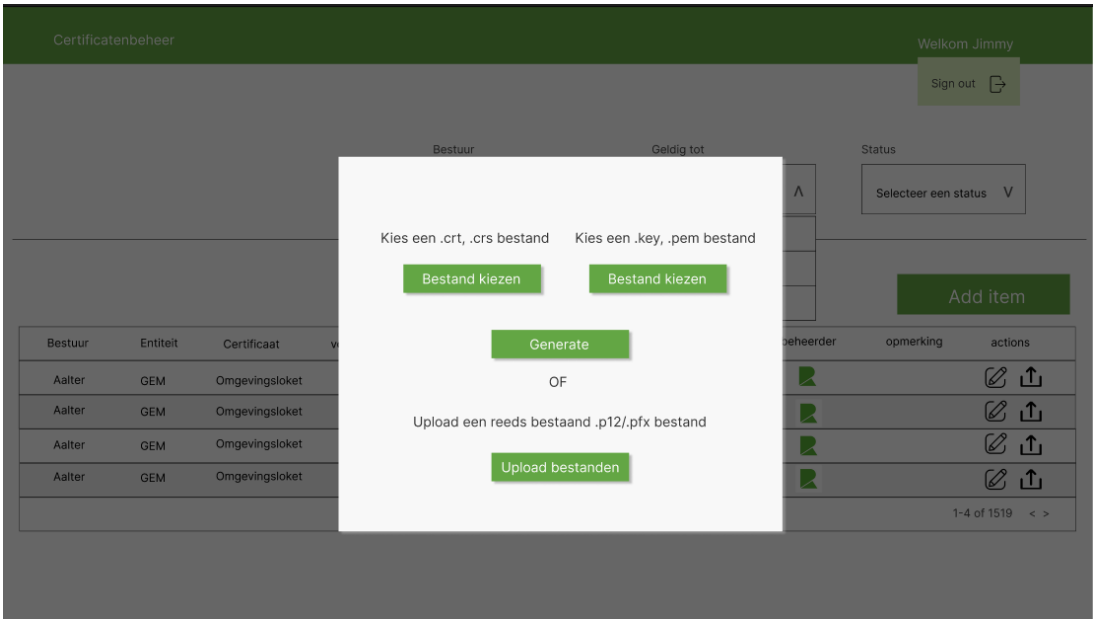
Beheerders:



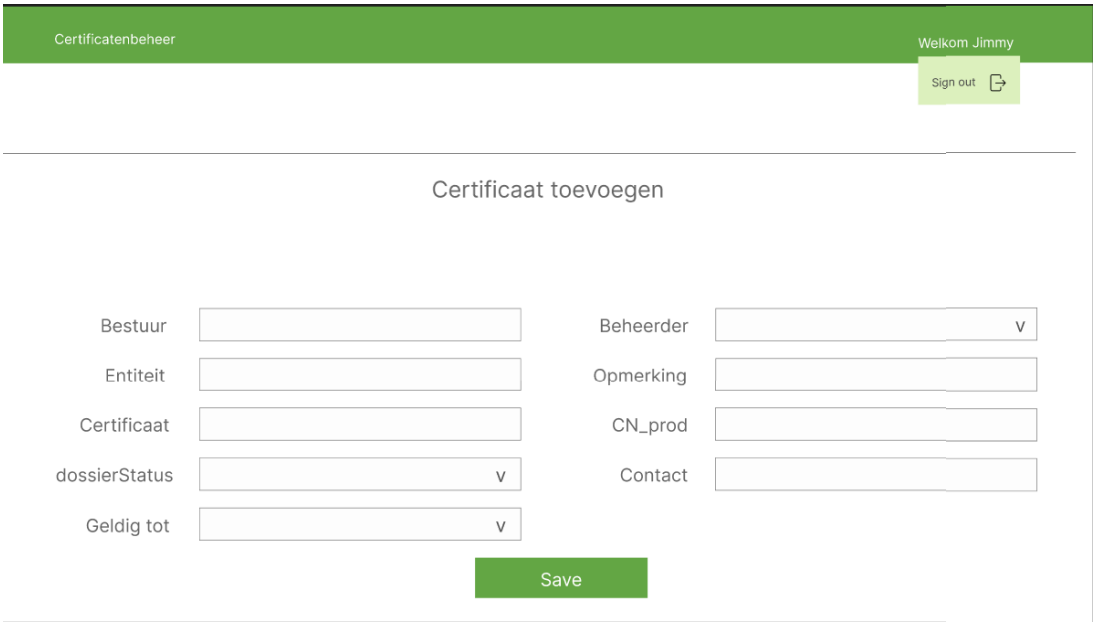
Gebruikers:



✓ Onze uploadpagina (ook PFX genereren)



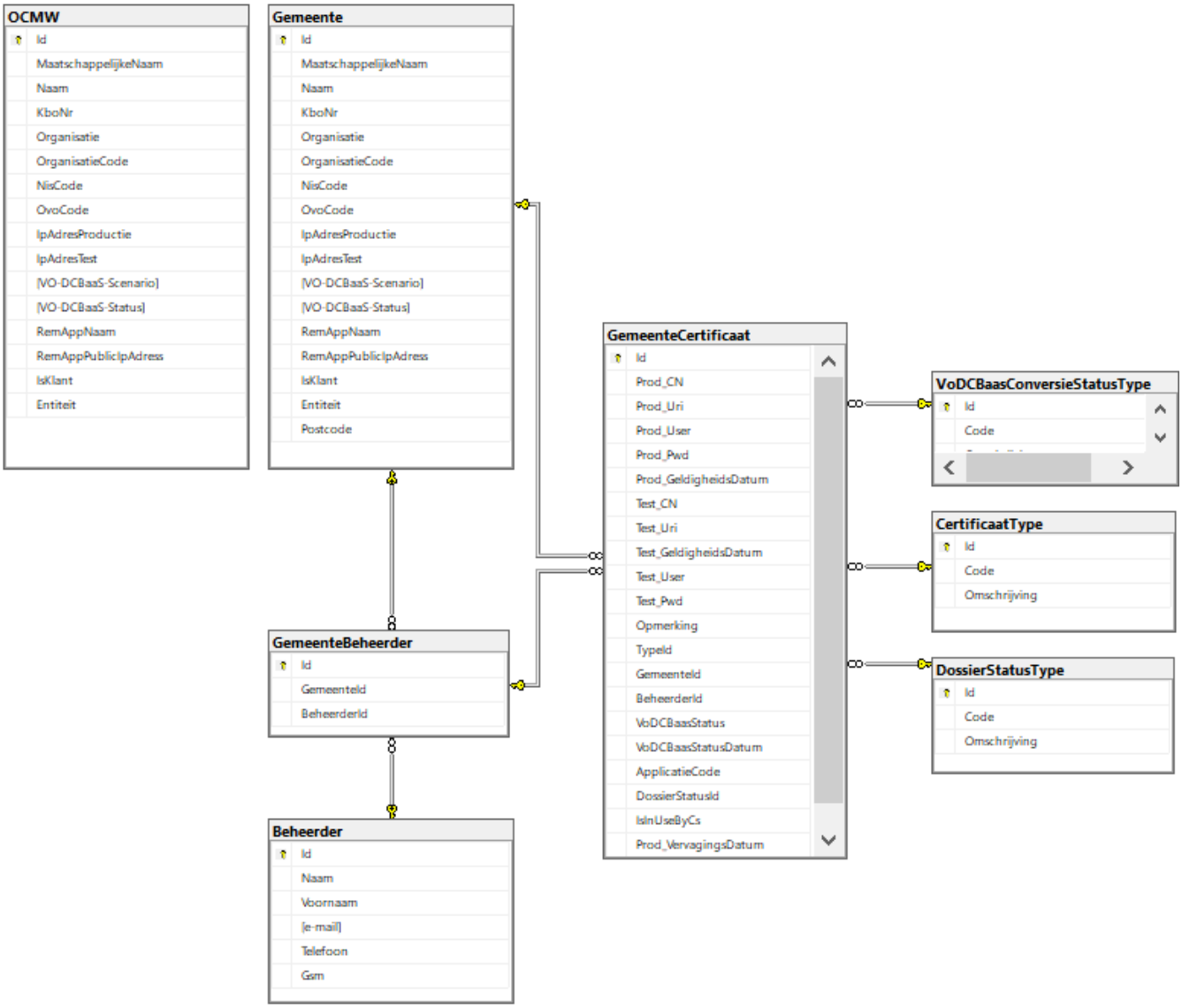
✓ Onze create pagina



7.3 Databank (ERD)

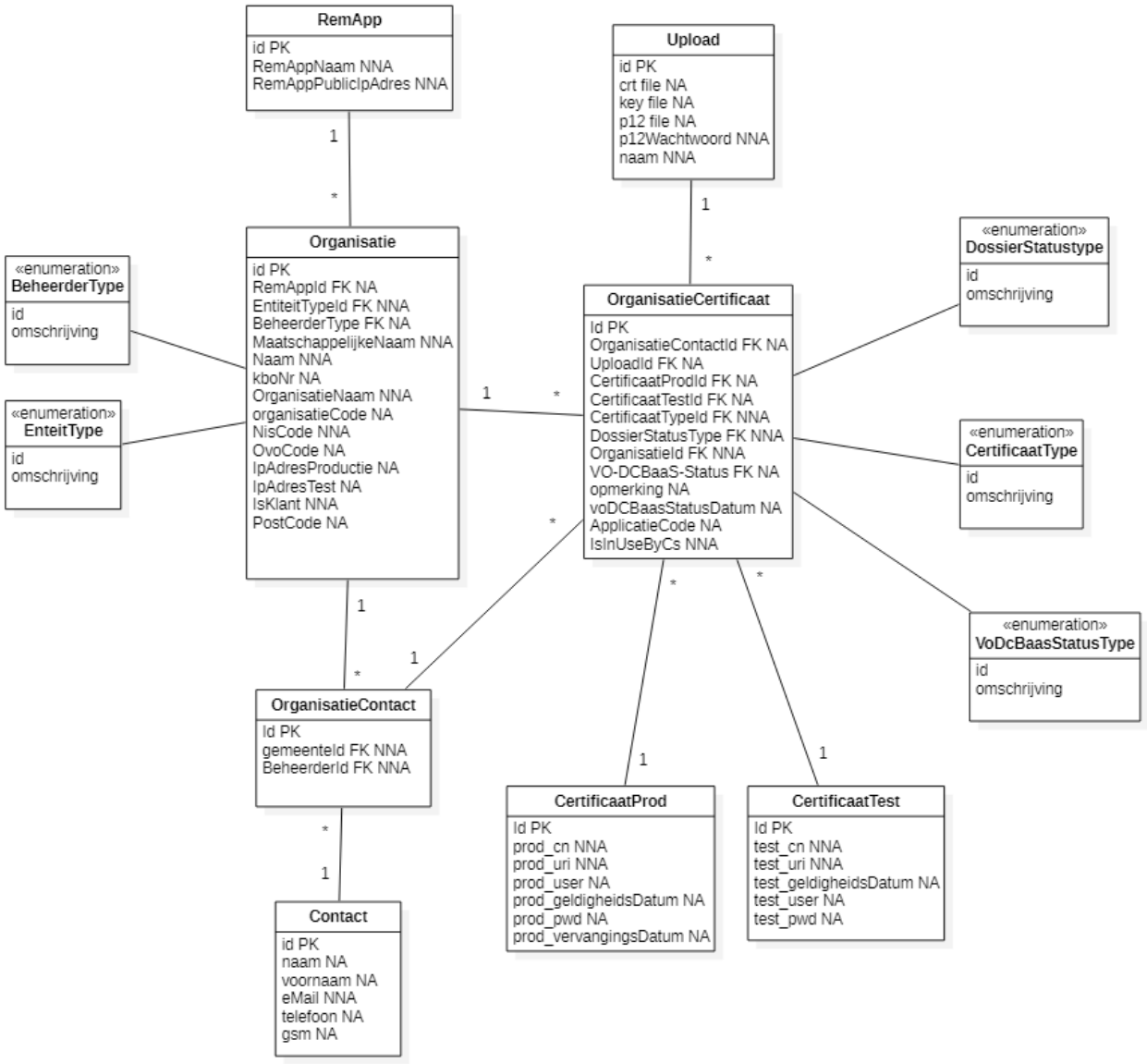
Oorspronkelijk gezien gingen we gebruik blijven maken van de oude databank. Deze was echter niet ideaal en daarom hebben we in overleg met onze stagebegeleider besloten om een nieuwe databank op te stellen. We hebben de beslissing genomen een nieuwe databank aan te maken aangezien er veel dingen onduidelijk waren of niet zo gebruikelijk waren. Denk bijvoorbeeld aan te weinig generalisatie, naamgevingen die niet duidelijk waren, onnodige dubbele data, enzovoort. Verder werd er ook niet veel gebruik gemaakt van de databank zelf om data op te halen. Ze maakten eerder gebruik van views om de data van de tabellen gemakkelijker op te kunnen halen. Nu is er veel meer sprake van generalisatie en beschikken we over minder nutteloze data. Verder hebben we de views ook niet meer nodig aangezien alles nu duidelijk en coherent is. We konden natuurlijk niet alles weglaten aangezien de data nog wel grotendeels hetzelfde zal moeten blijven. Ons vernieuwde diagram, ook wel een ERD genoemd, hebben we eerst uitgewerkt in StarUML.

✓ De oude databank

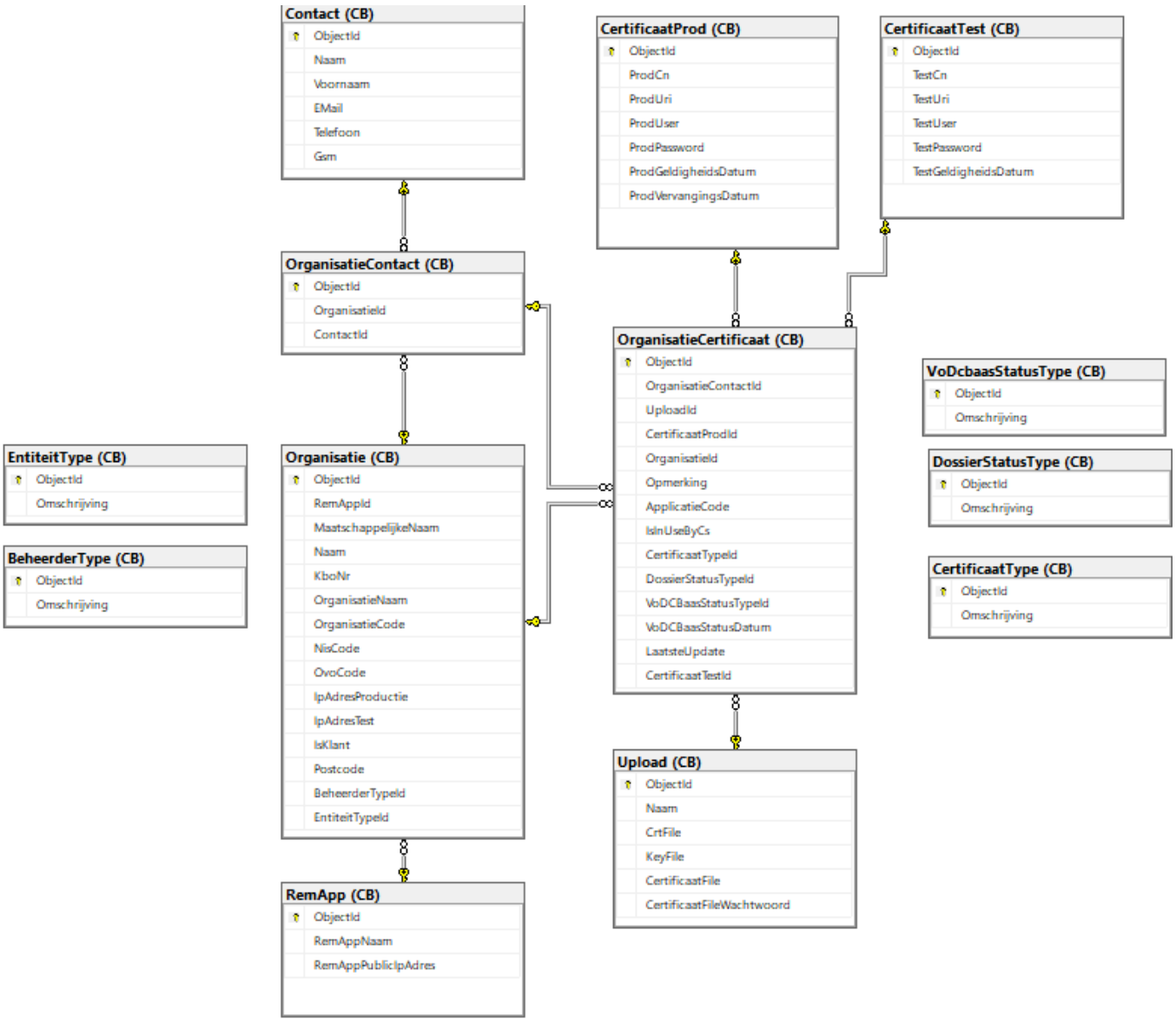


✓ De nieuwe databank

StarUML:



SQL-diagram:


Contact (CB)
ObjectId
Naam
Voornaam
EMail
Telefoon
Gsm
CertificaatProd (CB)
ObjectId
ProdCn
ProdUri
ProdUser
ProdPassword
ProdGeldigheidsDatum
ProdVervangingsDatum
CertificaatTest (CB)
ObjectId
TestCn
TestUri
TestUser
TestPassword
TestGeldigheidsDatum
OrganisatieContact (CB)
ObjectId
OrganisatieId
ContactId
OrganisatieCertificaat (CB)
ObjectId
OrganisatieContactId
UploadId
CertificaatProdId
OrganisatieId
Opmerking
ApplicatieCode
IsInUseByCs
CertificaatTypeId
DossierStatusTypeId
VoDCBaasStatusTypeId
VoDCBaasStatusDatum
LaatsteUpdate
CertificaatTestId
VoDcbaasStatusType (CB)
ObjectId
Omschrijving
DossierStatusType (CB)
ObjectId
Omschrijving
CertificaatType (CB)
ObjectId
Omschrijving
EntiteitType (CB)
ObjectId
Omschrijving
BeheerderType (CB)
ObjectId
Omschrijving
Organisatie (CB)
ObjectId
RemAppId
MaatschappelijkeNaam
Naam
KboNr
OrganisatieNaam
OrganisatieCode
NisCode
OvoCode
IpAdresProductie
IpAdresTest
IsKlant
Postcode
BeheerderTypeId
EntiteitTypeId
Upload (CB)
ObjectId
Naam
CrtFile
KeyFile
CertificaatFile
CertificaatFileWachtwoord
RemApp (CB)
ObjectId
RemAppNaam
RemAppPublicIpAdres

7.4 **Authenticatie**

Wie?

Binnen de applicatie zijn er drie mogelijke rollen aanwezig. Deze hebben elks andere rechten.

✓ Niet-ingelogde gebruiker
Geen rechten

✓ Ingelogde gebruiker
Leesrechten

✓ Beheerder
Lees- en schrijfrechten

Hoe?

Voor de authenticatie maken we gebruik van het onderdeel gebruikersbeheer van Remmicom. We connecteren hiermee via OpenID Connect. We maken zowel gebruik van authenticatie in de back-end als in de front-end. De bedoeling is eigenlijk dat we connecteren met de gebruikersbeheer API van Remmicom. Hier staat onze app dan bij met alle leden en hun rechten. Op deze manier kan men via gebruikersbeheer alle gebruikers van onze applicatie beheren. Onze interne applicatie connecteert dan met deze API om te kijken of de gebruikers geautoriseerd zijn om een bepaalde actie uit te voeren op onze applicatie.

✓ Back-end
De verschillende API-calls zullen beveiligd zijn. Dit betekent dat u zich moet autoriseren om de actie uit te mogen voeren, anders krijgt u een 401 unauthorized error van de API.

✓ Front-end
We kijken of de gebruiker ingelogd is (en eventueel een beheerder is) en op basis hiervan gaan we dan wel of niet toegang verlenen tot bepaalde routes. Verder kunnen we ook de knoppen die enkel de beheerder mag zien verbergen voor mensen die deze rechten niet hebben. Ook kunnen we in de navigatiebalk de naam van de gebruiker tonen.

8 RXJS

In dit hoofdstuk staat meer informatie vermeld omtrent de doelen en het nut van RxJS. Wij gebruiken RxJS om te zorgen voor een correcte implementatie van Observables in Angular. Dankzij de verschillende onderzoeken hebben wij dan ook verschillende best practices bijgeleerd omtrent het werken met Observables. Heel kort gezegd is een Observable iets dat kan zorgen voor het verspreiden van informatie doorheen een applicatie. Dit alles aan de hand van het observeren van een bron van informatie. Meer informatie hierover staat vermeld in de volgende onderdelen van dit document.

Alle informatie omtrent RxJS hebben we verkregen via een video van Pluralsight.

- ✓ Pluralsight, RxJS in Angular: Reactive Development, door Deborah Kurata.
- ✓ Zie bronvermelding 2.

8.1 Wat is RxJS?

- ✓ RxJS staat voor Reactive Extensions for JavaScript.
- ✓ RxJS kan je onder andere gebruiken met frameworks zoals Angular, React, Vue. Verder kan je het gebruiken met JavaScript en TypeScript.
- ✓ RxJS is een library die ons helpt bij het opstellen van asynchrone en event-based programma's door gebruik te maken van observables. Het is een manier om data en events te observeren en hierop te reageren terwijl ze door de applicatie bewegen.
- ✓ RxJS kan dus werken met asynchrone data zodat de applicatie normaal blijft werken terwijl we wachten tot de request geslaagd is.

8.2 Waarom RxJS?

- ✓ Eén techniek om met alle soorten data te werken.
- ✓ We kunnen gemakkelijk data samenstellen en samenbrengen.
- ✓ Het is eenvoudig om user interactions & data changes op te merken en hierop te reageren.
- ✓ RxJS is 'lui'. Hiermee bedoelen we dat alles enkel uitgevoerd wordt wanneer we het nodig hebben.
- ✓ RxJS beschikt over built-in error handling.
- ✓ We kunnen asynchrone acties ook annuleren. Als we bijvoorbeeld product A willen bekijken, maar deze actie plots annuleren en in de plaats hiervan product B willen bekijken, kunnen we de actie nog steeds annuleren om zo enkel product B te tonen.

8.3 Reactieve formulieren

Reactieve formulieren gebruiken we om reactieve aanpassingen op te merken wanneer de gebruiker iets aanpast in een formulier. Denk bijvoorbeeld aan een tekstvak dat rood wordt wanneer men iets verkeerd invult. Het systeem herkent meteen de foutieve waarden en reageert hierop.

8.4 Reactieve ontwikkeling

Hieronder staat een eenvoudig voorbeeld vermeld dat kan helpen bij het verkrijgen van een beter beeld omtrent reactief ontwikkelen.

Eenvoudig wiskundig voorbeeld
X = 5
Y = 3
Z = X + Y

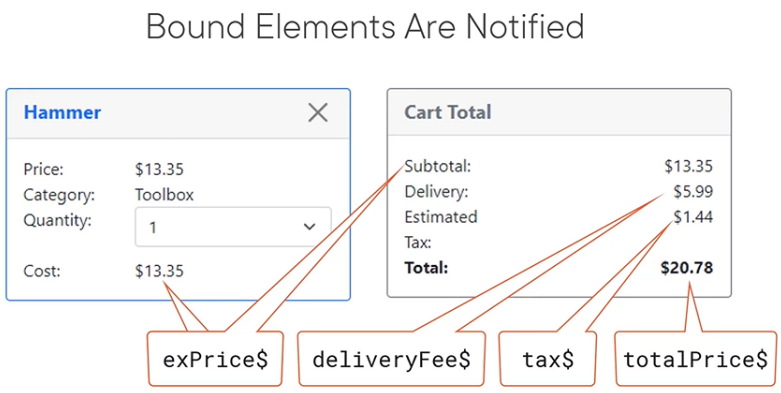
Wanneer X of Y aanpast, zal Z hier geen rekening mee houden aangezien dit enkel de eerste keer een waarde toegewezen krijgt. Het kan echter voorkomen dat we willen dat Z wél actief aanpast. Denk bijvoorbeeld aan een online webshop. Als de hoeveelheid van een product aangepast wordt door de gebruiker moeten we ook de totaalprijs actief mee aanpassen.

Optie 1: Getter Function
We roepen bij aanpassingen steeds de getter aan om Z te berekenen. Dit is echter niet altijd de beste optie. De totaalprijs van dat product zal wel actief mee aanpassen, maar de totaalprijs van ons winkelmandje zal niet mee aanpassen.

Optie 2: Event Handler
Als de gebruiker de hoeveelheid aanpast zal het event aangemaakt worden en zal de nieuwe prijs berekend worden. Ook deze optie heeft hetzelfde probleem. De totaalprijs van dat product zal wel actief mee aanpassen, maar de totaalprijs van ons winkelmandje zal niet mee aanpassen.

Optie 3: RxJS
We maken van de hoeveelheid van een project een observable, aangezien we de hoeveelheid moeten observeren. Deze kan namelijk aangepast worden. De syntax hiervoor is "qty$ = new Observable();". Het dollarteken gebruiken we om te laten weten dat die variabele geobserveerd kan worden bij het reageren op aanpassingen.
Als de quantity observable opmerkt dat er nieuwe aanpassingen zijn zal ook de exPrice variabele automatisch berekend worden. Dankzij de observable kunnen we opmerken wanneer er aanpassingen zijn gemaakt en hier op reageren door de andere zaken opnieuw te berekenen.

De afbeelding hieronder toont aan hoe de totaalprijs berekend wordt op basis van de wijzigende bron van informatie.

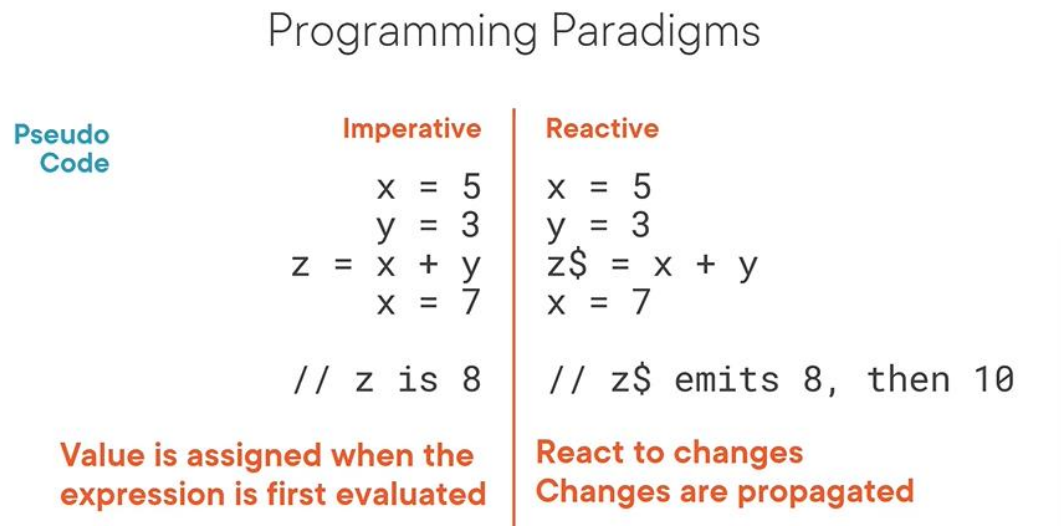


Bron afbeelding: zie bronvermelding 2

8.5 Imperatief VS Reactief

Imperatief => wordt enkel de eerste keer, bij initialisatie, berekend.
Reactief => wijzigt bij aanpassingen.

De afbeelding hieronder toont dit verschil ook nog eens aan op basis van een voorbeeld.



Bron afbeelding: zie bronvermelding 2

8.6 Wat is Reactieve ontwikkeling?

Reactive Development, ofwel reactieve ontwikkeling, is een declaratief programmeerparadigma dat zich bezighoudt met data streams en de verspreiding van verandering.

Code is reactief wanneer een verandering in input leidt tot een automatische verandering in output. Denk bijvoorbeeld aan het voorbeeldje, Z = X + Y. Als Z automatisch aanpast wanneer X of Y aanpast zijn we reactief aan het werk. Op deze manier kunnen we gemakkelijk reageren op gebruikersacties, state changes, samenstellen en combineren van data streams zoals bij het calculeren van de totaalprijs, communicatie tussen verschillende componenten zonder close coupling, ook error handling en management van states.

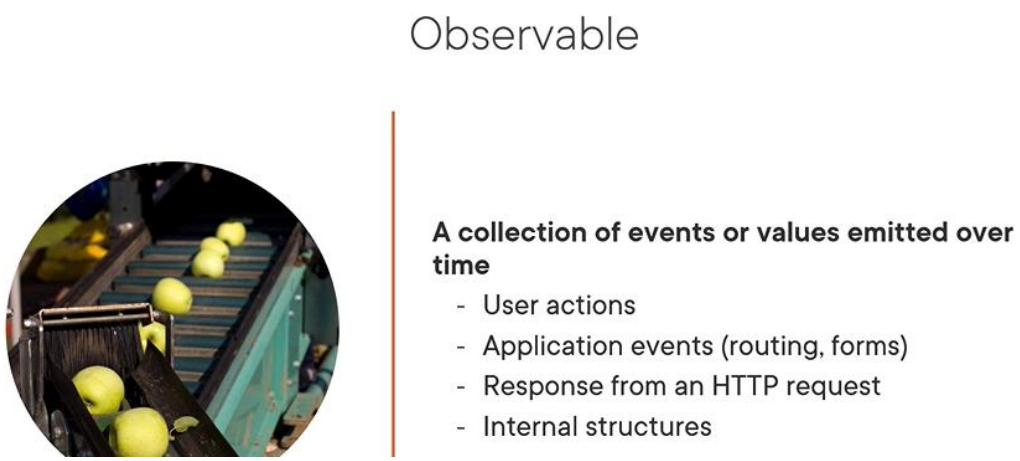
8.7 RxJS terminologie en syntax

Observable

Een observable is een object waar we notificaties van kunnen ontvangen. Het is een lijst van acties of waarden over een periode van tijd. Een observable zal een observer notificeren wanneer er een nieuwe waarde of actie is.

Hieronder vindt u wat meer uitleg hierover aan de hand van enkele afbeeldingen.

In de afbeelding hieronder ziet u wat meer informatie omtrent wat we allemaal kunnen zien als een observable. Een observable is eigenlijk een object dat iets in de gaten houdt. Denk bijvoorbeeld aan appels op een lopende band. Wanneer een appel aankomt bij het eindpunt, zal de observable dit doorgeven aan de personen die dit moeten weten. De personen die dit moeten weten zijn dan de observers.



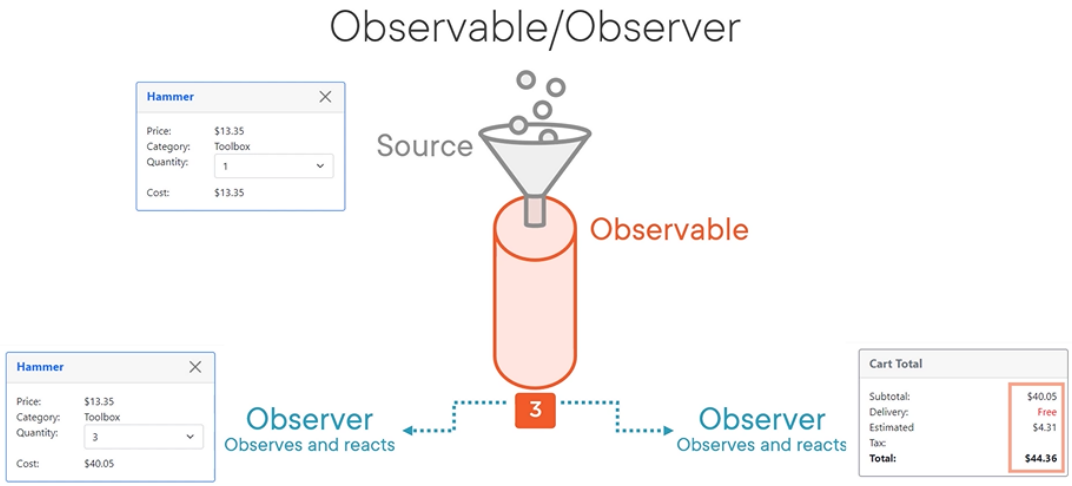
Bron afbeelding: zie bronvermelding 2

Hieronder staat ook nog een extra afbeelding die wat duidelijkheid kan scheppen. Hierbij zien we de bron van informatie, de observable en de observer.

De bron van informatie hieronder is het formulier van een webshop. Een gebruiker wilt iets bestellen. Ik zal als voorbeeld appelen gebruiken. De prijs van een appel is bepaald door de applicatie zelf. De gebruiker kan echter de hoeveelheid appelen aanpassen.

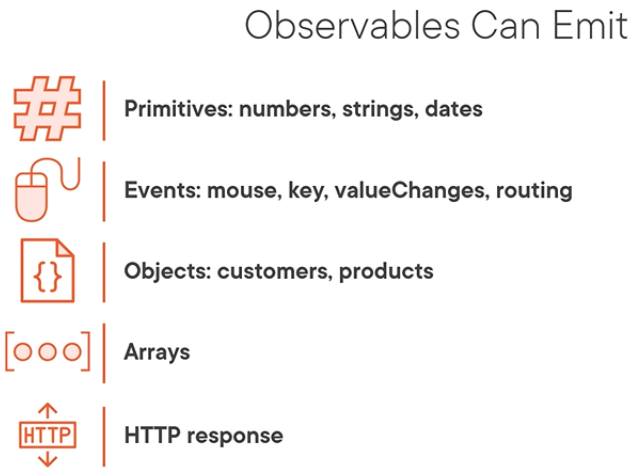
De observable houdt rekening met de bron van informatie. Deze weet de prijs van een appel, en weet ook hoeveel appelen een gebruiker wil kopen. Als de gebruiker de hoeveelheid appelen aanpast, zal de observable dit weten. Vervolgens zal hij dit laten weten aan de observers.

Een observer luistert naar een observable. In dit geval zal bijvoorbeeld de totaalprijs van ons winkelmandje wijzigen op basis van de hoeveelheid appelen. De observable liet ons weten dat de hoeveelheid appelen gewijzigd is. De observer krijgt dit te horen en past ook meteen de totale prijs aan. Dit is nu de prijs die u kan zien in uw winkelmandje.



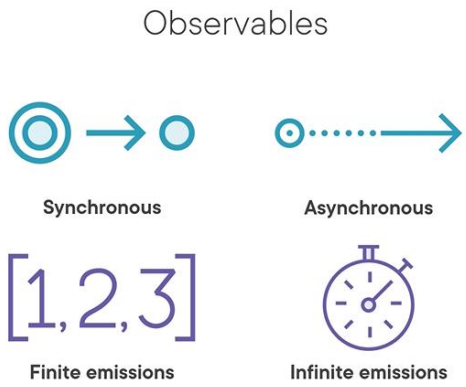
Bron afbeelding: zie bronvermelding 2

Hieronder staat ook een afbeelding die aangeeft wat een observable allemaal kan uitzenden. Denk bijvoorbeeld aan nummers, gebeurtenissen, objecten, enzovoort. Een observable kan zelfs een andere observable uitzenden.



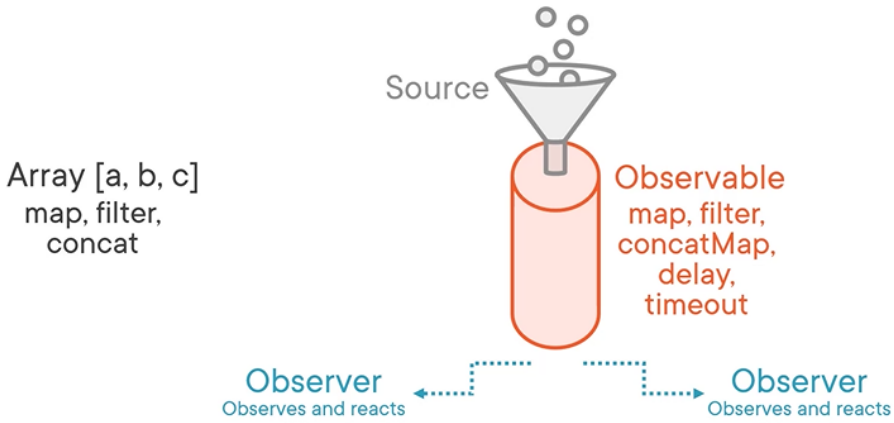
Bron afbeelding: zie bronvermelding 2

De afbeelding hieronder verduidelijkt ook dat er sprake kan zijn van synchrone en asynchrone observables. Bij synchrone observables zal alles lopen volgens plan. Bij asynchrone observables weten we niet van tevoren wat er allemaal kan gebeuren. Alle acties kunnen door elkaar plaatsvinden.



Bron afbeelding: zie bronvermelding 2

We kunnen vervolgens ook gebruik maken van de methode pipe(). Deze brengt het object door verschillende operaties. Zo kunnen we de geregistreerde data bijvoorbeeld filteren. Hierbij geven we aan wat de voorwaarden zijn van de data die we willen tonen. Als je bijvoorbeeld wil kijken hoeveel appelen je op een bepaalde dag hebt verkocht, dan kan je de informatie van deze appelen ophalen, waarbij de verkoopdatum gelijk is aan de datum naar keuze. De afbeelding hieronder stelt dit ook stelselmatig voor.



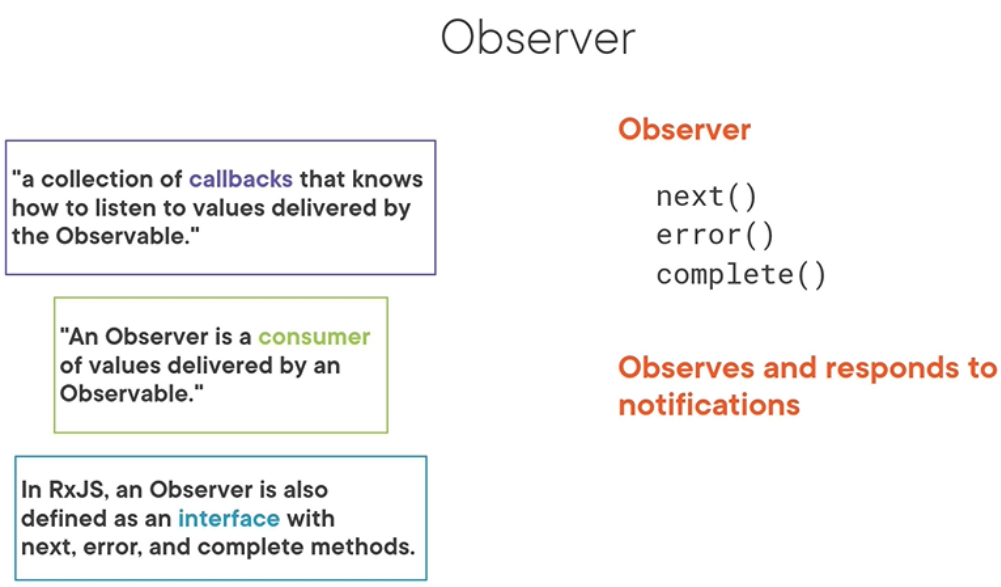
Bron afbeelding: zie bronvermelding 2

Observer

Een observer observeert observables en reageert op notificaties van observables.

Hieronder vindt u wat meer uitleg hierover aan de hand van enkele afbeeldingen.

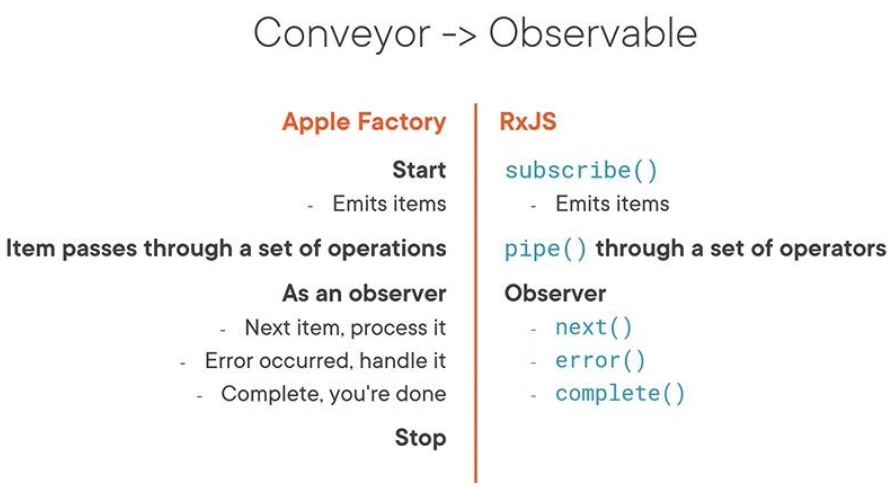
De afbeelding hieronder geef wat meer informatie omtrent observers. Zoals vooraf reeds vermeld luistert een observer naar de notificaties van observables. Het is dus een consument. Hij ontvangt namelijk data van een observable. Een observer is ook een soort van interface aangezien deze steeds beschikt over verschillende methodes. Denk aan de next(), error() en complete() methodes. Deze methodes kunnen we gebruiken om iets te doen nadat we iets gedaan hebben. Als we bijvoorbeeld data proberen op te halen, maar er vindt een error plaats, dan komen we in de error() methode terecht. Hier kunnen we dan een gepaste actie uitvoeren.



Bron afbeelding: zie bronvermelding 2

De afbeelding hieronder verduidelijkt de werking van observables en observers aan de hand van een voorbeeld. In de afbeelding kan je enkele acties zien die in een appelfabriek overeen zouden komen met de verschillende instructies in RxJS.

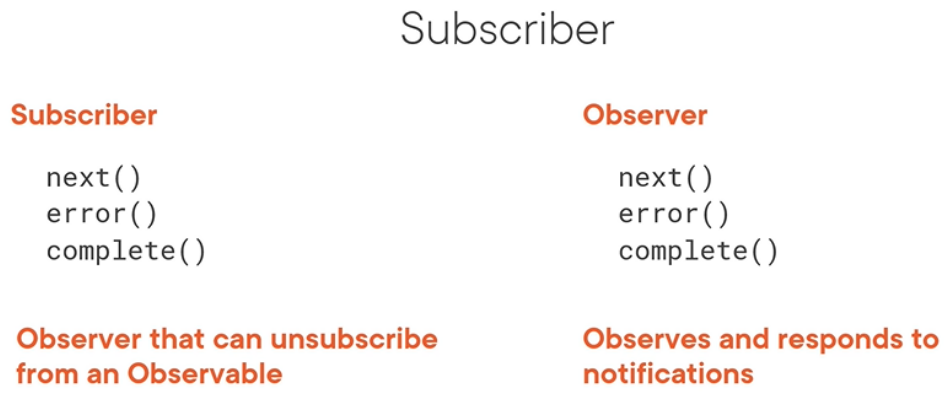
Een observer subscribed op een observable om hier data van te kunnen ontvangen. Dit is het startpunt. Nadien kan men pipe() gebruiken om operaties uit te voeren op het object. Hierbij kan je denken aan een appel die verschillende acties moet ondergaan in een appelfabriek. Een observer heeft vervolgens ook drie methodes, zoals vooraf reeds vermeld. Elk van deze methodes behandeld dan een bepaalde situatie. Op deze manier kunnen we aangeven wat we moeten doen wanneer we in zo'n situatie aankomen. Ter voorbeeld, wanneer we bij error() terechtkomen, kunnen we aangeven wat we moeten doen in geval van een probleem. In het geval van een appelfabriek kunnen we de loopband bijvoorbeeld stopzetten. In onze applicatie zouden we bijvoorbeeld een errorpagina kunnen tonen.



Bron afbeelding: zie *bronvermelding* 2

Subscriber

Een subscriber is een implementatie van een observer. Het kan dus alles doen wat een observer kan doen, maar aangezien het een daadwerkelijke implementatie is kan je dus ook weer unsubscriben om ervoor te zorgen dat we geen notificaties meer ontvangen. De actie 'unsubscribe' is van groot belang om te vermijden dat we data leaks krijgen.

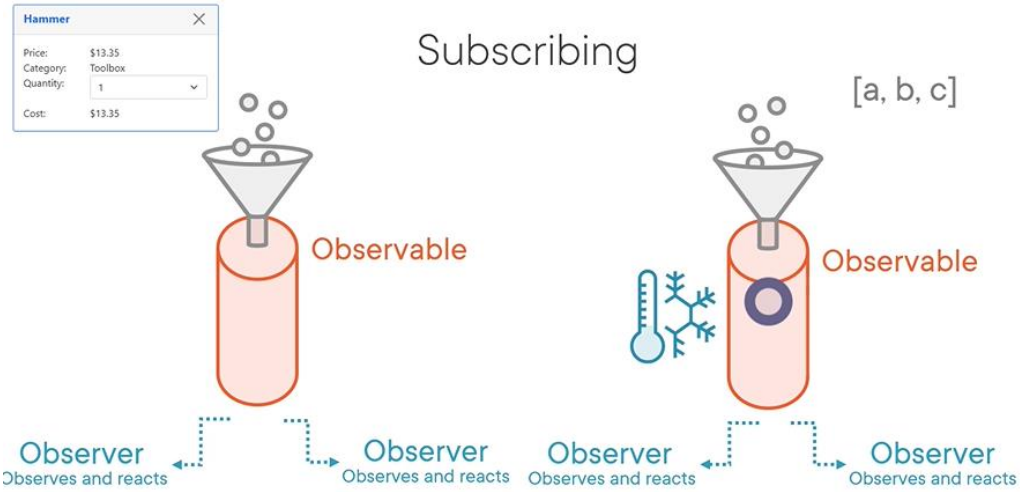


Bron afbeelding: zie bronvermelding 2

8.8 Subscribing

Er zijn zowel observables die al beginnen met het uitzenden van data voordat er subscribers zijn, als observables die pas beginnen met het uitzenden van data wanneer er een subscriber is. Een subscriber is dus een observer die luistert naar en reageert op notificaties van een observable. Vervolgens kan deze ook weer unsubscriben om data leaks te vermijden.

De afbeelding hieronder stelt nogmaals kort stelselmatig voor hoe dit werkt. In het ene geval zenden we dus altijd data uit (links), in het andere geval zenden we dus enkel data uit wanneer we een subscriber hebben (rechts). Een subscriber is een implementatie van een observer.



Bron afbeelding: zie bronvermelding 2

De afbeelding hieronder verduidelijkt hoe deze actie uitgevoerd kan worden in een applicatie. Hierbij subscriben we dus op een observable.

```
const sub = apples$.subscribe({
  next: apple => console.log(`Apple emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
});
```

Bron afbeelding: zie bronvermelding 2

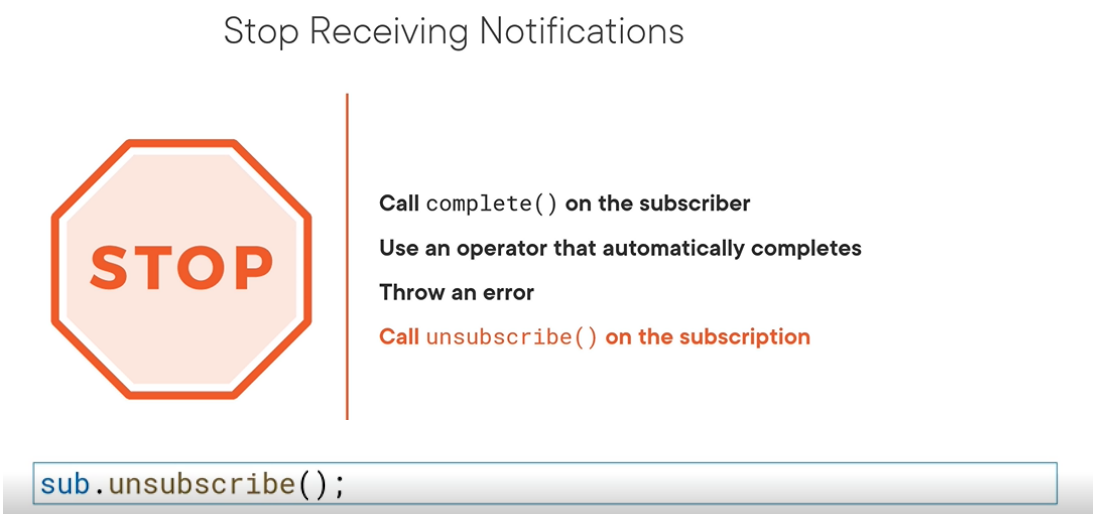
8.9 Unsubscribing

Het is ook erg belangrijk om steeds te unsubscriben wanneer we de pagina bijvoorbeeld verlaten. We doen dit om ervoor te zorgen dat we niet meer hoeven te luisteren naar de observables. Verder vermijden we zo ook memory leaks.

De afbeelding hieronder vertelt ons hoe we kunnen unsubscriben. Er zijn namelijk verschillende opties.

- ✓ We gebruiken complete() aan het einde
- ✓ We gebruiken een operator die automatisch de complete() methode implementeert
- ✓ We roepen zelf de unsubscribe() methode op

Verder zijn er ook nog andere mogelijkheden, zoals het stoppen met ontvangen van data wanneer we een bepaalde webpagina verlaten.



Bron afbeeldingen: zie bronvermelding 2

8.10 Creation functions

De afbeelding hieronder verduidelijkt hoe je een observable kan aanmaken.

Enkele opties staan hieronder vermeld

- ✓ New Observable()
- ✓ Of()
- ✓ From()

Creating an Observable

```
const apples$ = new Observable(appleSubscriber => {
  appleSubscriber.next('Apple 1');
  appleSubscriber.next('Apple 2');
  appleSubscriber.complete();
});
```

```
const apples$ = of('Apple1', 'Apple2');
```

```
const apples$ = from(['Apple1', 'Apple2']);
```

Bron afbeelding: zie bronvermelding 2

De afbeelding hieronder verduidelijkt ook het verschil tussen of() en from(). Het grootste verschil is dat 'of' de data zal blijven tonen zoals het is, terwijl 'from' de informatie eruit zal halen en deze zal tonen.

of vs. from

```
const apples = ['Apple 1', 'Apple 2'];
```

```
of(apples);
// [Apple1, Apple2]
```

```
from(apples);
// Apple1 Apple2
```

```
of(...apples);
// Apple1 Apple2
```

Bron afbeelding: zie bronvermelding 2

8.11 **Recap:**

De afbeelding hieronder is een korte schematische voorstelling van de belangrijkste onderdelen die besproken zijn. Aangezien ik niet heel de afbeelding opnieuw uit ga leggen, ga ik enkele belangrijke puntjes herhalen.

✓ Observable

Een observable houdt iets in de gaten en stuurt de informatie hiervan door naar de observers.

Voorbeeld: de klant wijzigt de hoeveelheid van een bepaald product op de webshop. De observable stuurt deze wijziging door naar de observers.

✓ Observer

Een observer luistert naar een observable. Vervolgens reageert hij op een correcte wijze.

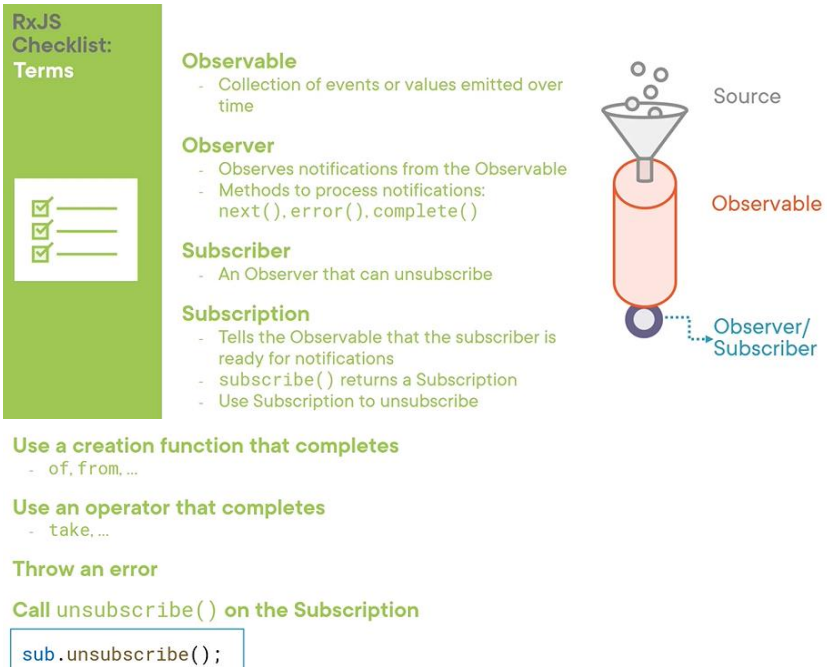
Voorbeeld: de observable vertelt de observer dat de hoeveelheid appelen die de klant wil bestellen gewijzigd is, waarna de observer de totaalprijs wijzigt.

✓ Subscriber

Een subscriber is een implementatie van een observable.

Het is belangrijk om steeds te unsubscriben om data leaks te vermijden. Er zijn echter bepaalde situaties waarbij een unsubscribe niet nodig is, aangezien deze dan automatisch uitgevoerd wordt. Hieronder staan enkele voorbeelden die een automatische unsubscribe implementeren.

✓ Complete()
✓ Sommige creation functions & operatoren (of, from, take, …)
  - Zij voeren vaak een complete() uit

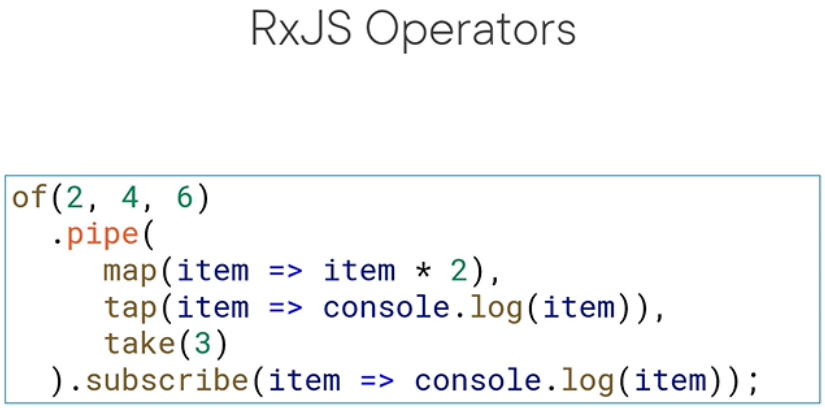


Bron afbeeldingen: zie bronvermelding 2

8.12 **RxJS Operators**

Een operator is een functie. Deze wordt onder andere gebruikt om uitgezonden items (observable) te transformeren en manipuleren. Gebruik de observable's pipe() methode om operatoren sequentieel toe te passen. Je kan een operator dus zien als een pipeline waarbij we een input hebben (source observable, de input), daarna onze pipeline (operator) die de input manipuleert / transformeert, en daarna onze output waar we op subscriben.

In de afbeelding hieronder gebruikt men een pipe() om nadien enkele operaties uit te voeren.



Bron afbeelding: zie bronvermelding 2

Enkele voorbeelden van mogelijke operatoren kan u terugvinden op de volgende website:

https://rxjs.dev/api

Zie ook bronvermelding 3.

9 Verder onderzoek

Hieronder staan nog enkele andere onderdelen vermeld waarnaar wij onderzoek hebben gedaan tijdens de eerste twee weken van onze stage. Pluralsight is één van de websites die we gebruikt hebben voor het verkrijgen van interessante informatie omtrent deze onderwerpen. Zie bronvermelding 1.

✓ Azure AD

We hebben enkele video's bekeken omtrent de werking van een .NET API met een Azure Active Directory ter authenticatie & autorisatie. We hebben achteraf echter besloten om de authenticatie te implementeren met het onderdeel gebruikersbeheer van Remmicom.

✓ Material UI & Sass

We hebben actief getest met Angular Material UI. Op deze manier hebben we onszelf al een beetje wegwijs gemaakt hiermee. Verder hebben we ook enkele informatieve video's bekeken omtrent dit onderwerp. Ook hebben we wat testing gedaan met Sass.

✓ Angular

Ter herhaling hebben we verschillende video's bekeken. Hier en daar hebben we ook een beetje getest.

✓ .NET 7 & C#

We hebben verschillende filmpjes bekeken om een korte herhaling te krijgen omtrent de werking van een .NET API en de programmeertaal C#.

✓ Entity Framework

We hebben verschillende filmpjes bekeken om een korte herhaling te krijgen omtrent het Entity Framework die ons kunnen helpen wanneer we via onze .NET API met onze databank moeten connecteren.

✓ Mocking & Unit Testing

Herhaling van Mocking & Unit Testing.

✓ SQL

Korte herhaling van het schrijven van SQL-scripts.

10 Conclusie

Dit is het einde van ons verslag in verband met onderzoek & analyse. Het onderzoek dat we verricht hebben heeft dus voornamelijk plaatsgevonden gedurende de eerste twee weken van onze stage. Nadien hebben we deze steeds aangevuld met de laatste informatie. Ons onderzoek is begonnen met het verkrijgen van een beter beeld omtrent de huidige werking van het certificatenbeheerproces, in combinatie met de mogelijke verbeteringen die wij zouden kunnen maken. Verder hebben we onder andere een prototype opgesteld en besproken met de stagebegeleider tot in hoeverre dit prototype overeenkomt met de werkelijke verwachtingen. Dit prototype konden we dan gebruiken als leidraad voor het ontwerpen van onze applicatie. Verder hebben we onderzoek gedaan naar de verschillende technologieën waar wij mogelijk gebruik van wouden maken tijdens ons project. Veel van deze technologieën hebben we onderzocht aan de hand van filmpjes. Voor zaken zoals SCSS hebben we voornamelijk zelf dingen uitgetest. Ook RxJS is een belangrijk onderdeel van deze onderzoeken geweest. Dankzij deze onderzoeken hebben we namelijk veel meer best practices uit kunnen voeren.

Hieronder staan de belangrijkste zaken die we onderzocht hebben nog eens kort vermeld.

- ✓ De concrete opdrachtbeschrijving in combinatie met de huidige werking van het certificatenbeheerproces en de mogelijke verbeteringen ervan
- ✓ Algemene keuzes
  - Angular
  - .NET 7
  - SCSS
  - SQL Server
  - Remmicom Packages
    - styling, componenten
- ✓ Project management tool
  - DevOps-omgeving
- ✓ Eisenanalyse
  - UC-diagram met beschrijving van de belangrijkste systeemeisen
  - Prototype om een beter beeld te krijgen van hoe alles eruit zal zien
  - ERD ter voorstelling van onze databank
  - Authenticatie & autorisatie, wie mag wat doen
- ✓ Terminologie van certificaten en IDP
- ✓ Technische onderzoeken zoals RxJS

Geschreven door:
Brent Verlinden & Nicholas Coorevits

11 Bronvermelding

1. *Online Courses, Learning Paths, and Certifications - Pluralsight*. (z.d.). https://www.pluralsight.com/
2. *RxJS in Angular: Reactive Development | Pluralsight*. (z.d.). https://app.pluralsight.com/course-player?clipId=33fa18ab-2cd1-482c-b0f1-3193abfdd778
3. *RxJS*. (n.d.). https://rxjs.dev/api