Rapport Project Databases

1. Status

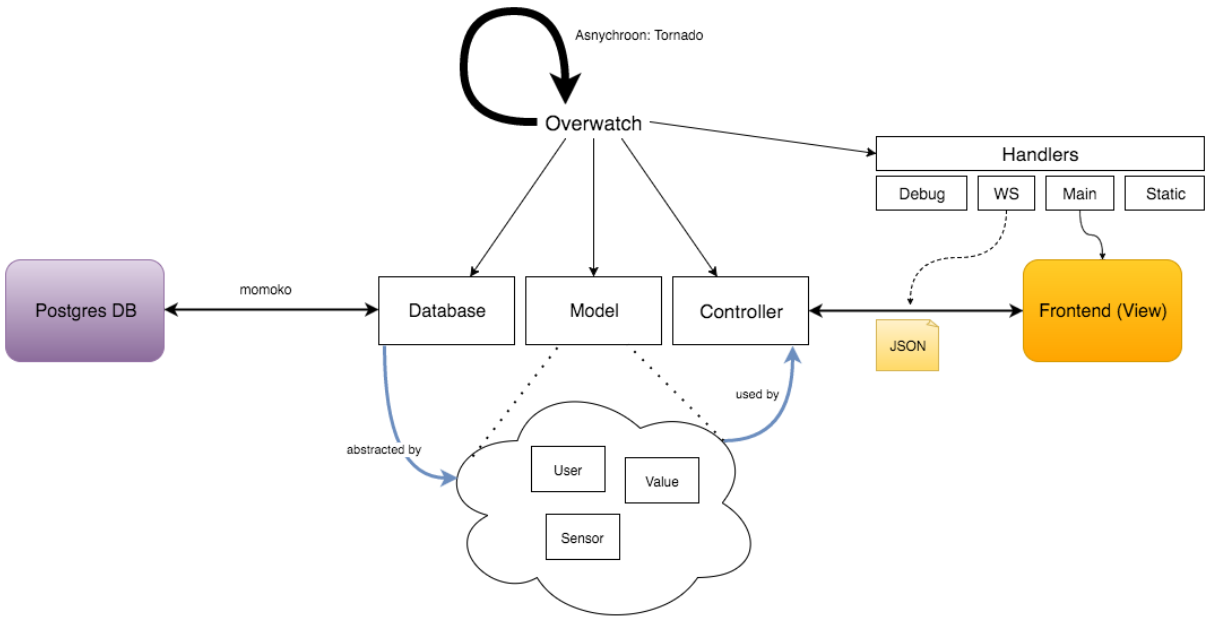
Werkende features

- Login en signup
- Opzetten database, backend etc.

Werkverdeling

- **Iedereen**: SQL
- **Sam:** HTML/CSS, sterk bezig met de layout v/d frontend en dus MDL en Google Charts.
- **Stijn**: Communicatie HTML en JS, sterk bezig met JS Framework (AngularJS). Verantwoordelijk voor frontend in zijn geheel.
- **Jeroen**: JavaScript, de infrastructuur achter de frontend en is ook verantwoordelijk voor communicate met backend.
- **Anthony**: Infrastructuur backend (in Python). Controller in MVC-termen.
- **Evert**: Algoritmes en aggregaties etc. Voornamelijk het Model in MVC-termen.

2. Design



Inleiding

We hebben ervoor gekozen om onze applicatie als SPA (Single-Page Application) te laten werken. Voor de backend gebruiken we Python 3.5, met Tornado als framework. Als database gebruiken we Postgres, met momoko als library om te communiceren. Voor de frontend gebruiken we Angular als framework, met MDL en Google Charts als voornaamste bouwstenen voor de UI. Communicatie tussen back end en front end gebeurt via websockets met een zelfgemaakte JSON API.

Backend

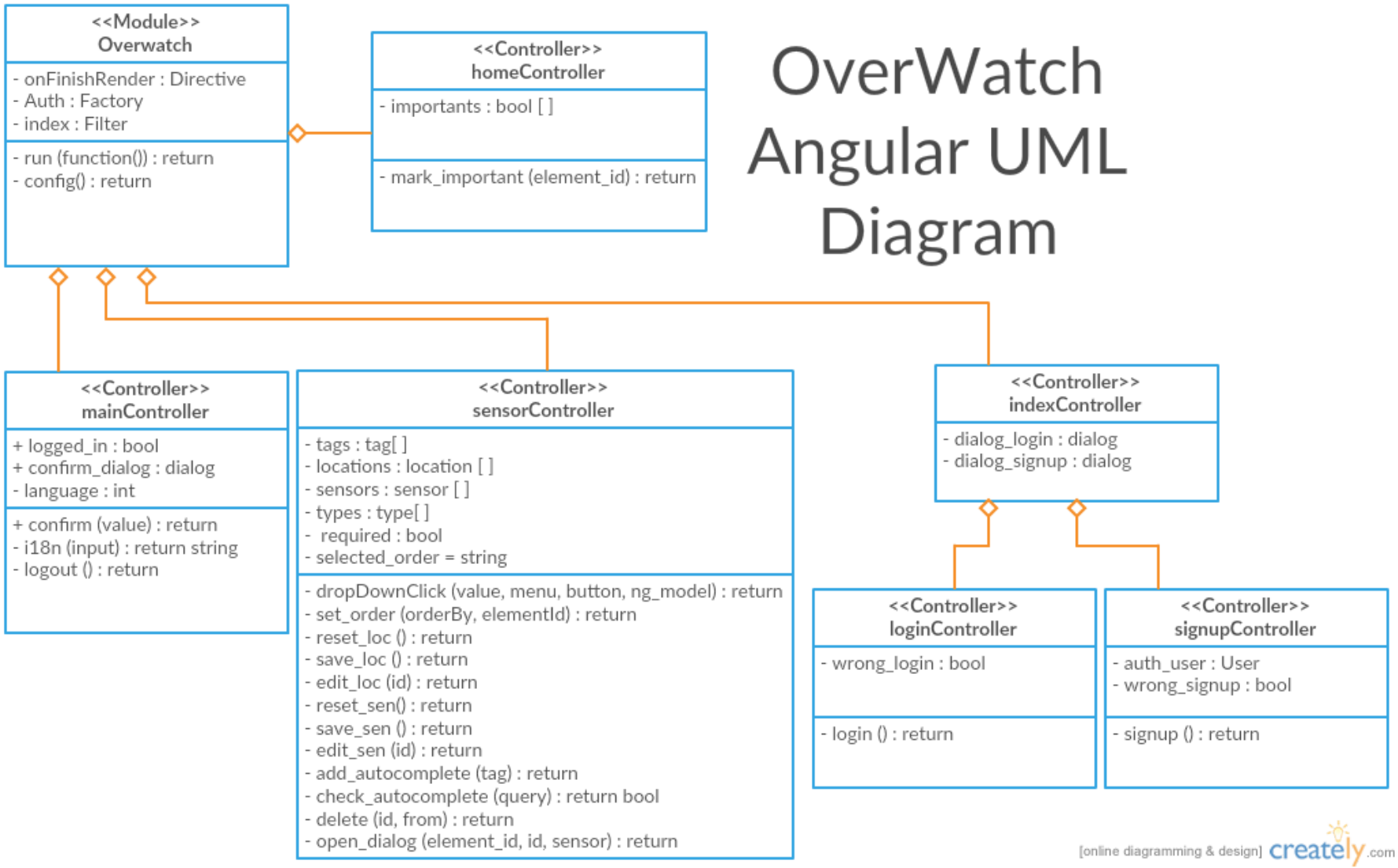
De back end draait zoals eerder gezegd op Python 3.5 (of nieuwer). Dit is geen voor de hand liggende keuze, want deze versie is niet zomaar op alle platformen beschikbaar omdat ze zo nieuw is. De reden hiervoor is de superieure support voor asynchrone code (`def async`, `await`, `async for`, ...). Tornado is namelijk een asynchrone library, dit zorgt voor een forse verbetering in de performantie. Tornado wordt aanzien als een iets moeilijkere library dan bv. Flask of Django, we hebben hier toch voor gekozen omdat de andere frameworks slecht overweg kunnen met langlopende connecties zoals die gebruikt worden voor websockets.

De achterliggende database is Postgres, en we gebruiken momoko om ermee te communiceren. Momoko zorgt voor een asynchrone manier van communicatie met de database, zodat het past binnen de rest van ons systeem.

Frontend

De frontend verschilt sterk van de meeste andere websites, we hebben namelijk voor een SPA gekozen. Dit geeft tot resultaat dat er eigenlijk maar één html pagina wordt ingeladen per ‘bezoek’ aan de website, en dat de rest via Javascript moet worden afgehandeld. Het framework dat we hiervoor gebruiken is Angular. Daarnaast gebruiken we Material Design Lite voor de layout en Google Charts voor de grafieken.

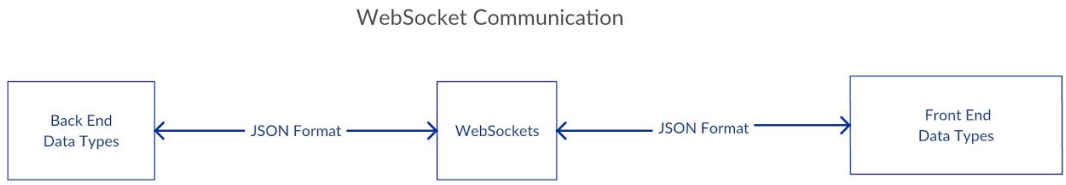
OverWatch Angular UML Diagram

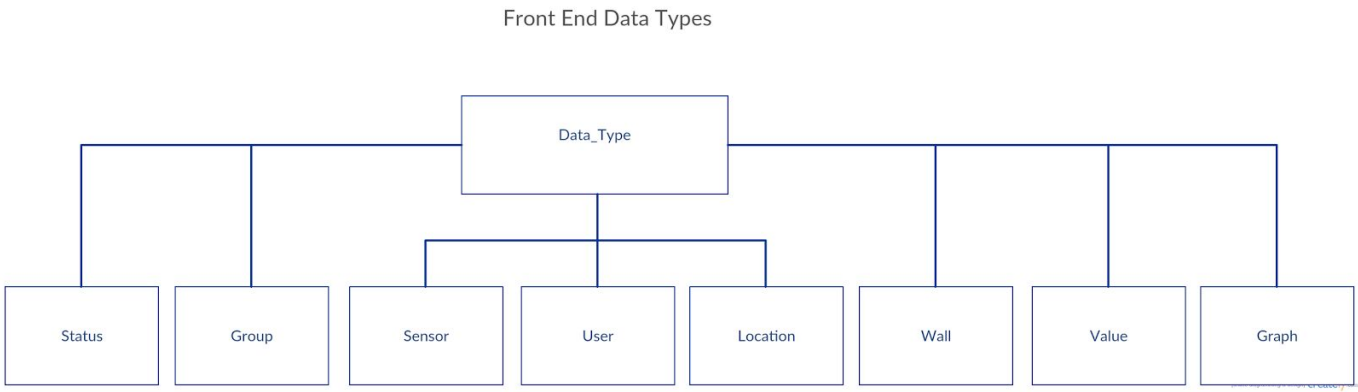


Communicatie

Nadat de gebruiker de initiële webpagina heeft ingeladen, maakt hij ook een verbinding met de websocket handler van onze server. Dit is de voornaamste en praktisch enige manier van communicatie tussen front end en back end. De communicatie zelf gebeurt door JSON-berichten heen en weer te sturen. We gebruiken hiervoor een zelfgemaakte JSON API. Deze API moet door zowel de front end als de back end gerespecteerd worden.

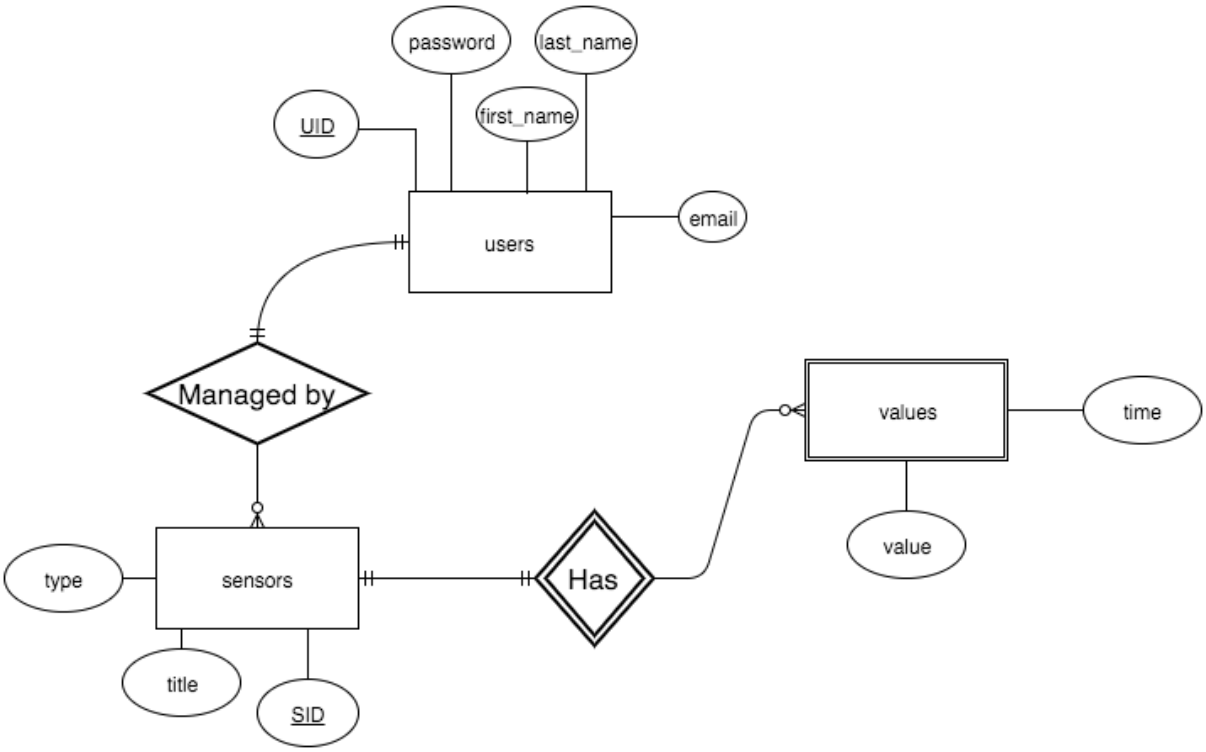
UML-Schema websocket communicatie





Database schema

Huidige database



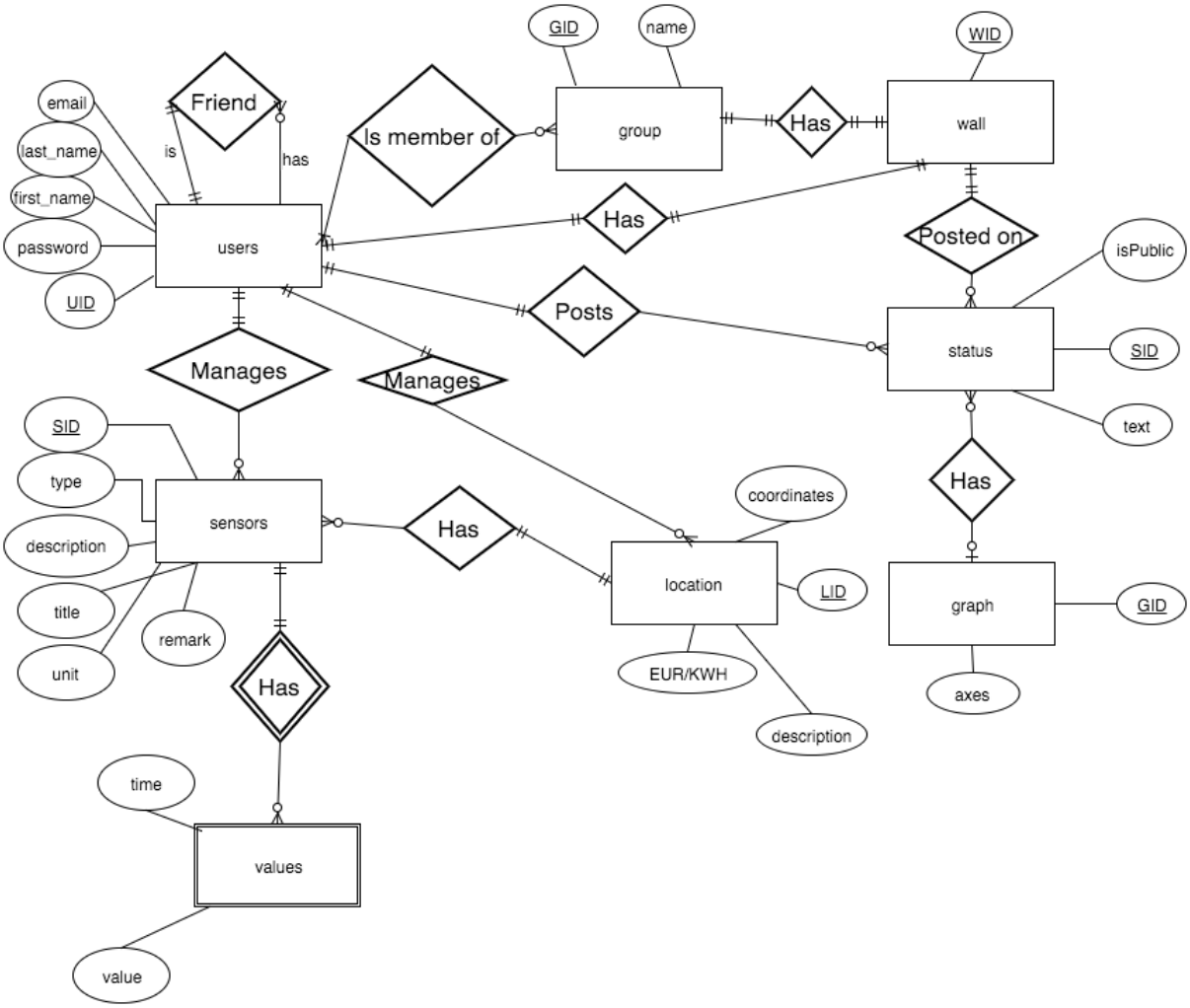
Uitleg huidige database

De uitleg van de entiteiten en hun attributen staat in de `CREATE TABLE statements` (Zie appendix).

Momenteel maken we gebruik van twee relaties:

1. Relatie users en sensors (“Managed by”):
    a. Elke sensor wordt beheerd door 1 user (referentiële integriteit)
    b. 1 user kan 0 of meerdere sensors beheren (0-many)
2. Relatie sensors en values (“Has”,weak relationship):
    a. Elke value behoort tot 1 sensor (referentiële integriteit)
    b. Een sensor kan 0 of meerdere values hebben (0-many)

Toekomstige database



Uitleg toekomstige database

De volgende uitleg is slechts een schets van de uiteindelijke database en zal dus naarmate het project vordert nog aangepast worden.
Uiteindelijk hebben we de volgende entiteiten met hun bijhorende attributen:

1. Users
    a. UID: PRIMARY KEY
    b. password: VARCHAR
    c. first name: VARCHAR
    d. last name: VARCHAR
    e. email: UNIQUE, NOT NULL (elke user logt in aan de hand van zijn email dus moet deze uniek en not null zijn)
    f. WID: verwijzing naar de ID van de wall van de user
2. Sensors
    a. SID: PRIMARY KEY
    b. type: VARCHAR
    c. title: VARCHAR

    d. description: VARCHAR
    e. unit: VARCHAR
    f. remark: VARCHAR
    g. UID: verwijzing naar de ID van de user die deze sensor beheert
3. Values (weak entity)
    a. time: TIMESTAMP
    b. value: DOUBLE PRECISION
    c. SID: verwijzing naar de sensor waartoe de bepaalde value behoort
4. Group
    a. GID: PRIMARY KEY
    b. name: VARCHAR
    c. WID: verwijzing naar de WID van de wall van de group
5. Wall
    a. WID: PRIMARY KEY
6. Status
    a. SID: PRIMARY KEY
    b. text: VARCHAR
    c. isPublic: BOOLEAN
    d. WID: verwijzing naar de WID van de wall van de status
    e. UID: verwijzing naar de UID van de user van de status
    f. GID: verwijzing naar de GID van de graph van de status, kan NULL zijn
7. Graph
    a. GID: PRIMARY KEY
    b. axes: VARCHAR (Moet nog verder uitgewerkt worden)
8. Location
    a. LID: PRIMARY KEY
    b. coordinates: VARCHAR
    c. EUR/KWH: VARCHAR
    d. description:VARCHAR

De toekomstige database zal de volgende relaties bevatten:

1. Relatie users en sensors (“Manages”):
    a. Elke sensor wordt beheerd door 1 user (referentiële integriteit)
    b. 1 user kan 0 of meerdere sensors beheren (0-many)
2. Relatie sensors en values (“Has”,weak relationship):
    a. Elke value behoort tot 1 sensor (referentiële integriteit)
    b. Een sensor kan 0 of meerdere values hebben (0-many)
3. Relatie users en locations (“Manages”):
    a. Elke user beheert 0 of meerdere locaties (0-many)
    b. Een locatie wordt beheerd door 1 user (referentiële integriteit)
4. Relatie users en users (“Friend”, gebruik van roles):
    a. Een user is een friend (referentiële integriteit)
    b. Een user kan 0 of meerdere vrienden hebben (0-many)
5. Relatie users en group (“Is member of”):
    a. Een user kan in 0 of meerdere groepen zitten (0-many)
    b. Een group kan 1 of meerdere users bevatten (1-many)

6. Relatie users en wall (“Has”):
    a. Elke user heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 user (referentiële integriteit)
7. Relatie users en status (“Posts”):
    a. Een user kan 0 of meerdere statussen posten (0-many)
    b. Een status behoort tot 1 user (referentiële integriteit)
8. Relatie status en graph (“Has”):
    a. Een status kan 0 of 1 graph bevatten (0-1)
    b. Een graph kan gebruikt worden in 0 of meerdere statussen (0-many)
9. Relatie status en wall (“Posted on”):
    a. Een status kan maar 1 wall hebben (referentiële integriteit)
    b. Een wall kan 0 of meerdere statussen bevatten (0-many)
10. Relatie group en wall:
    a. Een group heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 group (referentiële integriteit)
11. Relatie sensors en location (“Has”):
    a. Elke sensor heeft 1 locatie (referentiële integriteit)
    b. Een locatie kan 0 of meerdere sensoren bevatten (0-many)

De toekomstige database zal de volgende relaties bevatten:

1. Relatie users en sensors (“Manages”):
    a. Elke sensor wordt beheerd door 1 user (referentiële integriteit)
    b. 1 user kan 0 of meerdere sensors beheren (0-many)
2. Relatie sensors en values (“Has”,weak relationship):
    a. Elke value behoort tot 1 sensor (referentiële integriteit)
    b. Een sensor kan 0 of meerdere values hebben (0-many)
3. Relatie users en locations (“Manages”):
    a. Elke user beheert 0 of meerdere locaties (0-many)
    b. Een locatie wordt beheerd door 1 user (referentiële integriteit)
4. Relatie users en users (“Friend”, gebruik van roles):
    a. Een user is een friend (referentiële integriteit)
    b. Een user kan 0 of meerdere vrienden hebben (0-many)
    c. Belangrijke constraint hier is dat de eerste UID in de relatie kleiner is dan de tweede UID om redundancy te vermijden
5. Relatie users en group (“Is member of”):
    a. Een user kan in 0 of meerdere groepen zitten (0-many)
    b. Een group kan 1 of meerdere users bevatten (1-many)
6. Relatie users en wall (“Has”):
    a. Elke user heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 user (referentiële integriteit)
7. Relatie users en status (“Posts”):
    a. Een user kan 0 of meerdere statussen posten (0-many)
    b. Een status behoort tot 1 user (referentiële integriteit)
8. Relatie status en graph (“Has”):
    a. Een status kan 0 of 1 graph bevatten (0-1)

    b. Een graph kan gebruikt worden in 0 of meerdere statussen (0-many)
9. Relatie status en wall (“Posted on”):
    a. Een status kan maar 1 wall hebben (referentiële integriteit)
    b. Een wall kan 0 of meerdere statussen bevatten (0-many)
10. Relatie group en wall:
    a. Een group heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 group (referentiële integriteit)
11. Relatie sensors en location (“Has”):
    a. Elke sensor heeft 1 locatie (referentiële integriteit)
    b. Een locatie kan 0 of meerdere sensoren bevatten (0-many)

3. Product

Basisvereisten

We hebben de applicatie zelf opgedeeld in 5 pagina’s: de login pagina, een home pagina, een sensor pagina, een statistiek pagina en een pagina met de sociale media.

1. Login pagina
De login pagina is momenteel redelijk basic: Een login knop voor bestaande gebruikers, een signup knop voor nieuwe gebruikers en een welkomsttekstje voor iedereen.

2. Home pagina
De home pagina zal berichten en grafieken laten zien die door de applicatie als belangrijk worden beschouwd. De gebruiker kan zelf zijn bijdrage doen aan deze selectie door berichten en grafieken zelf als belangrijk te markeren.

3. Sensor pagina
Op deze pagina kan de gebruiker zijn sensoren en locaties beheren. De sensoren worden weergegeven in een lijst. Aangezien deze lijst heel groot kan worden, hebben we er voor gezorgd dat er als het ware verschillende “pagina’s” aanwezig zijn. Dit mechanisme van “pagination” is ook toegepast op de lijst met locaties.
Sensors en locaties toevoegen en aanpassen gebeurt via een dialog. Veel van de gevraagde input is puur tekst, maar bijvoorbeeld de tags hebben een aangepast inputmechanisme.

4. Statistiek pagina
Men kan data krijgen door sensoren, tags, huizen,... manueel te selecteren. In de toekomst zouden wij misschien een zoekfunctie inbouwen om dit gemakkelijker te maken. Er zijn ook knoppen voorzien om de data te aggregeren op type, locatie, tag of sensor.
De gebruiker zal ook moeten ingeven hoeveel dagen, maanden of jaren hij wilt terugkijken in de tijd.
De grafieken die worden weergegeven, kunnen dan worden gedeeld met vrienden of je kan ze aanduiden als belangrijk en deze grafieken zullen dan verschijnen op de home pagina.

5. Sociale media
Je kan grafieken delen met je vrienden. Je kan ook berichten schrijven. Deze berichten zullen zichtbaar zijn op de wall van de persoon naar wie je het bericht stuurt .Er zullen ook groepen beschikbaar zijn, elk met hun eigen wall, groepsleden en een administrator.

Al deze pagina's zijn ook voorzien van een functie die de hele applicatie vertaalt naar het Engels of Nederlands.

Extra functionaliteit

Extra functionaliteit vanuit het end-user perspectief is momenteel nog niet aanwezig in het project. We zullen dit plannen naarmate de basisvereisten verder vorderen.

4. Planning

Tegen volgende presentatie moeten vanzelfsprekend alle basisvereisten af zijn. Tegen deze presentatie proberen we alvast werkende sensor-management te hebben, met grafieken (mogelijks statische placeholders).

5. Appendix

SQL code

```
CREATE TABLE Users (
    UID          SERIAL PRIMARY KEY,
    password     VARCHAR,
    first_name   VARCHAR,
    last_name    VARCHAR,
    email        VARCHAR UNIQUE NOT NULL
);

CREATE TABLE Sensors (
    SID          SERIAL PRIMARY KEY,
    type         VARCHAR,
    title        VARCHAR,
    UID          INT REFERENCES Users
);

CREATE TABLE Values (
    SID          INT REFERENCES Sensors,
    time         TIMESTAMP,
    value        DOUBLE PRECISION,
    PRIMARY KEY (SID, time)
```

```
);
```

Tot op heden zijn er geen niet-triviale SQL queries.