topicus

Van Wiechen Module

Scriptie

Versie	1.1
Auteur	Liam Schippers, 439431
Datum	16 januari 2022
Onderwijsinstelling	Saxion Hogeschool, Enschede
Opleiding	HBO-ICT, Software Engineering
Afstudeerbegeleider	Peter Ebben
Bedrijfsbegeleider	Thijs Smeenk

Versiehistorie

Versie	Datum	Auteur	Wijzigingen
0.1	27-10-2021	Liam Schippers	Eerste opstelling
0.2	21-11-2021	Liam Schippers	Onderzoeken verwerkt
1.0	19-12-2021	Liam Schippers	Conceptversie
1.1	16-01-2022	Liam Schippers	Definitieve versie

Samenvatting

Voor de opleiding HBO-ICT van Saxion te Enschede is een afstudeeropdracht uitgevoerd binnen Topicus. Deze opdracht gaat over het bouwen van een module, die werkt binnen en buiten al bestaande applicaties van Topicus, waar ouders het Van Wiechenonderzoek in kunnen zien en kunnen invullen. Het Van Wiechenonderzoek is een onderzoek dat de ontwikkelingen volgt van kinderen van 0 - 4 jaar.

Huidige Situatie
Het huidige Van Wiechen zit verweven in het huidige KD+, het Digitale Dossier Jeugdgezondheidszorg (DD-JGZ) dat Topicus zelf ontwikkeld heeft. Wanneer een zorgprofessional het Van Wiechenonderzoek invult zal dit weggeschreven worden naar de database. Wanneer een zorgprofessional de resultaten van het Van Wiechenonderzoek wil inzien worden deze weer opgehaald uit de database en in het KD+ getoond.

Onderzoek
De hoofdvraag van deze afstudeeropdracht luidt: '*Hoe kan het Van Wiechen onafhankelijker gemaakt worden van het KD+ en Mijn Kinddossier?*'. Er zijn verschillende onderzoeken gedaan naar hoe het Van Wiechen onafhankelijker gemaakt kan worden. Er werdt gekeken hoe de Van Wiechen module eventueel op een cloud native manier geïmplementeerd kan worden binnen het applicatie landschap van Topicus. Hierbij moest wel rekening gehouden worden met de wetgevingen van de algemene verordening gegevensbescherming (AVG) en de wet aanvullende bepaling verwerking persoonsgegevens inde zorg (Wabvpz).

Conclusie
Uit het proof of concept is gebleken dat het niet alleen mogelijk is, maar ook haalbaar om een Van Wiechen module te bouwen die niet alleen AVG technisch klopt, maar ook binnen en buiten de huidige applicaties, het KD+ en Mijn Kinddossier, werkt. Dit is behaald door het toepassen van concepten zoals cloud native en web componenten. Doordat het proof of concept ook is opgebouwd met behulp van de technieken die Topicus zelf gebruikt is de kans op acceptatie en doorontwikkeling door Topicus zelf vergroot.

Inhoudsopgave

Inleiding

Voor de opleiding HBO-ICT van Saxion Enschede wordt er een afstudeeropdracht uitgevoerd door Liam Schippers voor het ICT-bedrijf Topicus. Dit document is de scriptie voor de afstudeeropdracht die is uitgevoerd binnen een tijdsbestek van 5 maanden. Deze opdracht gaat over het ontwikkelen van een module voor het Van Wiechenonderzoek voor bij het KD+. Het Van Wiechenonderzoek is onderzoek dat wordt uitgevoerd om de ontwikkeling van kinderen tussen de 0 en 4 jaar te volgen. Het KD+ is een Digitaal Dossier Jeugdgezondheidszorg ontwikkeld door Topicus. In dit document wordt onder andere ingegaan op het probleem, onderzoek, proces en behaalde resultaten.

1. Projectdefinitie

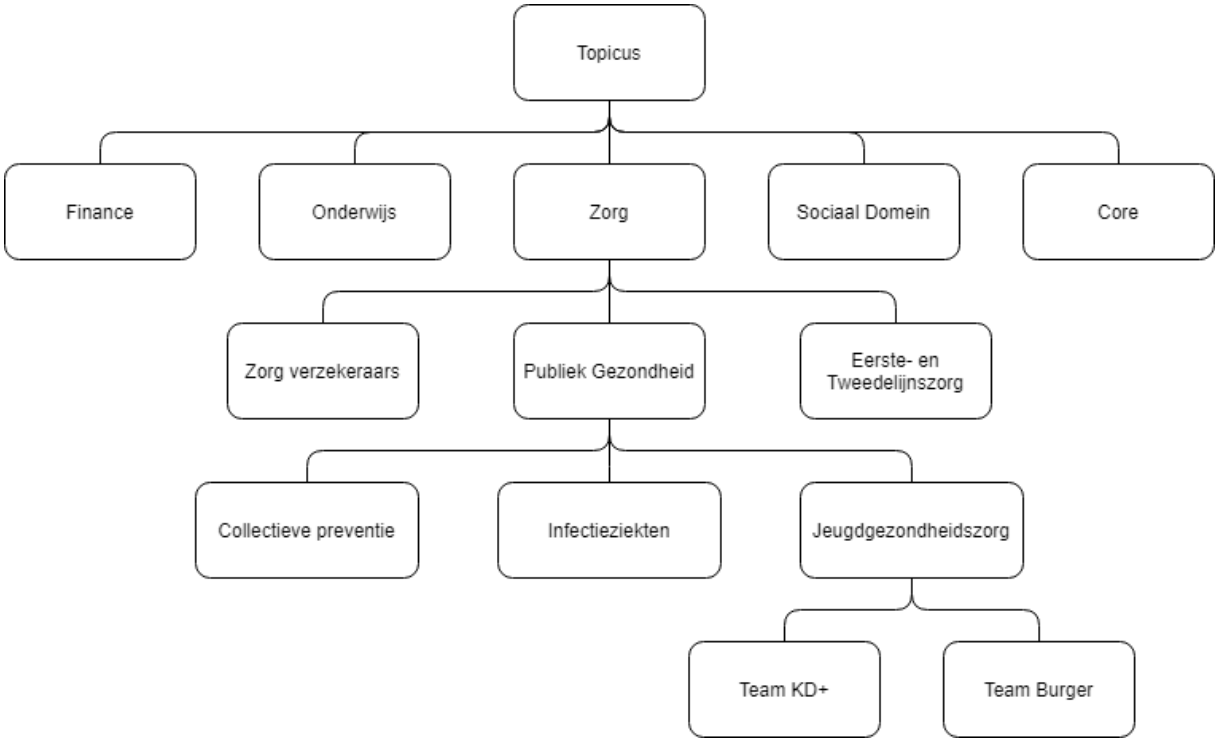
1.1. Het bedrijf

Topicus is een IT-bedrijf dat ontstaan is in Deventer. Topicus houdt zich bezig met het ontwikkelen van IT-oplossingen om met behulp van hun oplossingen optimale zorg, duidelijkheid en inzicht in financiën en betrouwbaar en kwalitatief onderwijs te leveren. Met behulp van hun IT-oplossingen streeft Topicus ernaar om het leven van mensen op zowel persoonlijk als professioneel gebied te verbeteren. Om deze doelen te bereiken creëert Topicus platformen waarin mensen, data en systemen met elkaar verbonden worden. Tegenwoordig telt Topicus meer dan 1.000 werknemers die verdeeld zijn over vijf verschillende divisies, namelijk Finance, Onderwijs, Zorg, Sociaal domein en Core.

De opdracht zal plaatsvinden binnen de divisie Zorg van Topicus. Divisie Zorg streeft naar optimale zorg. Dat betekent goede, toegankelijke maar ook betaalbare zorg voor patiënten, zorgverleners en mantelzorgers. Verder bestaat een divisie uit verschillende Business Lines. De Business Lines waaruit divisie Zorg bestaat zijn Zorgverzekeraars, Publieke Gezondheid en Eerste- en Tweedelijnszorg. Binnen de Business Line Publieke Gezondheid zal de opdracht zich bevinden.

Binnen de verschillende Business Lines houden verschillende multidisciplinaire teams zich bezig met verschillende projecten. De Business Line Publieke Gezondheid houdt zich bezig met collectieve preventie, infectieziekten en de jeugdgezondheidszorg. Deze opdracht valt dan binnen het project Jeugdgezondheidszorg.

Binnen het project Jeugdgezondheidszorg houden verschillende teams zich bezig met het ontwikkelen van software voor in de jeugdgezondheidszorg. Hieronder vallen twee ontwikkelteams, namelijk team KD+ en team Burger. Team KD+ houdt zich bezig met het ontwikkelen van KD+, het Digitale Dossier Jeugdgezondheidszorg (DD JGZ) dat Topicus aanbiedt. Team Burger houdt zich voornamelijk bezig met het Mijn Kinddossier, het ouder portaal waarmee ouders het dossier van hun kind kunnen inzien.



Figuur 1: Versimpeld Organogram Topicus

1.2. Context

Jeugdgezondheidszorg (JGZ)

De jeugdgezondheidszorg is de publieke gezondheidszorg voor kinderen van 0-19 jaar. Het is een wettelijke taak met als doel het bevorderen, beschermen en beveiligen van de gezondheid en de lichamelijke en geestelijke ontwikkeling van jeugdigen. Jeugdartsen en jeugdverpleegkundigen volgen de groei en ontwikkelingen van kinderen tijdens contact momenten. Zo kan de JGZ bij alle kinderen tijdig problemen signaleren en jeugdigen begeleiden of doorverwijzen. Voor het uitvoeren van de JGZ wordt een Digitaal Dossier Jeugdgezondheidszorg gebruikt.

Digitale Dossier Jeugdgezondheidszorg (DD JGZ)

Alle kinderen in Nederland in de leeftijd van 0 tot 19 jaar krijgen sinds 2010 een digitaal medisch dossier, namelijk het Digitale Dossier Jeugdgezondheidszorg (DD JGZ). In dit dossier bevordert de JGZ de ontwikkeling en gezondheid van alle kinderen. Het DD JGZ heeft verschillende voordelen, ten eerste is het beter leesbaar dan het papieren dossier. Verder is een digitaal dossier gemakkelijker over te dragen naar een andere JGZ-organisatie. (Digitaal Dossier Jeugdgezondheidszorg, n.d.)

KD+

Het KD+ is het DD JGZ systeem van Topicus zelf dat draait op productie sinds 2009. Het KD+ is zo ontworpen dat het een digitale versie is van het papieren dossier. Het KD+ is opgebouwd uit verschillende modules die op verzoek toegevoegd of verwijderd kunnen worden. Met behulp van dit dossier wordt zo de administratieve last geminimaliseerd om zo meer tijd te hebben voor de zorg (Topicus, 2021).

Mijn Kinddossier

Het Mijn Kinddossier is het ouderportaal dat aansluit op het KD+. Hiermee kan de ouder van het kind een inzicht krijgen in de ontwikkelingen van hun kind. Ook kunnen ouders afspraken plannen en wijzigen. Tot slot kunnen ouders actief deelnemen in het zorgtraject van hun kind. Dit doen ze dan door onder anderen zelf de ontwikkelingen van hun kind te volgen en voorafgaand een afspraak een digitale vragenlijst in te vullen (Topicus, 2021).

Van Wiechenonderzoek

Het Van Wiechenonderzoek is een onderzoek dat wordt uitgevoerd om de ontwikkeling van kinderen tussen de 0 en 4 jaar te volgen. Het onderzoek vindt plaats op het consultatiebureau. Het onderzoek bestaat uit meer dan 70 kenmerken die door een zorgprofessional getest en geregistreerd moeten worden.
De kenmerken zijn gericht op fijne motoriek, adaptie, taalontwikkeling, sociaal gedrag en persoonlijkheid. Een voorbeeld van een van deze kenmerken is blokjes stapelen, deze wordt pas bij een kind van anderhalf jaar getest. Bij dit kenmerk wordt gekeken of het kind blokjes kan stapelen.

De verschillende Van Wiechen-kenmerken worden afgenomen op de leeftijd die het beste past bij de 0-90 leeftijd, de leeftijd waarop 90% van de kinderen een positief scoort voor het kenmerk. Elk kenmerk moet in eerste instantie via eigen waarneming van de onderzoeker worden onderzocht. De onderzoeker zal dan een positief of negatief noteren voor het onderzochte kenmerk. Wanneer dit niet lukt, wordt bij sommige kenmerken op heteroanamnese overgegaan. Dit betekent dat de ouder dan aangeeft of ze het kenmerk hebben waargenomen. (Jeugdgezondheid, 2021)

Naast het volgen van de ontwikkeling van het kind is het onderzoek ook ondersteunend bij de gesprekken over de ontwikkeling van het kind tussen ouder en zorgprofessional. Tot slot is het doel van het Van Wiechenonderzoek om bij elk kind de eerste vier levensjaren het tempo en kwaliteit van de psychomotorische en neurologische ontwikkeling te volgen, ontwikkelingsstoornissen vroegtijdig op te sporen en eventuele verwijzingen te ondersteunen. (Jeugdinstituut, 2021)

1.3. Aanleiding

Binnen de jeugdgezondheidszorg (JGZ) wordt het Van Wiechenonderzoek afgenomen om zo de ontwikkelingen van jonge kinderen tussen de 0 en 4 jaar te volgen. Met behulp van het Van Wiechenonderzoek kunnen achterstanden in de ontwikkeling van het kind vroegtijdig worden opgespoord en indien nodig kan het kind worden verwezen.

Nu blijkt uit een TNO-onderzoek dat 92% van de ouders plezier heeft in het afnemen van de kenmerken van het Van Wiechenonderzoek. Uit dit onderzoek is de aanbeveling gedaan om instructievideo's te ontwikkelen voor de Van Wiechen-kenmerken, om zo de uitvoering en beoordeling van de kenmerken te verduidelijken. Met behulp van deze video's kunnen ouders voorafgaand aan een consult naar de ontwikkeling van hun kind kijken. Dit helpt ouders ook meer betrokken te zijn bij het onderzoek en de ontwikkeling van het kind. (TNO, 2018)

Nu blijkt uit de praktijk dat vanwege de coronamaatregelen er een directe behoefte is aan de Van Wiechenfilmpjes. Hoewel de filmpjes niet ontwikkeld zijn voor het vervangen van de beoordeling van de professional bieden ze ondersteuning voor de JGZ om zo toch de ontwikkeling van kinderen in zicht te houden. Nu is er, vanwege de coronamaatregelen, de ondersteuning van ouders benodigd om de ontwikkeling van kinderen op afstand te volgen. (Van Zoonen, et al., 2021)

Topicus wil nu graag een tool ontwikkelen voor hun applicatie dat ouders de mogelijkheid geeft om het Van Wiechenonderzoek zelf af te nemen. Ze zoeken niet naar een tool dat de zorgprofessional vervangt met het afnemen van het onderzoek, maar een tool die de zorgprofessional ondersteunt. Niet alleen hebben ouders belang bij deze tool, maar ook zorgprofessionals om zo de ontwikkelingen van een kind op afstand beter te kunnen volgen.

Het Van Wiechen Do It Yourself is een uitbreiding op het Van Wiechenonderzoek dat Topicus graag in hun applicaties wil realiseren. Het doel is om ouders meer betrokken te hebben bij de ontwikkeling van hun kind en om zo ook meer ondersteuning te bieden aan de zorgverleners als het om de ontwikkeling van het kind gaat.

1.4. Probleemstelling

Het huidige KD+ is zo ontworpen dat het een digitale versie is van het papieren dossier. Dit betekent ook dat het Van Wiechenonderzoek een één op één kopie is van de papieren versie. Deze vorm van het Van Wiechenonderzoek is dan ook alleen beschikbaar voor de zorgprofessional. Hierdoor hebben ouders niet de mogelijkheid om zelf de kenmerken van het Van Wiechenonderzoek af te nemen. Ook kunnen ouders niet de resultaten die voortkomen uit het Van Wiechenonderzoek inzien.

Ouders kunnen resultaten niet inzien

Alleen de zorgprofessionals kunnen de resultaten van het Van Wiechenonderzoek inzien. De ouders van het kind hebben geen inzicht in de resultaten die voortkomen uit het onderzoek. Verder hebben ouders niet alleen belang bij het inzien van de resultaten, wat blijkt uit een intern onderzoek van Topicus zelf, maar ook hebben zij recht op de inzage. Tot slot helpt dit ook JGZ-organisaties transpanter te zijn tegenover de ouders van het kind.

Niet genoeg flexibiliteit JGZ

Op het moment zijn er 14 momenten benodigd voor het afnemen van het Van Wiechenonderzoek. Zes van deze momenten vallen vaak niet samen met een contactmoment. Hierdoor zijn er niet genoeg contactmomenten voor de zorgprofessional om het Van Wiechenonderzoek af te nemen. Wanneer ouders het Van Wiechenonderzoek zelf in kunnen vullen kan de JGZ flexibeler te werk gaan en meer zorg op maat bieden voor het kind.

Ouder te afhankelijk van JGZ

Het komt vaak voor dat tijdens het uitvoeren van het Van Wiechenonderzoek, tijdens een contactmoment, het kind niet mee wil werken. Hierdoor is het vaak het woord van de ouder waar de professional op moet vertrouwen. Wanneer een kenmerk niet geconstateerd kan worden tijdens het consult wordt de ouder gevraagd of ze het kenmerk geconstateerd heeft. Hierdoor is een ouder altijd afhankelijk van de JGZ voor het afnemen van de Van Wiechenkenmerken en zou pas bij een contactmoment geconstateerd kunnen worden of er iets mis is met de ontwikkeling van het kind.

Complexiteit

Het KD+ bestaat al meer dan tien jaar en door de jaren heen is er veel data aan toegevoegd. Verder is het KD+ gebouwd met verouderde technieken, bestaat uit diverse

modulen en meer dan 800.000 regels aan code. Hierdoor is het niet gewenst om het KD+ uit te breiden met nieuwe functionaliteiten met behulp van de huidige technieken.

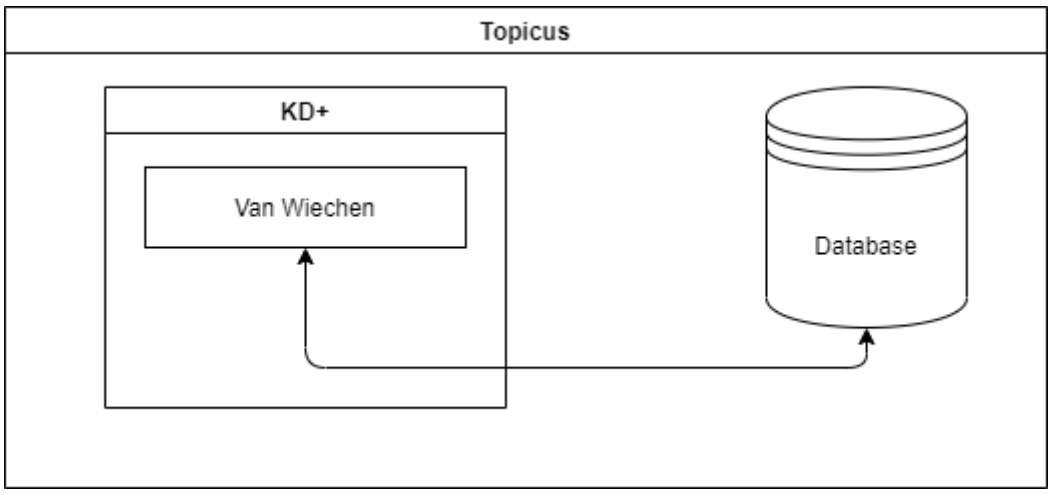
2. Project Doelstelling

Het doel van het project is om het Van Wiechen onafhankelijker te maken van het huidige KD+ en Mijn Kinddossier. Dit wordt bereikt door het opstellen van een aparte Van Wiechen module, die zowel binnen als los van het KD+ en Mijn Kinddossier kan draaien. Met behulp van deze nieuwe module zouden ouders ook de mogelijkheid krijgen om het Van Wiechenonderzoek zelf in te vullen en de resultaten van het Van Wiechenonderzoek in te zien.

De voordelen die hieruit voortkomen zijn dat ouders meer betrokken zullen zijn bij de ontwikkeling van hun kind. Ook helpt dit de JGZ transparanter te zijn tegenover de ouders. Tot slot wordt zo ook de oude codebase zo min mogelijk belast.

2.1. Huidige situatie

Om een zicht te krijgen op de huidige situatie is er een versimpeld diagram geschetst, zie figuur 2. Het huidige Van Wiechen zit verweven in het huidige KD+. Wanneer een zorgprofessional het Van Wiechenonderzoek invult zal dit weggeschreven worden naar de database. Wanneer een zorgprofessional de resultaten van het Van Wiechenonderzoek wil inzien worden deze weer opgehaald uit de database en in het KD+ getoond.



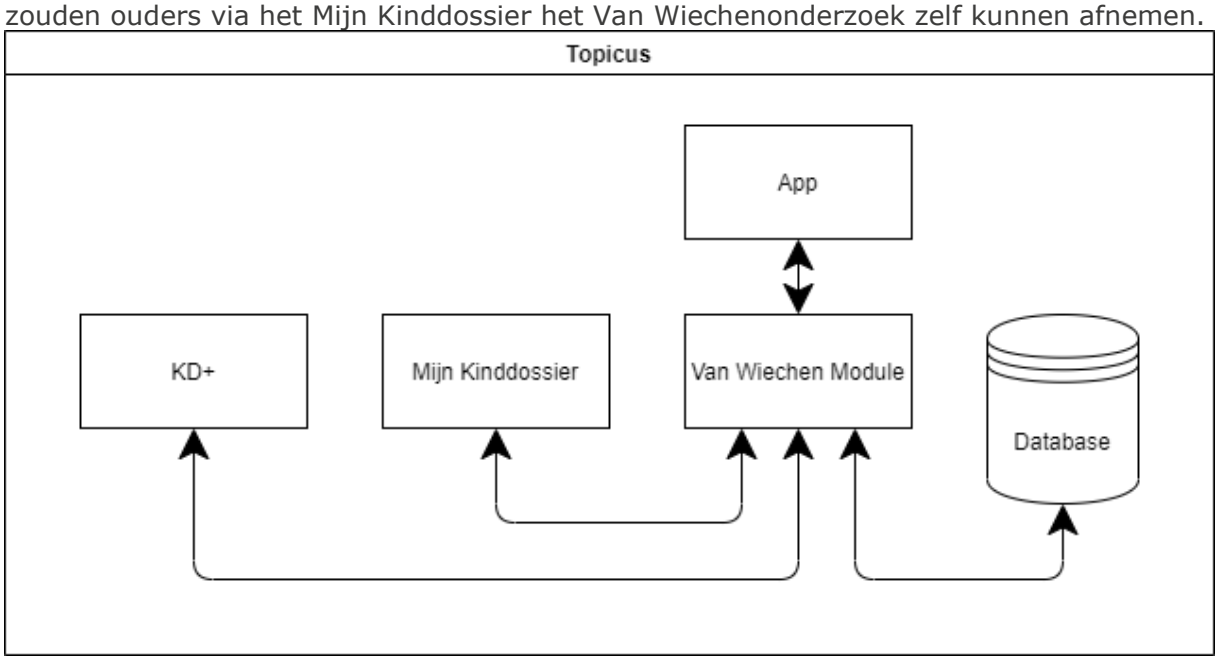
Figuur 2: Diagram Huidige implementatie

2.2. Gewenste situatie

Het Van Wiechen dat in het huidige KD+ zit wordt eruit getrokken en een eigen module van gemaakt. De nieuwe module zal dan weer communiceren met het KD+ en Mijn Kinddossier. Deze module zou dan ook eventueel verbinden met een aparte app, zodat het Van Wiechen ook als apart product zou kunnen worden aangeboden.

De nieuwe module zal dan bestaan uit een front-end en een backend. De backend zal dan de informatie verwerken en zorgt ervoor dat de juiste informatie bij de juiste applicatie in te zien is. Het front-end dat erbij hoort zal, naast Mijn Kinddossier en KD+, ook een mogelijkheid bieden voor ouders om het Van Wiechenonderzoek in te vullen. Het is gewenst dat het front-end dan mobile first gebouwd wordt. Dit houdt in dat de ontwerp focus op mobiel ligt en dan gaat doorontwerpen naar tablet of desktop.

In de gewenste situatie moeten zorgprofessionals via het KD+ nog steeds het Van Wiechenonderzoek kunnen afnemen en de resultaten in kunnen zien. Verder moeten ze ook de resultaten in kunnen zien die ouders hebben ingevuld. Ook moeten ouders via het Mijn Kinddossier de resultaten van het Van Wiechenonderzoek in kunnen zien. Tot slot zouden ouders via het Mijn Kinddossier het Van Wiechenonderzoek zelf kunnen afnemen.



Figuur 3: Gewenste situatie

3. Onderzoek

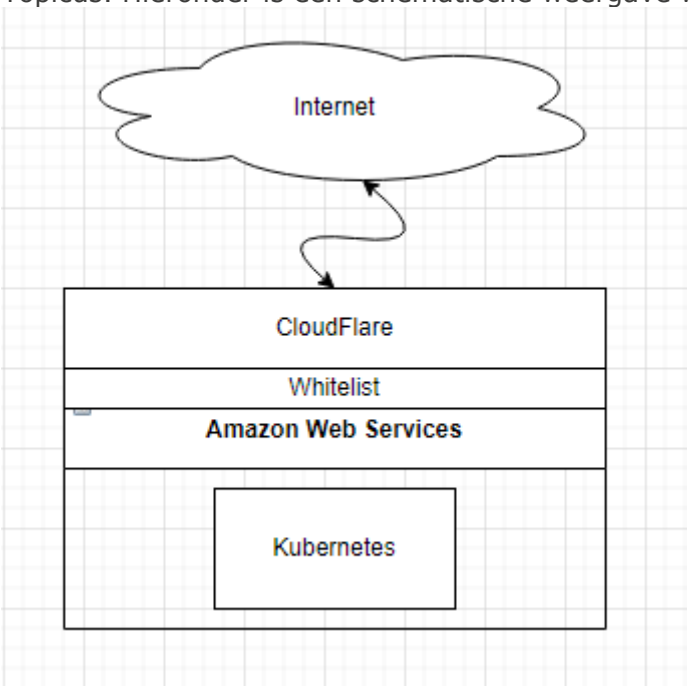
In dit hoofdstuk worden de resultaten die zijn voortgekomen tijdens het onderzoek aan het licht gebracht.

3.1. Huidige situatie

Om een beeld te krijgen van wat er al bestaat en hoe het werkt zal er onderzoek gedaan moeten worden naar hoe de huidige situatie is bij Topicus Jeugdgezondheidszorg (JGZ) en hoe het Van Wiechen geïmplementeerd is.

3.1.1. Hoe ziet de huidige architectuur eruit?

De verschillende applicaties van Topicus JGZ draaien in Kubernetes clusters binnen Amazon Web Service (AWS). Vanaf het internet zal het verkeer via CloudFlare binnenkomen en dan via een Whitelist zal het verkeer uiteindelijk bij het AWS komen van Topicus. Hieronder is een schematische weergave weergegeven van deze architectuur.



Figuur 4: Schematische Weergave Architectuur

CloudFlare

Cloudflare is een bedrijf dat Content Delivery Network (CDN), Domain Name System (DNS) en distibuted denial-of-service (DDoS) bescherming biedt (CloudFlare, 2021). Binnen de architectuur van Topicus zit de CouldFlare service tussen hun applicaties en

het internet. Gebruikers zullen altijd via CloudFlare met de applicaties van Topicus verbinden.
Een CDN refereert naar een geografisch gedistribueerde groep aan servers die samenwerken om snelle levering van internetcontent te bieden (CloudFlare, 2021).
De DNS is een systeem en netwerkprotocol dat een webadres omzet naar een Internet Protocol (IP) adres.

Whitelist
Tussen CloudFlare en AWS zit nog een Whitelist. De Whitelist geeft alleen IP-adressen toegang die via CloudFlare zijn binnengekomen tot de applicaties die binnen het AWS draaien. Zo houdt Topicus ongewenst netwerkverkeer buiten de deur.

Amazon Web Services (AWS)
Amazon Web Services is een cloud computing platform dat wordt aangeboden door Amazon. Cloud computing is het on-demand leveren van computer recources. Sinds 2021 bestaat AWS uit meer dan 200 services en producten (Wikipedia, Amazon Web Services, 2021). Topicus maakt gebruik van Amazon Web Services (AWS) om hun applicaties te beheren en te hosten.
Binnen AWS wordt er verder nog gebruikt gemaakt van verschillende services die worden aangeboden vanuit Amazon. Deze Services zijn Amazon Elastic Kubernetes Service (EKS), Amazon Relational Database Service (RDS) en Amazon Simple Storage Service (S3).

Amazon Elastic Kubernetes Service (EKS)
Topicus maakt gebruik van Amazon Elastic Kubernetes Service (EKS) de Amazon implementatie van Kubernetes. Kubernetes is een open-source container orchestration systeem voor het automatiseren van deployen, schalen en managen van computerapplicaties. Kubernetes werkt met verschillende container tools en draaien containers in een cluster, vaak met behulp van Docker Images. (Wikipedia, Kubernetes, 2021).

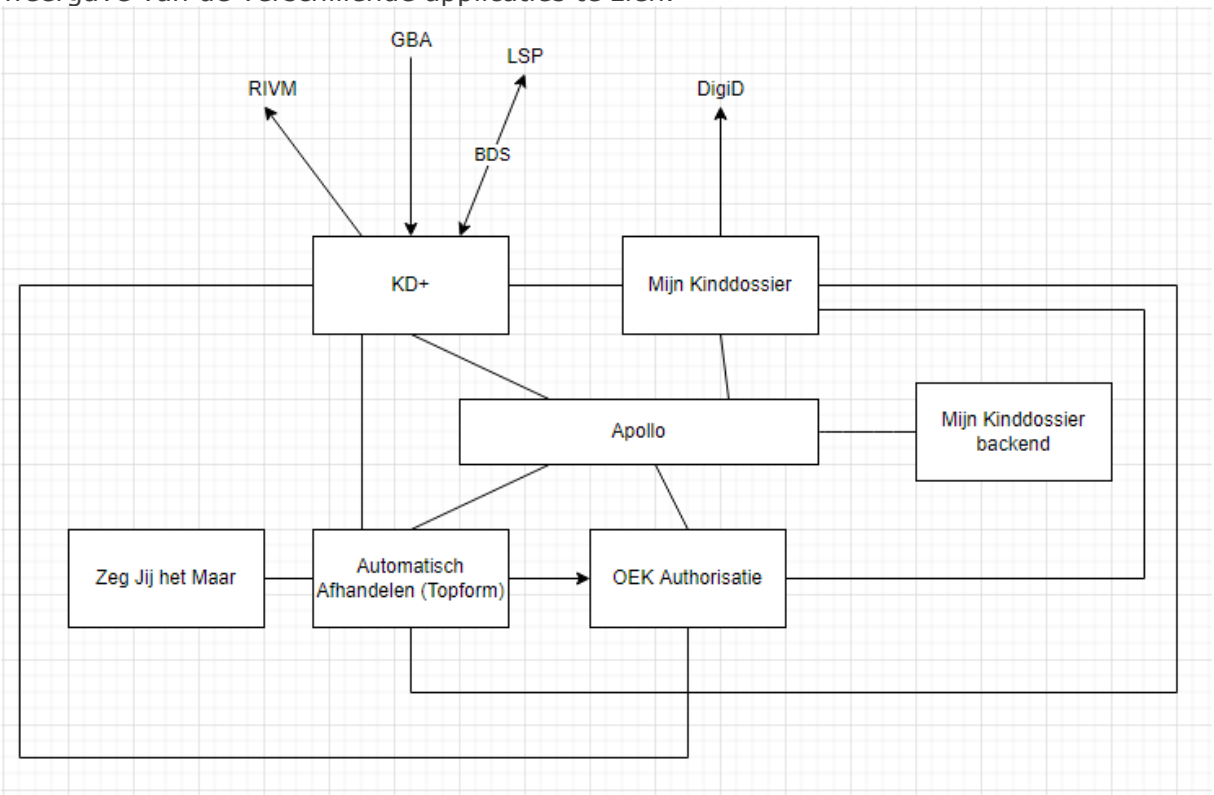
Amazon Relational Database Service (RDS)
Amazon Relational Database Service (RDS) is een gedistribueerde relationale database service aangeboden door AWS (Wikipedia, Amazon Relational Database Service, 2021).
Topicus maakt gebruik van het RDS voor het opslaan van data dat verwerkt wordt door hun applicaties.

Amazon Simple Storage Service (S3)

Amazon Simple Storage System Service (S3) is een service aangeboden door Amazon, met behulp van AWS, voor object storage (Wikipedia, Amazon S3, 2021). Topicus maakt gebruik van S3 voor file storage binnen hun applicaties.

3.1.2. Welke applicaties zijn er binnen de Topicus JGZ applicatielandschap?

Binnen Topicus JGZ worden verschillende applicaties ontwikkeld en onderhouden. Het is van belang om hier inzicht in te hebben, aangezien het Van Wiechen dat gebouwd moet worden ook met deze applicaties te maken krijgt. Hieronder is een schematische weergave van de verschillende applicaties te zien.



Figuur 5: Schematische weergave applicatielandschap

KD+

Het KD+ is het DD JGZ systeem van Topicus zelf, dat op productie draait sinds 2009. Het KD+ is zo ontworpen dat het een digitale versie is van het papieren dossier. Het KD+ is opgebouwd uit verschillende modules die op verzoek toegevoegd of verwijderd kunnen worden. Met behulp van dit dossier wordt zo de administratieve last geminimaliseerd om zo meer tijd te hebben voor de zorg (Topicus, 2021).

Mijn Kinddossier

Het Mijn Kinddossier is het ouderportaal dat aansluit op het KD+. Hiermee kan de ouder van het kind een inzicht krijgen in de ontwikkelingen van hun kind. Ook kunnen ouders afspraken plannen en wijzigen. Tot slot kunnen ouders er actief deelnemen in het zorgtraject van hun kind. Dit doen ze dan door onder andere zelf de ontwikkelingen van hun kind te volgen en voorafgaand aan een afspraak een digitale vragenlijst in te vullen (Topicus, 2021).

Apollo

Apollo is een data gateway voor het JGZ. Het Mijn Kinddossier verbindt met Apollo om van verschillende databronnen informatie op te halen, waaronder het KD+, Mijn Kinddossier Backend, Oek Authorisatie en Topform.

Mijn Kinddossier backend

Het Mijn Kinddossier Backend is de backend voor het mijn kinddossier.

Zeg Jij Het Maar

Met behulp van Zeg Jij het Maar wordt de mogelijkheid geboden om digitaal vragenlijsten af te nemen. Met behulp van een unieke inlogcode in combinatie met de geboortedatum kan er in het systeem worden ingelogd en de vragenlijst worden ingevuld. Wanneer een vragenlijst wordt afgerond worden de ingevulde antwoorden verstuurd naar het DD JGZ KD+. In het KD+ zijn ze vervolgens in te zien voor de zorgprofessional.

Topform

Binnen de TopForm applicatie(s) kunnen JGZ-organisaties nieuwe vragenlijsten aanmaken en de logicaregels omtrent deze vragenlijsten definiëren. Vragenlijsten zijn vragenlijsten die wijzigen op basis van de antwoorden van diegene die de vragenlijst invult. JGZ-organisaties kunnen vervolgens vanuit hun applicatie, bijvoorbeeld KD+, de aangemaakte vragenlijsten opvragen, klaarzetten voor ouders of jongeren en deze kunnen de vragenlijsten invullen vanuit bijvoorbeeld Mijn Kinddossier (App) of Zeg Jij Het Maar.

OEK Authorisatie

De OEK Authorisatie is de authorisatie service.

Rijksinstituut voor Volksgezondheid en Milieu (RIVM)

Tussen het RIVM en het KD+ wordt er informatie uitgewisseld voor het Rijks Vaccinatie Programma (RVP)

Gemeentelijke basisadministratie persoonsgegevens (GBA)
Het GBA was de benaming voor de Nederlandse registratie van de administratieve levensloop van alle natuurlijke personen die woonachting in Nederlands waren. Vanuit het GBA kan het KD+ de persoonsgegevens ontvangen van personen die geregistreerd moeten worden in het KD+.

(Landelijk SchakelPunt) LSP
Het LSP regelt het verkeer van berichten tussen zorgaanbieders en zorgverleners. Als het nodig is voor een behandeling én als de patiënt toestemming heeft gegeven, kunnen zorgaanbieders via het LSP medische gegevens bekijken (Patiëntfederatie, 2021). Met behulp van de LSP-module kunnen dossiers overgedragen worden naar organisaties die geen gebruik maken van KD+.

Basisdataset (BDS)
De basisdataset wordt gebruik om informatie tussen het LSP en het KD+ uit te wisselen. Het BDS zorgt voor een uniforme registratie in de JGZ waardoor onderlinge uitwisselingen en vergelijking van gegevens mogelijk wordt.

3.1.3. Hoe zit het huidige Van Wiechen in het KD+?

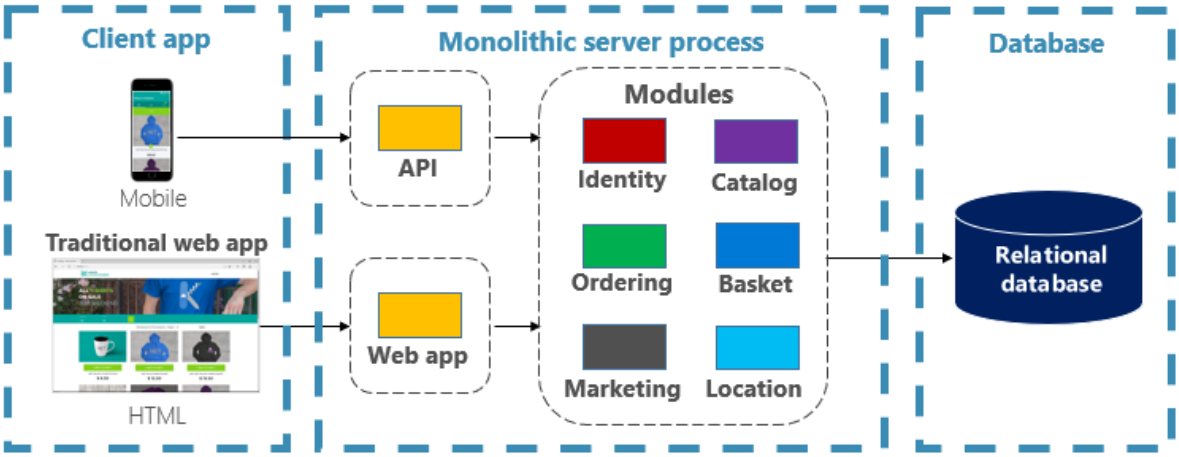
Het KD+ is opgebouwd met allemaal verschillende modules. Het Van Wiechen onderdeel van het KD+ zit verweven in de core module en het KD+. Dit betekent dat het Van Wiechen niet een eigen module is maar deel is van een andere module. Verder zijn het KD+ en Mijn Kinddossier allebei opgebouwd als monolitische applicaties. In figuur 7 is een voorbeeld te zien over hoe een monopolistische applicatie is opgebouwd.

Een monolithic architectuur, of manier van software ontwerpen is de traditionelere manier van het ontwerpen van software. Hierbij is de software samengesteld uit één stuk. Monolithic software is ontworpen om op zichzelf staand te zijn. Componenten van de software zijn dan onderling verbonden en afhankelijk van elkaar in plaats van losjes gekoppeld. In een strak gekoppelde architectuur, zoals de monolithic architectuur, moeten alle componenten aanwezig zijn voor de code om te werken. Zie figuur 4 voor een voorbeeld van een monolithic architectuur. (Wigmore, 2016)

Voordelen van een monolithic architectuur zijn bijvoorbeeld wel dat het een beter doorvoer heeft dan een modulaire aanpak. Verder kunnen ze ook makkelijker getest en gedebugd worden aangezien er minder variabelen zijn vanwege een minder aantal aan

elementen (Wigmore, 2016). Ook is een monolithic architectuur best recht toe recht aan te bouwen en te deployen. (Microsoft, 2021)

Nadelen van een monolithic architectuur zijn dat de software zo groot en complex wordt dat een enkel persoon niet de gehele codebase in zijn eentje kan begrijpen. Daarnaast kunnen aanpassingen aan de software ongewenste en kostelijke bijwerkingen hebben binnen de software. Nieuwe features en/of fixes kunnen kostelijk zijn om te implementeren. Bij elke release moet de gehele applicatie opnieuw gedeployed worden. Eén onstabiel component kan ervoor zorgen dat het gehele systeem crashed. Je zit vast aan de technologieën en frameworks die gebruikt zijn tijdens het bouwen. Het wordt lastiger om agile levering methodieken aan te houden. Architecturale erosie treedt in als de codebase verslechterd met nooit eindigende "special cases". Tot slot zul je op den duur een deel of de gehele code base moeten herschrijven. (Microsoft, 2021)



Figuur 6: Monolithic Architectuur (Microsoft, 2021)

3.2. Algemene verordening gegevensbescherming (AVG)

Sinds 25 mei 2018 is de Algemene verordening gegevensbescherming (AVG) van toepassing. Dit is een privacywetgeving die in de hele Europese Unie (EU) geldt (Persoonsgegevens, 2021). De AVG zorgt voor een gestandaardiseerde set aan regels voor verwerking van persoonsgegevens door particuliere bedrijven en overheidsinstanties in de hele EU. Het doel van de AVG is niet alleen om de bescherming van persoonsgegevens in de EU te garanderen, maar ook om het vrije verkeer van gegevens binnen de Europese interne markt te waarborgen. (Wikipedia, Algemene verordening gegevensbescherming, 2021)

3.2.1. Wat doet de Algemene verordening gegevensbescherming (AVG)?

De AVG biedt betere bescherming van de persoonsgegevens van burgers door middel van de volgende regels:
Transparantie: De persoon van wie de gegevens verwerkt worden, is hiervan op de hoogte, heeft hiervoor toestemming gegeven en kent zijn rechten
Gegevensbeperking: Alleen de noodzakelijke gegevens voor het leveren van product of dienst mogen gevraagd worden.
Doelbeperking: De ontvangen gegevens mogen alleen gebruikt worden voor een doel waar de burger de gegevens voor heeft verstrekt.
Juistheid: De persoonsgegevens moeten correct zijn en blijven.
Bewaarbeperking: De persoonsgegevens mogen niet langer bewaard worden dan nodig is voor het beoogde doel.
Integriteit en vertrouwelijkheid: De persoonsgegevens moeten beschermd worden tegen toegang voor onbevoegden, verlies of vernietiging.
Verantwoording: De verantwoordelijke moet kunnen aantonen aan deze regels te voldoen.

3.2.2. Wat zijn persoonsgegevens?

Persoonsgegevens zijn gegevens die aan een individu verbonden kunnen worden, of waarmee een individu geïdentificeerd kan worden. Voorbeelden hiervan zijn naam, foto, telefoonnummer, adres, bankrekeningnummer, e-mailadres, IP-adres, vingerafdruk en medische data. (Wikipedia, Algemene verordening gegevensbescherming, 2021)

3.2.3. Wanneer mogen persoonsgegevens verwerkt worden?

Aangezien het verwerken van persoonsgegevens een inbreuk is op de privacy van de mensen over wie het gaat, mogen persoonsgegevens alleen verwerkt worden als zonder deze gegevens het doel niet bereikt kan worden. Er is dus een goede reden nodig voor het verwerken van persoonsgegevens. In de AVG staan 6 redenen, ook wel juridisch grondslagen genoemd. Er is dus een grondslag nodig voor het verwerken van persoonsgegevens. Hieronder staan de 6 grondslagen die zijn verwerkt in de AVG.

U heeft toestemming van de persoon om wie het gaat.
Het is noodzakelijk om gegevens te verwerken om een overeenkomst uit te voeren
Het is noodzakelijk om gegevens te verwerken omdat u dit wettelijk verplicht bent.
Het is noodzakelijk om gegevens te verwerken om vitale belangen te beschermen.
Het is noodzakelijk om gegevens te verwerken om een taak van algemeen belang of openbaar gezag uit te oefenen.
Het is noodzakelijk om gegevens te verwerken om uw gerechtvaardigde belang te behartigen.

3.2.4. Zorgverlener en de AVG

Voor de zorgverlener komen er verschillende verplichtingen naar voren met de AVG. Een van deze plichten is de informatieverplichting, wat onder het kader transparantie valt. Informatieverplichting houdt in dat patiënten van tevoren recht hebben op heldere informatie over wat er met hun persoonsgegevens gedaan wordt en waarom. Verder moet de zorgverlener ook kunnen aantonen dat ze daadwerkelijk toestemming hebben gekregen van hun patiënt.
Ook zijn zorgverleners in veel gevallen verplicht om een register van verwerkingsactiviteiten bij te houden; een Data Protection Impact Assessment (DPIA) uit te voeren; functionaris voor de gegevensbescherming (FG) aan te stellen.

3.2.5. Wat zijn de principes waarmee AVG-compliance bereikt kan worden?

Er zijn twee principes om persoonsgegevens te beschermen. Deze twee zijn privacy by design en privacy by default. Privacy by design is dat in de ontwerpfase al rekening wordt gehouden met de persoonsgegevens beschermen. Een voorbeeld hiervan is om persoonlijke identifiers om te zetten naar artificial identifiers. Het andere principe is privacy by default. Dit houdt in dat de standaard privacy instelling de privacy van de persoon respecteren totdat de persoon zelf toegang geeft. Een voorbeeld hiervan is dat in een (web)formulier geen vakjes al van tevoren zijn aangevinkt. (Breejen, 2021)

3.2.6. Wat valt buiten de scope van de opdracht?

Buiten de scope van de opdracht valt het bepalen van de grondslag aangezien die al is vastgesteld. Aangezien de jeugdgezondheidszorg een wettelijke taak valt hij onder de grondslag, omdat het wettelijk verplicht is.
Wat verder nog meer buiten de scope van de opdracht valt is:

- De informatieverplichting van de zorgverlener
- Het bijhouden van het register van verwerkingsactiviteiten
- Data Protection Impact Assessment (DPIA) uitvoeren
- Functionaris voor de gegevensbescherming (FG) aan te stellen

3.2.7. Conclusie

In conclusie mag er volgens de Algemene verordening gegevensbescherming persoonsgegevens verwerkt worden, omdat de jeugdgezondheidszorg een wettelijke taak is. Wel moeten de persoonsgegevens die verwerkt worden zo min mogelijk zijn. Alleen de persoonsgegevens die benodigd zijn om het Van Wiechenonderzoek uit te kunnen voeren mogen verwerkt worden. Daarnaast moeten deze persoonsgegevens beschermd worden. Verder zal tijdens het ontwerpen van de software rekening gehouden moeten worden met privacy by design en privacy by default.

3.3. Wet aanvullende bepaling verwerking persoonsgegevens in de zorg (Wabvpz)

In de Wet aanvullende bepaling verwerking persoonsgegevens in de zorg (Wabvpz) wordt het gebruik van het Burgerservicenummer (BSN) en bepaalde vormen van elektronische uitwisseling van medische gegevens tussen zorgverleners geregeld. Daarnaast geeft deze wet patiënten het recht om hun dossier elektronisch in te mogen kijken of er een afschrift van te ontvangen. Tenslotte zijn in de Wabpvz ook artikelen opgenomen over de logging. (AVG-Helpdesk, 2021)

3.3.1. Wanneer mag je gebruik maken van het BSN?

In de Wabpvz staat dat een BSN alleen verwerkt mag worden als dit wettelijk is toegestaan. Dit houdt in dat je BSN mag worden gebruikt in de administratie om te controleren of de patiënt wel de persoon is wie die beweert te zijn. Wanneer is vastgesteld dat het BSN en de bijbehorende persoonsgegevens correct zijn wordt het BSN gebruikt in het uitwisselen van medische gegevens tussen. (Rijksoverheid, 2021)

3.3.2. Welke regels gelden er rond het uitwisselen van medische gegevens?

Volgens de Wabpvz mag een zorgaanbieder niet zomaar de medische gegevens beschikbaar stellen voor een andere zorgaanbieder. De zorgaanbieder moet eerst vastleggen of de client hier toestemming voor heeft gegeven. Tot slot moet een zorgaanbieder verifiëren of de persoonsgegevens correct zijn met het BSN. Tijdens deze verificatie moeten ze ook kijken of de identiteit van de client overeenkomt.

3.3.3. Wanneer mag een Medisch dossier worden ingezien?

De Wabpvz geeft patiënten het inzage recht om hun dossier in te zien. De vorm waarop de inzage kan worden verleend aan de patiënt wordt hier niet expliciet beschreven. Er zijn verschillende manier van inzage die toegestaan zijn, waaronder een PDF aan de patiënt leveren, de patiënt via een portaal het dossier laten inzien. Zelfs inzage op een scherm bieden bij de zorgaanbieder behoort tot de mogelijkheden. ((Elektronisch) inzage verlenen, 2021)
Binnen het DD JGZ hebben gezaghebbende ouder(s) het recht van inzage. Voordat een ouder het dossier mag inzien moet deze ouder eerst geïdentificeerd zijn, dit is om na te gaan of de ouder daadwerkelijk de gezaghebbende ouder is die in het DD JGZ geregistreerd staat. (Nederlands Centrum Jeugdgezondheid, 2020)

3.3.4. Welke logging moet er plaats vinden?

Volgens de Wabpvz moet er bijgehouden worden wie medische informatie uit het dossier beschikbaar gesteld heeft, en dan via een elektronisch uitwisselingssysteem, hierbij moet ook de datum vermeld worden. Daarnaast moet er bijgehouden worden wie bepaalde informatie heeft ingezien of opgevraagd, hierbij moet ook de datum vermeld worden. (Beatrix, 2020)

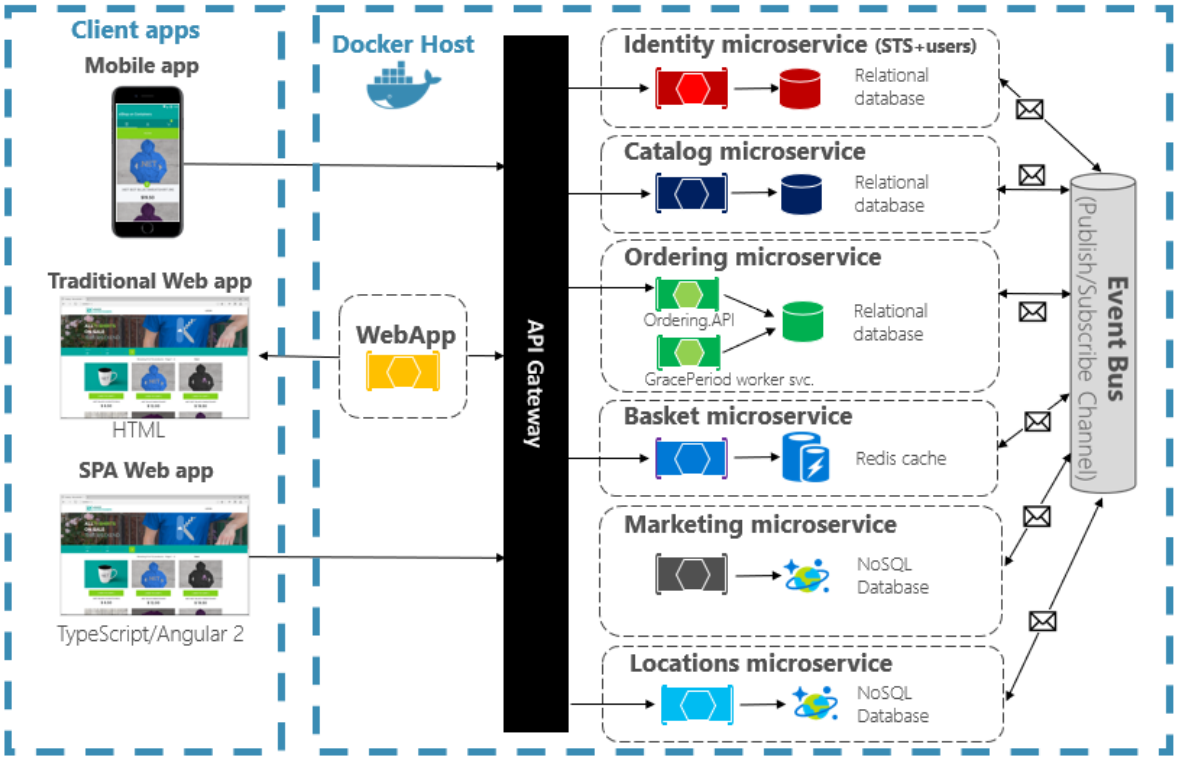
3.3.5. Conclusie

In conclusie met de Wet aanvullende bepaling verwerking persoonsgegevens in de zorg (Wabpvz) moet er rekening gehouden worden met wie wel en niet de gegevens kan inzien. De medische gegevens mogen alleen gedeeld worden wanneer de gegevens correct zijn en de cliënt hier daadwerkelijk toestemming voor heeft gegeven. Verder moeten ouders altijd geïdentificeerd worden wanneer ze het DD JGZ of een deel ervan in willen zien. Tot slot moet er bijgehouden worden wie de gegevens inziet, hierbij moet genoteerd worden welke persoon het heeft ingezien en op welke datum de inzage heeft plaatsgevonden.

3.4. Cloud Native

Het huidige KD+ en Mijn Kinddossier zijn gebouwd als monolitische applicaties die in de cloud draaien. Doordat het monolitische applicaties zijn maken ze niet gebruik van de vele voordelen die de cloud omgeving biedt. Hierdoor is er onderzoek gedaan naar het concept cloud native om een manier te vinden om beter gebruik te maken van de cloud omgeving.

Cloud native technologieën stellen bedrijven in staat om schaalbare applicaties te bouwen en runnen in een moderne, dynamische omgeving zoals de cloud. Containers, service meshes, microservices, immutable infrastructuur en gedeclareerde APIs illustreren deze benadering van ontwikkelen. Deze technieken maken losjes gekoppelde systemen mogelijk die veerkrachtig, beheersbaar en waarneembaar zijn. Gecombineerd met robuuste automatisering stellen ze teams in staat om frequent en voorspelbaar ingrijpende wijzigingen door te voeren met minimale inspanning. (Foundation, 2021)



Figuur 7: Cloud native design (Microsoft, 2021)

3.4.1. Voordelen cloud native

Het ontwikkelen van een cloud native applicatie heeft verschillende voordelen. Een van de grootste voordelen is dat de applicatie dan uit allemaal kleinere geïsoleerde services bestaat. Doordat alles geïsoleerd is hoeft niet het gehele systeem gereleaset te worden maar alleen de geïsoleerde service. Vanwege de isolatie van services is de code ook niet

zo ingewikkeld, hierdoor brengt het minder kosten met zich mee om een nieuwe feature te implementeren. Verder is het systeem ook betrouwbaarder aangezien wanneer een component van het systeem niet werkt kan er nog steeds gebruik gemaakt worden van de rest van het systeem. Met behulp van een cloud infrastructuur is een cloud native applicatie ook zeer schaalbaar. Tot slot door het bouwen van microservices zit je niet vast aan technologieën. Voor de verschillende microservices kan er gebruikt gemaakt worden van de technologie die het beste bij de taak van de microservice past.

3.4.2. Nadelen cloud native

Cloud native brengt ook verschillende nadelen met zich mee. Het grootste nadeel van een cloud native applicatie is de zeer complexe architectuur die opgezet moet worden, hierdoor is cloud native ook niet de best manier om een minimum viable product (MVP) te ontwikkelen. Het testen van het gehele systeem is ook ingewikkeld aangezien het uit allemaal kleinere services bestaat. Tot slot zit er wat meer overhead werk in het ontwikkelen van een cloud native architectuur, aangezien een team dat werkt aan een service ook de API moet uitbreiden met de functionaliteiten van de service.

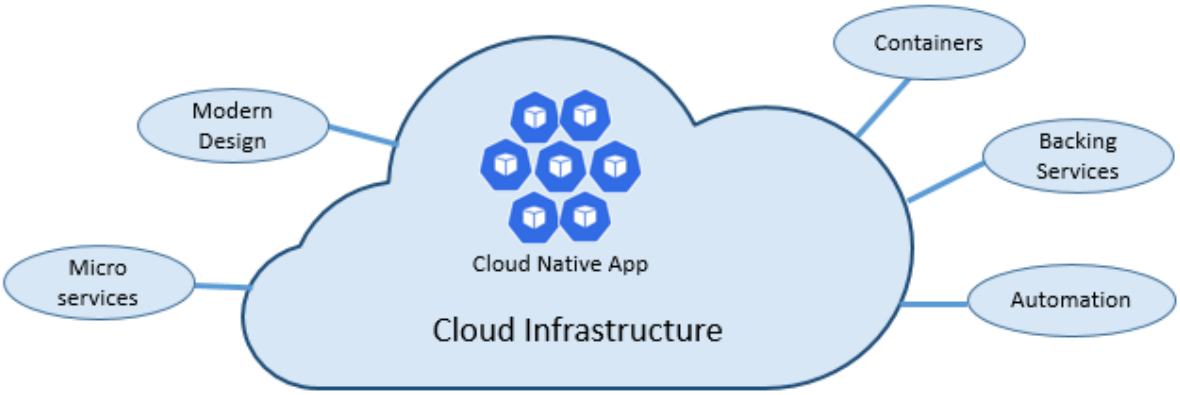
3.4.3. Waarom Cloud native?

Cloud native brengt verschillende voordelen met zich mee over een monolithic applicatie. Voornamelijk dat nieuwe features minder kosten om te implementeren en te deployen naar productie. Dit komt doordat alle service geïsoleerd zijn in een cloud native applicatie en onafhankelijker van elkaar zijn dan bij een monolithic applicatie. Verder brengt cloud native ook vrijheid van technologieën met zich mee, hierdoor zul je niet vastzitten aan een oud-of-date framework, wat kan gebeuren bij het gebruiken van een monolitich architectuur. Verder is het systeem ook meer betrouwbaar aangezien alle componenten geïsoleerd zijn, hierdoor zal het systeem nog steeds bruikbaar zijn. In tegenstelling tot een monolithic applicatie waar alle componenten correct moeten werken om het gehele systeem werkend te hebben. Tot slot is een cloud native applicatie ook schaalbaarder dan een monolithic applicatie. Met een monolithic applicatie moet je de gehele applicatie schalen. Met behulp van een cloud native applicatie hoef je alleen maar de service te schalen die geschaald moet worden.

3.4.4. Hoe bouw je cloud native?

Een cloud native systeem wordt gebouwd met behulp van verschillende pilaren om zo tot een cloud native systeem te komen. De verschillende pilaren zijn Modern Design,

Microservices, Containers, Backing Services en Automation. In figuur 8 zijn de fundamentele pilaren te zien. In de volgende hoofdstukken zullen de verschillende pilaren worden uitgelegd.



Figuur 8: Fundamentale pilaren Cloud native (Microsoft, 2021)

3.4.5. Modern Design

Binnen de methodologie van het bouwen van cloud-based applicaites is, volgens microsoft en google de Twelve-Factor Application het meest geaccepteerd. De Twelve-Factor Application beschrijft een set aan principes en practices die developers kunnen volgen om zo een applicatie geoptimaliseerd voor een cloud omgeving te bouwen. Met de Twelve-Factor Application wordt er extra aandacht gegeven aan portability tussen verschillende omgevingen en automatisering. Hieronder is een tabel te vinden met de verschillende factoren van de Twelve-Factor Application.

Tabel 1: Twelve-Factor Application (Wiggens, 2017)

Factor	Uitleg
1. Code base	Elke codebase moet zijn eigen repository hebben.
2. Dependencies	Alle dependencies moeten gedeclareerd worden via een dependency manifest zoals package.json voor elke microservice. Wanneer code gedeeld moet worden tussen twee applicaties moet dit via dependencies.
3. Configuraties	Alle configuraties moeten in de omgeving worden opgeslagen en niet in de source code.
4. Backing Services	Alle bijkomende services zoals database moeten als een service behandeld worden. Dit houdt in dat de verbindingen met de services per omgeving geconfigureerd worden.

5. Build Release Run	Build, release, run houdt in dat je het deployment proces opdeelt in drie herhaalbare stappen, namelijk build, release en run.
6. Processes	Elke microservice focust zich op zijn eigen proces. Dit betekent dat elke microservice op zijn eigen proces focust zonder te weten de staat van het proces bij te hoeven houden.
7. Port Binding	Elke microservice is identificeerbaar via hun poort nummer voor de andere microservices.
8. Concurrency	Elke applicatie moet automatisch kunnen schalen. Er moeten nieuwe instanties bij kunnen komen wanneer nodig en instanties weg worden gehaald wanneer deze niet meer nodig zijn.
9. Disposability	Er moeten nieuwe instanties van de applicties bij kunnen komen wanneer nodig en instanties weg worden gehaald wanneer deze niet meer nodig zijn.
10. Dev/Prod parity	Hou de verschillende omgevingen zo vergelijkbaar mogelijk. Probeer zo veel mogelijk dezelfde technologische stack te gebruiken in de verschillende omgevingen.
11. Logging	Logging als event streams
12. Admin processen	Run admin managementprocessen als eenmalige taken

3.4.6. Microservices

Cloud native systemen maken vaak gebruik van de microservice architectuur. Microservices worden gebruikt vanwege hun agility. Voordelen van het gebruik maken van microservices is dat elke service geïsoleerd is. Verder hebben microservices een onafhankelijke lifecycle en kunnen onafhankelijk ontwikkeld worden. Ook kunnen ze onafhankelijk van elkaar gedeployed worden zonder dat het gehele systeem gedeployed hoeft te worden, waardoor je snel een nieuwe feature kan releasen. Een ander voordeel is dat microservices onafhankelijk kunnen schalen. Hierdoor kan in plaats van de gehele applicatie te schalen alleen de microservice die het nodig heeft te schalen. Met behulp van microservices kun je ook aan factor 1: codebase en factor 6: Processes voldoen.

3.4.7. Containers

Volgens de Cloud Native Computing Foundation is het plaatsen van een microservice in een container een van de eerste stappen voor het bouwen van een cloud native applicatie. Het draaien van een microservice in een container kan behaald worden door een container image te creëren van de microservice. Deze container images kan dan in

een container gedraaid worden op een machine waar een container runtime engine op geïnstalleerd is. Elke set aan containers heeft zijn eigen set aan dependencies en draaien apart van elkaar.
Voordelen van het gebruik maken van een container zijn dat de microservice die dan in een container wordt gedraaid consistent en geïsoleerd is op de verschillende omgevingen. Verder kunnen containers draaien op elk systeem dat een container runtime engine geïnstalleerd heeft. Tot slot nemen containers niet zoveel resources in zoals een volledige virtuele machine. Met behulp van containers kun je ook aan factor 2: dependecies voldoen.

Container orchestration
Met het gebruik van containers wordt het ook essentieel om gebruik te maken van container orchestration. Container orchestration automatiseert het managen van containters. Met behulp van container orchestration zul je ook Factor 8: concurrency en Factor 9: disposability behalen.

3.4.8. Backing services

Cloud native applicaties zijn afhankelijk van bijkomstige services, zoals data stores, monitoring, identity service. Deze services worden ook wel backing services genoemd. Verschillende cloud providers bieden een rijk assortiment van backing services aan, zodat je ze zelf niet hoeft te managen. De cloud provider zorgt ervoor dat de resource beschikbaar is en geschaald wordt, regelt de security en onderhoud de resource. Vaak worden dan monitoring, redundancy en beschikbaarheid meegeleverd met de service.
Een best practice is om een backing service als een bijbehorende resource van een microservice te behandelen. Met de configuratie informatie opgeslagen in een externe configuratie. Zo behaal je ook Factor 3: configuraties en Factor 4: backing service.
Zo kunnen backing services ook gemakkelijk gekoppeld en losgekoppeld worden van een microservice. Met de configuraties kan de microservice ook gemakkelijk gekoppeld worden aan de verschillende backing services voor de verschillende omgevingen. Tot slot door aan elke microservice zijn eigen backing services te hangen kan ook Factor 6: statelessness behaald worden.

3.4.9. Automation

Een concept dat veel gebruikt wordt voor automatisering bij het bouwen van een cloud native applicatie is Infrastructure as Code (IaC). Hierdoor zijn je infrastructuur en deployments geautomatiseerd, consistent en herhaalbaar.

Tot slot met behulp van Continous Integration en Continous Deployment kan factor 5: Build, run, release behaald worden.
Vanwege de automatisering kunnen er snel verandering aangebracht worden aan de applicatie en deze veranderingen kunnen dan ook snel naar de consument gebracht worden met minimale inspanning.

4. Requirements

In dit hoofd stuk staan de verschillende requirements beschreven. Deze requirements zijn voortgekomen uit gesprekken met de stakeholders en onderzoeken.

De verschillende requirements zijn geprioriteerd volgens de MoSCoW methode. Hiermee wordt de prioriteit van de requirements verdeeld over Must (M), Should (S), Could (C) en Would (W). De Must requirements moeten in het eindresultaat terugkomen, als er niet aan deze requirements voldaan worden kan het project niet succesvol afgerond worden. De Should requirements zijn zeer wenselijk, maar zonder deze requirements kan het project nog steeds succesvol worden afgerond. De Could requirements zullen alleen aan bod komen wanneer er tijd genoeg is en alle Must en Should requirements al zijn behaald. Tot slot is er de Would, deze requirements zijn het minste belangrijk en zullen niet aan bod komen, maar kunnen voor een vervolg wel interessant zijn.

Verder heeft elke requirement een ID dat bestaat uit een categorie letter, B voor business, U voor user en S voor system, en een getal. Zo kan er makkelijker naar de verschillende requirements verwezen worden. De business requirements komen van de top van het bedrijf en zijn gefocust op de functie, doel en resultaat van het project. De user requirements zijn behoeftes vanuit het perspectief van de gebruiker van de applicatie. Tot slot zijn system requirements de requirements waar het systeem zich aan moet voldoen.

Tabel 2: Requirements

ID	Requirement	MoSCoW
B1	Als ouder het Van Wiechenonderzoek kunnen afnemen	M
B2	De software moet voldoen aan de Algemene Verordening Gegevensbescherming (AVG)	M
B3	De software moet voldoen aan de Wet aanvullende bepalingen verwerkingen persoonsgegevens zorg (Wabvpz)	M
B4	Ouders moeten ondersteund worden met de Van Wiechenfilmpjes tijdens het beantwoorden van een kind-vraag	C
B5	De software moet onafhankelijk van het KD+ en Mijn Kinddossier werken	M
B6	De oude codebase moet zo min mogelijk belast worden	M

U1	Zorgverleners moeten de door ouders ingevulde resultaten kunnen inzien	M
U2	Zorgverleners moeten hun eigen ingevulde resultaten in kunnen zien	M
U3	Ouders moeten de door zorgverleners ingevulde resultaten kunnen inzien	M
U4	Ouders moeten hun eigen ingevulde resultaten kunnen inzien	M
U5	De ouder moet alleen de Van Wiechenvragen kunnen beantwoorden die van belang zijn voor hun eigen kind.	S
U6	Ouders mogen alleen de Van Wiechen resultaten van hun eigen kind inzien	M
U7	Ouders mogen alleen het Van Wiechenonderzoek van hun eigen kind invullen	M
U9	Zorgverleners moeten de door ouders ingevulde resultaten verifiëren	C
S1	De app moet zowel in Wicket werken als buiten Wicket om	M
S2	De gebouwde software moet Cloud Native zijn	M
S3	Ouders moeten files kunnen uploaden om als bewijs last te dienen	C
S4	De microservice moet kunnen draaien in een Docker container	M
S5	De Van Wiechen backend moet een microservice worden	M
S6	De Van Wiechen microservice moet een eigen database krijgen om zijn data in op te kunnen slaan	M
S7	Er moet een gateway API komen om zo de Van Wiechen microservice aan te kunnen roepen	M
S8	De Van Wiechen app moet mobile first gebouwd worden	S
S9	Het Van Wiechen moet beschikbaar zijn in het KD+	M
S10	Het Van Wiechen moet beschikbaar zijn in het Mijn Kinddossier	M

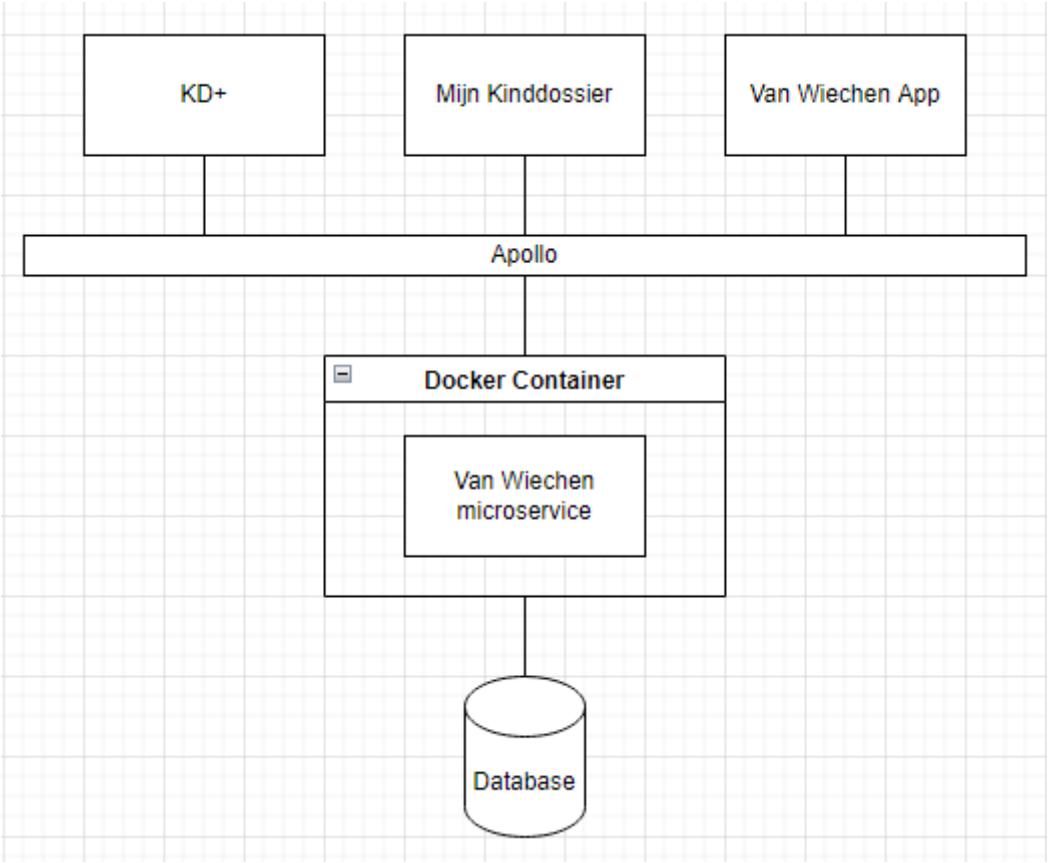
5. Implementatie

5.1. Technische opbouw

Er is voor gekozen voor een microservice architectuur voor de applicatie. Hier is voor gekozen omdat de applicatie zo cloud native mogelijk gebouwd moet worden, zie requirement S2. Met behulp van de microservice architectuur is de gehele applicatie flexibel en beter afgestemd met de cloud omgeving. Verder zal dit ook onafhankelijkheid bieden van het KD+ en het Mijn Kinddossier, zie requirements S1, S9 en S10.

Verder door een microservice te bouwen wordt het Van Wiechen geïsoleerd binnen de architectuur. Zo wordt de codebase voor zowel KD+ en Mijn Kinddossier zo min mogelijk aangetast. Verder door de isolatie zal het deployen van de Van Wiechen microservice makkelijker worden, aangezien je dan alleen de microservice hoeft te deployen en niet de gehele applicatie. Hierdoor zal de microservice ook makkelijker te onderhouden zijn, doordat het zijn eigen codebase heeft dat geïsoleerd is van de rest van de codebase. Tot slot zal de Van Wiechen microservice makkelijk te schalen zijn. Aangezien je alleen de microservice hoeft te schalen en niet de gehele applicatie.

De applicatie zal bestaan uit verschillende onderdelen, waaronder de microservice zelf, het KD+, het Mijn Kinddossier, de Van Wiechen app, een gateway API en tot slot zal er een database aan de microservice gekoppeld worden. In figuur 9 is een schematische weergave van de architectuur gegeven om een idee te geven over hoe het gehele systeem in elkaar gaat zitten.



Figuur 9: Schematische weergave architectuur

KD+

Het KD+ is het Digitale Dossier Jeugdgezondheidzorg (DD JGZ) systeem dat Topicus heeft ontwikkeld. Via het KD+ zal de zorgprofessional het Van Wiechenonderzoek zelf kunnen invullen en de resultaten van het onderzoek in kunnen zien.

Mijn Kinddossier

Het Mijn Kinddossier is het ouder portaal dat bij het KD+ hoort. De ouder zal via het Mijn Kinddossier het Van Wiechenonderzoek zelf kunnen invullen en de resultaten van het onderzoek in kunnen zien.

Van Wiechen App

De Van Wiechen app wordt een aparte applicatie waar ouders ook het Van Wiechenonderzoek zelf kunnen invullen en de resultaten kunnen inzien. De Van Wiechen app is gebouwd met behulp van het Angular Framework. Angular is een populair web framework dat gebruikt wordt voor het bouwen van webapplicaties. Er is gekozen om een webapplicatie te bouwen omdat deze zowel via de webbrowser als mobiele browser bruikbaar is. Er is gekozen om gebruik te maken van het Angular Framework omdat het team waarmee gewerkt wordt ook gebruik maakt van het Angular Framework. Zo valt het binnen de expertise van het team en kan het gemakkelijker doorontwikkeld worden.

Gateway API

Er wordt gebruikt gemaakt van een gateway API. Deze wordt dan gebruikt om niet alleen met de Van Wiechen microservice te communiceren, maar ook met de andere services van het JGZ Team. Dit is om zo alles te isoleren van elkaar. Zo hoeven de front end applicaties maar met een API te praten om informatie van de verschillende services te halen. De gateway API gaat gebouwed worden met Apollo en graphql, aangezien dit al bestaat.

Van Wiechen Microservice

De Van Wiechen microservice gaat de backend van de Van Wiechen module worden. Er is gekozen om er een microservice van te maken aangezien het aansluit op requirement S5 en S2. Eén van de pilaren van cloud native is microservice. Zo zal het Van Wiechen onafhankelijk zijn van het KD+ en Mijn Kinddossier. De microservice gaat gebouwd worden met behulp van het Ktor framework. Ktor is een framework voor het bouwen van asynchrone client en server applicaties geschreven in Kotlin. Ktor kan gebruikt worden voor het bouwen van een microservice. Er is gekozen voor Ktor met Kotlin omdat het team waarmee gewerkt wordt hier ook gebruik van maakt. Zo valt het binnen de expertise van het team en kan de microservice gemakkelijker doorontwikkeld worden.

Database

De database gaat gebruikt worden om de data van het Van Wiechen op te slaan. Door het een eigen database te geven wordt zo de module onafhankelijker van het gehele systeem en kan ook cloud native meer geforceerd worden.

Docker

Met behulp van Docker kan de software in een container draaien. Dit zorgt ervoor dat de microservice geïsoleerd gedraaid kan worden. Vanwege de isolatie kan deze microservice de andere services/applicaties niet storen. Tot slot kunnen er meerdere instanties van de microservice op een device draaien, omdat containers ervoor zorgen dat de microservice niet alle gedeelde resources inneemt.

Met behulp van container orchestration worden verschillende taken geautomatiseerd en beheerd. Zo wordt ervoor gezorgd dat er altijd een container met de microservice beschikbaar is, zorgen ze voor het schalen van containers en dat de container de resources heeft om te draaien. Verder zorgt container orchestration ervoor dat containers gedeployed worden, dat ze de juiste configuratie hebben en wanneer een container faalt,

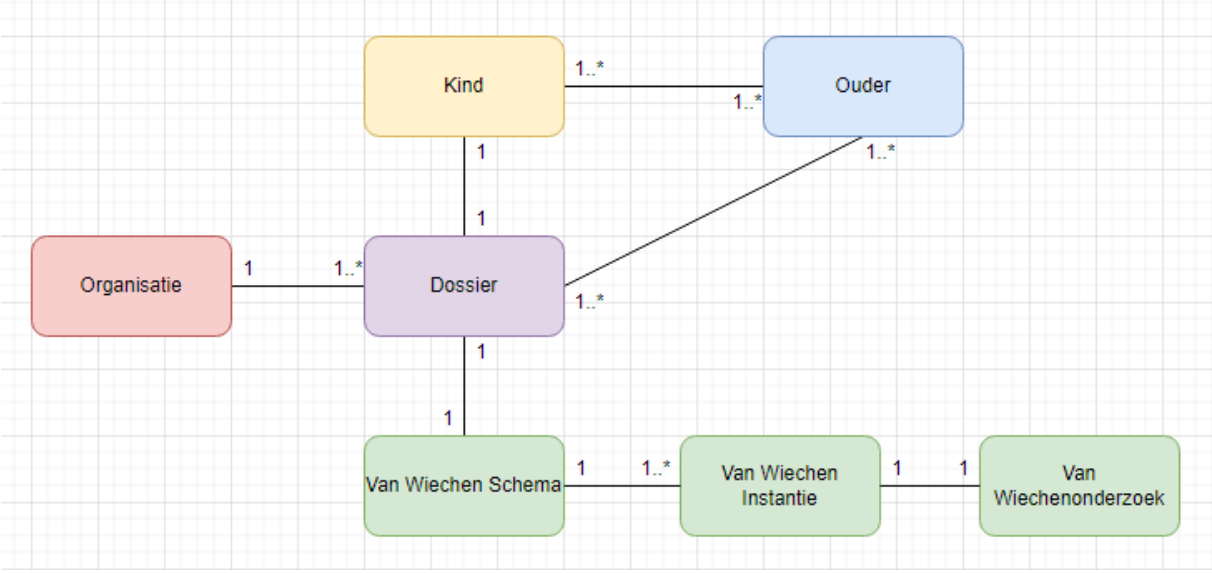
dat die container wordt afgesloten en een nieuwe instantie deze gefaalde container vervangt.

5.2. Van Wiechen buiten KD+ en Mijn Kinddossier

De Van Wiechen microservice moet onafhankelijk van het MK en het KD+ kunnen werken. Dit betekent dat ook organisaties die niet gebruik maken van de applicatie. Om dit te kunnen bereiken moeten er enkele functionaliteiten geïmplementeerd worden.

Professional moet het Van Wiechen kunnen klaarzetten voor de ouder (Zie hoofdstuk 5.2.1 Van Wiechen Klaarzetten)
Ouder moet kunnen inloggen (Zie hoofdstuk 5.3.1. Login)
Ouder moet een profiel kunnen maken

Er kunnen meerdere ouders aan een dossier/kind gekoppeld worden. Een dossier kan maximaal maar een Van Wiechen Schema bezitten. Het Van Wiechen Schema kan meerdere Van Wiechen instanties bevatten. In een instantie wordt opgenomen wat een ouder heeft ingevuld bij de Van Wiechen Kenmerken.



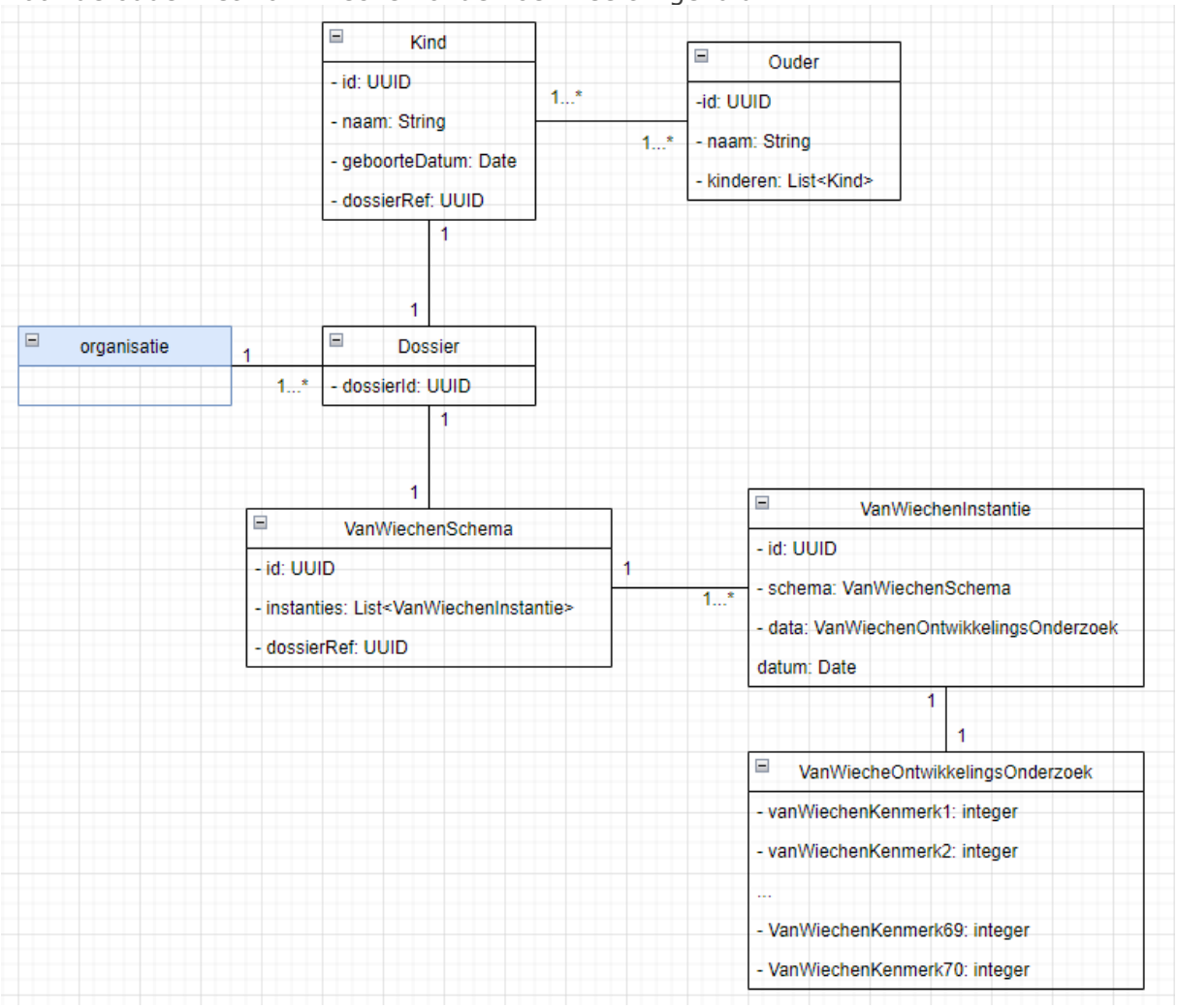
Figuur 10: Domein Model Diagram

Beschrijving van het domein Model

Om het Van Wiechen onderzoek te kunnen invullen zal een ouder een koppeling moeten hebben met het dossier van het kind. Vanuit hier kan dan het Van Wiechen onderzoek opgehaald kunnen worden en ingevuld door de ouder.

Een dossier zal altijd deel uitmaken van een organisatie en gekoppeld zijn aan het bijbehorende kind. Wanneer een ouder meerdere kinderen heeft zal deze ouder ook een koppeling met meerdere dossiers kunnen hebben.

Een Van Wiechen onderzoek heeft meerdere instanties. Elke instantie is een moment waar de ouder het Van Wiechen onderzoek heeft ingevuld.

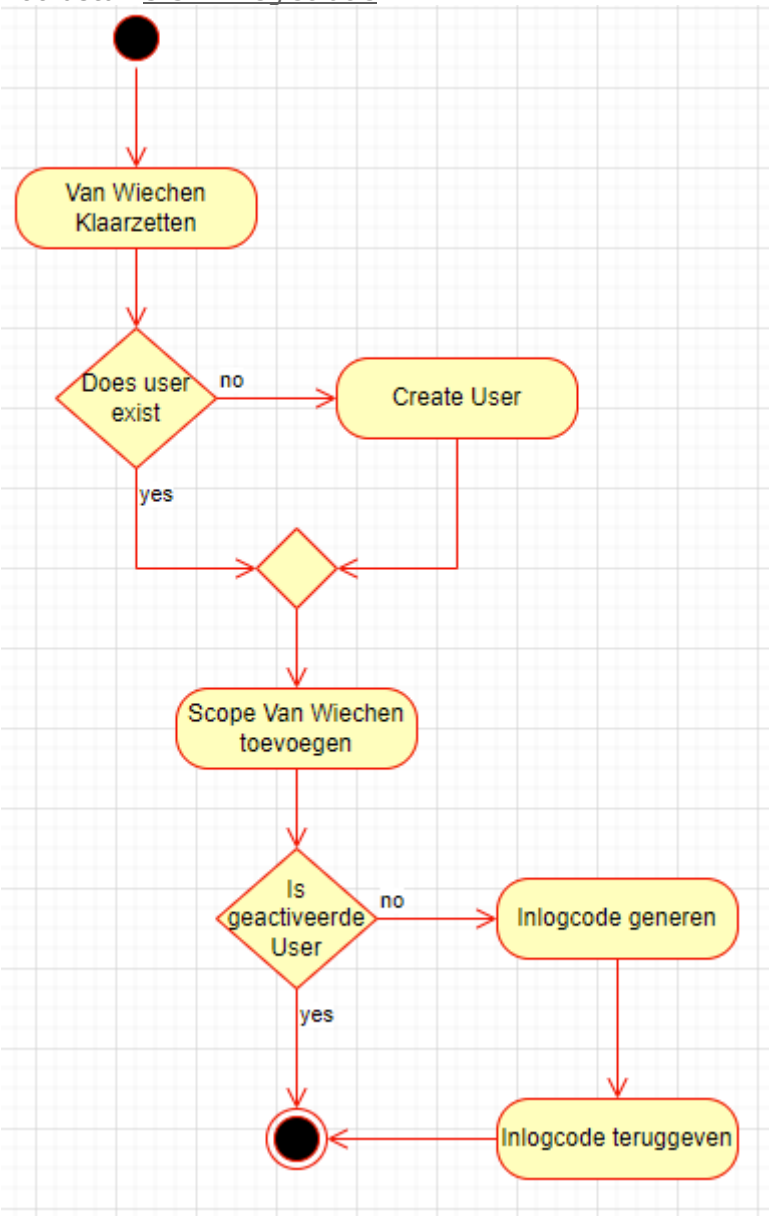


Figuur 11: Class Diagram

In figuur 11 is te zien hoe het domein model gerepresenteerd wordt als klasse diagram binnen de applicatie. Het Van Wiechenontwikkelings onderzoek bestaat uit alle 70 kenmerken. In de Van Wiechen isntantie wordt de datum van invulling genoteerd als datum.

5.2.1. Van Wiechen Klaarzetten

Om de Van Wiechen applicatie te gebruiken als ouder, zal een professional het Van Wiechen het voor de ouder klaar moeten zetten. Dit is omdat je niet weet om welke ouder het gaat en om welk kind. De professional zal weten om welk kind het gaat en zal de ouder aan het kind moeten koppelen.
De informatie die de professional nodig heeft om het Van Wiechen klaar te kunnen zetten is het BSN van de ouder. De ouder zal gekoppeld worden aan het kind zodat de ouder het Van Wiechenonderzoek voor het kind kan invullen. Hieronder is een activity diagram te vinden over het klaarzetten van het Van Wiechen. Voor registratie van een ouder zie hoofdstuk 5.3.2. Registratie



Figuur 12: Activity Diagram ouder registreren

5.3. Authenticatie en Autorisatie

Authenticatie en autorisatie wordt geïmplementeerd met behulp van de zorg identity provider (Zorg IDP). In de zorg IDP wordt dan de ouder opgeslagen samen met tot welke Van Wiechenonderzoeken de ouder toegang heeft. De gegevens van de ouder die dan worden opgeslagen bij de zorg IDP zijn dan de naam van de ouder, een user ID gebaseerd op het BSN van de ouder en tot welke Van Wiechenonderzoeken de ouder toegang heeft.
Er is gekozen om deze gegevens van de ouder op te slaan omdat alleen deze gegevens benodigd zijn voor de applicatie. Dit valt in lijn met de AVG regel, bewaar alleen de benodigde persoonsgegevens. Op deze manier valt de verantwoordelijkheid van deze gegevens ook buiten de gebouwde applicatie. Ook kan zo alleen de gebruiker bij zijn/haar persoonsgegevens, dit is een stukje privacy by default wat ook in lijn ligt met de AVG.
De authenticatie en autorisatie wordt geïmplementeerd met behulp van OpenID Connect. OpenID Connect wordt gebruikt omdat het zowel authenticatie afhandelt als autorisatie. Met behulp van OpenID connect kan de gebruiker zijn identiteit worden opgehaald en de autorisatie van de gebruiker.
De gebruiker krijgt een set aan tokens terug vanuit de Zorg IDP, waaronder een ID token, JSON Web Token (JWT Token) en een refresher token. Wat deze tokens inhouden wordt hieronder uitgelegd.
De verschillende endpoints van de microservice zullen ook beschermd worden door middel van JWT tokens. Dit is ook een stukje privacy by design om zo de applicatie ook zo AVG mogelijk te maken.

ID Token

De ID Token bevat de identiteit van de gebruiker. Hier zullen dan de volgende items in staan: de naam van de gebruiker, hun user id en een lijst van dossiers waar ze toegang tot hebben. De ID Token zal er dan als volgt uitzien:

```
{
    “user_id”: “5eb195de-e6f7-47a2-8ddb-162eec301bb5”
    “name”: “Liam Schippers”
    “dossiers”: [ “bc782df9-881c-4928-8d38-9f5ac9f755cd”, “b0e204f3-711a-4433-9012-d0dd2ea33bc6” ]
}
```

Syntax: JSON

JWT Token

De JWT Token wordt gebruikt om toegang te krijgen tot de microservice. Met behulp van de JWT Token wordt de gebruiker geverifieerd in de microservice. Zonder een geldig JWT token heeft de gebruiker geen toegang tot de endpoints van de microservice.

Refresher Token

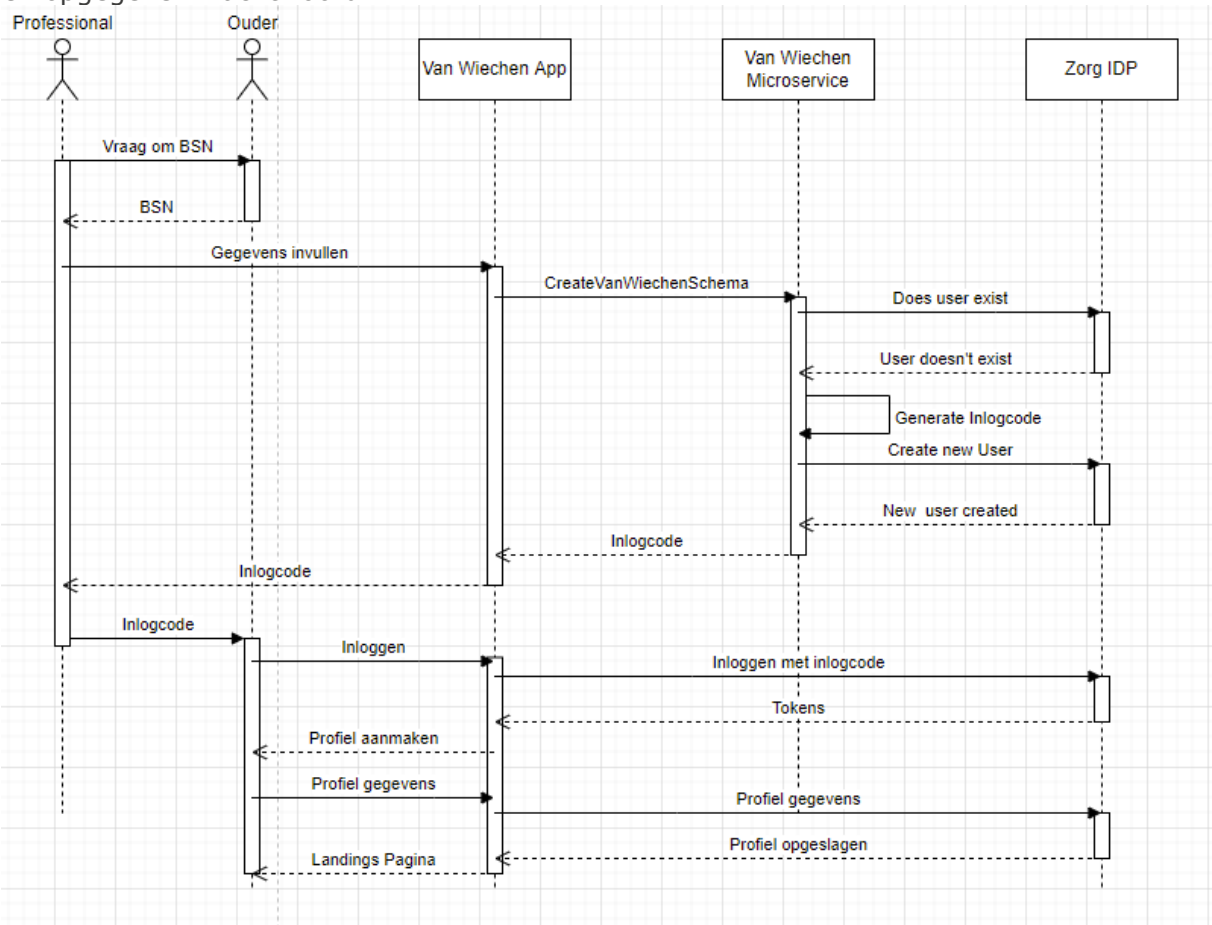
De refresher token wordt gebruikt om de JWT Token te vernieuwen wanneer de JWT token verloopt. Dit voorkomt dat een gebruiker steeds opnieuw in moet loggen wanneer de JWT token is verlopen. Zodra de JWT Token verlopen is zal er een refresher token gelden, een nieuwe JWT Token wordt opgehaald bij de Zorg IDP. Wanneer de refresher token verloopt zal de gebruiker opnieuw in moeten loggen.

5.3.1. Login

Een ouder die gebruik wil maken van de Van Wiechen applicatie zal altijd geauthentiseerd moeten worden door de Zorg IDP. Hier zullen ze dan met hun credentials moeten inloggen om toegang te krijgen tot de applicatie. Wanneer een gebruiker heeft ingelogd zal de gebruiker een id token, JWT Token en een refresher token ontvangen van de Zorg IDP. De gebruiker zal eerst bij de Zorg IDP geauthentiseerd worden. Na de autorisatie krijgt de gebruiker zijn tokens.

5.3.2. Registratie

Een gebruiker registreren zal gebeuren wanneer een professional het Van Wiechen het klaarzet voor de ouder. Tijdens het klaarzetten zal de professional de informatie van het kind invullen, waaronder de naam en de geboortedatum, verder zal de professional ook het BSN van de ouder nodig hebben. Het BSN zal dan omgezet worden naar een UUID en er wordt een inlogcode gegenereerd. Het BSN omzetten naar een UUID is een vorm van privacy by design, waarbij een persoonlijke identifier, het BSN, wordt omgezet naar een artificial identifier, een UUID. Deze inlogcode zal dan aan de ouder gegeven moeten worden zodat zij voor de eerste keer kunnen inloggen.
Tijdens de eerste keer inloggen zullen ouders gevraagd worden om een profiel te maken, wat bestaat uit hun naam invullen. Daarnaast zal de ouder gevraagd worden de inlogcode naar een wachtwoord te veranderen, zodat zij kunnen inloggen met hun BSN en opgegeven wachtwoord.



Figuur 13: Nieuwe ouder registreren

6. Realisatie

6.1. Backend (Microservice)

De backend wordt gebouwd als een microservice. Hier is voor gekozen omdat met behulp van het cloud native concept microservices makkelijker te schalen zijn dan monolitische applicaties. Verder biedt dit ook meer onafhankelijkheid van het KD+ en Mijn Kinddossier. Zo kan de backend ontwikkeld worden met weinig tot geen impact op het ontwikkelproces van de al bestaande applicaties.

Kotlin
Kotlin is een programmeertaal die is ontworpen als een beter alternatief voor Java door JetBrains. Net als Java wordt Kotlin gecompileerd naar Java bytecode die door de Java Virtual Machine uitgevoerd kan worden. De voordelen van Kotlin ten opzichte van Java zijn dat er minder boilerplate code geschreven hoeft te worden, maar ook dat het naast object georiënteerd programmeren, functioneel programmeren ondersteund.

Ktor
De microservice gaat gebouwd worden met behulp van het Ktor framework. Ktor is een framework, ontwikkeld door JetBrains, voor het bouwen van asynchrone client en server applicaties geschreven in Kotlin. Ktor kan gebruikt worden voor het bouwen van een microservice. Er is gekozen voor Ktor met Kotlin omdat het team waarmee gewerkt wordt hier ook gebruik van maakt. Zo valt het binnen de expertise van het team en kan de microservice gemakkelijker doorontwikkeld worden.

Ktor is opgebouwd uit verschillende modules, die weer opgebouwd zijn uit verschillende features. In figuur 14 is te zien hoe een Ktor programma is opgebouwd. De main functie wordt aangeroepen met de argumenten om de applicatie te configureren. Daarna wordt de Application.module uitgevoerd, hier wordt onder ander de database, routing en graphql geconfigureerd.

Figuur 14: Ktor main file

Persistence

Voor persistente dataopslag wordt er gebruik gemaakt van een PostgreSQL database. Er is gekozen om een relationele database te gebruiker omdat de data die verwerkt wordt zeer gestructureerd is. Verder is de PostgreSQL database te gebruiken met Amazon Relational Database Service (RDS). Er is gekozen om gebruik te maken van het RDS omdat de applicatie gehost zal worden in de Amazon Web Service omgeving van Topicus.

Exposed

Exposed is een object-relational mapping (ORM) framework voor kotlin. ORM is een techniek om data van een database te kunnen queryen en aan te kunnen passen op een object georiënteerde manier. Exposed zorgt ervoor dat er Kotlin gebruikt kan worden om zo de database data te manipuleren in plaats van allemaal SQL queries te schrijven. Exposed kan gebruikt worden in twee verschillende vormen, namelijk DSL (Domain Specific Language) en DAO (Data Access Object).

De DSL API van Exposed is vergelijkbaar met SQL queries (Exposed, 2021). Verder ondersteund DSL ook type safety, waarbij types worden gechecked door de compiler. Doordat DSL zo vergelijkbaar is met SQL queries, zijn SQL queries gemakkelijk om te zetten naar DSL queries. Een nadeel van DSL is dat er code duplicatie voor kan komen omdat verschillende cases hun eigen parameters nodig hebben.

De DAO API van Exposed is meer vergelijkbaar met ORM frameworks zoals Hibernate (Exposed, 2021). Bij de DAO API representeert een object een rij in de database. Verder ondersteund het CRUD operaties: Create, Update, Read en Delete. Hierdoor is het de DAO API best gemakkelijk te gebruiken.

Er is gekozen om de DSL API te gebruiken van Exposed. Hier is voor gekozen omdat er bij de DAO API te veel magie onder water gebeurt. Doordat er minder magie plaatsvindt met de DSL API is er meer controle over de data die gemanipuleerd wordt en is het gemakkelijker te debuggen.

In figuur 15 is te zien hoe een tabel gedefinieerd wordt met behulp van de DSL API. In dit geval wordt er een UUID-tabel gemaakt. Dit betekent dat de tabel in de database automatisch een UUID als primary key krijgt toegekend. Verder worden er ook referenties gemaakt naar andere tabellen. Aangezien de tabellen die gerefereerd worden ook UUID-tabellen zijn wordt in de database de UUID die de primay key is voor de tabel in deze tabel ingevoerd als forgein key.

```
object VanWiechenSchemaInstantieTable: UUIDTable() {
    val schemaId = reference( name: "schema_id", VanWiechenSchemaTable)
    val data = reference( name: "vanwiechenontwikkelingsonderzoek_id", VanWiechenOntwikkelingsOnderzoekTable)
    val datum = text( name: "date")
}
```

Figuur 15: tabel definiëren

In figuur 16 is te zien hoe een rij van de tabel in figuur 15 eruitziet als een Kotlin data class.

```
@Serializable
data class VanWiechenSchemaInstantie (
    val id: String,
    val schemaId: String,
    val data: VanWiechenOntwikkelingsOnderzoekModel,
    val datum: String,
)
```

Figuur 16: Database informatie als klasse

In figuur 17 is een voorbeeld te zien van hoe zowel de data class (figuur 16) als tabel (figuur 15) worden gebruikt om data weg te schrijven naar de database. In dit gevval wordt er een batch insert gedaan. Dit betekent dat voor elk item in een lijst een aparte insert wordt uitgevoerd om de data van dat item weg te schrijven naar de database. In de batch insert wordt nog wel elk item naar de juiste kolom gemapped moet worden.

```
fun insertVanWiechenInstanties(vanWiechenSchemaInstanties: List<VanWiechenSchemaInstantie>): List<ResultRow> {

    val result = VanWiechenSchemaInstantieTable.batchInsert(vanWiechenSchemaInstanties) { instanties ->
        this[VanWiechenSchemaInstantieTable.schemaId] = UUID.fromString(instanties.schemaId)
        this[VanWiechenSchemaInstantieTable.data] =
            VanWiechenOntwikkelingsOnderzoekRepository.insertVanWiechenOntwikkelingsOnderzoek(instanties.data)
        this[VanWiechenSchemaInstantieTable.datum] = instanties.datum
    }

    return result
}
```

Figuur 17: Exposed Batch insert functie

Flyway

Flyway is een open-source database-migration tool. Met behulp van Flyway kan de database geupdate worden van de ene versie naar een andere versie met behulp van migrations. Deze migrations kunnen geschreven worden in SQL.

In figuur 18 is te zien hoe Flyway geconfigureerd is. Eerst wordt de datasource, oftewel de database, toegevoegd aan de configuratie. Daarna wordt de datasource geladen. Nadat de datasource geladen is print hij de migraties die zijn uitgevoerd met de .info() commando. Tot slot voert Flyway de nog niet uitgevoerde migraties uit via de .migrate() commando.

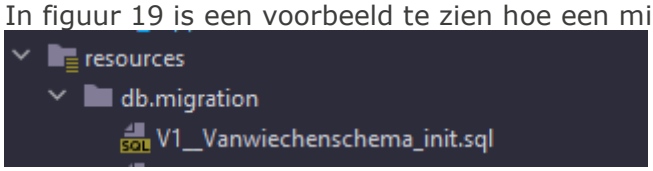
```
private fun runFlyway(datasource: DataSource) =
    Flyway.configure()
        .dataSource(datasource)
        .load()
        .also { it: Flyway!
            it.info()
            it.migrate()
        }
```

Figuur 18: configuratie Flyway

Migratie scripts voor Flyway staan in de volgende folder *$PROJECT_ROOT/resources/db/migrations*. De migratie scripts hebben een bepaalde naamgeving, namelijk *<Prefix><Versie>__<Beschrijving>.sql*.

- *<Prefix>* Het prefix voor de scripts is standaard V.
- *<Versie>* Dit is het versie nummer. Wanneer de versie bijvoorbeeld 1.1 is zal de bestandsnaam er als volgt uitzien: *V1_1__voorbeeld_script.sql*.
- *<Beschrijving>* Dit is de beschrijving van het migratie script. Deze wordt van de versie gescheiden door een dubbele underscore.

In figuur 19 is een voorbeeld te zien hoe een migratie script in een ktor project zit.



Figuur 19: Flyway migratie file

6.2. Gateway API

Een gateway API is een API die alle backend services verbindt met de front-end. Zo hoeft de front-end niet met alle verschillende services verbonden te worden maar met een enkele API. Dit is vooral toepasbaar op een microservice architectuur, aangezien een microservice architectuur uit allemaal verschillende microservices bestaat waar de front-end allemaal mee moet verbinden. Het JGZ team heeft al een gateway API gebouwd met behulp van Apollo GraphQL. De bestaande gateway API zal dan verder uitgebreid worden.

GraphQL

GraphQL is een query language die zit tussen backend en front-end. Het stelt de backend in staat om precies aan te geven welke data er beschikbaar is. Verder stelt het de front-end in staat om alleen de gegevens op te halen die van belang en niets meer. Er is gekozen voor GraphQL omdat, in tegenstelling tot REST, de front-end precies weet welke data er beschikbaar is en alleen de data op kan halen die de front-end nodig heeft. Dit voorkomt over- en under-fetching. Over-fetching is dat er te veel data wordt opgehaald vanuit de API en under-fetching is dat er te weinig data wordt opgehaald vanuit de API.

Apollo

Apollo is een platform dat GraphQL implementeert om zo data te kunnen overdragen tussen de front-end en de achterliggende services en APIs. Apollo GraphQL wordt gebruikt als een gateway API.

GraphQL API

Elke Graph in GraphQL maakt gebruik van een schema om te definiëren wat voor type data het bevat. GraphQL maakt dan gebruik van drie verschillende schema types, namelijk object types (zie figuur 20), het query type en het mutation type (zie figuur 21). Het object type is een object dat opgehaald of aangepast moet kunnen worden. Met behulp het query type kunnen objecten daadwerkelijk opgehaald worden. Het mutation type wordt gebruikt wanneer de data aangepast moet kunnen worden.

```
type VanWiechenSchemaInstantie {
  id: String
  schemaId: String
  data: VanWiechenOntwikkelingsOnderzoek!
  datum: String
}
```

Figuur 20: GraphQL Schema

```
extend type Mutation {
  newVanWiechenOuder(body: InputVanWiechenSchema): VanWiechenSchemaModel
}
```
Figuur 21: GraphQL Mutation declarative

GraphQL maakt gebruik van resolvers. Resolvers zijn verantwoordelijk voor het object type voorzien van de bijbehorende data. Wanneer er een GraphQL query binnenkomt, van het query of mutation type, zal deze afgehandeld worden door de bijhorende resolver. In figuur 22 is een voorbeeld van een mutation resolver te zien. Deze mutation resolver handelt de mutation newVanWiechenOuder af. Deze zorgt ervoor dat er een nieuw VanWiechenSchema opgeslagen wordt met de bijbehorende VanWiechenInstanties. De resolver zal de data doorsturen naar de bijbehorende satasource, in dit geval de Van Wiechen microservice.

```
export const newVanWiechenOuder: MutationResolvers<Context>['newVanWiechenOuder'] =
  async (_, variables, { dataSources }) => {

    const schemaId = uuidv4().toString()
    const instanties = mapToInstanties(variables.body.instanties, schemaId)

    return await dataSources.vanWiechenGraphQl.postVanWiechenOuder({
      id: schemaId,
      instanties: instanties,
      clientId: variables.body.clientId,
    })
  }
```
Figuur 22: Apollo Resolver

In de datasource klasse zal de interactie met de datasource afgehandeld worden. In dit geval zal er een mutation worden uitgevoerd bij de Van Wiechen microservice. De mutation die wordt uitgevoerd is dan de postVanWiechenResult mutation. In figuur 23 is te zien hoe de mutation op wordt gesteld en wordt gequeried.

```
export class VanWiechenGraphQl extends RESTDataSource {
  constructor(baseURL: string) {
    super()
    this.baseURL = baseURL
  }

  async postVanWiechenOuder(
    body: VanWiechenSchemaModel
  ): Promise<VanWiechenSchemaModel> {
    return await this.post('/graphql', {
      query: `mutation PostVanWiechenResults($input: VanWiechenSchemaModel!) {
        postVanWiechenResults(body: $input) {
          id
        }
      }`,
      variables: {
        input: body,
      },
    }).then((response) => response.data.postVanWiechenResults)
  }
}
```

Figuur 23: Apollo Datasource

In Figuur 24 is te zien hoe een mutation gedefinieerd wordt met behulp van KgraphQL. Wanneer hij binnenkomt zal de mutation afgehandeld worden door de bijhorende resolver. De resolver is in figuur 25, deze zal de data doorsturen naar de database, wat in dit geval de datasource is.

```
mutation( name: "postVanWiechenResults") {  this: MutationDSL
    description = "Voegt een nieuw Van Wiechen Onderzoek to aan de database"

    resolver { body: VanWiechenSchemaModel -> VanWiechenResolver().postResolver(body) }
}
```

Figuur 24: KraphQL Mutation

```
suspend fun postResolver(body: VanWiechenSchemaModel): VanWiechenSchemaModel {
    return VanWiechenRepository.save(body)
}
```

Figuur 25: KraphQL Resolver

6.3. Front-end

De Van Wiechen app is gebouwd met behulp van het Angular Framework. Angular is een populair web framework dat gebruikt wordt voor het bouwen van webapplicaties. Er is gekozen om een webapplicatie te bouwen omdat deze zowel via de webbrowser als mobiele browser bruikbaar is. Er is gekozen om gebruik te maken van het Angular Framework omdat het team waarmee gewerkt wordt ook gebruik maakt van het Angular Framework. Zo valt het binnen de expertise van het team en kan het gemakkelijker doorontwikkeld worden.

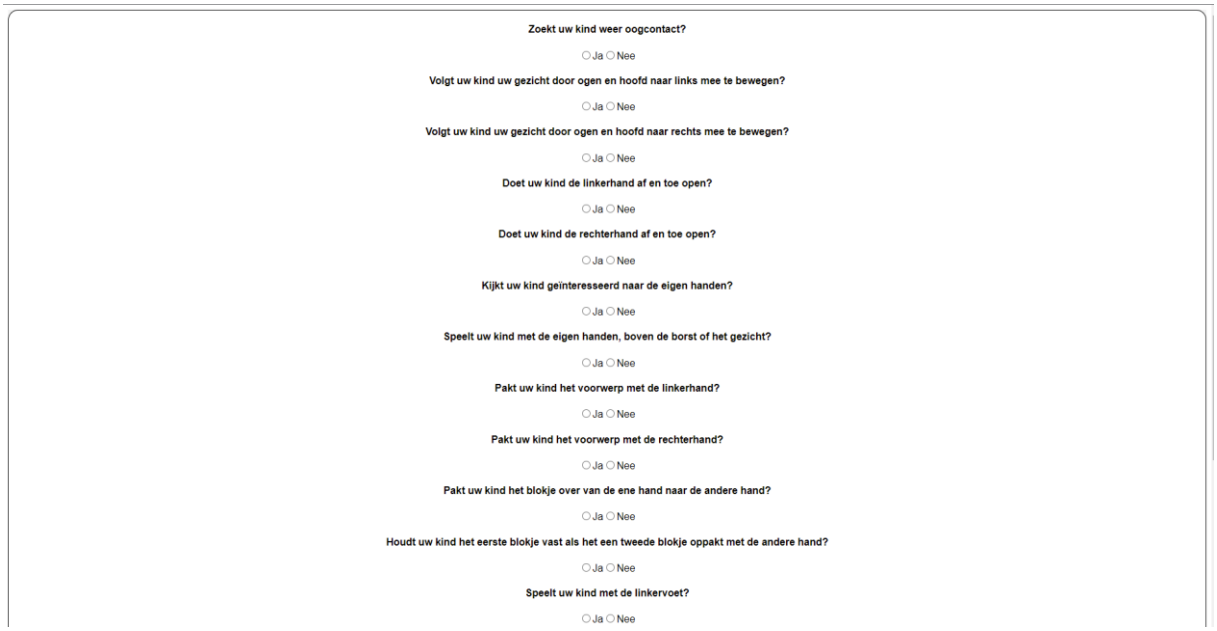
6.3.1. Web component

Web components is een set aan verschillende technologieën die het mogelijk maken om herbruikbare elementen te maken die gebruikt kunnen worden in verschillende web apps. Deze componenten kunnen dan gebruikt worden in verschillende JavaScript libraries of frameworks die gebruik maken van HTML. Omdat voor het Mijn Kinddossier en de Van Wiechen App veelal dezelfde functionaliteiten gerealiseerd moeten worden, wordt er gebruik gemaakt van web componenten. Met behulp van web componenten hoeven de componenten maar in één applicatie gerealiseerd te worden en kan dat component gebruikt worden in de andere applicatie.

6.3.2. Implementatie web components

Om met Angular web componenten te kunnen bouwen wordt er gebruik gemaakt van een library gemaakt Angular elements. Angular elements zijn Angular componenten die verpakt zijn custom elements, ook wel web componenten genoemd. Een custom element zorgt ervoor dat je een HTML tag kunt definiëren die gebouwd en beheert wordt door Javascript code.

Voordat een Angular component naar een web component omgezet kan worden moet er natuurlijk eerst een Angular component gebouwd worden. Een Angular component bestaat uit een HTML, CSS en een Typescript bestand. De HTML en CSS bestanden zorgen samen voor de user interface en opmaak van het bestand en het Typescript bestand voorziet de pagina van onderliggende logica. In figuur 20 is het Van Wiechen vragenlijst component te zien die ouders in kunnen vullen.



Figuur 26: Van Wiechen vragenlijst component

Nadat het Angular component is gebouwd moet het gedefinieerd worden als een custom element met behulp van de Angular elements library. In figuur 21 is te zien hoe het component in figuur 20 wordt gedefinieerd als custom element. Eerst wordt er gekeken of de applicatie in development modes gedraaid wordt of een andere modus. Zolang de applicatie niet in development modes draait zal er een custom element gemaakt worden van het component. Daarna wordt gecheckt of het custom element niet al gedefinieerd is, wanneer die dat niet is wordt hij gedefinieerd. Tijdens het definiëren van het web component geef je de HTML tag mee waarmee het component aangeroepen kan worden, in dit geval is het *vanwiechen-ouder-invul-component*.

```
export class AppModule {

  constructor(private injector: Injector) {
    if (!isDevMode()) {
      const el = createCustomElement(InvullenOuderComponent, {injector: this.injector})

      if (!customElements.get('vanwiechen-ouder-invul-component')) {
        customElements.define('vanwiechen-ouder-invul-component', el)
      }
    }

  }

  ngDoBootstrap(appRef: ApplicationRef): void {
    if (isDevMode()) {
        appRef.bootstrap(AppComponent);
    }
}
```

Figuur 27: Custom elements definiëren

De web componenten moeten nadat ze gedefinieerd zijn ingeladen worden binnen Wicket. Nu worden ze nog ingeladen vanuit locale bestanden die uit het build script van Angular komen. Idealiter zouden deze bestanden automatisch geüpdatet worden wanneer er een nieuwe versie van de Van Wiechen app gepubliseerd wordt. Zowel de CSS als Javascript moeten ingeladen worden, dit wordt gedaan met behulp van de volgende twee regels code:

```
response.render(CssHeaderItem.forUrl("/assets/javascript/van-wiechen-ouder-component.css", "screen"));
```

```
response.render(JavaScriptHeaderItem.forUrl("assets/javascript/van-wiechen-ouder-component-es2015.js", "van-wiechen-es2015"));
```

Nadat de web componenten zijn ingeladen kunnen de componenten in het Wicket framework gebruikt worden. Er moeten dan drie verschillende items gedefinieerd worden, namelijk de HTML, de pagina waar het component op komt en het component zelf. In Figuur 28 is te zien hoe het web component in de HTML toegevoegd wordt binnen Wicket.

```
<!DOCTYPE html>
<html xmlns:wicket="http://wicket.apache.org">
<wicket:panel>
    <div wicket:id="vanWiechenComponent"></div>
</wicket:panel>
</html>
```

Figuur 28: HTML component in Wicket

```
@Override
protected void onInitialize() {
    super.onInitialize();

    add(new VanWiechenWebComponent( id: "vanWiechenComponent"));
}
```

Figuur 29: Web component initialiseren

In figuur 30 wordt het component gedefinieerd in het Wicket framework. Binnen het Wicket framework wordt een nieuw component gemaakt die als het web component dient. Dit component wordt dan gebruikt om het web component in een pagina te laden.

```java
public class VanWiechenWebComponent extends WebMarkupContainer {

    public VanWiechenWebComponent(String id) { super(id); }

    @Override
    protected void onComponentTag(ComponentTag tag) {
        super.onComponentTag(tag);

        tag.setName("vanwiechen-ouder-invul-component");
    }
}
```

Figuur 30: Web component in Wicket

Het uiteindelijke resultaat is te zien in figuur 31.

Figuur 31: Van Wiechen in Mijn Kinddossier

6.4. Deployment

Om de applicatie te kunnen draaien en te deployen wordt er gebruik gemaakt van Docker Images en Infrastructure as Code (IaC). Met behulp van de dockerfiles te zien in figuur 32 en 33 worden er docker images gebouwd. Deze docker images kunnen dan weer gebruikt worden om de applicaties in containers te laten draaien.

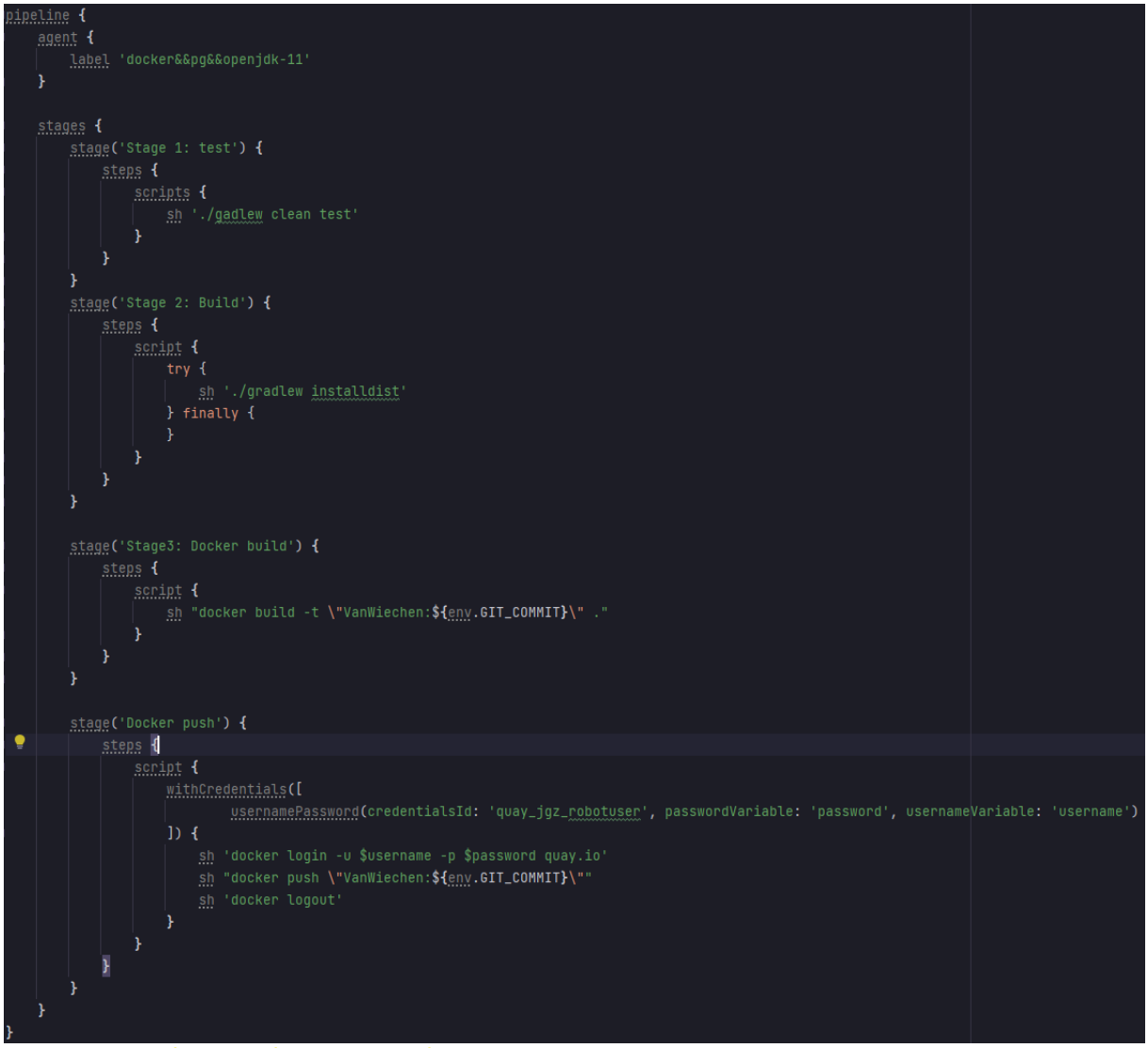
```
FROM adoptopenjdk/openjdk11-openj9

EXPOSE 8095

RUN mkdir /app

COPY ./build/install/VanWiechen /app

WORKDIR /app

CMD ["./bin/VanWiechen"]
```

Figuur 32: Dockerfile Van Wiechen microservice

```
dockerfile > ...
1   FROM node:16.13.2 as build
2   WORKDIR /app
3   COPY . .
4   RUN npm install
5   RUN npm run build --prod
6
7   FROM nginx:1.17.1-alpine
8   COPY --from=build /app/dist/van-wiechen-frontend /usr/share/nginx/html
```

Figuur 33: Dockerfile Van Wiechen app

Deze dockerfiles zullen geconsumeerd worden door hun bijhorende Jenkins pipeline die het project zal builden en releasen. In figuur 34 is de Jenkins pipeline van de Van Wiechen microservice te zien. Deze pipeline wordt uitgevoerd wanneer een pull request naar de master branch wordt gemerged. De Jenkins zal dan eerst alle unit testen uitvoeren. Nadat de testen zijn geslaagd wordt het project gebouwd. Met de artefacten die voortkomen uit het bouwen van het project wordt er een docker image gebouwd. Nadat de docker image gebouwd is zal deze gereleased worden naar quay.io.



```
pipeline {
    agent {
        label 'docker&&pg&&openjdk-11'
    }

    stages {
        stage('Stage 1: test') {
            steps {
                scripts {
                    sh './gadlew clean test'
                }
            }
        }
        stage('Stage 2: Build') {
            steps {
                script {
                    try {
                        sh './gradlew installdist'
                    } finally {
                    }
                }
            }
        }

        stage('Stage3: Docker build') {
            steps {
                script {
                    sh "docker build -t \"VanWiechen:${env.GIT_COMMIT}\" ."
                }
            }
        }

        stage('Docker push') {
            steps {
                script {
                    withCredentials([
                            usernamePassword(credentialsId: 'quay_jgz_robotuser', passwordVariable: 'password', usernameVariable: 'username')
                    ]) {
                        sh 'docker login -u $username -p $password quay.io'
                        sh "docker push \"VanWiechen:${env.GIT_COMMIT}\""
                        sh 'docker logout'
                    }
                }
            }
        }
    }
}
```

Figuur 34: Jenkins Pipeline Van Wiechen microservice

Uiteindelijk wordt met behulp van de deploy pipeline. In deze pipeline moet de deployment en de service voor Kubernetes worden ingesteld. De deployment is verantwoordelijk voor een set aan containers draaiend te houden en de service zorgt ervoor dat de container benaderbaar is in het netwerk. In figuur 35 is een voorbeeld van een deployment te zien voor de Van Wiechen app. In figuur 36 is een voorbeerd van de dervice voor de Van Wiechen app te zien. De deployment en service worden dan met behulp van zogeheten “building blocks” ontwikkeld door Topicus omgezet naar terraform bestanden, bestanden die kuberneters begrijpt, met deze pipeline. Deze bestanden worden dan geapplied op het cluster waar de applicatie dan draait in pods in het kubernetische kluster.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: van-wiechen-frontend
  namespace: {{ namespace }}
  labels:
    app: van-wiechen-frontend
spec:
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: van-wiechen-frontend
  replicas: {{ apps.van-wiechen-frontend.replicas }}
  template:
    metadata:
      labels:
        app: van-wiechen-frontend
    spec:
      imagePullSecrets:
        - name: quay-topicus-zorg-jgz-jenkins-previder-pull-secret
      containers:
        - name: app
          env:
            - name: APOLLO_URL
              value: http://apollo.{{ namespace }}.svc.cluster.local:4000
          image: quay.io/topicus_zorg/van-wiechen-frontend:{{ apps.van-wiechen-frontend.image_tag }}
          resources:
            requests:
              memory: {{ env.resources.van-wiechen-frontend.requests.memory }}
              cpu: {{ env.resources.van-wiechen-frontend.requests.cpu }}
            limits:
              memory: {{ env.resources.van-wiechen-frontend.limits.memory }}
          ports:
            - containerPort: 80
```

Figuur 35: Deployment Van Wiechen app test

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vanwiechen-frontend
  namespace: {{ namespace }}
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: NodePort
  sessionAffinity: ClientIP
  selector:
    app: van-wiechen-frontend
```

Figuur 36: Service Van Wiechen app test

6.5. Cloud Native

Om de applicatie zo cloud native mogelijk te maken zijn de volgende stappen genomen aan de hand van de twelve-factor application (Zie hoofdstuk … modern design).

Factor 1: codebase
Volgens Factor 1 van de twelve-factor app moet de source code van de applicatie bijgehouden worden via een version control system zoals Git. Dit is bereikt door de source code van de applicatie bij te houden in GitHub. Daarnaast heeft de applicatie zijn eigen repository in GitHub.

Factor 2: Dependencies
Factor 2 houdt in dat dependencies expliciet gedefinieerd wordt. De dependecies moeten dan ook gepackaged kunnen worden in een container. Dit kan bereikt worden door alle depencies te declareren in een dependency manifest, zoals gradle of package.json. Hier is niet helemaal aangehouden omdat de web componenten via een artefact die handmatig toegevoegd moet worden ingeladen worden in Wicket.

Factor 3: Config
De derde factor houdt in dat voor de verschillende omgevingen verschillende configuraties bestaan. Deze configuraties extern worden opgeslagen in configuratie variabelen per omgeving. Dit wordt bereikt door gebruik te maken van de pipeline van Topicus. Hier worden de verschillende configuratie variabelen opgeslagen per omgeving. Zie figuur 35. Hier worden verschillende variabelen per omgeving geconfigureerd.

Factor 4: backing Services
Volgens factor 4 moet je backing services, zoals bijvoorbeeld een database, als gewone services. Dit is behaald door de connectie string en credentials van de database extern op te slaan in de configuraties. Zo kan de database makkelijk uitgewisseld worden tussen de verschillende omgevingen. De backing services worden geconfigureerd via IaC waar ze per omgeving gekoppeld worden aan de applicatie.

Factor 5: Build, Release, Run
Build, release, run houdt in dat je het deployment proces opdeelt in drie herhaalbare stappen, namelijk build, release en run. Dit wordt breikt door hoe de deployment van de applicatie is opgezet (hoofdstuk 6.4 Deployment). Hier is te zien dat de applicatie eerst gebouwd wordt, dan gereleased wordt als docker image met behulp van een Jenkins

pipeline. Nadat hij gereleased is als docker image kan hij gerund worden op de juiste omgeveing met behulp van IaC

Factor 6: Processes

Het factor processes houdt in dat de applicatie gebouwd moet worden als een proces. Dit betekent dat de microservice stateless zou moeten zijn. Dit wordt bereikt doordat er geen state opgeslagen wordt in de microservice. De microservice slaat of haalt data op uit de database. Hierdoor zal de gebruiker zijn gegevens niet verliezen wanneer de service wegvalt en met een andere verbindt.

Factor 7: Port binding

Met behulp van de services worden kunnen de verschillende services in de backend aan elkaar verbonden doormiddel van porten. Met behulp van IaC kunnen de applicaties met behulp van hun poorten aan elkaar gebonden worden.

Factor 8: Concurrency

Concurrency houdt in dat de microservcies horizontaal schaalt. Dit betekent dat er meerdere instanties van dezelfde microservice draait. Door gebruik te maken van docker containers en container orchestration wordt dit proces geautomatiseerd.

Factor 9: disposability

Volgens factor 9 disposability moeten containers kunnen afsluiten en snel kunnen opstarten. Dit zorgt ervoor dat de microservice makkelijk kan schalen. Dit wordt ook bereikt doormiddel van container orchestration. Dit zorgt ervoor dat containers worden opgestart wanneer nodig en afgesloten wanneer ze niet meer nodig zijn.

Factor 10: Dev/Prod parity

Factor 10 houdt in dat alle omgevingen zo veel mogelijk hetzelfde zijn. Omdat in AWS alles in docker containeres zal draaien worden de applicaties ook al lokaal in docker containers gedraait. Verder wordt er ook gewoon gebruik gemaakt van een PostgreSQL in de lokale stack aangezien deze ook gebruik wordt binnen AWS.

Factor 11: Logs

Alle logs worden met behulp van statsD verstuurd naar het elasticsearch kibana cluster. Alle output van de container komen hierdoor ook terecht in kibana.

6.6. AVG

Om de applicatie zo veel mogelijk te bouwen volgens de AVG-regeling zijn de volgende acties ondernomen.

Privacy by design
Een voorbeeld van privacy by design is het pseudonimisering van het Burgerservicenummer. Hierbij wordt een Burgerservicenummer (BSN) vervangen door een UUID. Zo wordt het lastiger om de pseudonimisering van het BSN te herleiden naar een persoon.

De api wordt beschermd door middel van JWT-tokens. Gebruikers kunnen alleen bij de data die de api serveert wanneer ze zijn ingelogd via de zorg IDP. Zo kunnen niet geauthentiseerde gebruikers niet bij de data die staat opgeslagen.

Privacy by default
Gebruikers mogen ook niet standaard bij alle informatie wanneer ze zijn ingelogd. Ouders mogen alleen bij de dossiers waar ze toegang voor hebben gekregen. Deze toegang zal altijd door een andere partij gegeven aan een gezaghebbende ouder moeten worden, waaronder een zorgverlener binnen de jeugdzorg.

Data Minimalisatie
Om zo min mogelijk persoonsgegevens op te slaan, worden alleen de persoonsgegevens opgeslagen die nodig zijn voor de applicatie, zie AVG regel, sla alleen de benodigde gegevens op. Hieronder vallen de voor-, achternaam en geboortedatum van het kind. De voor- en achternaam van een geregistreerde ouder die bij het dossier mag. Het dossier ID en tot slot het Van Wiechenonderzoek zelf.

- De voor- en achternaam van het kind worden opgeslagen zodat het kind geïdentificeerd kan worden wanneer de applicatie geen toegang heeft tot het dossier.
- De geboortedatum van het kind wordt opgeslagen zodat de leeftijd van het kind kan worden vastegesteld.
- Het dossier ID wordt opgeslagen zodat wanneer een ouder het Van Wiechenonderzoek invult het bij het juiste dossier/kind wordt opgeslagen.
- De voor- en achternaam van de ouder wordt opgeslagen om de ouders met toegang te kunnen identificeren.

7. Conclusie

Geconcludeerd kan worden dat Van Wiechen onafhankelijker gemaakt kan worden van het KD+ en Mijn Kinddossier. De oplossing die is bedacht past binnen het applicatielandschap van Topicus en is opgebouwd met technieken waar binnen Topicus gebruik van maakt. Hierdoor is de kans op acceptatie en doorontwikkeling door Topicus zelf vergroot.

Uit het proof of concept is gebleken dat er een oplossing gebouwd kan worden die onafhankelijk is van het KD+ en Mijn Kinddossier, zie requirement B5 (4. Requirements). Doormiddel van het toepassen van het cloud native concept microservices kon zo de oplossing zo onafhankelijk mogelijk gebouwd worden.

Web components maken het mogelijk om de Van Wiechen module te implementeren binnen en buiten Wicket, zie requirement B6 (4. Requirements). Door het gebruik van web componenten kunnen de features van de Van Wiechen app bij het KD+ en Mijn Kinddossier geïmplementeerd worden met zo min mogelijke belasting op hun codebase.

Met behulp van dit proof of concept is gebleken dat het niet alleen mogelijk is, maar ook haalbaar om een Van Wiechen module te bouwen die niet alleen AVG technisch klopt, maar ook binnen en buiten de huidige applicaties, het KD+ en Mijn Kinddossier, werkt. Dit kan behaald worden met behulp van concepten zoals cloud native en web componenten. Zo is er een proof of concept opgezet die voldoet aan de volgende requirements: B1, B2, B3, B5 en B6 (zie hoofdstuk 4: requirements).

De gekozen oplossing blijkt niet de optimale oplossing te zijn. Dit is omdat er gaandeweg suboptimale keuzes zijn gemaakt tijdens het ontwerpen en implementeren van de oplossing. Zo is tijdens de ontwikkeling GraphQL geïmplementeerd conform een monolitische applicatie in plaats van een microservice architectuur, welke beter past bij een cloud native applicatie. Verder is de applicatie ook niet helemaal gebouwd volgens de twelve factor application. Hierdoor zijn er nog wel aanpassingen die plaats kunnen vinden. Hoe deze problemen opgelost kunnen worden staat in het volgende hoofdstuk, hoofdstuk 8: advies, beschreven

8. Advies

Voor Topicus is het van belang om te weten wat de vervolgstappen zijn voor het ontwikkelen van de Van Wiechen Module. Daarom worden er in dit hoofdstuk verschillen aanbevelingen en adviezen gegeven over de doorontwikkeling.

Om de oplossing zo cloud native mogelijk te maken zal er een library gebouwd moeten worden voor de gedeelde componenten. Op het moment worden de web componenten gebouwd in het front-end van de applicatie en worden de artefacten gekopieerd naar het Wicket framework. Volgens de twelve-factor app, factor 2: dependencies, zal alle code die gedeeld wordt binnen twee services moeten worden gebundeld in een package en via een depency declaration manifest zoals package.json of gradle wordt toegevoegd aan de applicatie. Zo hoeven er geen handmatige acties worden uitgevoerd om de laatste versie van de web componenten te gebruiken en is de app ook niet afhankelijk van een service die online moet draaien.

De user interface op het moment is nog zeer minimaal. Dit zorgt ervoor dat de applicatie er niet professioneel uit ziet en de gebruikservaring niet optimaal is. Er zou een UI overhaul plaats moeten vinden, met behulp van een user experience en user interface designer. De UI zal dan mobile first ontworpen moeten worden, zie requirement S8 (hoofdstuk 4. requirements). Dit houdt in dat de UI eerst voor mobiel wordt ontworpen en dan voor devices met grotere schermen.

Voor een betere seperations of concern wordt er aangeraden om Apollo federation te implementeren. Op het moment is de software zo gebouwd dat er een monolitische graph is. Met behulp van Apollo federation wordt het mogelijk om bij een (micro)service alleen het deel van de graph te implementeren waar hij verantwoordelijk voor is,

Met KGraphQL, de huidige library die gebruikt wordt om GraphQL in Kotlin te implementeren, is het waarschijnlijk niet mogelijk om gebruik te maken van federation. Als federation geïmplementeerd moet worden zal de API omgeschreven moeten worden met behulp van een andere library die wel federation ondersteund. De GraphQL Kotlin library is een voorbeeld van een library die federation wel ondersteund.

9. Reflectie

De uitgevoerde opdracht was erg groot, omdat er verschillende onderzoeken uitgevoerd moesten worden op complexe vraagstukken. Hierdoor is er veel tijd gaan zitten in het onderzoeken naar deze vraagstukken. Er is ook veel tijd komen zitten in het leren werken met technologieën, zoals Ktor, Apollo en GraphQL, maar ook het toepassen van nieuwe concepten zoals web components, microservices en cloud native. Dit heeft als voordeel dat de gebouwde applicatie doorontwikkeld kan worden door de teams binnen Topicus. Het nadeel is dat de ontwikkeling langzamer gaat aangezien er geleerd moest worden hoe er met deze technologieën gewerkt moest worden.

Daarnaast zijn het KD+ en Mijn Kinddossier grote en complexe applicaties die gebouwd zijn met een verouderd framework, het Wicket framework. En omdat de oplossing ook moest zowel binnen deze applicaties als buiten de applicaties moest er onderzoek gedaan worden naar hoe de architectuur van deze applicaties eruitziet en hoe de potentiële oplossing hiertussen paste. Daarnaast moest er ook begrepen op een basisniveau worden hoe het Wicket framework werkte.

Doordat de focus van de eerste twee sprints lag in het implementeren van de functionaliteiten binnen het KD+ en het Mijn Kinddossier is er veel tijd gaan zitten in het ontwikkelen van functionaliteiten die hierop aansluiten. Daarna is de focus naar het ontwikkelen van de Van Wiechen app gegaan. Hierdoor is er een beetje gewerkt aan verschillende functionaliteiten die nog niet helemaal geïmplementeerd zijn. Maar hierdoor kan er wel aangetoond worden dat de oplossing een alleenstaande applicatie kan zijn maar ook binnen het KD+ en Mijn Kinddossier kan werken. Het was handiger om de scope van de opdracht te focussen op alleen de app om zo een goede basis te kunnen leggen. En later met deze basis het ook te kunnen implementeren binnen het KD+ en Mijn Kinddossier.

Literatuurlijst

(Elektronisch) inzage verlenen. (2021). Opgehaald van AVG-Helpdesk: https://www.avghelpdeskzorg.nl/onderwerpen/rechten-van-betrokkenen/elektronisch-inzage-verlenen

AVG-Helpdesk. (2021). *Wabvpz*. Opgehaald van AVG-Helpdesk: https://www.avghelpdeskzorg.nl/onderwerpen/wabvpz

Beatrix, A. K. (2020, Juli 1). *Wet aanvullende bepalingen verwerking persoonsgegevens in de zorg*. Opgehaald van Overheid.nl Wettenbank: https://wetten.overheid.nl/BWBR0023864/2020-07-01

Breejen, A. d. (2021, Juni 10). *Privacywetgeving AVG, wanneer ben je compliant?* Opgehaald van KVK: https://www.kvk.nl/advies-en-informatie/wetten-en-regels/privacywetgeving-avg-wanneer-ben-je-compliant/

CloudFlare. (2021, September 15). Opgehaald van Wkipedia: https://en.wikipedia.org/wiki/Cloudflare

CloudFlare. (2021). *What is a CDN?* | *How do CDNs work?* Opgehaald van CloudFlare: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

Digitaal Dossier Jeugdgezondheidszorg. (sd). Opgehaald van ddjgz: https://www.ddjgz.nl/

Exposed. (2021, September 4). *DAO*. Opgehaald van Exposed Wiki: https://github.com/JetBrains/Exposed/wiki/DAO

Exposed. (2021, September 4). *DSL*. Opgehaald van Exposed Wiki: https://github.com/JetBrains/Exposed/wiki/DSL

Foundation, C. N. (2021). *Cloud native definition*. Opgehaald van cncf: https://www.cncf.io/about/who-we-are/

Jeugdgezondheid, N. C. (2021). *Van Wiechen-kenmerken*. Opgehaald van Nederlands Centrum Jeugdgezondheid: https://www.ncj.nl/van-wiechen/kenmerken/

Jeugdinstituut, N. (2021). *Van Wiechenonderzoek (VWO)*. Opgehaald van nji: https://www.nji.nl/instrumenten/van-wiechenonderzoek-vwo

Microsoft. (2021, januari 19). *Introduction to cloud-native applications*. Opgehaald van Microsoft: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduction

Nederlands Centrum Jeugdgezondheid. (2020, July 17). *Handleiding digitale inzage in het digitaal dossier jeugdgezondheidszorg.* Opgehaald van Nederlands Centrum Jeugdgezondheid: https://assets.ncj.nl/docs/416f7354-d0d3-4a6f-85ff-474b2949cc0c.pdf#page=8&zoom=100,96,492

Patiëntfederatie. (2021, Februari 15). *Landelijk Schakelpunt (LSP)*. Opgehaald van Patiëntfederatie Nederland: https://kennisbank.patientenfederatie.nl/app/answers/detail/a_id/471/~/landelijk-schakelpunt-%28lsp%29

Persoonsgegevens, A. (2021). *Introductie AVG*. Opgehaald van Autoriteit Persoonsgegevens: https://autoriteitpersoonsgegevens.nl/nl/onderwerpen/algemene-informatie-avg/algemene-informatie-avg

Rijksoverheid. (2021). Opgehaald van Burgerservicenummer (BSN) in de zorg: https://www.rijksoverheid.nl/onderwerpen/privacy-en-persoonsgegevens/burgerservicenummer-bsn/bsn-in-de-zorg

TNO. (2018). *Volg de ontwikkeling van kinderen met de Van Wiechenfilmpjes*. Opgehaald van TNO: https://www.tno.nl/nl/aandachtsgebieden/gezond-leven/roadmaps/youth/diy-van-wiechenfilmpjes/

Topicus. (2021). *JeugdgezondheidsZorg*. Opgehaald van Topicus: https://topicus-screening-prevention.com/oplossingen-voor/jeugdgezondheidszorg

Van Zoonen, R., Vlasblom, E., Lanting, C., De Wolff, M., Van der Pal, S., & Schönbeck, Y. (2021). *Implementatieplan Van Wiechenfilmpjes.* TNO.

Wiggens, A. (2017). *The Twelve-Factor App*. Opgehaald van The Twelve-Factor App: https://12factor.net/

Wigmore, I. (2016, May). *monolithic architecture*. Opgehaald van What Is: https://whatis.techtarget.com/definition/monolithic-architecture

Wikipedia. (2021). *Algemene verordening gegevensbescherming*. Opgehaald van Wikipedia: https://nl.wikipedia.org/wiki/Algemene_verordening_gegevensbescherming

Wikipedia. (2021). *Amazon Relational Database Service*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Amazon_Relational_Database_Service

Wikipedia. (2021). *Amazon S3*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Amazon_S3

Wikipedia. (2021). *Amazon Web Services*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Amazon_Web_Services

Wikipedia. (2021). *Kubernetes*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Kubernetes