Sarea - Samen zoeken

Beheer en deployment

Versie 1.0

Inhoudsopgave

1. Inleiding

1.1. Doel van dit document

In dit document is beschreven hoe de Sarea applicatie opgegezet kan worden op een technische structuur. Dit omvat zowel een ontwikkelomgeving als een productieomgeving. De Sarea applicatie is opgesplitst in vier componenten: Website, Mobiele app, Kaartmodule en Backend. Deze vier componenten zijn per hoofdstuk beschreven en hebben een eigen methode voor het uitrollen en beheren.

Voor het beheren van de applicatie is zowel het beheren van de applicatie in productie uitgelegd evenals het onderhouden van de code die is geschreven. Hierbij worden een aantal configuratiemogelijkheden gegeven en wordt de werking van de huidige code uitgelegd. Aan het einde hiervan zal een advies gegeven worden voor mogelijke verbeterpunten voor de toekomst.

1.2 Referenties

Titel	**Afkorting**	**Versie**	**Auteur**	**Vindplaats**
Software Architectuur Document	SAD	2.0	Sarea ontwikkel teams	Gezamenlijke drive
Symfony documentatie	SYMF	4	Symfony	https://symfony.com/doc
Testrapport Sarea	TRS	1.0	Sarea ontwikkel teams	Gezamenlijke drive

2. Backend

In dit hoofdstuk wordt het uitrollen van de backend uitgelegd en de technieken die gebruikt worden voor het beheren, lees doorontwikkelen, van de backend voor sarea. De backend bestaat uit een REST API die is gemaakt middels het PHP framework Symfony 4. De handleiding wordt geschreven vanuit het perspectief van ontwikkelaars van de backend en gaat uit dat de hoofdfolder van de code het beginpunt is. Deze folder zal naar alle waarschijnlijkheid *backend-sarea* heten.

2.1 Opzetten omgeving

2.1.1. Docker

De backend van Sarea gebruikt een redelijk grote lijst van afhankelijkheden. Al deze afhankelijkheden staan in *composer.json*. Om deze te installeren zullen eerst het één en ander geïnstalleerd moeten worden. Om ervoor te zorgen dat de backend hardware onafhankelijk kan functioneren is er gebruikt gemaakt van een docker containers. Voor het opzetten van deze containers staat in de subfolder *docker/php-fpm* een bestand *Dockerfile*.

In deze dockerfile staan alle benodigdheden om een omgeving op te zetten waar de backend in kan draaien. Deze benodigdheden zitten voornamelijk in PHP-extensies, maar ook een aantal andere toevoegingen. Om gebruik te kunnen maken van docker containers zal docker beschikbaar moeten zijn op het platform waar de backend op moet functioneren. Voor de installatie van docker verwijzen wij u naar de website van docker.

Nadat docker is geïnstalleerd en het systeem toegang heeft tot het opzetten van een container middels het commando *docker-compose,* is het opzetten van de containers een kwestie van het uitvoeren van dat commando vanuit de hoofdfolder in een terminal. De omgeving wordt nu opgebouwd in een container en zal gebruikt kunnen worden voor het verdere beheer van de backend van sarea. Bij het opbouwen van de container worden de afhankelijkheden die in *composer.json* staan ook geïnstalleerd middels het *composer install* commando.

In de hoofdfolder staan een aantal .env bestanden. Deze bestanden bevatten gevoelige informatie die nodig is om de applicatie te laten functioneren zoals database connectie gegevens, applicatieomgeving variabelen en de CORS instellingen. Het is belangrijk dat deze niet in de productieomgeving worden blootgegeven. Dit is op te lossen door gebruik te maken van het commando *composer dump-env prod.* Voor een ontwikkelomgeving kunnen deze variabelen worden aangepast naar lokale gegevens.

Om te testen of de omgeving inderdaad in productie is, is het mogelijk om naar de /docs route te gaan. Als deze pagina goed geladen wordt, dan is de backend live.

2.1.2. Commands

Bij het opzetten van de docker container worden een aantal commands uitgevoerd. Als er belang is bij het opzetten van een lokale ontwikkelomgeving dan zullen deze commands handmatig moeten worden uitgevoerd. Om dit uit te kunnen voeren moeten wel eerst php en composer geïnstalleerd zijn op het systeem.

Het is nu mogelijk om het commando *bin/console sarea:deployment:deploy* in de terminal uit te voeren. Dit commando zal een aantal vragen stellen waarna het .env bestand wordt gekopieerd naar .env.local en de configuraties die worden aangegeven zullen worden ingesteld.

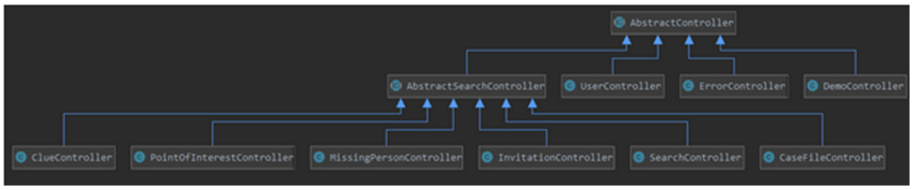
Het is mogelijk om nieuwe commands toe te voegen, maar dit is niet noodzakelijk voor het doorontwikkelen of functioneren van sarea. Mocht er toch belang zijn bij het toevoegen van andere custom commands dan is het mogelijk in de folder *src/Console/Commands* nieuwe commandos toe te voegen. Zie hiervoor ook https://symfony.com/doc/current/console.html.

2.2 Onderhouden van code

In dit deel wordt vanaf request tot response omschreven hoe de code te werk gaat. Om deze code goed te onderhouden is het sterk aangeraden om deze stijl en werkwijze toe te blijven passen. Mocht er belang zijn bij een andere structuur dan is het aangeraden om deze nieuwe structuur in zijn geheel te implementeren.

2.2.1. Controllers

Aanvragen vanuit de website en/of app komen binnen op endpoints. Deze endpoints zijn gedefinieerd in de controllers. In onderstaand figuur is een klassendiagram van de controllers zoals deze nu bekend zijn. In dit klassendiagram is te zien dat alle controllers overerven van de AbstractController. Deze AbstractController is afkomstig uit Symfony zelf en moet gebruikt worden voor het opzetten van een controller.



Alle entiteiten binnen de sarea-backend hebben een eigen controller mits deze entiteit data bevat die opgevraagd of bewerkt kan worden door een gebruiker van de applicatie. Omdat meerdere entiteiten gerelateerd zijn aan een zoekactie is er een extra abstracte controller tussen gezet.

2.2.1.1. Aanmaken nieuwe endpoint

Om een nieuw endpoint beschikbaar te maken is het mogelijk om in de bijbehorende controller een nieuwe functie te schrijven. Door gebruik te maken van de @Route annotatie kan aangegeven worden op welke url de endpoint beschikbaar moet zijn. Voor meer details over de mogelijkheden binnen het schrijven van een Route verwijzen we u door naar de documentatie van Symfony over Routes.

Als een aanvraag informatie meestuurt dan zal dit gaan middels een json body. Om deze json body op te vragen moet het request als een parameter in de functie worden gegeven. Dit kan als volgt *functie(Request $request).* De json body is nu op te vragen middels *$request->getContent*. Deze moet gedecode worden middels json_decode waarbij de assoc variabele true moet zijn. Dit ziet er als volgt uit *json_decode($request->getContent, true).*

Het is binnen de functie nu mogelijk om de gebruiker die de aanvraag doet op te vragen middels *$this->getUser().* Afhankelijk van wat het endpoint zijn taak is, is het mogelijk om bewerkingen te doen op entiteiten middels de service van die entiteit. Hierover staat meer in het hoofdstuk over de Service.

Als de functionaliteit van de endpoint is gemaakt is het belangrijk om de documentatie hiervan bij te houden. Deze documentatie wordt gedaan middels swagger. Swagger maakt gebruik van annotations om documentatie te genereren. Het is voor efficiëntie van ontwikkelen handig om deze up-to-date te houden bij alle endpoints. Om aan te geven wat er verwacht wordt aan gegevens in een request moet er gebruik worden gemaakt van de annotatie @SWG\Parameter. In deze annotation kan dan weer @SWG\Schema gegeven worden waarin alle velden worden gedefinieerd die nodig zijn bij een aanvraag. Een voorbeeld hiervan is te vinden in de SearchController.

2.2.1.2. Bewerken bestaande controller

Bij het bewerken van een bestaande controller is het belangrijk om goed op te letten dat alles nog voldoet aan de use-cases. Na een bewerking is het mogelijk om behat testen uit te voeren die de bekende scenario's afloopt en kijkt of alle endpoints nog functioneren zoals verwacht. Als hierin een fout optreedt moet er gekeken worden of dit scenario niet meer geldig is of dat er regressie is ontstaan in de functionaliteit van de code. In het geval dat het scenario niet meer geldig is, dan moet de behat test hierop worden aangepast. Hierover staat meer geschreven in het hoofdstuk over behat testen.

Als na het bewerken van een endpoint dusdanig veel aanpassingen hebben plaatsgevonden, dan zal de documentatie hiervan worden bijgewerkt. Net als bij het aanmaken van nieuwe endpoints is dit in de annotaties verwerkt. Voer waar nodig de aanpassingen in de annotations door.

2.2.2. Services

2.2.2.1. Entiteit services

In de servicelaag vindt de eerste splitsing tussen het publieke domein en het sarea domein. Waar bij de controllers van alles kan worden aangeleverd aan data, zullen de services van de entiteiten alleen de data verwerken die relevant is binnen het Sarea domein. Net als met de controller is het mogelijk dat elke entiteit zijn eigen service heeft. De service biedt de mogelijkheid om een entiteit aan te maken en aan de entitymanager te geven om deze op te slaan in de database.

Bij het verwerken van gegevens is het mogelijk dat domein gerelateerde problemen optreden. Hiervoor worden zelf-gedefinieerde domein excepties gegooid, welke opgevangen horen te worden in de controller. De controller zet ze om in een response welke de buitenwereld mag lezen. Een voorbeeld hiervan zit in de UserService. Als een gebruiker probeert te registreren met een e-mailadres die al in gebruik is, dan zal de UserService de UserAlreadyExistsException gooien die wordt afgevangen in de UserController.

2.2.2.2. Overige services

Naast de entiteit services zijn er een aantal services die gebruikt kunnen worden. Binnen de applicatie wordt op een aantal plekken gebruik gemaakt van UUIDs. Hiervoor is een generator in de servicepackage. Deze wordt onder andere gebruikt voor het uploaden van fotos. Elke foto krijgt een eigen UUID. Deze uuids worden gekoppelt aan de entiteit waar deze bij hoort. Naast de UUID generator zijn er ook services voor het versturen van e-mails en het uploaden van foto bestanden.

2.2.3. Repositories

Omdat er gebruik wordt gemaakt van doctrine beschikt de applicatie over repositories. Deze repositories bieden de mogelijkheid om queries op te bouwen voor de database en deze uit te voeren. Als er binnen de applicatie gegevens van of naar de database moeten, dan is het mogelijk om query's in deze repositories toe te voegen en te gebruiken. Voor meer details over het opbouwen van een query middels deze repositories raden wij de lezer aan om de documentatie van doctrine te raadplegen.

2.2.4. Security voters

Voor het regelen van permissies wordt gebruik gemaakt van security voters. De voters zijn een onderdeel van symfony. Er kan een permissie string worden meegegeven als eerste parameter en een entiteit die bekeken moet worden als tweede parameter. Om permissie controles toe te voegen is het mogelijk om één van de bestaande voters aan te passen of een nieuwe te maken. Net als bij Controllers en Services is het hanteren van een voter per entiteit toegepast voor het structureren van gegevens.

Bij het toevoegen van een permissie controle kan er in een klasse die de Voter extend een switch statement worden gemaakt met een extra case. Deze switch-case zoekt de permissie controle die hoort bij de meegegeven string op en voert deze uit met het object dat is toegevoegd. In de case kan de daadwerkelijke permissie worden bekeken en een boolean teruggeven waarbij *true* betekent toegang toegestaan.

2.2.5. Views

Afhankelijk van de permissies van iemand die data opvraagt moeten er bepaalde gegevens weggelaten worden bij het terugsturen van data naar de gebruiker. De data die wel teruggestuurd moet worden is gedefinieerd in Views. Door een entiteit aan de bijbehorende view mee te geven en dan de *render()* functie uit te voeren, wordt er json teruggegeven die alleen de velden bevat die de aanvrager mag zien.

2.3. Database

Zoals omschreven in hoofdstuk 2.1 wordt de connectie url in de .env gegeven. Deze wordt verder gebruikt voor de connectie met de database. Het beheren van de database binnen de applicatie gaat middels Doctrine. Een ERD van de huidige database is gegeven in het Software Architectuur document.

2.3.1. SQL

Doordat er gebruik wordt gemaakt van MySQL is het mogelijk om met een SQL manager direct SQL statements naar de database te sturen. Dit is makkelijk voor het uitlezen van data die in productie is.

2.3.2. Migraties

Onder migraties wordt het aanpassen, toevoegen of verwijderen van entiteiten bedoeld. Doordat er gebruik wordt gemaakt doctrine is het mogelijk om met het commando *make:entity* een nieuwe entiteit toe te voegen of aan te passen. Tijdens het uitvoeren van dit commando worden een aantal vragen gesteld die de entiteit opbouwen. Na het uitvoeren van *make:entity* zullen meteen de bijbehorende entiteiten verschijnen als klassen in de code. Deze zijn terug te vinden in *src/Entity.*

Naast het aanmaken van een PHP klasse voor de entiteit wordt er ook een migratie aangemaakt. Deze staat in *src/Migrations* en wordt gebruikt voor het uitvoeren van SQL die de entiteit ook aanmaakt in de database. Dit wordt gedaan door *doctrine:migrations:migrate* uit te voeren. In de database zelf staat een tabel die bijhoudt op welke versie de database staat en zal bij het uitvoeren van dit commando alleen de opvolgende migraties nog uitvoeren.

Om een tabel te verwijderen uit de database is het mogelijk om de PHP klasse van die entiteit te verwijderen. Doctrine zal nu de entiteit niet meer als onderdeel van het domein zien en er niet meer mee werken. Aangemaakte repositories en services zullen hierbij ook handmatig moeten

worden verwijderd. Deze actie zal echter niet de tabel uit de database verwijderen. Dit zal via een SQL statement moeten gebeuren.

2.4. Testen

2.4.1. Behat

Er wordt binnen de applicatie gebruik gemaakt van behat testen. Deze testen kijken naar de use-cases die gerealiseerd moeten worden en hebben hier scenario's bij gemaakt. Om deze scenario's op te kunnen bouwen wordt er gebruik gemaakt van een context. Deze zijn te vinden in *src/Behat.* In de folder *features/…* zijn de daadwerkelijke testen geschreven. Hier wordt door middel van de context een scenario opgebouwd en naar het antwoord van de backend gekeken. Een uitgebreide omschrijving en werking van behat testen zijn gegeven in het testrapport.

2.5. Nog te realiseren

Nog niet alle use-cases zijn binnen de backend gerealiseerd evenals een aantal duidelijke improvements die kunnen worden gemaakt binnen de applicatie. Zo is het nog niet mogelijk om chatberichten aan te leveren in de backend en is er nog geen input validatie aanwezig op alle input. Een suggestie voor input validatie is om gebruik te maken van de Validators.

In de huidige staat wordt er alleen gebruik gemaakt van behat testen om scenario's te testen. Voor een volledige teststructuur is het aan te raden om nog unittesten te maken evenals systeemtesten om het geheel van sarea te testen.

Vanwege de context waarbinnen de applicatie is ontwikkeld is het niet mogelijk geweest om een volledig werkende mailing service te maken. De service componenten zijn wel geschreven, maar omdat er geen daadwerkelijke mailserver beschikbaar was zijn deze componenten nog niet volledig functioneel. Dit is nog een belangrijk verbeterpunt voor de toekomst omdat zowel registratie als uitnodigingen van zoekacties afhankelijk zijn van deze mailingservice.

3. Kaart

3.1 Beheer - Algemeen

3.1.1 Versiebeheer

De kaart-module broncode staat onder Git versiebeheer, de gehele broncode staat in 1 repository. Deze repository is overhandigd tijdens de oplevering.

3.1.2 Opzetten development-omgeving

Tijdens het ontwikkelen is gebruikt gemaakt van Jetbrains IDEA-gebaseerde IDEs, de configuratiebestanden hiervan zijn meegenomen in de broncode repository en zijn direct te importeren.
De development-omgeving is onderverdeeld in 4 modules:

- KAART_APP:
  React-native module die de Kaart-functionaliteit voor de App implementeert

- KAART_WEBSITE:
  React module die de kaart-functionaliteit voor de website implementeert

- KAART_COMMON:
  Node module die 'common' functionaliteit implementeert voor de KAART_APP/KAART_WEBSITE modules. Wordt gebruikt als dependency door beide modules.

- KAART_SERVER:
  IDEA Python Flask project die de backend kaart-functionaliteit implementeert

Naast deze modules bestaat de development omgeving nog uit een aantal losse bestanden, en de mappen "lua" en "builds". De map "lua" bevat de Lua Interpreter versie 5.3, deze wordt gebruikt voor een aantal build scripts. De map "builds" is de output-map voor distributie-builds.

De losse bestanden zijn:

- *app-module-build-container.sh* Dit bash script bouwt de app kaart-module in een Docker container
- *build-app.sh* Dit bash script bouwt de app kaart-module lokaal
- *build-app.yml* Docker-compose configuratie voor de app kaart-module
- *build-web.sh* Dit bash script bouwt de website kaart-module lokaal
- *BUILD_APP.lua* Dit Lua script bouwt de app kaart-module lokaal, en voert een versie-increment uit voor kaart-common.

- *BUILD_WEB.lua* Dit Lua script bouwt de website module lokaal, en voert een versie-increment uit voor kaart-common en kaart-website.
- *Dockerfile* Docker configuratie voor de app kaart-module.

Aan externe ontwikkel tooling is nodig:
- Node.js: >10.17.0
- NPM: >6.13.1
- Yarn: > 1.13.0
- Android SDK (Gebruikt gemaakt API level 28, eerdere versies ongetest)

De configuratiebestanden gaan ervan uit dat deze tools beschikbaar zijn in de %PATH% en %ANDROID_HOME% systeemvariabelen.

3.2 Kaart server

3.2.1 Deployment

Voor het deployen van de server voor een kleine hoeveelheid aan gebruikers kan gebruik worden gemaakt van de meegeleverde Docker bestanden. In dit document wordt geen rekening gehouden met grote schaal deployment over meerdere servers. Er zal in dit hoofdstuk enkel worden uitgelegd hoe de Docker deployment kan worden gedaan.

Voor basisinstallatie van de server wordt aangeraden om voor een Linux distributie te kiezen. Een aantal distributies die gebruikt kunnen worden zijn het betaalde Red Hat Enterprise Linux(RHEL), CentOS (gratis versie van RHEL) en Debian.

Op de server moet Docker en docker compose geïnstalleerd staan om gebruik te maken van de docker compose bestanden. De losse Docker bestanden kunnen worden gebruikt zonder Docker, door gebruik te maken van Podman.

Met Docker geïnstalleerd ga naar de folder kaart-server. Maak een kopie van .env.example en noem deze .env. In het .env bestand staan een aantal lege configuratie velden. Vul deze in voor het starten van de server. De .env stap is optioneel en vervangen worden door het meegeven van de variabelen in het startcommando.

10.2.g

Met de variabelen ingesteld start de server met het commando: “docker-compose -f deploy.yml up”.
De server staat nu aan maar is nog niet gereed voor gebruik. Voordat de server in gebruik kan worden genomen buiten testdoeleinden moet encryptie worden opgezet. Dit kan worden gedaan door middel van nginx ervoor plaatsen met een ssl certificaat van Let’s encrypt. Het verder opzetten van encryptie is buiten de scope van dit document.

3.2.2 Beheer server

Voor het ontwikkelen van de server kan er gebruik worden gemaakt van docker. Hierbij moeten dezelfde variabelen worden ingesteld als bij de deployment met uitzondering van DATA_LOCATION. Met de variabelen ingesteld kan “docker-compose up” worden gebruikt om de development server te starten. Deze server herlaad automatisch bij code veranderingen.

3.3 App

Voor het bouwen van de app module zijn er twee mogelijkheden. De gebruikte ontwikkel tools nodejs en Yarn kunnen geïnstalleerd worden, of er kan gebruik worden gemaakt van docker. Als eerst zal de docker methode worden besproken. Om dit uit te voeren moet docker geïnstalleerd staan. Als bash beschikbaar is kan er een hulp script worden uitgevoerd om alles verder te doen. Dit is het script app-module-build-container.sh. Bash is de standaard terminal op MacOS en Linux op Windows moet een alternatief worden gebruikt of handmatig de commandos invoeren die in dit bestand staan. Als het process is geslaagd bevindt de module export zich nu in de builds folder als een .tgz bestand.
Nu voor de methode zonder Docker. Er wordt van uitgegaan dat Node en Yarn op de computer geïnstalleerd staan. Er is een bouwscript beschikbaar(build-app.sh) in de hoofdfolder, deze werkt echter alleen op Fedora. Om het handmatig te doen zijn de volgende stappen nodig:

1) Ga naar de kaart-common folder en kopieer alle .js bestanden naar de folder kaart_app/src/kaart-common
2) In de folder kaart_app draai het commando “yarn install” uit om alle afhankelijkheden te downloaden.
3) Voor het commando “yarn pack” uit om de module te maken. Deze staat in huidige folder als een .tgz bestand.

3.3.1 Ontwikkelomgeving

Voor het ontwikkelen van de app moeten er drie programma's geïnstalleerd staan. Dit zijn node, yarn en Android SDK. Voor het verkrijgen van de Android SDK moet eerst Android studio geïnstalleerd worden. Het gebruik van de kaart-common module kan gedaan worden zoals beschreven in de voorgaande. De kaart_app bevat buiten de te exporteren logica ook een App.js bestand. Dit is een simpele test app die is gebruikt voor het testen van de module. De

test app kan uitgevoerd worden door in de kaart_app folder dependencies te installeren met “yarn install” en met “react-native run-android” de test app te starten.

3.4 Website

Voor het maken van een website module build worden de volgende stappen uitgevoerd:

1) Maak een /**dep** folder aan in de **kaart-website** map, indien deze niet bestaat.
2) (Optioneel) Increment de versie van kaart-common aan in de package.json van kaart-common
3) Voer het commando **NPM pack** uit in de folder **kaart-common**
4) Kopieer de nieuw gemaakte tarball naar de /**dep** folder van **kaart-website**
5) (Optioneel) Pas de versie van de kaart-common dependency aan in de package.json van kaart-website naar de nieuwe nummer uit stap 2.
    a) Voer het commando **NPM install** uit in de folder **kaart-website**
6) Pas de versie van kaart-website aan in de package.json van kaart-website
7) Voer het commando **NPM pack** uit in de folder **kaart-website**

De optionele stappen zorgen ervoor dat de gemaakte builds nieuwe versienummers hebben, dit voorkomt problemen met de NPM cache.

3.4.1 Linux

Ga in de hoofdfolder en voer het build-web.sh bestand uit. Als alles goed ging bevindt de gebouwde module zich nu in de builds folder als .tgz bestand. Deze methode is niet getest en kan mogelijk niet werken. De web module is ontwikkeld op Windows dus de Windows build scripts hebben wellicht betere resultaten.

3.4.2 Windows

Voor een complete build, voer het "BUILD_WEB.lua" script uit, het eindresultaat van dit script is een NPM tarball voor de kaart-website module in the /builds/ folder.

Indien geen nieuwe versie van kaart-common hoeft worden uitgevoerd kan ook het "NPM pack" commando uitgevoerd worden in de kaart-website folder.

4. Mobiele app

4.1 Benodigdheden

Voor het beheren en publiceren van de mobiele applicatie zijn een aantal dingen nodig. Zonder (één van) deze onderdelen zal het erg lastig worden.

- Een computer
- Een telefoon of een emulator waarop getest kan worden
- Een ontwikkelomgeving
- Technische kennis van Javascript en React Native

4.1 Algemeen beheer

Het beheren van de app vereist flinke technische kennis. Als de app goed beheerd gaat worden is het zaak dat de beheerders zich comfortabel maken met react native. Iemand met technische kennis zal snel een weg kunnen vinden in de code. Tijdens de ontwikkeling is ervoor gezorgd dat elk bestand op een logische plek staat. Zo zijn er bijvoorbeeld voor *components*, *features* en *styles* aparte mappen gemaakt. Ook is er een map waar alle gebruikte API's in staan.

Alvorens iemand bezig gaat met ontwikkelen moet er een aantal dingen worden geïnstalleerd. Natuurlijk moet er een ontwikkelomgeving worden geïnstalleerd. Voor de afgelopen ontwikkeling is de IDE van Jetbrains gebruikt maar dat is een persoonlijke keuze voor de ontwikkelaar.

4.1.1 Versiebeheer

Ook voor de mobiele app is met behulp van Gitlab gebruikt gemaakt van Git tijdens de ontwikkeling. Dat wil zeggen dat de code volledig inzichtelijk is inclusief de progressie. Deze code is overhandigd tijdens de oplevering.

4.1.2 NPM en node modules

Tijdens de ontwikkeling is gebruik gemaakt van NPM als pakketbeheerder. Dit betekent dat NPM geïnstalleerd moet worden door ontwikkelaar of beheerder. Het gebruik van NPM is enigszins vergelijkbaar met dat van de website. Voor het binnenhalen van de benodigde packages is het commando *npm install* gemaakt.

Als alle packages eenmaal zijn opgehaald kan de lokale *metro-server* gestart worden. Met dit onderdeel 'praat' de emulator of de telefoon. Het starten van deze server kan met het commando *npm start.*

4.3 Publiceren van de app

Met name de deployment voor de mobiele app is erg overzichtelijk in vergelijking met de andere onderdelen. Wie de app wil gebruiken moet deze geïnstalleerd hebben op de telefoon. Op het moment van schrijven is de mobiele applicatie enkel voor Android klaar voor gebruik, maar door de keuze van React Native is het relatief simpel om de app te draaien op een iPhone. Er zijn twee manieren waarop de huidige app gepubliceerd kan worden.

4.3.1 APK-bestand

De eerste optie is om via de ontwikkelomgeving een apk-bestand te exporteren. Eenmaal geëxporteerd kan het bestand worden geïnstalleerd op een Android-telefoon.

4.3.2 Google Play Store

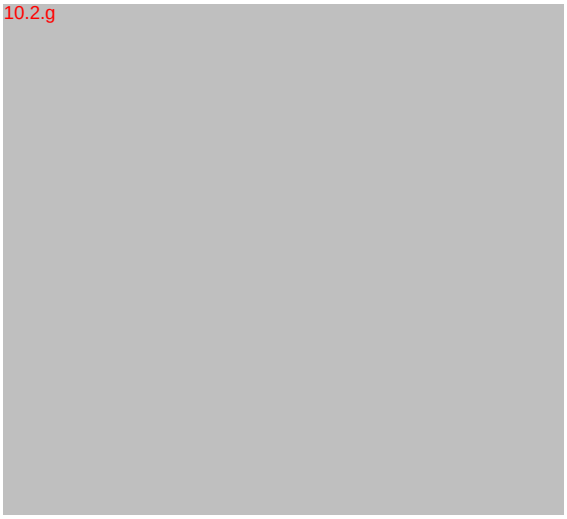
Als er wordt besloten de mobiele app openbaar te maken kan hij worden gepubliceerd op de Android Play Store. Hiervoor moet een aab-bestand worden gemaakt. Dit is een Android App Bundle waarmee Google een geoptimaliseerd apk-bestand kan maken. Eenmaal gepubliceerd bij Google wordt er weer een apk-bestand van gemaakt. Voor het publiceren van de applicatie is de handleiding die Google aanbiedt handig.

4.4 Configuratie

In de code kunnen een aantal dingen geconfigureerd worden. Denk hierbij aan adressen van servers en endpoints voor de backend.

4.4.1 IP-adressen servers

In het project moet aangegeven worden wat de IP-adressen zijn van de externe services. In dit geval zijn dat de backend en de kaart-server. De configuratie voor de backend is gedefinieerd in het bestand *constants.json* in de map *API*. In dit bestand wordt aangegeven wat het IP-adres is van de backend alsook de 'routes' voor de verschillende endpoints. Een gelijknamig bestand voor de kaart-server is te vinden in de hoofdmap.



Afbeelding X. Deel van configuratie voor de backend. Met een basis-URL en verschillende endpoints.

Afbeelding X. De configuratie voor de kaart.

4.4.2 Taal en vertalingen

De mobiele app kan ingesteld worden op verschillende talen. Hiervoor wordt gebruik gemaakt van de veelgebruikte module *i18n*. Deze module beschikt over de mogelijkheid om makkelijk talen toe te voegen. Op het moment van schrijven zijn bestanden gemaakt voor Engels en Nederlands. In afbeelding X is te zien hoe de vertalingen er in zo'n bestand uit zien.

Met opmerkingen [1]: Afbeeldingsnummering

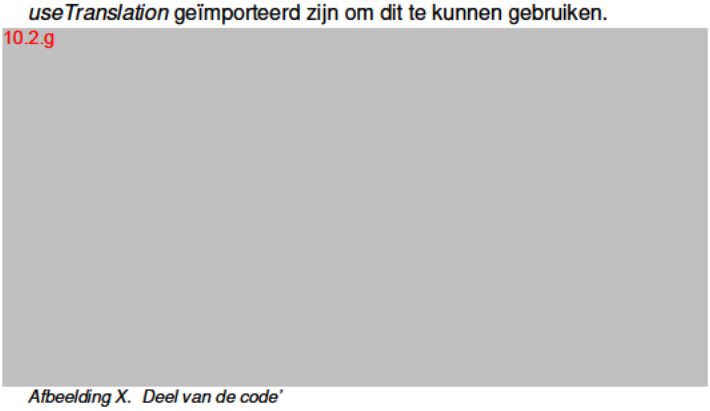
Met opmerkingen [2]: gaan we het afbeeldingen of figuren noemen?

10.2.g

Afbeelding X. Deel van het 'vertaalbestand'

In de code wordt er vervolgens op de volgende manier naar verwezen. In het component moet *useTranslation* geïmporteerd zijn om dit te kunnen gebruiken.

10.2.g



Afbeelding X. Deel van de code'

Als er teksten aangepast moeten worden in de app kan dat dus simpelweg in de vertaalbestanden. Deze zijn te vinden in de map *locales*. Ook het toevoegen van nieuwe items om te vertalen kan simpel gedaan worden. Creëer een nieuw item door de stijl van Afbeelding X te gebruiken. Vervolgens kan met het commando: "npm run extract" gebruiken om ervoor te zorgen dat het nieuwe item direct op de juiste plaats neergezet wordt in alle vertaalbestanden. Hierna hoeft alleen de tekst zelf handmatig aan het item toegevoegd te worden voor elk vertaalbestand.

5. Website

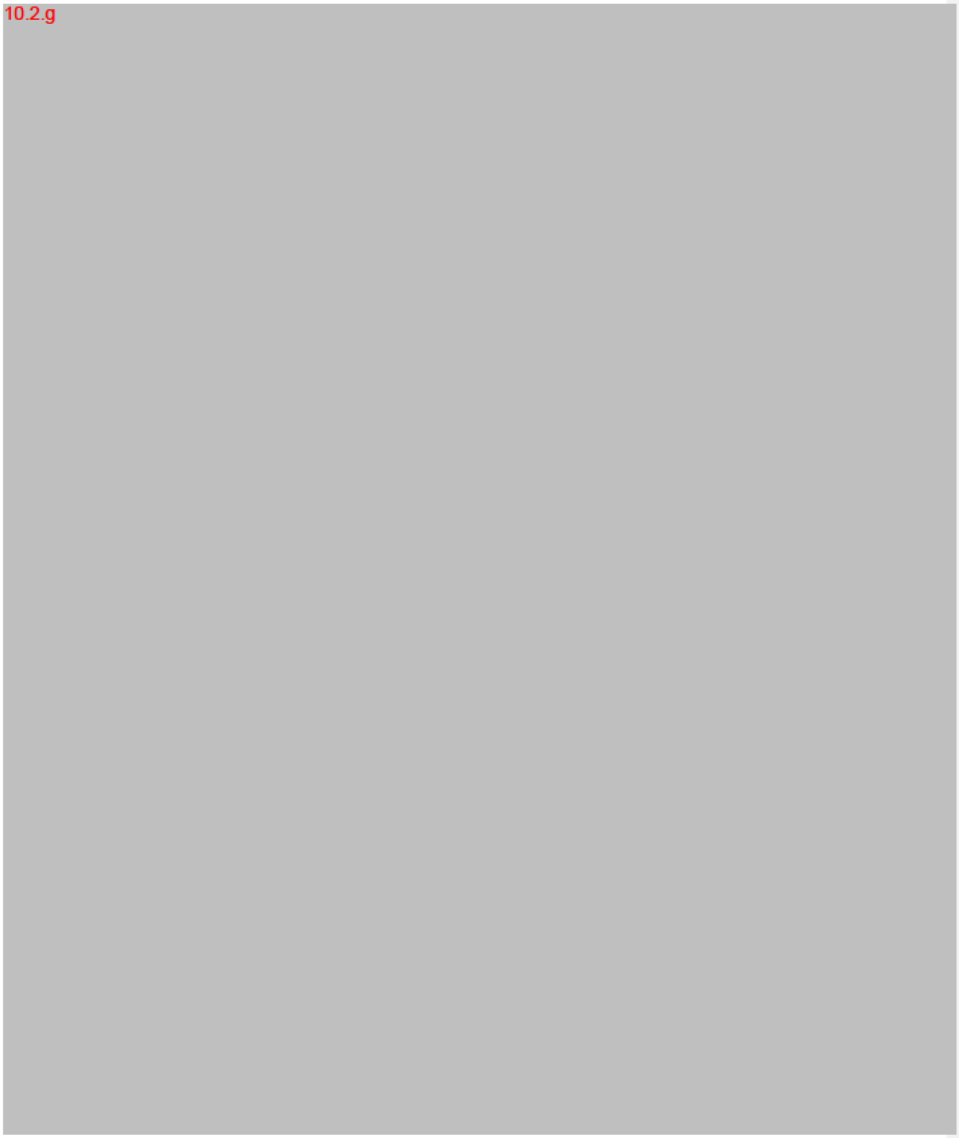
5.1 Benodigdheden

1. Om gebruik te maken van het product is internetverbinding nodig.
2. Basiskennis serverbeheer.
3. Een website server waarop de website gehost kan worden.
4. Een computer waarmee het opzetten gedaan wordt.

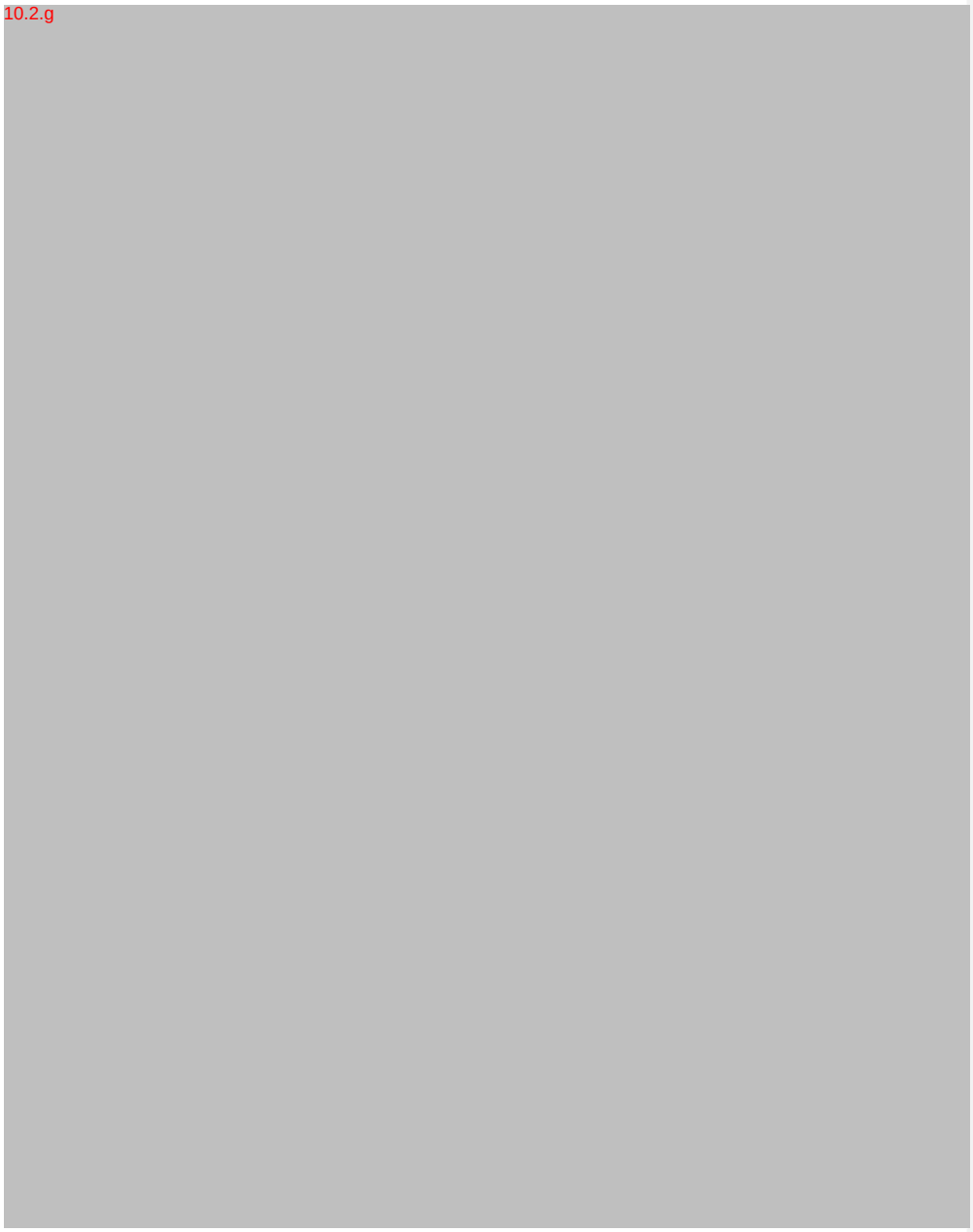
5.2 Stappenplan installatie website

5.3 Connectie met de backend

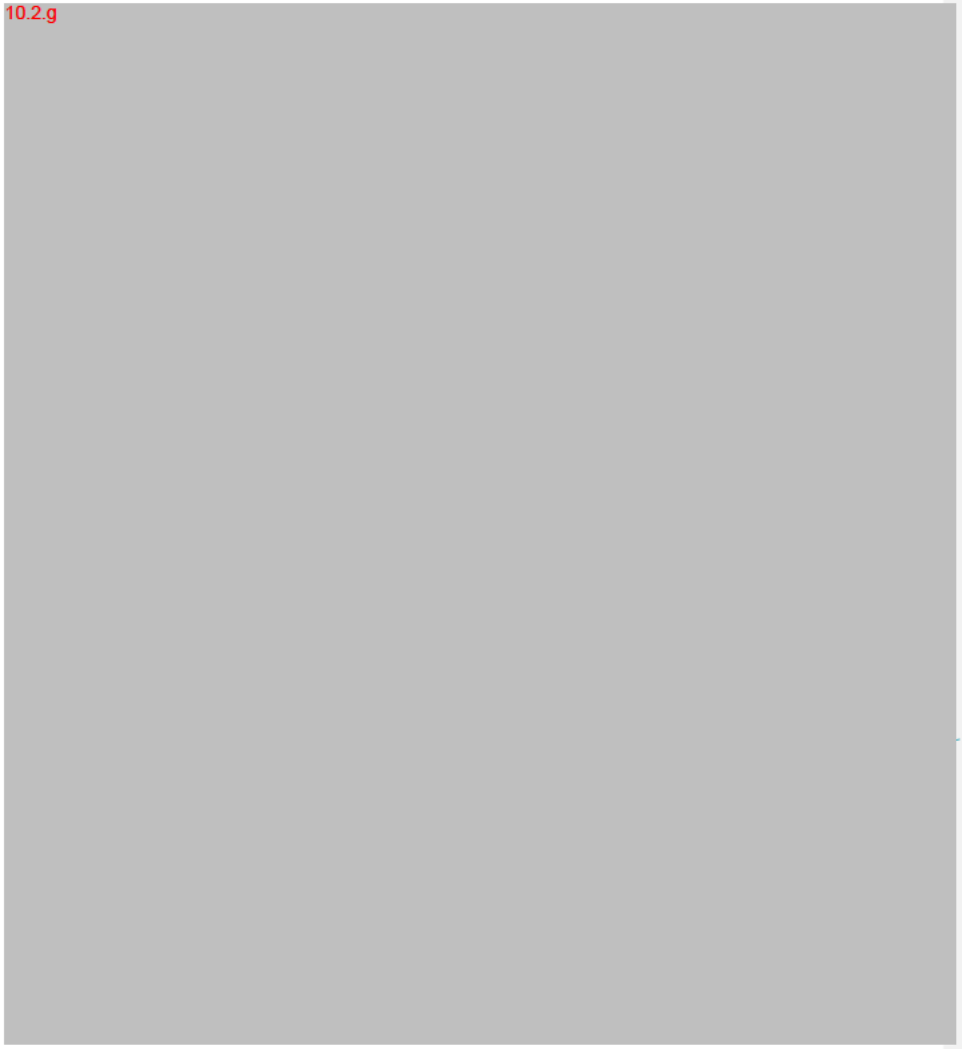
Het website gedeelte van de Sarea is puur een voorkant van de applicatie. Dit wil zeggen dat het voor een uiterlijk zorgt en alle functionaliteit heeft voor een goede werking in de webbrowser. Dit noemen we dan ook een "frontend applicatie" omdat het alleen de voorkant verzorgt. Om daadwerkelijk informatie op te slaan die gegeven kan worden door deze voorkant is een backend nodig. In dit document staat beschreven hoe de backend opgezet kan worden. Om de koppeling nu te maken van de frontend naar de backend is vrij gemakkelijk.


10.2.g

10.2.g



10.2.g



Met opmerkingen [3]: Zin iets beter formuleren