# Inhoudsopgave

# 1 Inleiding

## 1.1 Context

Pozyx Labs is een Gents bedrijf dat werd opgericht in augustus 2015. Het specialiseert in in- en outdoor positioning en voorziet hardware oplossingen voor accurate positionering. Het Pozyx-systeem maakt gebruik van ultra-wideband (UWB), een nieuwe draadloze radiotechnologie, die positionering op een aantal centimeter nauwkeurig mogelijk maakt.

## 1.2 Probleemstelling

Momenteel heeft Pozyx slechts één commercieel product: het DIY-product dat in staat is een beperkt aantal devices te tracken. In de toekomst is het echter de bedoeling een grootschalig platform te ontwikkelen dat tracking van duizenden devices toelaat. Hiervoor is een webapplicatie nodig die real-time tracking data weergeeft. De hoeveelheid weergegeven data is voorlopig onbeperkt. Voorbeelden zijn: geschiedenis (afgelegde route) van tags op een gepersonaliseerd platform, triggers wanneer een bepaalde gebeurtenis zich voordoet (x devices binnen zich in regio y) of informatie over de tags zelf (id, battery life...).

### 1.2.1 Doelstelling

Dit vakoverschrijdend eindproject heeft als doel de eerder beschreven cloud-applicatie te ontwikkelen. Met een groep van zes zullen we in twaalf weken de (zowel volgens ons als volgens Pozyx) meest prioritaire features opnemen in de webapplicatie. Ons einddoel is een gebruiksvriendelijke applicatie die de meest prioritaire functionaliteiten bevat af te leveren.

### 1.2.2 Overzicht

Dit document bundelt de analysefase van het project. In het sectie 2 worden de gebruikersaspecten toegelicht a.d.h.v. use case diagrammen, de sprint backlog en de geselecteerde features per sprint. Sectie 3 handelt over de systeemarchitectuur. Deze wordt toegelicht aan de hand van klassen- en sequentiediagrammen.

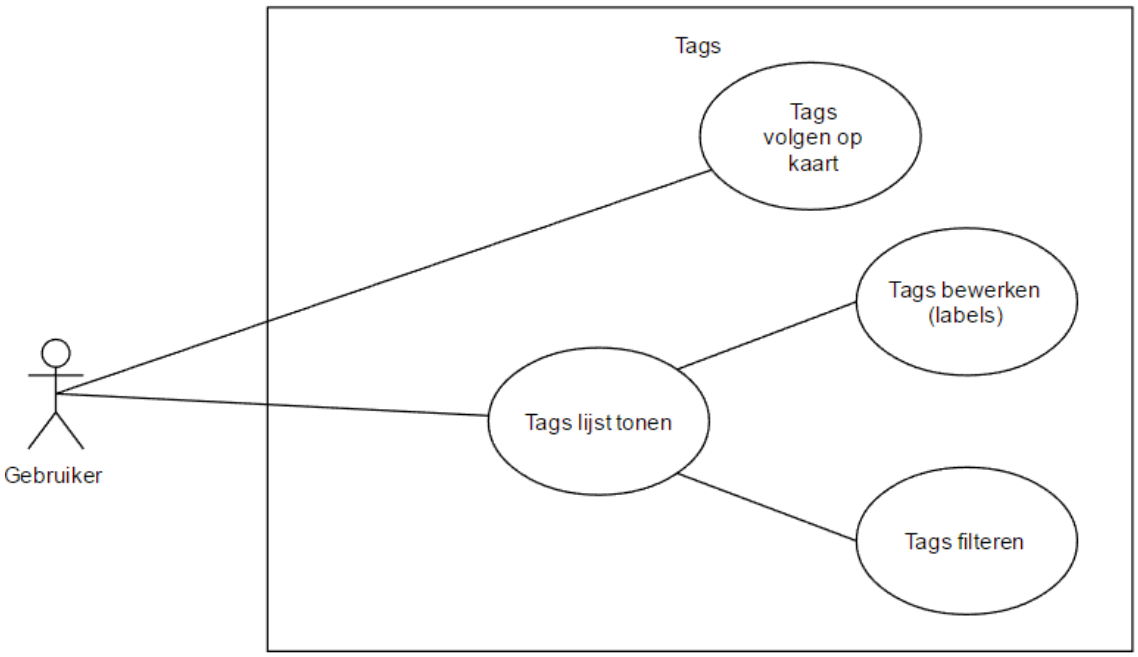
# 2 Gebruikersaspecten

## 2.1 High-level requirements

De webapplicatie moet een gebruiksvriendelijke real-time visualisatie bieden van alle tags en anchors (voorzien tags met nodige informatie om de positie te bepalen) op de gepersonaliseerde plattegrond. De gebruiker van de applicatie moet zelf plattegronden (opdeelbaar in zones), tags, anchors én triggers kunnen toevoegen en beheren.

## 2.2 Use case diagrammen

### 2.2.1 Tags

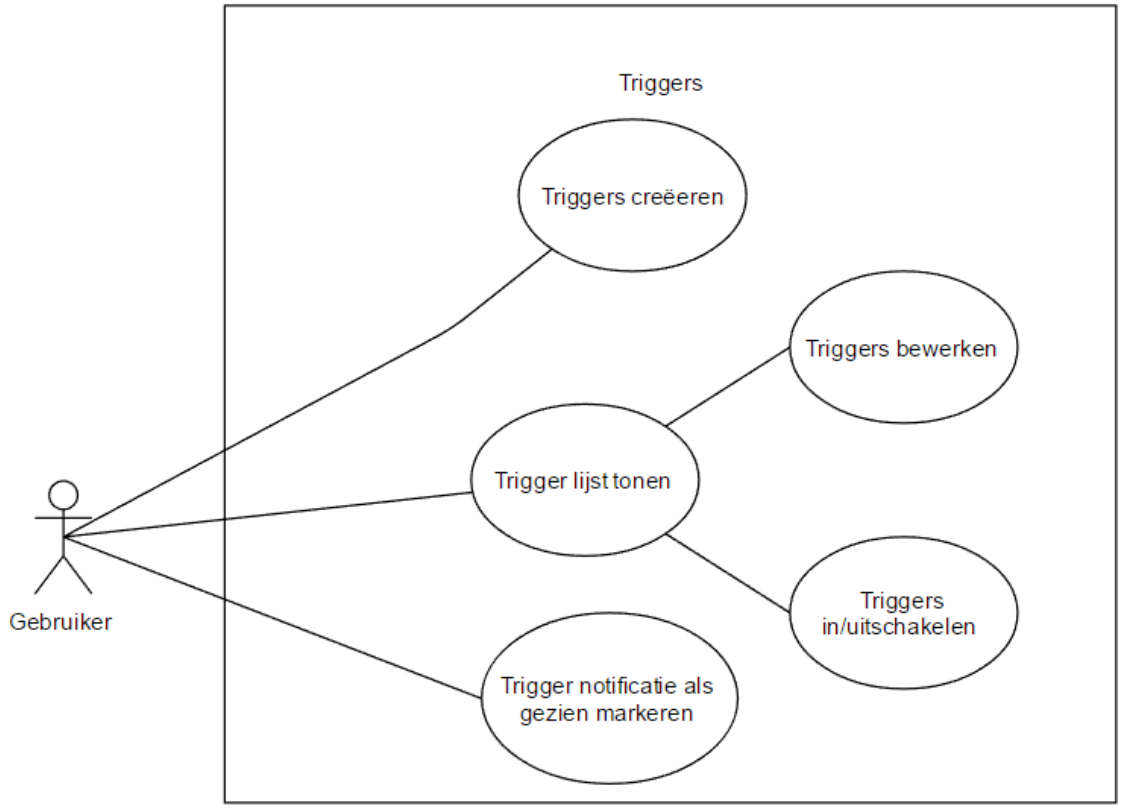



Figuur 1: Use case diagram van tags

Figuur 1 toont het use case diagram horende bij Tags. Er worden 5 verschillende cases erkend:

1. Tags volgen op de kaart: Deze use case maakt het mogelijk om specifieke geselecteerde (dmv filters) tags te volgen op de kaart.
2. Tags lijst tonen: Een gebruiker kan bestaande tags opgelijst weergeven.
3. Tags filteren: Opgelijste tags kunnen op basis van label, batterijpercentage, naam of ID gefilterd worden.
4. Tags bewerken: Deze use case voorziet de functie om tags een bepaald label te geven of ontnemen.

### 2.2.2 Triggers




Figuur 2: Use case diagram van triggers

Figuur 2 toont het use case diagram horende bij Triggers. Er worden zes verschillende cases erkend:

1. Trigger creëren: Binnen de pagina 'Triggers' heeft een gebruiker de mogelijkheid meerdere triggers aan te maken.

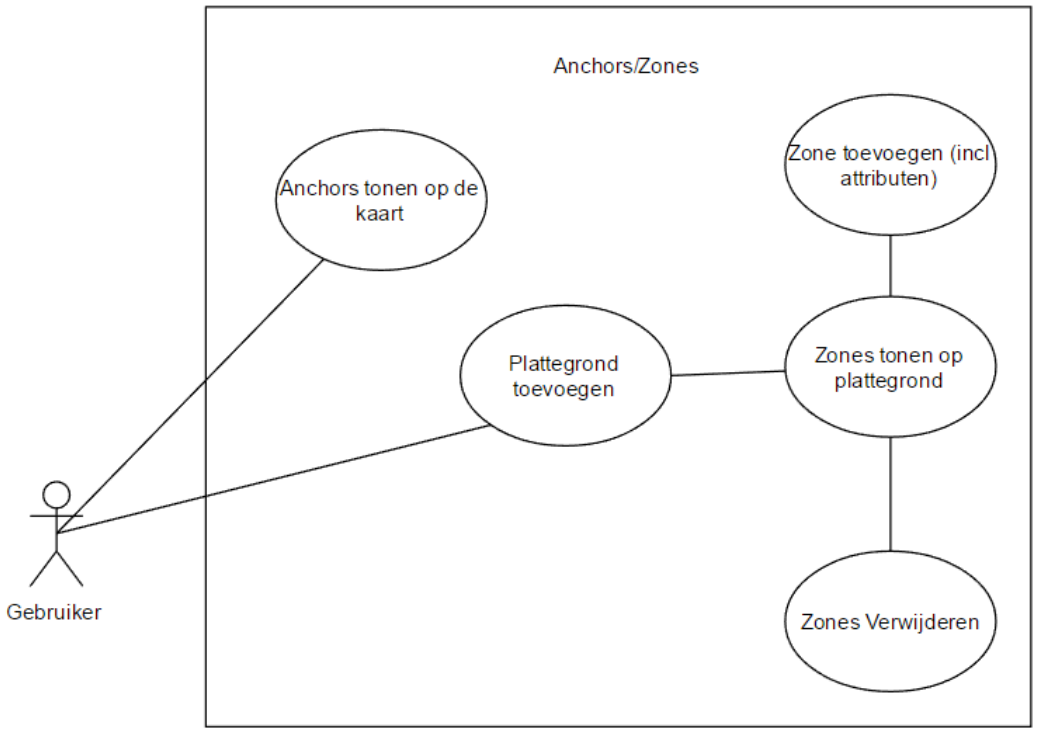
2. Trigger notificatie als gelezen markeren: Wanneer een trigger getriggerd wordt, krijgt de gebruiker een notificatie. De gebruiker heeft de mogelijkheid deze notificatie als 'gezien' te markeren, en kan via de notificatie meer informatie over de trigger opvragen.

3. Triggerlijst tonen, bestaande uit twee subcases:

   - Triggers bewerken
   - Triggers in- en uitschakelen

   De gebruiker kan de bestaande triggers opgelijst weergeven, en een opgelijste trigger selecteren, bewerkenen als (in)actief instellen.

### 2.2.3 Anchors en Zones




Figuur 3: Use case diagram van Anchors en Zones

1. Plattegrond toevoegen: Een gebruiker kan een persoonlijke plattegrond uploaden waarop in een latere fase tags en anchor worden weergegeven.
2. Anchors tonen op de kaart: Het moet mogelijk zijn om alle anchorpunten te zien op de kaart.
3. Zones tonen op de plattegrond: Een gebruiker kan zelf kiezen of hij eerder gecreëerde zones wenst te zien op de plattegrond of niet.
4. Zone toevoegen: Een gebruiker kan een zone toevoegen door een polygon te tekenen op de plattegrond. Hierna wordt verplicht een naam verwacht, en is een kleur kiezen voor de zone optioneel.
5. Zones verwijderen: Een gebruiker heeft de mogelijkheid zones te verwijderen wanneer hij de toggle button aanklikt.

## 2.3 Feature list / Backlog

De feature list is opgesteld uit User Stories en is omgezet en opgedeeld in specifieke taken in de backlog. De prioriteit van een item komt overeen met de sprint waarin het item geïmplementeerd wordt. Tabel 1 handelt over tags. Vervolgens worden anchors en zones behandeld in Tabel 2 en ten slotte komen triggers aan bod in de laatste tabel. De feature lists worden telkens gesorteerd op hun prioriteit. Deze komt overeen met elke sprint, zo zijn essentiële zaken opgenomen in de eerste twee sprints en worden de overige zaken opgeschoven naar de laatste sprint.

| | **User stories** | **Taken (backlog)** | **Sprint** |
|---|---|---|---|
| 1 | Tags moeten real-time op een plattegrond weergegeven worden. | 1. Endpoint om alle huidige posities van een tag te krijgen.<br>2. Tags weergeven in frontend. | 1 |
| 2 | Het moet mogelijk zijn om een lijst van tags weer te geven. | 1. Endpoint om alle tags te krijgen.<br>2. Lijst van alle tags in de UI tonen. | 1 |
| 3 | Het moet mogelijk zijn een lijst van tags te filteren op attributen/labels | 1. Endpoint om tags te sorteren.<br>2. Material-ui gebruiken om frontend filtering op te bouwen.<br>3. Visuele weergave van niet gefilterde tags uitschakelen in front-end. | 2 |

Tabel 1: Tags Backlog

| | **User stories** | **Taken (backlog)** | **Prioriteit** |
|---|---|---|---|
| 1 | Het moet mogelijk zijn om een kaart toe te voegen | 1. Plattegrond endpoints programmeren.<br>2. Plattegrond opslaan in static file.<br>3. Uploadknop op website plaatsen. | 1 |
| 2 | Het moet mogelijk zijn om een plattegrond met anchors te zien | 1. Endpoint om de positie van tags op te vragen.<br>2. GUI met kaart en tags toevoegen in de UI. | 1 |
| 3 | Zones moeten kunnen worden toegevoegd door een polygon te creëeren. | 1. Endpoint om plattegrond in ruimten op te delen<br>2. Mogelijkheid om zones te creëren in GUI toe te voegen<br>3. Zones opslaan in database | 2 |
| 4 | Het moet mogelijk zijn om een lijst van anchors te tonen. | 1. Endpoint anchors op te vragen. | 2 |

| | | | |
|---|---|---|---|
| 5 | Een zone moet kunnen worden verwijderd. | 1. Endpoint om zone te verwijderen.<br>2. Zone uit databank verwijderen.<br>3. Optie om zone verwijderen toevoegen in de UI. | 2 |
| 6 | Zones moeten kunnen worden gevisualiseerd op de kaart | 1. Optie om zone al dan niet te visualiseren in de UI.<br>2. Endpoint om zones op te vragen. | 2 |

Tabel 2: Anchors & zones Backlog

| | **User stories** | **Taken (backlog)** | **Prioriteit** |
|---|---|---|---|
| 1 | Er moet een notificatie worden getoond wanneer een trigger zich voordoet | 1. Trigger detecteren op de server<br>2. Beperkte trigger data weergeven in popover list. | 3 |

| | | | |
|---|---|---|---|
| 2 | Het moet mogelijk zijn een trigger te creëren | 1. Team brainstorm ivm triggers. Wat? Hoe?<br>2. Ui walktrough om Trigger te doorlopen<br>3. Endpoints voorzien | 3 |
| 3 | Er moet een overzicht van alle triggers getoond worden | 1. Endpoint om alle triggers te tonen (actieve, uitgeschakelde en in ontwerp).<br>2. Triggers opslaan in databank.<br>3. Lijst van triggers weergeven in UI. | 3 |
| 4 | Een trigger moet kunnen worden in/uitgeschakeld | 1. Endpoint om status van triggers te wijzigen en doorvoeren naar backend.<br>2. Toggle optie om status van een trigger te wijzigen toevoegen aan UI. | 3 |
| 5 | Trigger moeten bewerkt kunnen worden (naam, filters,..) | 1. Endpoints om een trigger te wijzigen.<br>2. UI voorziening (editable textfields,..) | 3 |

Tabel 3: Triggers Backlog

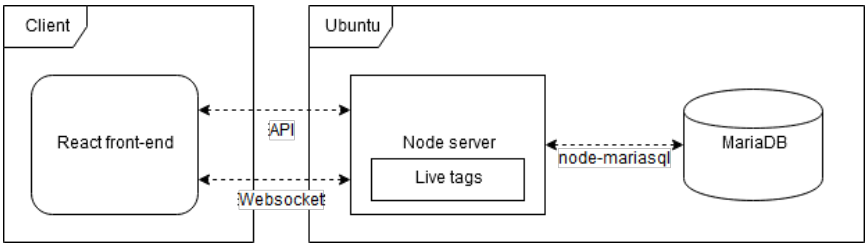
## 2.4 Geselecteerde features per sprint

De vermelde prioriteit in de backlog komt telkens overeen met de sprint waarin we dit item willen uitwerken. De eerste sprint bestaat voornamelijk uit essentiële zaken die nodig zijn voor de basisfunctionaliteit van de applicatie. In de tweede sprint werd gepland zowel zones en triggers volledig uit te werken. Door het wegvallen van een teamlid bleef het enkel bij zones. Sprint 3 werd nadien volledig besteed aan het implementeren van triggers.

# 3 Systeemarchitectuur

## 3.1 Deployment Diagram

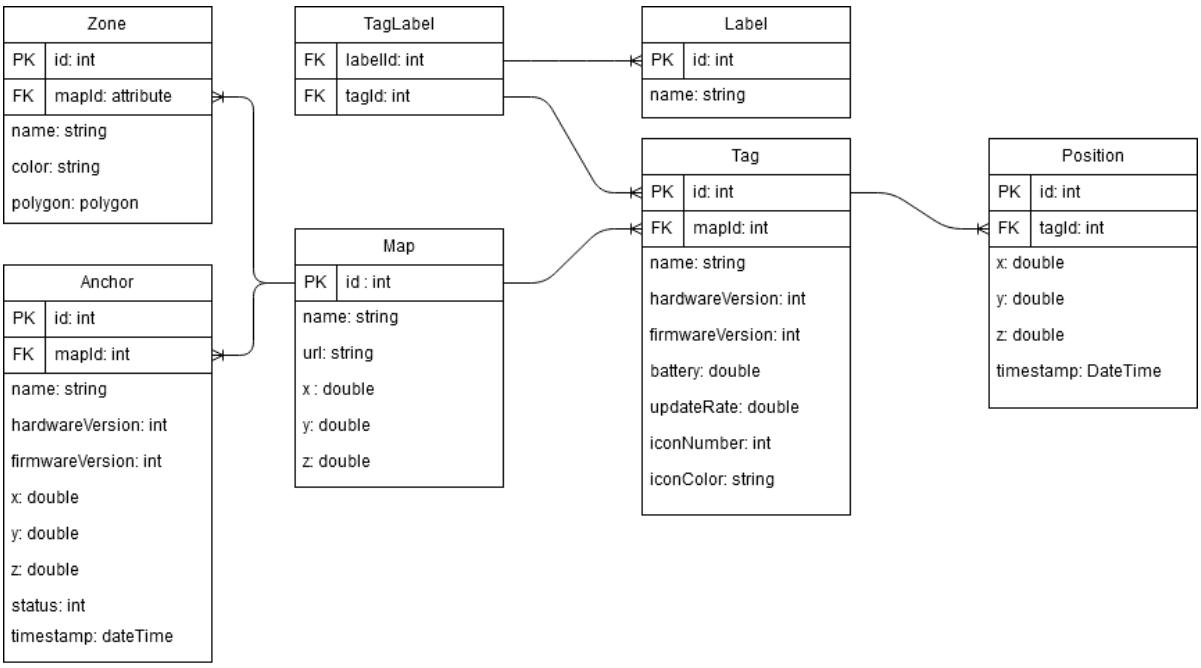
Onze applicatie bestaat uit een ReactJS front-end en NodeJS back-end met een MariaDB databank. Zoals te zien figuur 4 wordt er op twee manieren gecommuniceerd tussen clients en de back-end. De eerste manier is via de API. Hiermee kan de kaart met tags en anchors opgevraagd worden, kunnen labels aangebracht worden bij tags en worden zones en triggers aangemaakt en gewijzigd. Het tweede communicatie kanaal is een websocket. Hiermee wordt de live data van tags zoals hun locatie doorgegeven als de live data rond triggers zoals notificaties wanneer een trigger zich voordoet. De Node server maakt gebruik van een script om de bewegingen van tags te simuleren. Dit script kan ook aangesproken worden via de API, deze functionaliteit wordt verder besproken in sectie 4.3.




Figuur 4: Deployment diagram

## 3.2 Databank

We maken gebruik van een relationele databank om de gegevens bij te houden. Zoals te zien in Figuur 5 staat de map (kaart) centraal. Elke map stelt een fysieke locatie voor waar tags en anchors aanwezig zijn. Tags kunnen op hun beurt één of meerdere labels hebben en labels kunnen tot één of meerdere tags behoren. Voor elke tag worden ook de historische posities bijgehouden in taglocation, deze zijn voorzien van een timestamp. Voor elke kaart kunnen zones en triggers aangemaakt worden.

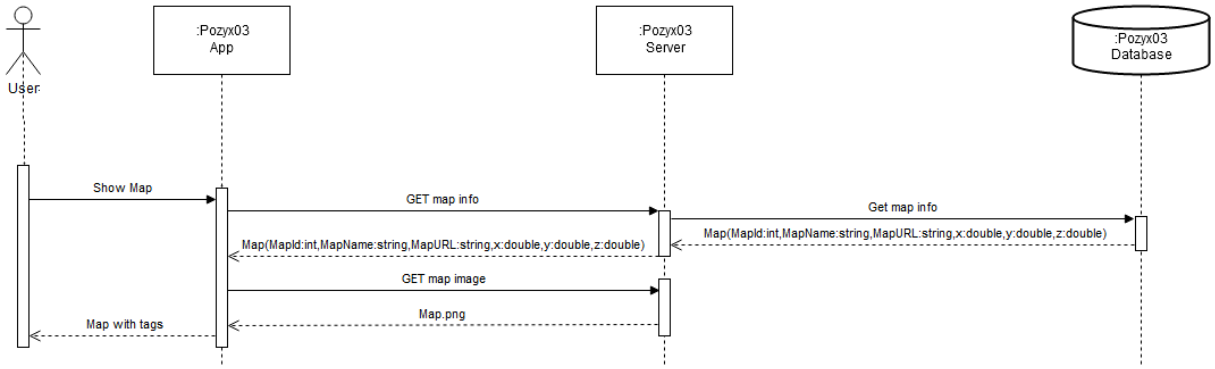



Figuur 5: Databank diagram

## 3.3 Sequentiediagrammen

### 3.3.1 Kaart

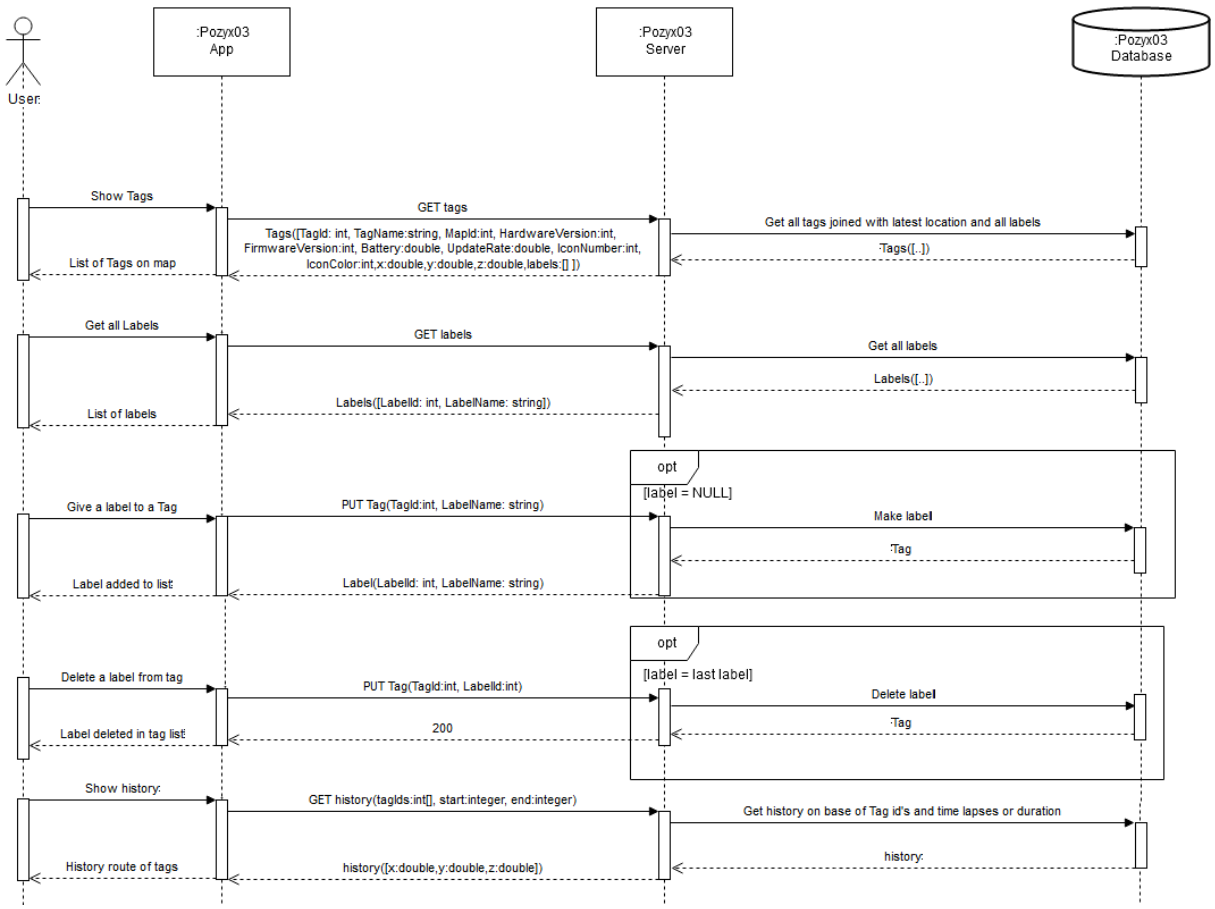
De gebruiker kan kiezen welke kaart zichtbaar is via het kaart icoon rechtsboven. Figuur 6 toont de achterliggende stappen. Eerst wordt de kaart info opgehaald en vervolgens wordt de bijhorende achtergrond afbeelding opgehaald.




Figuur 6: Sequentie diagram: Kaart

### 3.3.2 Tags

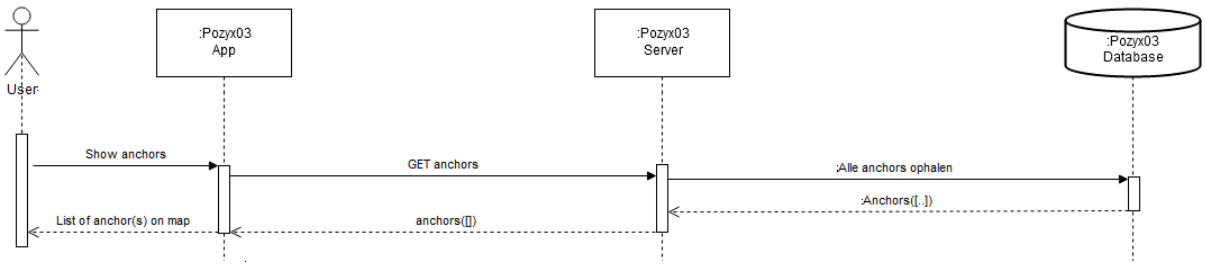
Het beheer van tags is opgedeeld in meerdere sequentie diagrammen. Figuur 7 omvat deze. Het oplijsten gebeurt door de optie tags te kiezen in het linker menu. De tags worden met hun id, naam, labels en batterij opgelijst. Vervolgens is het mogelijk om labels toe te kennen aan tags en labels te ontnemen.




Figuur 7: Sequentie diagrammen: Tags

### 3.3.3 Anchors

Het oplijsten van alle anchors is mogelijk door in het linkermenu de optie anchors te kiezen. De anchors worden vervolgens met id, naam, firm- en hardware version opgelijst. Figuur 8.



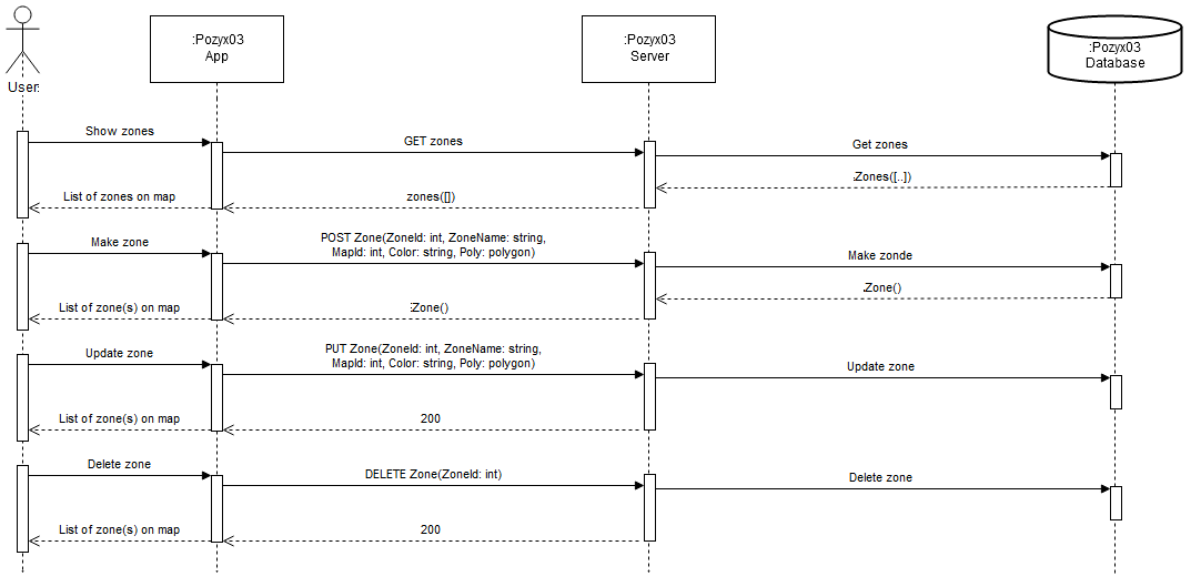

Figuur 8: Sequentie diagram: Anchors

### 3.3.4 Zones

Boven de plattegrond is er een menubalk die het beheren van zones mogelijk maakt.

De knop 'create zone' begint het aanmaken van een nieuwe zone. Per klik worden de hoekpunten van de nieuwe zone als polygon bepaald en men sluit de figuur door bij het laatste punt te dubbelklikken. Het dialoogvenster om de zone op te slaan laat toe de zone te benoemen en een kleur toe te kennen. De wijzigingen worden uiteindelijk bevestigt of verworpen met de 'create zone' of 'cancel' optie. De polygon wordt opgeslagen in de databank (a.d.h.v. de coordinaten van de punten), samen met een zoneId, mapId, kleur en naam van de zone (Figuur 9).

Linksboven de menubalk van zones is er een toggle die zones kan verbergen. Verder is er nog een tweede toggle optie om zones te verwijderen. In deze modus krijgen de zones een kruis in de linkerbovenhoek om ze één per één op de plattegrond te verwijderen. Hiervoor moeten de zones eerst zichtbaar staan.




Figuur 9: Sequentie diagrammen: Zones

### 3.3.5 Triggers

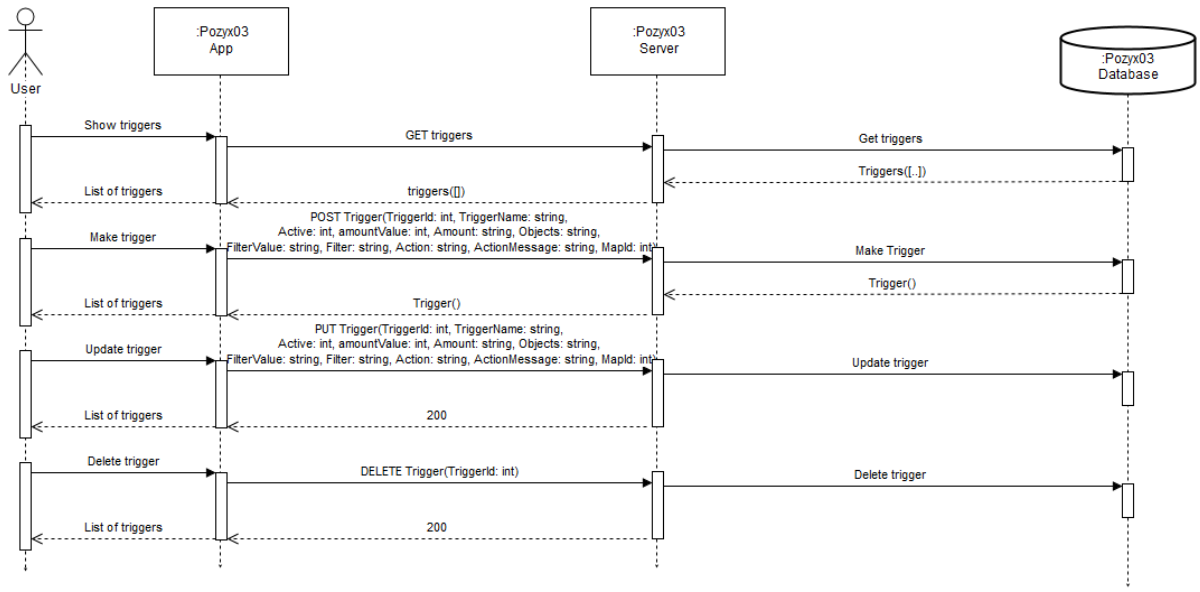
Het menu 'Triggers' begeleid de gebruiker om interactief en eenvoudig nieuwe triggers aan te maken. Triggers zijn controle en analyse configuraties die automatisch een notificatie uitsturen wanneer er aan de vooraf geselecteerde voorwaarden wordt voldaan. Op basis van use cases is dit menu uitgewerkt om zoveel mogelijk scenario's te dekken en tegelijk toch ongecompliceerd te blijven.

Men begint het bouwen van een trigger door op de knop rechtsonder het scherm te klikken, zoals aangegeven in de instructies op het scherm. Als tweede stap kan een filter worden toegevoegd, dit is waar de trigger telkens op controleert. Dergelijke opties zijn "Binnen Zone", "Buiten Zone", "Met Label", "Batterijpercentage" en "Naam bevat"(Figuur 10). Onder zone-opties kan een bepaalde zone worden geselecteerd waar de tags mee interacteren, zoals in de gespecificeerde zone aanwezig zijn of deze verlaten. Label en naam gaan specifiek de tags zelf filteren en tenslotte geeft batterijpercentage de mogelijkheid om de trigger te configueren aan de hand van een bereik waarbinnen het batterijpercentage moet liggen om een notificatie te krijgen.

Telkens is er de optie om te selecteren hoeveel tags er minimaal/exact/maximaal moeten voldoen aan de geselecteerde voorwaarden. Dit zorgt ervoor dat er scenario's kunnen worden toegevoegd in verband met maximale bezetting in een ruimte, een minimale voorraad of aanwezigheid van producten of mensen binnen een bepaalde omgeving en daarnaast nog veel andere belangrijke use cases.

Het laatste scherm laat de gebruiker een zinvolle naam geven aan de trigger en geeft de optie mee om deze op actief in te stellen. De finish-knop bevestigt de wijzigingen.

Wanneer er aan alle voorwaarden van een trigger wordt voldaan komt er bij het notificatieicoon een nieuwe melding.




Figuur 10: Sequentie diagrammen: Triggers

# 4 Testplan

We maken zowel gebruik van manuele als van geautomatiseerde testen. Er wordt zowel manueel getest of de front-end werkt zoals beschreven in de use cases als via unit testing. Ook het testen van triggers wordt geautomatiseerd adhv een script dat vaste scenario's overloopt.

## 4.1 Unit testen

Voor de front-end worden verschillende unit testen voorzien.

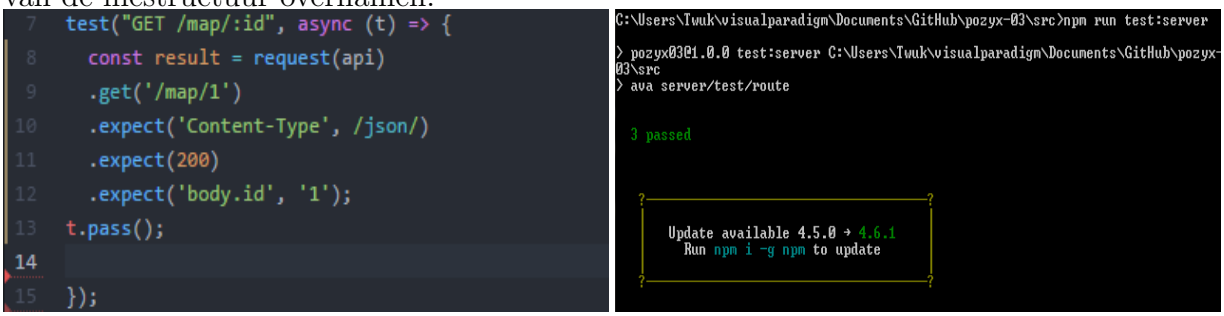
**Component Testen** Aangezien er in de frontend met **React** gewerkt wordt, wordt alles wat waarneembaar is voor de gebruiker geïmplementeerd als JSX-componenten. Op deze componenten wordt onder andere getest of zij de juiste *props* meekrijgen en correct hun *Child-components* weergeven op basis van die props.

**Reducer Testen** De state van de componenten wordt beheerd aan de hand van **redux**. Redux slaat alle data van de applicatie op binnen een *store* en gebruikt zogenaamde *reducers* om de data binnen die store te transformeren. Reducer Testen garanderen dat data die de state binnenkomen en buitengaan steeds correct worden verwerkt.

**Saga Testen** Redux is op zich synchroon. Voor alle *asynchrone* zaken binnen de applicatie wordt gebruik gemaakt van **redux-saga**. Momenteel worden sagas gebruikt voor het maken van API-calls en het opzetten van de websocket-verbinding voor de live posities. Dit kan wel eens foutlopen, het testen van deze sagas ligt dus voor de hand.

**Ava Testen** Als grote overkoepelende testfactor opteerden we voor Ava testen. Deze maken het mogelijk om alle javascript te testen in combinatie met bovenstaande componenten. Spijtig genoeg door de jeugdige versie van AVA in combinatie met BABEL hebben we enkel een kleine API test kunnen schrijven waarbij 3 calls gecontroleerd worden of er effectief het juiste JSON-object teruggegeven worden. De andere testen die we uitprobeerden gaven fouten omdat ze door onze BABEL versie niet de juiste naamgeving

van de filestructuur overnamen.




## 4.2 Manuele testen

De functionaliteit van de frontend wordt ook manueel worden getest. Hiervoor wordt gecontroleerd aan de hand van de use cases en de unit tests of de frontend zowel grafisch als functioneel het gewenste gedrag vertoont.

Voor verschillende uitgewerkte scenarios werden use cases bedacht. Tags kunnen zones intreden of verlaten, met een minimaal of maximaal aantal aanwezig zijn binnen een zone of hun batterij kan te laag worden. Hierop werden de unit tests gebaseerd en bijgevolg de vereiste mogelijkheden om tags te beheren werden geïmplementeerd. Het is dus mogelijk om tags op te laden, te ontladen of hun batterijpercentage bevriezen. Verder kunnen ze teleporteren naar een bepaalde plaats, naar een doel toe bewegen en eventueel blijven stilstaan op die plaats. In Tabel 4 zijn er twee voorbeelden van acties die kunnen worden uitgevoerd, in dit geval gespecificeerd voor elke individuele tag.

Tijdens de manuele testen werd er gecontroleerd of er problemen ontstonden bij overlappende zones (onnauwkeurigheden) of tags die eventueel vastliepen buiten het canvas of in de hoeken van het scherm. Er werden geen problemen gevonden bij het simulatie van tags.

## 4.3 Trigger testen

Om triggers te testen kan men zelf scenario's bedenken. De scenario's waarbij een trigger zich moet voordoen kunnen dan aan de hand van het live script gesimuleerd worden. Indien de trigger zich effectief voordoet bij het testen van het scenario dan is de test succesvol.

### 4.3.1 Voorbeeld test

Een trigger gaat af zodra 2 rode tags zich in de living bevinden. Om dit scenario te kunnen testen is er eerst een trigger nodig die filtert op tags met label "rood"én die zich in de specifieke zone "living"bevinden (in dit voorbeeld heeft living als zone id 2). Vervolgens kunnen we het scenario simuleren door het live script alle tags naar zelf gekozen locaties buiten de living te laten teleporteren en ze naar welgekozen punten in de living te laten lopen. Zodra er 2 rode tags in de zone aangekomen zijn wordt de trigger afgevuurd. Listing 1 toont hoe het trigger object voor dit scenario eruit ziet.

Listing 1: Trigger body zoals in een PUT request naar */api/map/1/trigger/1*

```
{
  "name": "At least 2 red tags in living",
  "json" : {
    "active": true,
    "comparator": {
      "type": "atLeast",
      "value": 2
    },
    "action": {
      "type": "notify",
      "value": ""
    },
    "filters": [
      {
        "type": "inZone",
        "value": 2
      },
      {
        "type": "label",
        "value": "rood"
      }
    ]
  }
}
```

Listing 2 toont hoe het simulatie script eruit zou zien. We gaan ervan uit dat de punten in "teleportLocations"buiten de zone "living"zijn en de punten in "targetLocations"binnen de zone liggen. Elk achtereenvolgende locatie in beide arrays (teleportLocations en targetLocations) is voor een specifieke tag. De vlag "haltAtTarget"geeft aan dat de tags moeten stoppen zodra ze aangekomen zijn aan hun respectievelijke target locations.

Listing 2: Scenario body zoals in een POST request naar */command/scenario*

```
{
    "type": "TELEPORT",
    "scenario": {
        "teleportLocations": [
            {"x": 0, "y": 0},
            {"x": 20, "y": 0},
            {"x": 40, "y": 0},
            {"x": 60, "y": 0},
            {"x": 80, "y": 0},
            {"x": 100, "y": 0}
        ],
        "targetLocations": [
            {"x": 50, "y": 50},
            {"x": 50, "y": 50},
            {"x": 50, "y": 50},
            {"x": 50, "y": 50},
            {"x": 50, "y": 50},
            {"x": 50, "y": 50}
        ],
        "haltAtTarget": true
    }
}
```

Het is ook mogelijk batterij verbuik en opladen te simuleren. Listing 3 is hiervan een voorbeeld. Analoog wordt voor elke tag het wenselijke gedrag in een array meegegeven.

Listing 3: Scenario body zoals in een POST request naar */command/scenario*

```
{
    "type": "BATTERY",
    "scenario": {
        "modes": [
            "FREEZE",
            "CHARGE",
            "CHARGE",
            "CHARGE",
            "CHARGE",
            "DRAIN",
            "DRAIN",
            "DRAIN",
            "DRAIN"
        ]
    }
}
```

### 4.3.2 Testresultaten

Het testen van triggers is getest met controlestructuren "minder dan" en "meer dan" met een of meerdere combinaties van volgende filters: "in zone", "buiten zone", "met label x". Deze testen werden lokaal uitgevoerd gedurende sprint 3 door Pieter-Jan Vandenberghe en nog eens live getoond in onze demo op 16/05. De resultaten zijn positief voor deze gevallen. De batterij simulatie is getest op werking maar niet opgenomen in het testen van triggers omdat hiervan geen feedback is in de front-end (de gebruiker kan niet live verifiëren wat er met de batterij gebeurt, enkel bij het opvragen van tags in lijst worden deze opgehaald uit db).

# 5 Installatiehandleiding

## 5.1 Vereisten

- `node 5.0.0` (via `https://nodejs.org`) of recenter (inclusief npm) met volgende package(s):
  - `node-gyp` (instructies onder installatie)
- Enkel voor development: je favoriete JavaScript editor met volgende package(s):
  - `linter-eslint`
- Enkel voor (frontent) development: Redux DevTools for Chrome

## 5.2 Installatie

Er zijn slechts enkele stappen vereist om het project op poten te zetten, de overige packages worden automatisch geïnstalleerd.

1. Clone of download de repository via de UGent Github: `https://github.ugent.be/iii-vop2017/pozyx-03.git` of `https://github.ugent.be/iii-vop2017/pozyx-03/archive/master.zip`
2. Open je favoriete terminal en ga naar de root van het project
3. Indien je nog geen node-gyp hebt installeer eerst de vereisten voor node-gyp (voor zover je deze nog niet hebt):

- Unix:
  - `python v2.7` via `https://python.org`
  - `make` via `https://gnu.org/software/make/`
  - Een C/C++ compiler toolchain, zoals GCC
- Mac OS X:
  - `python v2.7` via `https://python.org`

- `Xcode` via `https://developer.apple.com/xcode/download/`
- Inclusief `Command Line Tools`, terug te vinden onder `Xcode -> Preferences -> Downloads`
- Deze stap installeert `gcc` en de `make` toolchain

- Windows:
  - in console **met admin rechten**: `npm install --global --production windows-build-tools`

4. Vervolgens installeer je node-gyp zelf: `npm install --global node-gyp`
5. Zodra je node-gyp hebt kan je alle packages installeren met `npm install`

## 5.3 Configuratie Databank

We maken gebruik van de **MariaDB** databank en **Sequelize** ORM. De nodige plugins werden bij installatie reeds voorzien.

De configuratie van de databank verbinding gebeurt in ../server/src/config/, daar dient een `config.json` file voorzien te worden.

Voorbeeld:

Listing 4: Database productie configuratie

```
{
  "production": {
    "database": "pozyx",
    "host": "127.0.0.1",
    "username": "username",
    "password": "password",
    "dialect": "mariadb"
  }
}
```

De waarden "username" en "password" dienen worden te vervangen door de login gegevens die worden gebruikt om te connecteren met de databank.

In de test omgeving wordt gebruik gemaakt van een filler script dat de databank opvult:

```
npm run seed
```

## 5.4 Development

Volgend commando start alle benodigde tools voor fullstack development (webpack dev + hot reloading, nodemon, etc):

```
npm run dev
```

## 5.5 Productie

Build de applicatie eerst:

```
npm run build
```

Vervolgens kan de productie server draaien:

```
npm run prod
```

Met het commando `npm start` worden beide bovenstaan commando's in volgorde uitgevoerd.

## 5.6 API

De api documentatie wordt gegenereerd met [apiDoc]. Volgend commando genereert de documentatie:

```
npm run doc
```

Vervolgens is de documentatie terug te vinden onder ../server/public/apidoc/