Dataopslag Vermissingen in Sarea

Verantwoordingsverslag

10.2 e.

Versie 1.0
04 januari 2021

Gegevens

Afstudeerder: 10.2.e.
Opleiding: Hanzehogeschool, HBO-ICT, Software Engineering
Studentnummer: 10.2.e.

Afstudeerbedrijf: Innovatiehuis Politie Noord-Nederland
Digital Society Hub - Zernikepark 10 – Groningen
10.2.eg @politie.nl

Bedrijf begeleider: 10.2.e.
10.2.e. @sarea-samenzoeken.nl
06 10.2.e.

Docenten: 10.2.e.
10.2.e. @pl.hanze.nl
06- 10.2.e.

10.2.e.

10.2.e.
10.2.e. @pl.hanze.nl

Inhoudsopgave

1 Inleiding

In Nederland worden er per jaar 40.000 mensen bij de politie als vermist opgegeven (Slachtofferhulp Nederland, 2015). Slechts een klein deel hiervan is langdurig vermist. Bij een vermissing kan het zijn dat er een zoekactie wordt opgezet. De meeste mensen missen kennis en ervaring in het zoeken naar vermisten en het organiseren van zoekacties. Hier is waar het initiatief Sarea hulp wil bieden.
Sarea is een app die burgers moet gaan helpen bij het opzetten van zoekacties naar vermiste personen. Voor het ontwikkelen van de app wordt samengewerkt met 10.2.g Vierdejaars Hanzestudenten van de opleiding software engineering hebben een nieuw prototype voor Sarea ontwikkeld. Een gewenste functionaliteit van Sarea is om de gebruikers gebieden aan te wijzen met een grote kans om de vermiste terug te vinden. Er is door 10.2.g een algoritme gemaakt om deze locaties te bepalen. Dit algoritme is gebaseerd om heuristiek. De wens is om een algoritme te maken dat gebaseerd is op data. Er is echter een probleem voor het realiseren van dit algoritme. Op dit moment is er geen geschikte databron om het algoritme mee te ontwikkelen. Om dit algoritme te realiseren is een gestructureerde set aan vermissingsdata nodig. Het doel van dit project is het verwezenlijken van een opslagmethode voor vermissingsdata waarmee het zoekalgoritme ontwikkeld kan worden.
In dit document zal worden behandeld hoe deze opslagmethode is gerealiseerd door de student binnen het afstudeerproject. De onderdelen van dit project die verderop besproken worden zijn de projectaanpak, het opstellen van de requirements, het onderzoek dat is gedaan naar databases het ontwerp en het realiseren van de applicatie.

2 Project aanpak

In dit hoofdstuk zal een korte beschrijving worden gegeven van hoe het project is gestructureerd.
In het begin is er een plan van aanpak geschreven. Dit projectplan is na enkele iteraties goedgekeurd. Het project is in vier grote fasen onderverdeeld. Een nieuwe fase begint op het moment dat de voorgaande is afgerond. Deze fasen zijn:

1. Requirements opstellen
2. Dataopslag onderzoeken
3. Applicatie ontwerpen
4. Ontwerp realiseren.

De eerste drie van deze fasen volgen een klassieke watervalstructuur. De realiseerfase daarentegen maakt gebruik van agile. Terwijl de realisatie voortborduurt op het ontwerp, is er ruimte gegeven voor feedback om de richting van de applicatie bij te sturen. Het iteratief werken bij het realiseren moet ook ruimte geven om technische problemen te verbeteren.

3 Requirements

Als eerste grote stap binnen het afstudeertraject zijn de requirements opgesteld. De resultaten hiervan zijn te lezen het requirements bestand. Voor het opstellen van requirements zijn verschillende informatiebronnen gebruikt. Als eerste zijn er de initiële eisen van de opdrachtgever en 10.2.g opgesteld. Hier is de richting bepaald van wat de applicatie moet kunnen, namelijk het invoeren, opslaan en beheren van vermissingsdata. Hierbij moet de dataset uitbreidbaar zijn omdat er geen definitief antwoord is op de vraag welke informatie wel of niet relevant is om op te slaan. Er zijn wel een aantal gegevens die in elk geval opgeslagen moeten worden. Hieronder vallen het geslacht van de vermiste, het moment van de vermissing en locaties waarop de vermiste is gezien.
Een tweede bron van variabelen om op te slaan komt uit documenten. Er is een onderzoeksrapport van 10.2 g gebruikt waarin een versie van het zoek algoritme werd getest. Hierbij is een beperkte set aan gegevens gebruikt die willekeurig gegenereerd waren. Een tweede document is een lijst aandachtspunten die agenten moeten noteren bij het invoeren van een vermissing.
Als vervolg is er twee agenten gevraagd naar hoe zij te werk gaan bij meldingen van een vermissing. Met een agent is het huidige systeem bekeken en daarbij hoe er op dit moment verschillende zaken worden teruggevonden. Hierin is te zien dat alles wordt opgeslagen als rapport, en deze rapporten zijn terug te vinden door het meegeven van sleutelwoorden die in de tekst voorkomen.
De laatste invoer is door het stellen van vragen. Een voorbeeld van deze vragen is hoe de opdrachtgever data beveiliging voor zich ziet. Hier is onder andere de tweestrijd voorgelegd over gebruiksvriendelijkheid en veiligheid. Als er geen account nodig is om in te loggen dan moet het verkeer tot een intern netwerk worden beperkt en kan 10.2.g hier niet bij zonder op de locatie van de politie te werken. Als er met accounts wordt gewerkt dan moet iedereen een extra wachtwoord onthouden. Wanneer er één account door iedereen wordt gedeeld dan vallen er geen restricties meer op te leggen bij individuele gebruikers. Hier is het antwoord vanuit de opdrachtgever gekomen om een gebruiker accounts te gebruiken.

Er zijn voor het rondmaken van de requirements meer gedetailleerde vragen gesteld die betrekking hebben tot wat er moet komen. Hierbij zijn is naast functionaliteit ook naar limieten gekeken. De gekozen oplossingsrichting is een website voor invoer. Als gevolg hiervan moet worden gekozen binnen welke webbrowsers de website tenminste moet functioneren. Bij navraag is gebleken dat bij de politie zowel de computers als smartphones Google Chrome geïnstalleerd hebben staan. Hierom is Google Chrome gekozen als minimum van webbrowsers die ondersteund moet worden. Wel is hierbij de wens gekomen om het functioneren niet te beperken tot enkel Google Chrome.

De uiteindelijke requirements zijn na het opstellen besproken met de opdrachtgever. Op deze lijst aan requirements is een MoSCoW analyse gedaan. De eerste prioritering is gedaan door de student op basis van wat essentieel is om technisch te functioneren en het belang dat de opdrachtgever op verschillende onderdelen legt. Deze prioritering is met de opdrachtgever besproken en aangepast op basis van zijn inzichten en belangen. Na het bereiken van een overeenstemming is verdergegaan met het volgde onderdeel van het project.

Tijdens het opstellen van de requirements is er een mogelijk probleem aan het licht gekomen. Dit probleem is dan ook direct aan de opdrachtgever voorgelegd. Het probleem is dat de beoogde gebruikers (politieagenten) geen interesse hebben in het gebruik van de applicatie. Als dit systeem parallel moet worden ingevoerd naast het huidige dan is er een kans dat dit systeem wordt verwaarloos omdat agenten niet alles twee keer in willen vullen.

4 Onderzoek

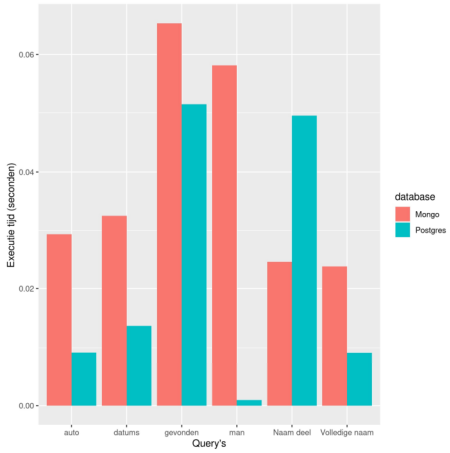
In het stagetraject is er een onderzoek gedaan naar verschillende databasen. De reden voor onderzoek naar databasen is dat dataopslag de kern is van de opdracht.
Voor het onderzoek zijn twee vragen gesteld om te beantwoorden. De eerste vraag is wat voor datamodel past het best bij de gegevens die moeten worden opgeslagen over vermissingen, en de tweede vraag is welke database presteert beter wanneer er meerdere records worden opgeslagen. Voor dit onderzoek is er meer interesse in de verschillen tussen datamodellen dan in snelheidsverschillen tussen de implementaties hiervan. Hierom is er maar één van de vele SQL-databases gebruikt.
Als eerste stap is gezocht naar de verschillende soorten SQL en NoSQL databasen. Van deze verschillende soorten is nagegaan of deze toegevoegde waarde hebben om verder te gaan vergelijken. De uiteindelijke soorten databasen die zijn gekozen om verder te vergelijken zijn de relationele-, document- en graaf databasen. Een verder overzicht van de verschillende soorten is te zien in het *onderzoeksverslag*.
Voor het selecteren van een uiteindelijke database is één extra restrictie meegegeven, namelijk dat de database gratis te gebruiken moet zijn. Dit betekent dat commerciële databasen zoals Oracle en Microsoft SQL Server niet gebruikt zullen worden. Van de verschillende databasen zijn de graaf en document database gekozen op basis van hun populariteit. Hierom zijn de databasen MongoDB en Neo4J gekozen. In het geval van de SQL-database is naast populariteit ook op persoonlijke kennis en ervaringen afgegaan. Hierom is PostgreSQL gebruikt in plaats van het voor websites populaire MySQL/MariaDB (MariaDB is een afsplitsing MySQL met als doel dezelfde functionaliteit te blijven bieden). De reden achter deze keuze is dat PostgreSQL meer functionaliteit ondersteunt dan MySQL zoals array's en UUID-velden. Volgens de PostgreSQL ontwikkelaars is er geen database die zegt alle SQL functionaliteit te ondersteunen.

Voor het kunnen testen van de database op prestaties moet er data worden ingevoerd. Er is gekozen om voor 100.000 vermissingen te gaan. Als de applicatie voor alle vermissingen die bij de politie binnen komen wordt gebruikt dan staat dit gelijk aan de hoeveelheid data van 2 1/2 jaar. Omdat het met de hand te veel is om de beoogde 100.000 vermissingen te maken en in te vullen is hier een generator voor gemaakt. Deze generator is geschreven in JavaScript en maakt een aangegeven hoeveelheid dummy data aan. De reden dat dit in JavaScript is geschreven is vanwege bekendheid met de taal. De data voor alle testscenario's wordt in een keer gemaakt om te voorkomen dat er prestatie verschillen zijn door verschillen in de datasets, waarbij de verhoudingen gunstiger uitpakken voor één van de databasen. Om van het gegenereerde object aan dummy data naar de database te komen zijn verschillende backends geschreven. Elke backend zet de data om in een query om de database mee te kunnen vullen.

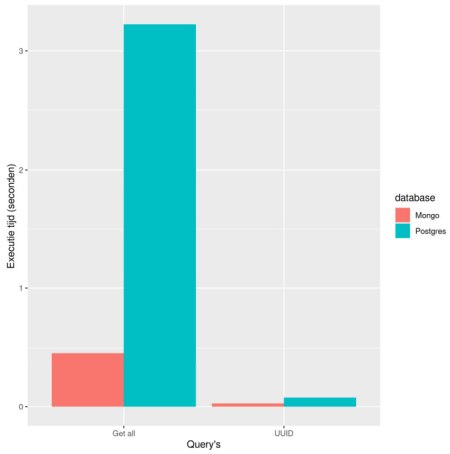
Tijdens dit proces is besloten om niet verder te gaan met het testen van Neo4J. Wanneer nodes werden verbonden is het niet gelukt om de queries snel genoeg te krijgen. Het is een geval van uren wachten op 1000 vermissingen invoeren tegenover PostgreSQL en MongoDB die binnen vijf minuten 100.000 kunnen verwerken. Het heeft te veel tijd gekost om te proberen Neo4J op snelheid te krijgen zonder succes.

In de uiteindelijke resultaten waren de prestaties in lijn met de verwachtingen. Het ophalen van data wordt in minder tijd uitgevoerd door MongoDB dan PostgreSQL, waarschijnlijk omdat MongoDB geen join-operaties gebruikt. Daar tegenover staat dat PostgreSQL met de striktere datastructuur beter is in het doorzoeken van de data. Bij het zoeken op geslacht was het grootste verschil te zien doordat hier een enumeratie type was gebruikt. Overal had geen van beide prestaties waarop deze afgekeurd zou kunnen worden. Hierdoor is er meer waarde gekomen op de voor- en nadelen van de datastructuur van de databases

Figuur 1: Tijden voor het doorzoeken van de database



Figuur 2: Hele vermissing laden



Een doorslaggevende reden is dat na het uitvoeren van de testen de realisatie is gekomen dat er geen veel-op-veel relatie was opgenomen. Mocht dit wel zijn gedaan dan had MongoDB een minder logisch datamodel

gekregen. PostgreSQL is gekozen als database om te gebruiken omdat het meer mogelijkheden biedt voor het weergeven van data.

5 Ontwerp

In dit hoofdstuk wordt het ontwerpproces besproken en redenaties achter de keuzes. Voor het ontwerp is er gekozen geen implementatie details op te nemen. Dit is gedaan vanwege een gebrek aan ervaring met de server programmeertaal en het frontend framework. Als een structuur zou worden opgeschreven zal deze tijdens het implementeren naar alle waarschijnlijkheid volledig gewijzigd worden. Wat wel is meegenomen zijn de gebruikersinterface, web API, databasemodel, beveiligingsoverwegingen en deployment methode.

5.1 Ontwerp richting

De basis voor het ontwerp is een JSON web API, database en web frontend. Er is een centrale database en server om alle data gecentreerd te kunnen bewaren in plaats van deze te verspreiden over vele verschillende computers. De server levert een web API waar de frontend gebruik van maakt. Hierdoor is de frontend niet direct afhankelijk van de database. Dit is te zien in figuur 3.

Figuur 3: Applicatie lagen

Een tweede en belangrijkere reden is zodat input validatie kan worden gedaan op de ingevoerde data en de gegevens die terug worden gestuurd gestructureerd kunnen worden. Hiermee kan aan de eis worden voldaan dat de vermissingsgegevens beschikbaar zijn in JSON.

De server heeft een REST geïnspireerd ontwerp maar is niet volledig RESTful. In tabel 1 staat een overzicht van welke REST eigenschappen wel en niet zijn gebruikt. Een voordeel van een REST architectuur is dat het makkelijker wordt om op te schalen in de toekomst door het aanzetten van een extra server.

Tabel 1: RESTful eigenschappen

Feature	**Meegenomen in de server**
Uniform interface	✓
Client–server	✓

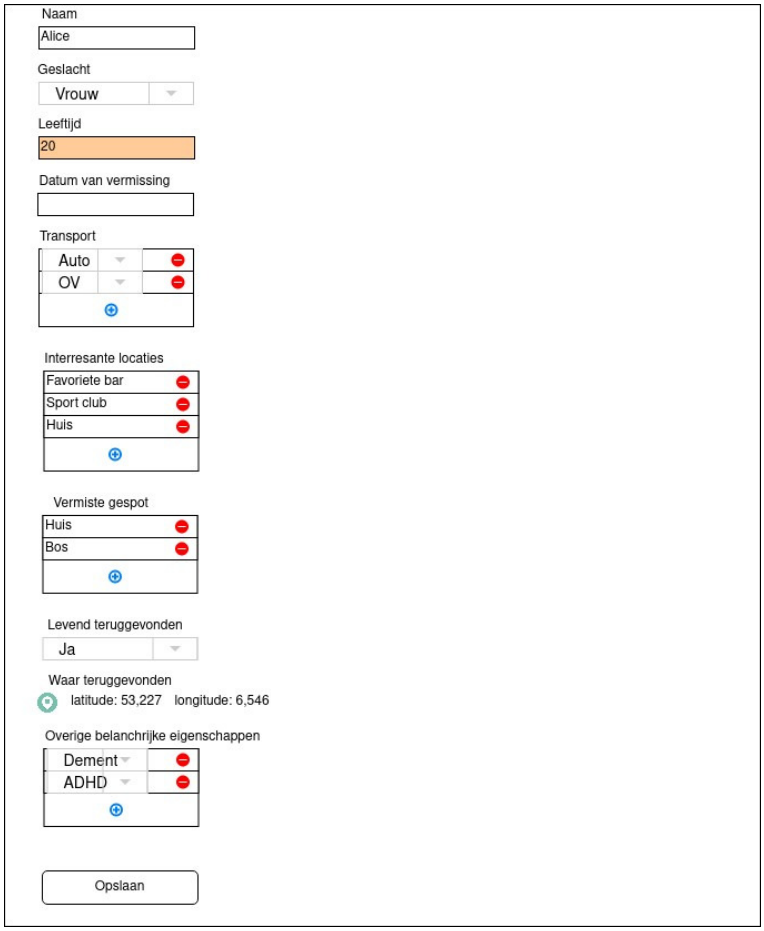
Stateless	✓
Cacheable	X (is mogelijk maar niet *geïmplementeerd*)
Layered system	X (monolithic design)
Code on demand (optional)	X

Als laatst is er een single page application frontend met progressive web app (PWA) functionaliteit. Het maken van een single page application ging het best samen met de wens voor een PWA. Er is hiermee de mogelijkheid om informatie invoer te doen over meerdere pagina's zonder te moeten wachten op een server die de status bewaard.

Elk component in deze structuur is onafhankelijk van de erboven gelegen laag. Hierdoor kan de API worden gebruikt om data op te halen en op te slaan vanuit een andere applicatie zonder dat er aanpassingen nodig zijn.

5.2 Gebruikersinterface

Voor het bepalen van de gebruikersinterface zijn mockups gemaakt. Voor het design is veel geleend van Google's material design (Google, 2020). Dit geeft een bekende basis om vanuit te werken en geeft toegang tot verschillende bestaande material design bibliotheken.
In de mockups zijn alle pagina's gemaakt waar gebruiksinteractie mee is. Een van deze mockups is te zien in figuur 4. Voor het invullen van vermissingen is geprobeerd een balans te vinden tussen de pagina's overzichtelijk houden en de hoeveelheid navigatie te beperken.
De mockups zijn aangepast op basis van twee bronnen. Als eerst is er feedback geweest vanaf de opdrachtgever over het hebben van bepaalde informatievelden. Ten tweede zijn de mockups met een CMD student doorgelopen om feedback te krijgen op het design. Hieruit zijn kleine dingen gekomen over het vormgeven en plaatsen van sommige object op het scherm. Een feedbackpunt van de CMD-student dat niet verder is meegenomen is het vullen van de lege stukken met een afbeeldingen.
Figuur 4: Vermissing invoer mockup



5.3 Gebruikte backend programmeertaal

Voor de gebruikte technologieën was het moeilijk om een keuze te maken. Dit komt doordat er geen duidelijkheid is over wie hierna gaat werken aan het project. Dit betekend dat niet bekend is wat de vaardigheden van de volgende ontwikkelaar zijn en hier dus geen rekening mee kan worden gehouden. Een eigen criteria dat is gebruikt voor het kiezen tussen de programmeertalen is betrouwbaarheid. Een concrete eis aan de hand van dit criteria is om type errors op compile time te kunnen vangen. Dit moet de kans op runtime errors in een eventuele productieomgeving verlagen. Een ander criteria dat vanaf de kant van 10.2.g is gekomen is dat er makkelijk mee verder te werken is. De uiteindelijke keuze voor de backend programmeertaal is Go. Hieronder is meer te lezen over de overwegingen.

Als startpunt zijn alle programmeertalen genomen waar de student eerder code in heeft geschreven. Hierbij voldoen alle talen waarin meer dan de standaard hello world is geschreven. Als eerst zijn de geïnterpreteerde talen weggehaald om runtime errors door type errors tegen te gaan. Hierdoor vallen veel van de populaire keuzes af zoals python, JavaScript en PHP. Als een transpiled language valt TypeScript ook hieronder. TypeScript heeft type annotaties maar omdat het transpiled naar JavaScript zijn deze garanties op runtime niet meer aanwezig. De server logica kan worden

gevalideerd door TypeScript maar als er data wordt gestuurd anders dan verwacht dan vallen alle garanties om en is het gelijk aan JavaScript.
Tussen de gecompileerde talen is er onderscheid gemaakt tussen garbage collected en manual memory management talen. In de categorie van manual memory management vallen C, C++ en Rust. Hiervan zullen C en C++ niet gebruikt worden omdat er te makkelijk memory bugs ontstaan. Volgens Cimpanu zijn 70% van alle security bugs bij Microsoft veroorzaakt door memory management problemen. Google en Mozilla melden vergelijkbare nummers. Rust heeft een compiler die controleert op deze problemen waardoor deze maar weinig voor kunnen komen.
Vervolgens zijn er de talen met een garbage collector namelijk Java, C# en Go. Java en C# zijn erg gelijkwaardig. Beiden hebben een groot aanbod aan bibliotheken om te gebruiken en uitstekende IDE ondersteuning. Met de komst van .net Core is C# net als Java cross platform geworden. Er is hierna verdere focus gelegd op de vergelijking met C# omdat het naar eigen ervaring een betere versie van Java is om te programmeren. Een van deze verbeteringen is LINQ. LINQ is een query syntax van C# om door datastructuren te filteren.

Na de eerste overwegingen waren de overgebleven talen Rust, Go en C#. Elk van deze talen heeft een andere sterke kwaliteit om deze te overwegen. Voor Rust is het de strenge compiler die afdwingt om error states op te lossen waardoor er geen dataraces of null pointer exceptions ontstaan. De kwaliteit van Go is dat het makkelijk te leren is en hierdoor eenvoudiger voor een volgende om mee verder te gaan. De kracht van C# is dat het een bekende bewezen oplossing is voor veel applicaties. Nu zullen deze drie talen uitgebreider worden behandeld.
Als eerste is er Rust. Rust is de jongste van de drie talen waarvan de eerste stabielere versie in 2015 uitkwam. Sinds deze tijd is Rust elk jaar verkozen tot de meeste geliefde programmeertaal in de stackoverflow developer survey (Stackoverflow, 2019). De belofte van Rust is extreem snelle applicaties kunnen maken zonder bang te hoeven zijn voor het veroorzaken van memory safety problemen of dataraces. Rust heeft het krachtigste type system van de drie talen. Onderdeel hiervan is een ergonomische manier om met error waarden om te gaan. Verder heeft Rust Cargo als standaard voor build en dependency management wat veel wegheeft van npm. Een meegekregen voordeel van Rust is dat het potentieel sneller is dan alternatieven gelet op web framework benchmarks (TechEmpowered, 2020). Rust kent echter enkele nadelen. Als eerste zijn de compileer tijden lang. Het is vergelijkbaar met C++ op dit gebied. Als tweede is het dependency landschap nog jong met missende bibliotheken en meer wijzigingen in de bestaande. Verder is de er geen IDE ervaring op het niveau van Java en C#. De bestaande oplossingen hebben meer fouten en minder functionaliteit. De enige IDE debugger is zit in het betaalde product CLion. Als laatste kost het meer tijd om Rust te leren dan andere talen.

Als tweede is er de programmeertaal Go. Go is ontwikkeld door Google als alternatief voor C++ en beschikbaar gekomen in 2009. Het is bedoeld voor het schrijven van simpele, betrouwbare en efficiënte software. Go biedt een goede onboarding tutorial en is daarmee in de tijd van een weekend te leren. De standaard bibliotheek biedt veel tools voor netwerken waardoor er minder afhankelijkheid is van externe dependencies op dat vlak. Voor het schrijven van een webserver is dan ook geen groot framework nodig. De standaard bibliotheek biedt niet alle gewenste functionaliteit voor een webserver maar meer dan andere talen. Wat verder in de taal zit is een eenvoudige manier voor parallelle executie met Goroutine. Een ander aspect dat al is meegeleverd met Go is een test framework. Een verdere hulp bij het werken met Go is dat de compiler foutmeldingen geeft bij sommige format fouten, wat een goede code formatting afdwingt. Als laatst is er een gespecialiseerde IDE van JetBrains beschikbaar genaamd GoLand.
Naast deze voordelen zijn er enkele nadelen om te benoemen. Het grootste directe obstakel met Go is het gebruik van dependencies. Go is oorspronkelijk ontwikkeld voor Google wat betekend dat alles in één monorepo zit. Hierom was het werken met externe dependencies geen prioriteit in Go en is het dependency systeem pas in 2019 geïntroduceerd. Een tweede nadeel is dat de taal functionaliteiten mist die wel beschikbaar zijn in andere talen. Het meest bekende voorbeeld hiervan is de afwezigheid van generics.

Als derde is er C#. C# is ontwikkeld door Microsoft als een alternatief op Java. Origineel was C# alleen beschikbaar voor Windows met het .net framework. Sinds de release van .NET Core in 2016 is C# crossplatform te gebruiken. C# heeft de best mogelijke IDE ondersteuning. Door de overeenkomsten is C# makkelijk te leren voor iemand die bekend is met Java. Voor het maken van webapps in C# is er het ASP.NET framework. Dit is een uitgebreid en welbekende manier voor het maken van webservers. Een nadeel is dat het een zeer groot framework is en daarmee lastig is om mee van start te gaan.

Na het afwegen van de voor- en nadelen van de talen is gekozen voor Go om de backend in te realiseren. In eerste instantie is er alleen rekening gehouden met het gebruik van twee dependencies voor de backend. Dit zijn een database driver en een routing bibliotheek.

5.4 Frontend

Omdat de frontend een website is zijn er weinig keuzes in programmeertalen om te gebruiken. Er is gekozen om TypeScript te gebruiken voor de frontend in plaats van JavaScript. TypeScript is een super set van JavaScript wat betekend dat alles dat mogelijk is in JavaScript ook met TypeScript kan worden gedaan. Het voordeel van TypeScript zijn de type annotaties waardoor de compiler kan beredeneren over type fouten. Dit

kan fouten opsporen en programmeurs ondersteunen door overal te verduidelijken met welke data er gewerkt kan worden.

De grotere vraag voor het maken van de frontend was welk framework er gebuikt moest worden. Hiervoor is de focus initieel beperkt tot de drie grootste JavaScript frameworks op dit moment namelijk: Angular, React en Vue.
Als een hulpmiddel voor het kiezen van een framework is gebruik gemaakt van de state of js survey van 2019. In deze survey geeft 35,8% van de mensen aan Angular gebruikt te hebben en niet weer te willen gebruiken (Sacha Greif, 2019). Dit is reden om meer te focussen op de verschillen tussen React en Vue die een positieve beoordeling krijgen van ontwikkelaars.

Vue en React hebben veel overeenkomsten. Een verschil dat er wel is tussen React en Vue is hoe ze worden ontwikkeld. React is een Facebookproject waar Vue een community project is zonder bedrijf dat de regie heeft. Beide van deze situaties kan als positief of negatief worden beschouwd afhankelijk van je perspectief. Een tweede verschil is hoe de twee omgaan met essentiële dependencies om met de frameworks te werken. Vue heeft deze dependencies zoals state management en router als onderdeel van het project waar deze bij React complete gescheiden projecten zijn. Als laatst is er het verschil in hoe de programmeur de applicatie schrijft met deze verschillende frameworks. Bij React wordt alles in JavaScript bestanden geschreven en is de view vormgegeven met JSX. JSX is een extensie van de JavaScript syntax. Aan de andere kant deelt Vue alles op in een meer bekende manier voor websites met aparte HTML, JavaScript en CSS-bestanden, deze kunnen samen worden gevoegd in Vue bestanden. Als laatst is Vue aan de student aangeraden door drie verschillende personen. Er is gekozen om Vue te gebruiken voor de frontend. De doorslaggevende factor voor deze keuze is dat Vue makkelijker te begrijpen is voor iemand bekend met standaard web technologieën (HTML, CSS).

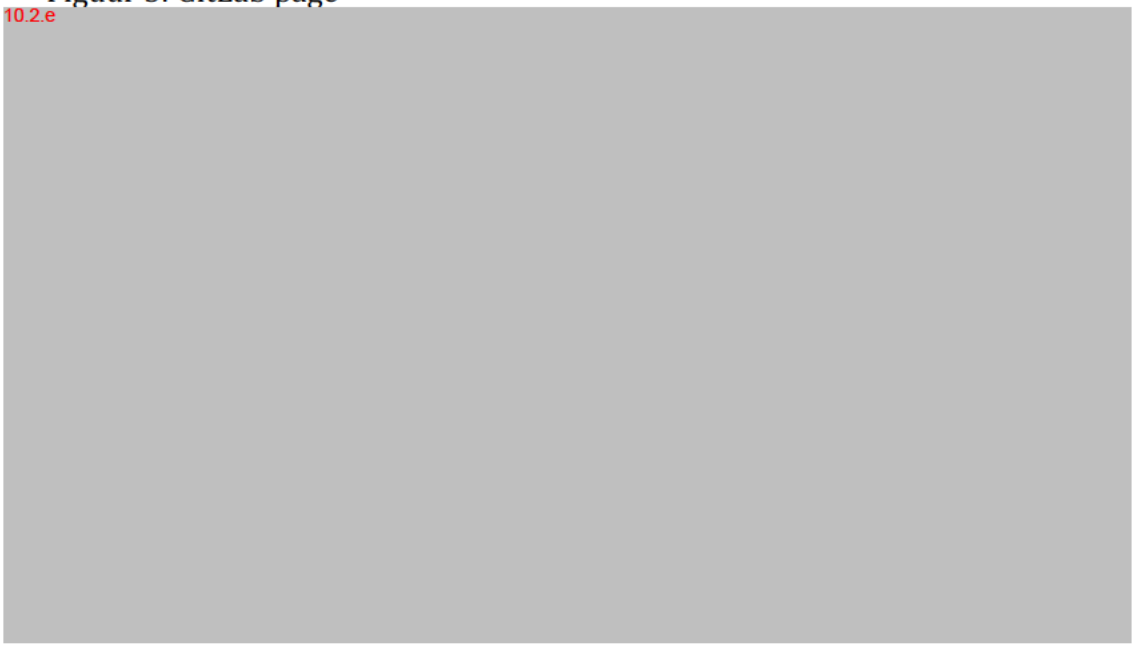
6 Realiseren

In dit hoofdstuk wordt het realiseren van het ontwerp besproken. Als eerst wordt de gebruikte ontwikkelomgeving beschreven. Daarna wordt beschreven hoe de applicatie is ontwikkeld. Tot slot wordt besproken hoe alles is getest en gedocumenteerd.

6.1 Ontwikkelomgeving

Er is voor het werken aan het project geen fysieke hardware beschikbaar gesteld door de politie dus alles is ontwikkeld en getest op hardware en software zoals die vooraf in bezit waren van de student. Dit houdt in dat de ontwikkelomgeving alleen bekend is te werken op de Linux distributie Fedora. Door het gebruik van eigen omgeving is ervoor gekozen om Podman en Podman Compose te gebruiken als container technologie voor de ontwikkelomgeving. Docker werkt op dit moment niet standaard op Fedora (dit heeft te maken met het verschil tussen cgoup versies 1 en 2).
De website zelf is getest onder Fedora in Firefox en Chromium en op Android in Firefox bèta (de bèta bevat een grote verandering die nog niet in de stabiele versie zit) en Google Chrome.
Als test server is een VPS gebruikt die beschikbaar is gesteld door de Hanze hogeschool. Dit is een VPS die draait op Azure met Ubuntu 18.04, 2 CPU cores en 4GB RAM. Op de VPS is de deployment met Docker en Docker Compose gedaan. Het doel van de VPS is het verifiëren van de Docker Compose bestanden voor deployment en voor demonstratie aan anderen.
Versiebeheer is gedaan met Git en GitLab voor het beheer van de online repository (figuur 5).
Figuur 5: GitLab page

10.2.e



6.2 Ontwikkelproces

Het realiseren van het product is in een iteratief proces gebeurt. Een groot deel van de server en website zijn herschreven om de kwaliteit te verbeteren en ruimte te maken voor nieuwe functionaliteit. Een van deze onderdelen is de interne datastructuur voor vermissingen van de website. Als eerst zijn de velden naam, leeftijd en geslacht op één pagina geplaatst met alle data lokaal bewaard. Deze data is vanaf daar naar een globale store verplaatst zodat het veranderen van pagina niet alle ingevoerde gegevens kan laten verdwijnen. Dit geeft de nodige functionaliteit om meerdere pagina's te gebruiken voor het invoeren van gegevens. Hierna is een tweede opslag gemaakt voor vermissingen die van de server zijn geladen. Binnen de vermissingspagina is logica toegevoegd voor het wisselen tussen de verschillende vermissingen gebaseerd op de URL. Met alle informatie opgeslagen in een globale store kan er gebruik worden gemaakt van meerdere pagina's om informatie in te vullen. Ook deze pagina's bevatten logica om te wisselen tussen de verschillende soorten vermissingen (bestaande en nieuwe). Dit geeft duplicatie van logica met als tussen resultaat dat geen van alle pagina's een goede implementatie heeft om met de verschillende soorten te werken. Hierom is er een adapter component gemaakt dat wordt geplaatst binnen de router. Deze adapter reageert op de parameters in de URL en werkt als middleman tussen de store en de webpagina. De pagina ontvangt de juiste vermissing state van de adapter en geeft update informatie terug die de adapter doorstuurt naar het juiste stuk data. In figuur 6 is te zien hoe de adapter op een component wordt gezet binnen de router.

Figuur 6: Adapter gebruik

Om data mutaties makkelijk door te laten voeren is er een impliciete interface die voor de nieuwe en bestaande vermissingsdata is geïmplementeerd. Om aan te geven wat voor mutatie moet worden doorgevoerd wordt naast de data een enum type meegegeven aan de adapter die het soort data aangeeft. Er zijn enums gebruikt in plaats van strings om bugs door typefouten en magische strings te voorkomen.

De JSON van de server wordt niet direct gebruikt door de website. Er zijn transformatie functies om de JSON om te zetten in de interne datastructuur. Het nadeel van deze aanpak is dat er meer logica moet worden toegevoegd bij het uitbreiden van informatievelden. De reden dat dit is gedaan is om de interne datastructuur los te koppelen van de JSON waarmee communicatie wordt gedaan. Deze twee konden hierdoor los worden doorontwikkeld.

Aan de server kant zijn er minder drastische iteraties geweest. De meeste iteraties zijn geweest in de create en update functies van vermissingen. Met elke iteratie waarbij nieuwe gegevens zijn toevoegen zijn deze bijgewerkt om deze gegevens te kunnen verwerken. Wanneer dit in een losse SQL-query moest gebeuren dan is deze logica met een refactor in een andere functie geplaatst om alles leesbaarder te houden. Het resultaat van het opsplitsen in functies is dat de hoofd create en update functies heel herhaaldelijk bestaan uit een repetitie van:

1. Roep functie aan
2. Als een error is teruggegeven zet de statuscode naar 500
3. Als een error is teruggegeven breek uit de functie met een return

Het bijwerken/aanmaken van een vermissing wordt binnen een transactie gedaan. Door het vroegtijdig afbreken van de functie worden de veranderingen binnen de transactie teruggedraaid.

Een andere verandering die later aan de server is toegevoegd is het gebruik van een logging framework. Na het toevoegen van functionaliteit in de grotere functies werd het lastig om terug te vinden waar de errors vandaan komen. Hierom worden errors zo dicht mogelijk bij de bron gelogd. Een grote bron van deze fouten had betrekking tot fouten in SQL-query's.

In de onderstaande afbeelding staat een meting van de grote van de code repository. De output is gegenereerd door Tokei 11.2. Tokei is een programma dat statistieken laat zien over de code. Het geeft verschillende nummers zoals de hoeveelheid regels code in een specifieke taal en de regels commentaar die daarbij zijn geplaatst.

Language	Files	Lines	Code	Comments	Blanks
Go	22	2864	2541	91	232
HTML	1	20	19	1	0
JavaScript	2	18	18	0	0
JSON	4	263	263	0	0
Markdown	3	386	386	0	0
SQL	24	89	88	1	0
SVG	2	150	150	0	0
Plain Text	1	2	2	0	0
TypeScript	16	1217	980	151	86
Vue	26	3343	2453	571	319
YAML	4	197	181	0	16
Total	105	8549	7081	815	653

Figuur 1tokei van de repository

6.3 Testen

Het testen van de applicatie is tijdens het project grotendeel handmatig uitgevoerd. Echter, richting het einde is er een verandering gekomen door de wens om alsnog integratietesten te doen. Deze integratietesten zijn gerealiseerd voor de server met de database. De integratietesten beslaan nu alleen het creëren en opvragen van vermissingen.
Voordat de integratietesten konden worden toegevoegd moest de structuur van de server worden aangepast. Tot op dat punt werd de hele server gebootstrapt binnen de main functie. Ook waren alle database connectie pools geïmplementeerd als een globale contanten.

Om de integratie testen geautomatiseerd te kunnen uitvoeren is een apart Docker Compose bestand gemaakt. Deze heeft vooroordelen over variabelen zoals het database wachtwoord omdat deze niet sterk hoeft te zijn voor een lokale test omgeving.

Het maken van de integratietesten heeft een aantal kleine bugs naar boven gebracht. Wanneer een nieuwe test faalde is de bug eerst verholpen. De test wordt hierna gebruik als verificatie dat de bug niet meer bestaat. Achteraf kijkend had het schrijven van de integratietesten vanaf het begin gemoeten. Dit had een snellere feedbackloop kunnen geven op veranderingen voor het vinden van bugs.

Verder kwaliteitscontrole is gedaan met behulp van linters. Voor de server is Gofmt uitgevoerd elk moment dat een bestand wordt opgeslagen. Aan de website kant is gebruik gemaakt van eslint. Omdat er geen vaste standaard is voor JavaScript en ook geen is aangeleverd is de eigen stijl gedefinieerd in de eslint instellingen. Hiervoor zijn de instellingen van een persoonlijk project als basis genomen.

Er is geen CI opgezet voor de testen en linters omdat het een eenpersoons project is. Als er meer tegelijk aan het werk waren geweest dan was dit wel nodig geweest.

6.4 Documentatie

De documentatie van de code is in de code zelf gedaan. Dit geeft een betere connectie tussen de twee zodat verschillen minder snel ontstaan. In het geval van de server kan er een programma worden uitgevoerd om documentatie pagina's te genereren voor het project en gebruikte dependencies. Voor inzicht in de API is er een markdown bestand gemaakt met daarin alle routes gedocumenteerd. Deze is te vinden in de server folder van de code repository. In figuur 7 is de beschrijving van de login route te zien.

Figuur 7: API markdown login route

Login

route: /login

method: POST

authentication required: no

request body:

```
{
    "email":    10.2.g
    "password":
}
```

7 Conclusie

In dit project is er gekeken naar het opslaan van vermissingsdata. Er is een systeem ontwikkeld om deze gegevens gestructureerd in op te slaan. Hiervoor zijn een database, een webserver en een PWA-website gemaakt.

Het ontwikkel traject is anders verlopen dan geanticipeerd. Binnen het proces werd Nederland getroffen door de pandemie net na het opstellen van de requirements. Na dit moment is er enkel vanuit huis gewerkt. Dit heeft het afstuderen vermoeilijkt door het niet langer in persoon kunnen spreken met verschillende belanghebbenden.
Achteraf kijkend heeft de vooraf opgezette projectstructuur goed gewerkt. Door de functionaliteit van de applicatie te laten bijsturen tijdens het realiseren zijn er nuttige veranderingen gekomen. Dit wil niet zeggen dat de executie zonden problemen is. Het proces had beter gekund bijvoorbeeld door nauwkeurig bij te houden welke veranderingen er zijn geweest en op welke momenten deze plaatsvonden.
Tijdens de ontwerpfase is er onderzoek gedaan naar verschillende databases. Hierin is gezien dat er veel verschillende soorten databases zijn. Bij het testen van de verschillende databases vertoonden deze grotendeels prestaties in lijn met de verwachtingen. De uitzondering hierop is de graafdatabase die te langzaam was om verder te kunnen testen. Helaas is er over het hoofd gezien om prestatie testen te doen met veel-op-veel relaties. Verder valt alleen niet te zeggen of de uitkomsten vergelijkbaar zouden zijn als er een andere selectie aan databases was gemaakt om te testen.

Het gerealiseerde product kan op dit moment worden gebruikt als beginstap voor het verder ontwikkelen van dit concept. Wat is neergezet moet beschouwd worden als een prototype. Er zijn genoeg ideeën over om meer gegevens toe te voegen. Naast het opslaan van de gegevens kan de API worden gebruikt om applicaties tegenaan te bouwen.
Dit systeem kan worden gebruikt om data in te voeren voor het ontwikkelen van het zoekalgoritme. Maar omdat de webserver niet afhankelijk is van de gebruiker kunnen er ook applicaties op worden gemaakt die nu nog niet bedacht zijn. In een ideale situatie kan het gat tussen dit project en de huidige gegevens invoer van de politie overbrugd worden zodat alle informatie op beide plekken beschikbaar komt zonder deze twee keer in te vullen.

Literatuur

Cimpanu, C. (2016). *Microsoft: 70 percent of all security bugs are memory safety issues.*

Google. (2020). *Material system.* Verkregen van https://material.io/design/introduction#principles

PostgreSQL contributors. (2020). *Sql conformance.* Op 2020-05-28 verkregen van https://www.postgresql.org/docs/current/features.html

Sacha Greif, R. B. (2019). *State of js.* Op 2020-04-17 verkregen van https://stateofjs.com/

Slachtofferhulp Nederland. (2015). *Jaarverslag slachtofferhulp nederland 2014.* Verkregen van https://www.slachtofferhulp.nl/globalassets/media/corporate-downloads/voor-professionals/ over-slachtofferhulp/jaarverslagen/jaarverslag-2014---slachtofferhulp-nederland---uitkomst-bieden.pdf

Stackoverflow. (2019). *Developer survey results 2019.* Op 2020-04-17 verkregen van https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted

TechEmpowered. (2020). *Web framework benchmarks.* Op 2020-04-17 verkregen van https://www.techempower.com/benchmarks/