# Rapport Project Databases

**Inhoudsopgave**

# 1. Status

## Werkende features

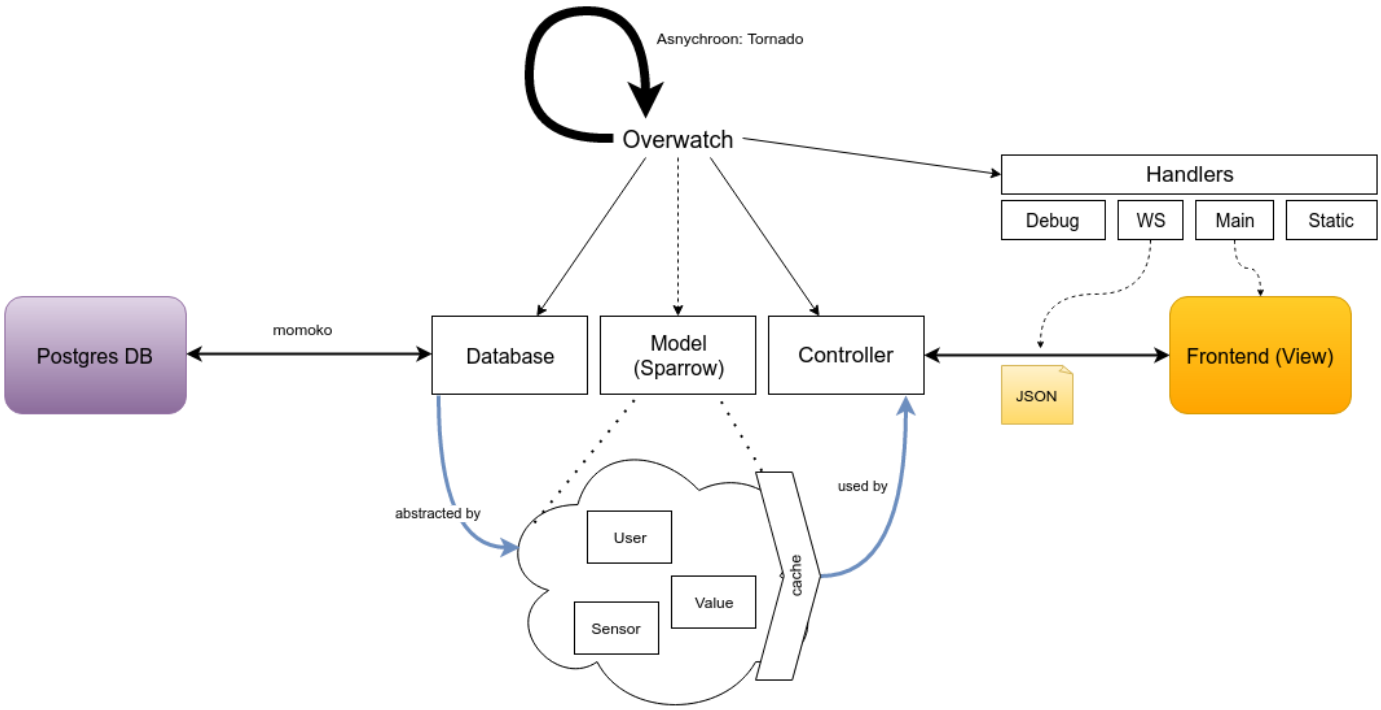
- Sensormanagement
- Locatiemanagement
- Statistics
- Sociale tab

## Werkverdeling

- **Sam:** HTML/CSS, sterk bezig met de layout v/d frontend en dus MDL en 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 de backend.
- **Anthony**:  Infrastructuur backend (in Python). Controller in MVC-termen, SQL
- **Evert**: Algoritmes en aggregaties etc., Sparrow (dus het Model in MVC), SQL

# 2. Design

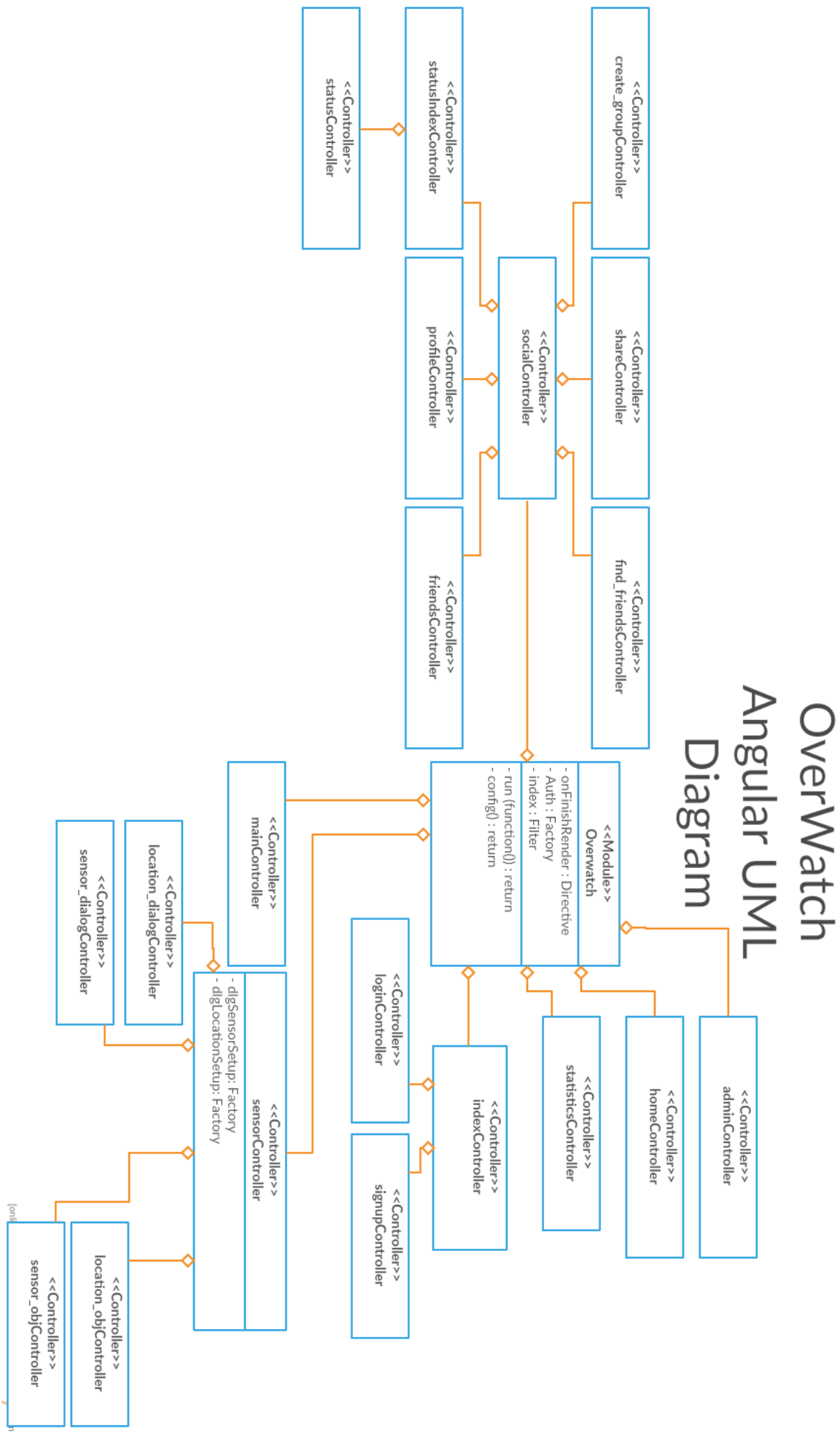
## Backend




Deel van de opdracht is vertrouwd geraken met SQL, en daarom is het ook logisch dat kant-en-klare relational models zoals Django's ORM niet toegestaan zijn. Omdat we toch de flexibiliteit van zo'n framework op prijs stelden, hebben we zelf zo'n framework gemaakt. Om modulariteit te bevorderen staat deze zelfs in een aparte repository en heeft een eigen naam: Sparrow. Er is ook eigen documentatie voor geschreven: [sparrow.readthedocs.org](sparrow.readthedocs.org). De naam staat voor Single Page Application Real-time Relational Object Wrapper. Zoals aangegeven in het schema hierboven, neemt Sparrow praktisch het hele Model voor zijn rekening (per klasse hebben we dan ook maar een tiental lijnen code). Buiten de flexibiliteit zorgt het framework ook voor caching en betere bescherming tegen SQL-injection aanvallen.

## Frontend

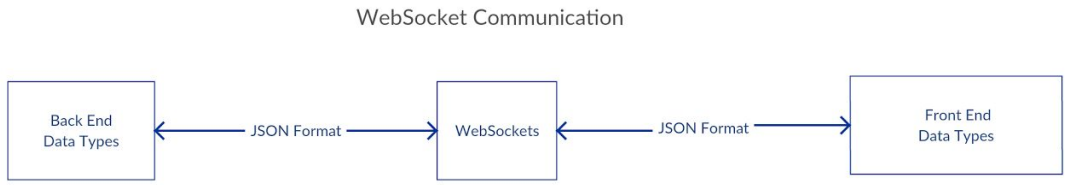
Qua onderliggend design is er niet veel veranderd. Hieronder het nieuwe UML diagram van de Angular code.

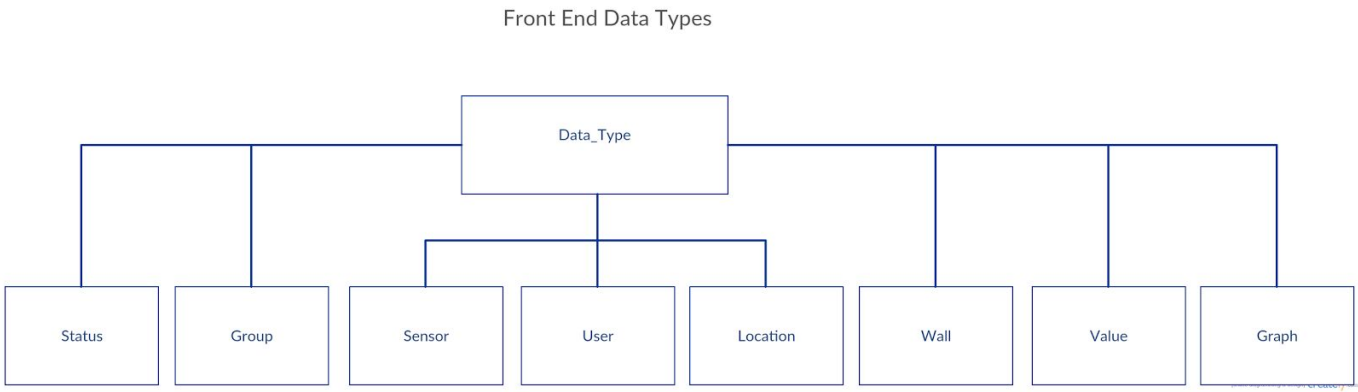

OverWatch
Angular UML
Diagram
<<Module>>
Overwatch
- onFinishRender : Directive
- Auth : Factory
- index : Filter
- run (function()) : return
- config() : return
<<Controller>>
adminController
<<Controller>>
homeController
<<Controller>>
statisticsController
<<Controller>>
indexController
<<Controller>>
signupController
<<Controller>>
loginController
<<Controller>>
sensorController
- dlgSensorSetup: Factory
- dlgLocationSetup: Factory
<<Controller>>
mainController
<<Controller>>
location_dialogController
<<Controller>>
sensor_dialogController
<<Controller>>
sensor_objController
<<Controller>>
location_objController
<<Controller>>
create_groupController
<<Controller>>
shareController
<<Controller>>
socialController
<<Controller>>
find_friendsController
<<Controller>>
friendsController
<<Controller>>
profileController
<<Controller>>
statusIndexController
<<Controller>>
statusController

# 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.

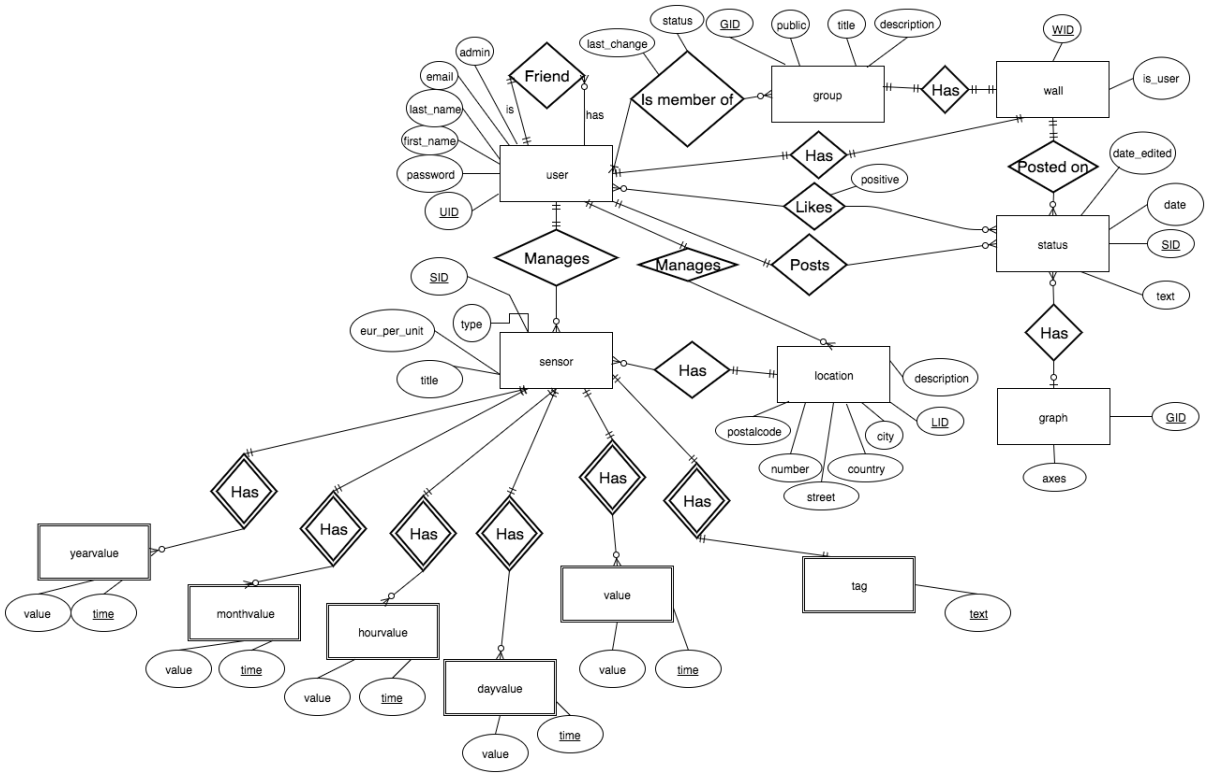
*UML-Schema websocket communicatie*



# Database

## Database schema




## Uitleg database

### Entiteiten

Voor de entiteiten en hun attributen verwijzen we naar de appendix die de `CREATE TABLE` en `CREATE TYPE` statements bevat.

### Relaties

De database bevat de volgende relaties:

1. Relatie user en sensor (“Manages”):
    a. Elke sensor wordt beheerd door 1 user (referentiële integriteit)
    b. Een user kan 0 of meerdere sensors beheren (0-many)
2. Relatie sensor en de verschillende soorten values (“Has”,weak relationship):
    a. Elke value behoort tot 1 sensor (referentiële integriteit)
    b. Een sensor kan 0 of meerdere values hebben (0-many)
    c. De overige value soorten zijn nodig voor de aggregatie van de gegevens
3. Relatie user en location (“Manages”):
    a. Elke user beheert 0 of meerdere locaties (0-many)

    b. Een locatie wordt beheerd door 1 user (referentiële integriteit)
4. Relatie user en user (“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 redundantie te vermijden
5. Relatie user 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)
    c. Bevat de attributen:
        i. Status: Om aan te geven welke rol de user in de group speelt (member, admin etc)
        ii. Last change: Om de laatste verandering aan te geven
6. Relatie user en wall (“Has”):
    a. Elke user heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 user (referentiële integriteit)
7. Relatie user en status (“Likes”):
    a. Een user kan 0 of meer statussen liken (0-many)
    b. Een status kan geliked worden door 0 of meer users (0-many)
    c. Bevat de attributen:
        i. positive: Geeft aan of het een like of dislike was
8. Relatie user en status (“Posts”):
    a. Een user kan 0 of meerdere statussen posten (0-many)
    b. Een status behoort tot 1 user (referentiële integriteit)
9. 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)
10. 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)
11. Relatie group en wall(“Has”):
    a. Een group heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 group (referentiële integriteit)
12. Relatie sensor en location (“Has”):
    a. Elke sensor heeft 1 locatie (referentiële integriteit)
    b. Een locatie kan 0 of meerdere sensoren bevatten (0-many)
13. Relatie sensor en tag (“Has”,weak relationship):
    a. Een sensor kan 0 of meer tags hebben (0-many)
    b. Een tag behoort tot 1 sensor (referentiële integriteit)

# 3. Product

## Basisvereisten

We hebben de applicatie zelf opgedeeld in 6 pagina's: de login pagina, een home pagina, een sensor pagina, een statistiek pagina, een pagina voor administrators en een pagina met de sociale media.

### 1. Login pagina

De login pagina is eenvoudig: Een login knop voor bestaande gebruikers, een signup knop voor nieuwe gebruikers en in het groot het logo van onze website.

### 2. Home pagina

De home pagina doet dienst als een soort splash screen. Je kan er niets doen, het is als het ware de "inkomhal" van onze website.

### 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 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 dialoogvenster. 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. Er zijn ook knoppen voorzien om de data te aggregeren op type, locatie, tag of sensor.
De gebruiker zal moeten ingeven hoeveel dagen, maanden of jaren hij wil terugkijken in de tijd.
Alle grafieken die in 1 sessie worden opgevraagd (= als de pagina niet wordt vernieuwd) zullen blijven staan. Zo kan de gebruiker de verschillende grafieken vergelijken.
De grafieken die worden weergegeven, kunnen dan worden gedeeld met vrienden of met leden van een bepaalde groep.

### 5. Sociale media

Op de sociale pagina('s) kan je vrienden toevoegen, groepen maken, je profielfoto bewonderen (veranderen gebeurt via gravatar.com),...
Je kan grafieken delen met je vrienden, dit doe je door in de statistics pagina een grafiek te genereren en dan op de share knop te klikken in de grafiek. Er zullen ook groepen beschikbaar zijn, elk met hun eigen wall en groepsleden.

### 6. Admin pagina

De admin pagina is op analoge wijze opgebouwd als de statistics pagina. Het verschil zit er in dat de admin in plaats van sensors enkel gebruikers te zien krijgt. Voor het genereren van een rapport zal de admin dan ook de mogelijkheid worden aangeboden om comments bij de gegenereerde grafieken te zetten. Deze pagina is ook print-vriendelijk gemaakt.

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

## Extra functionaliteit

We hebben een aantal extra features bedacht voor onze website. Het verdere verloop van het project zal uitwijzen welke we al dan niet kunnen implementeren.

- Live updates van data
- Company Rebranding
- Superadmin (cfr sudo)
- Extra types en eenheden
- Social media integratie (Facebook, Twitter, …)
- Extrapolatie en interpolatie van data
- Grafieken als belangrijk markeren
- De home pagina een functie geven (grafieken suggereren, belangrijke statussen van vrienden,...)
- Likes en comments op statussen
- Customizable Reports

# 4. Planning

Tegen volgende presentatie moet vanzelfsprekend zoveel mogelijk extra functionaliteit af zijn. Bovendien gaan we ook meer aandacht investeren in de autorisatie van bepaalde requests wat de beveiliging van de gegevens bevordert.

# 5. Appendix: SQL code

## Create table/Create type statements

### Wall

```
CREATE TABLE table_Wall (
	WID SERIAL,
	is_user BOOL NOT NULL,
	PRIMARY KEY (WID)
);
```

### User

```
CREATE TABLE table_User (
	UID SERIAL,
	first_name VARCHAR NOT NULL,
	last_name VARCHAR NOT NULL,
	password VARCHAR NOT NULL,
	email VARCHAR UNIQUE NOT NULL,
	admin BOOL NOT NULL,
	wall_WID INT NOT NULL,
	FOREIGN KEY (wall_WID) REFERENCES table_Wall,
	PRIMARY KEY (UID)
);
```

### Location

```
CREATE TABLE table_Location (
	LID SERIAL,
	description VARCHAR NOT NULL,
	number INT NOT NULL,
	street VARCHAR NOT NULL,
	city VARCHAR NOT NULL,
	postalcode INT NOT NULL,
	country VARCHAR NOT NULL,
	user_UID INT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User,
	PRIMARY KEY (LID)
);
```

### Sensor and type for electricity, water etc

```
CREATE TYPE type_type AS ENUM ('water', 'electricity', 'gas', 'other')
```

```
CREATE TABLE table_Sensor (
	SID SERIAL,
	type type_type NOT NULL,
	title VARCHAR NOT NULL,
	EUR_per_unit DOUBLE PRECISION NOT NULL,
	user_UID INT NOT NULL,
	location_LID INT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User,
	FOREIGN KEY (location_LID) REFERENCES table_Location,
	PRIMARY KEY (SID)
);
```

### Value (General)

```
CREATE TABLE table_Value (
	value DOUBLE PRECISION NOT NULL,
	time INT NOT NULL,
	sensor_SID INT NOT NULL,
	FOREIGN KEY (sensor_SID) REFERENCES table_Sensor,
	PRIMARY KEY (sensor_SID, time)
);
```

### YearValue

```
CREATE TABLE table_YearValue (
	time INT NOT NULL,
	value DOUBLE PRECISION NOT NULL,
	sensor_SID INT NOT NULL,
	FOREIGN KEY (sensor_SID, sensor_SID) REFERENCES table_Sensor,
	PRIMARY KEY (sensor_SID, time)
);
```

### MonthValue

```
CREATE TABLE table_MonthValue (
	time INT NOT NULL,
	value DOUBLE PRECISION NOT NULL,
	sensor_SID INT NOT NULL,
	FOREIGN KEY (sensor_SID, sensor_SID) REFERENCES table_Sensor,
	PRIMARY KEY (sensor_SID, time)
);
```

### DayValue

```
CREATE TABLE table_DayValue (
	time INT NOT NULL,
	value DOUBLE PRECISION NOT NULL,
	sensor_SID INT NOT NULL,
	FOREIGN KEY (sensor_SID, sensor_SID) REFERENCES table_Sensor,
	PRIMARY KEY (sensor_SID, time)
);
```

### HourValue

```
CREATE TABLE table_HourValue (
	time INT NOT NULL,
	value DOUBLE PRECISION NOT NULL,
	sensor_SID INT NOT NULL,
	FOREIGN KEY (sensor_SID, sensor_SID) REFERENCES table_Sensor,
	PRIMARY KEY (sensor_SID, time)
);
```

### Tag

```
CREATE TABLE table_Tag (
	text VARCHAR NOT NULL,
	sensor_SID INT NOT NULL,
	FOREIGN KEY (sensor_SID) REFERENCES table_Sensor,
	PRIMARY KEY (sensor_SID, text)
);
```

### Status

```
CREATE TABLE table_Status (
	SID SERIAL,
	date INT NOT NULL,
	date_edited INT NOT NULL,
	text VARCHAR NOT NULL,
	author_UID INT NOT NULL,
	wall_WID INT NOT NULL,
	FOREIGN KEY (author_UID) REFERENCES table_User,
	FOREIGN KEY (wall_WID) REFERENCES table_Wall,
	PRIMARY KEY (SID)
);
```

### Like

```
CREATE TABLE table_Like (
	positive BOOL NOT NULL,
	status_SID INT NOT NULL,
	user_UID INT NOT NULL,
```

```
	FOREIGN KEY (status_SID) REFERENCES table_Status,
	FOREIGN KEY (user_UID) REFERENCES table_User,
	PRIMARY KEY (status_SID, user_UID)
);
```

### Friendship

```
CREATE TABLE table_Friendship (
	user1_UID INT NOT NULL,
	user2_UID INT NOT NULL,
	FOREIGN KEY (user1_UID) REFERENCES table_User,
	FOREIGN KEY (user2_UID) REFERENCES table_User,
	PRIMARY KEY (user1_UID, user2_UID)
);
```

### Group

```
CREATE TABLE table_Group (
	GID SERIAL,
	title VARCHAR NOT NULL,
	description VARCHAR NOT NULL,
	public BOOL NOT NULL,
	wall_WID INT NOT NULL,
	FOREIGN KEY (wall_WID) REFERENCES table_Wall,
	PRIMARY KEY (GID)
);
```

### Membership and type for admin, member, …

```
CREATE TYPE status_type AS ENUM ('ADMIN', 'MEMBER', 'PENDING', 'BANNED')
CREATE TABLE table_Membership (
	status status_type NOT NULL,
	last_change INT NOT NULL,
	user_UID INT NOT NULL,
	group_GID INT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User,
	FOREIGN KEY (group_GID) REFERENCES table_Group,
	PRIMARY KEY (user_UID, group_GID)
);
```

## Niet triviale SQL queries

### Tags

Query om tags terug te geven rekening houdende met het text attribuut (Living, keuken, enz) om autocompletion van tags in de frontend te voorzien.

```
SELECT * FROM table_Tag WHERE table_Tag.text IN (
	SELECT MIN(table_Tag.text) FROM table_tag GROUP BY table_Tag.text)
```