Rapport Project Databases

Inhoudsopgave

1. Status

Werkende features

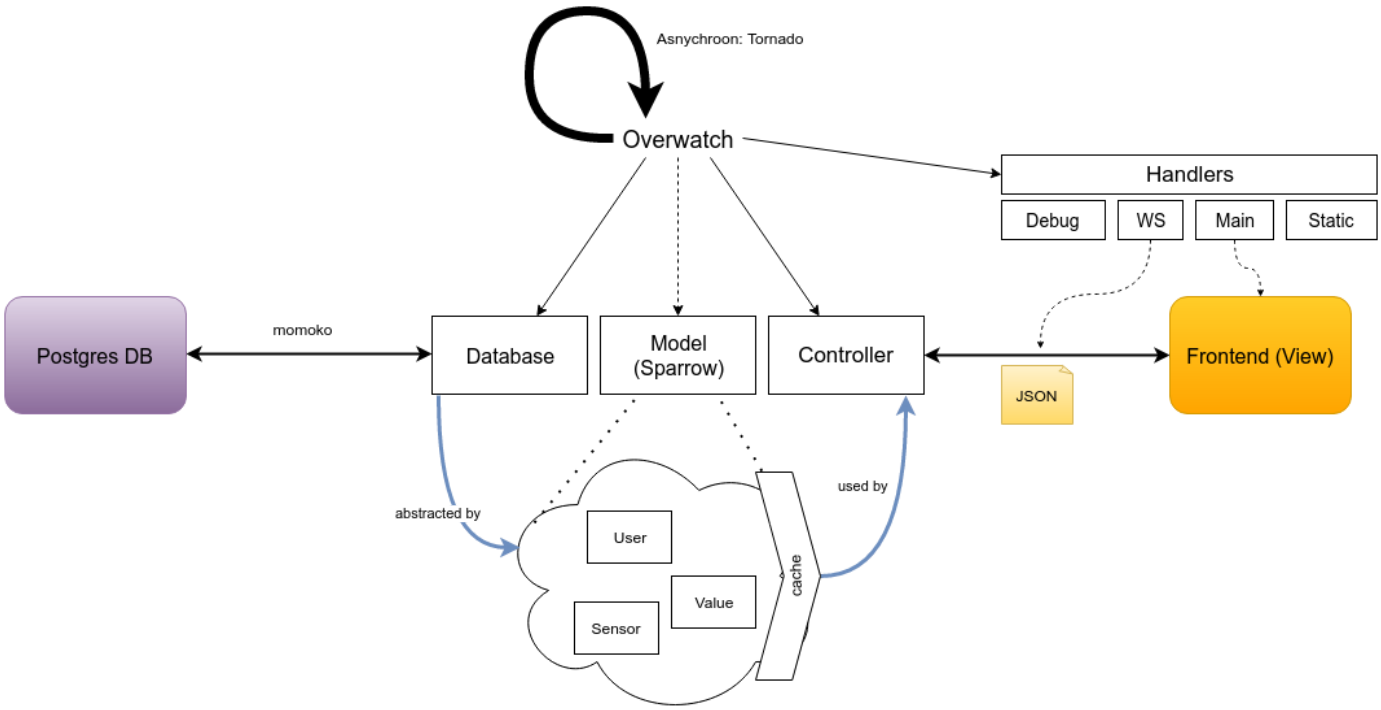
- Basisvereisten
- Extra functionaliteit:
    - Live updates (zowel voor grafieken als andere delen van de website)
    - Live data
    - Home pagina (bevat belangrijke live grafieken)
    - Fullscreen grafieken
    - Real-life sensoren
    - Help pagina

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

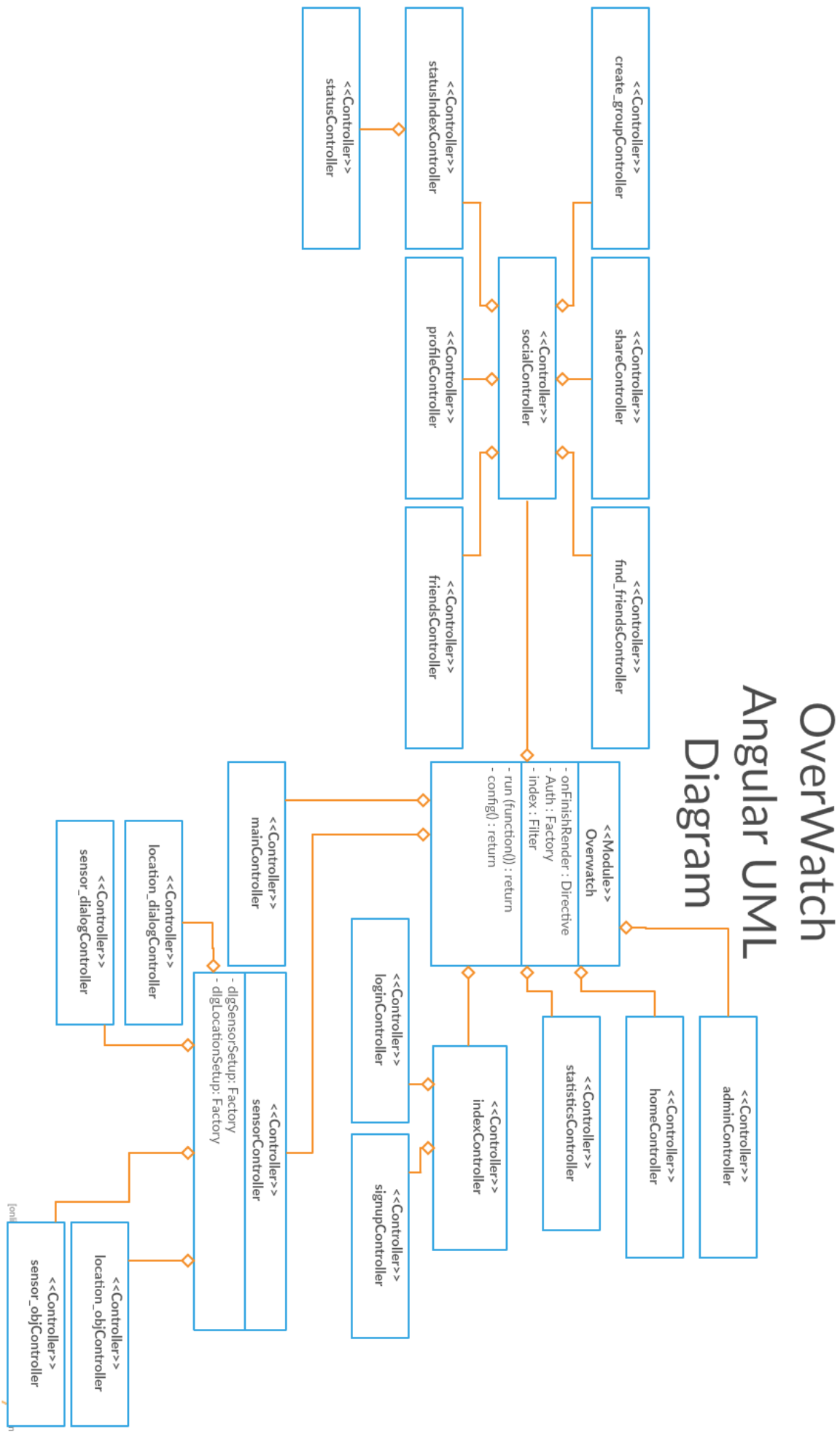
Backend



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. 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

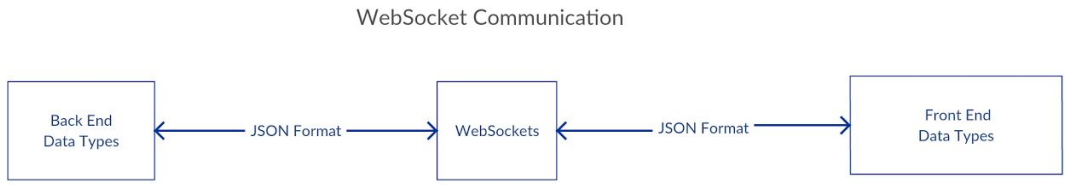
Qua onderliggend design is er niet veel veranderd. Hieronder het nieuwe UML diagram van de Angular code.

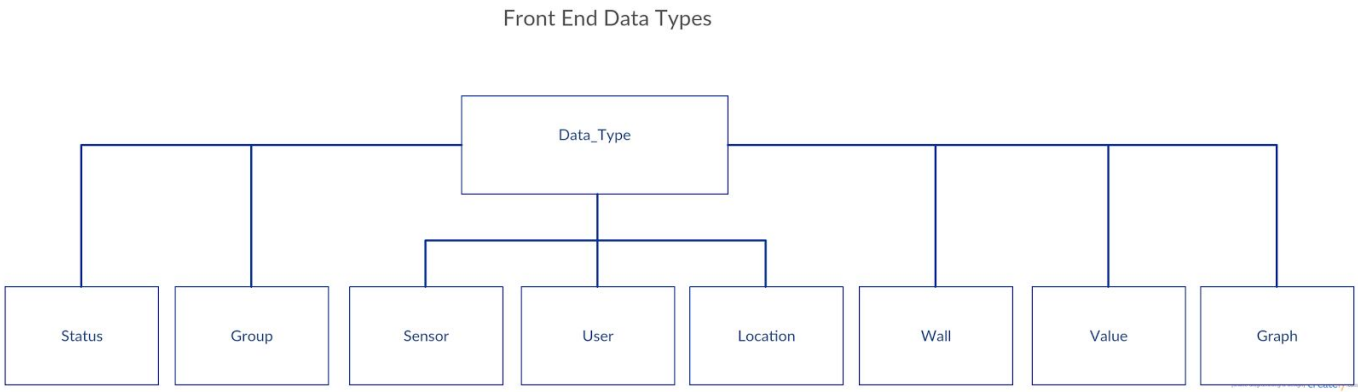

OverWatch
Angular UML
Diagram
<<Module>>
Overwatch
- onFinishRender : Directive
- Auth : Factory
- index : Filter
- run (function()) : return
- config() : return
<<Controller>>
adminController
<<Controller>>
homeController
<<Controller>>
statisticsController
<<Controller>>
indexController
<<Controller>>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

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



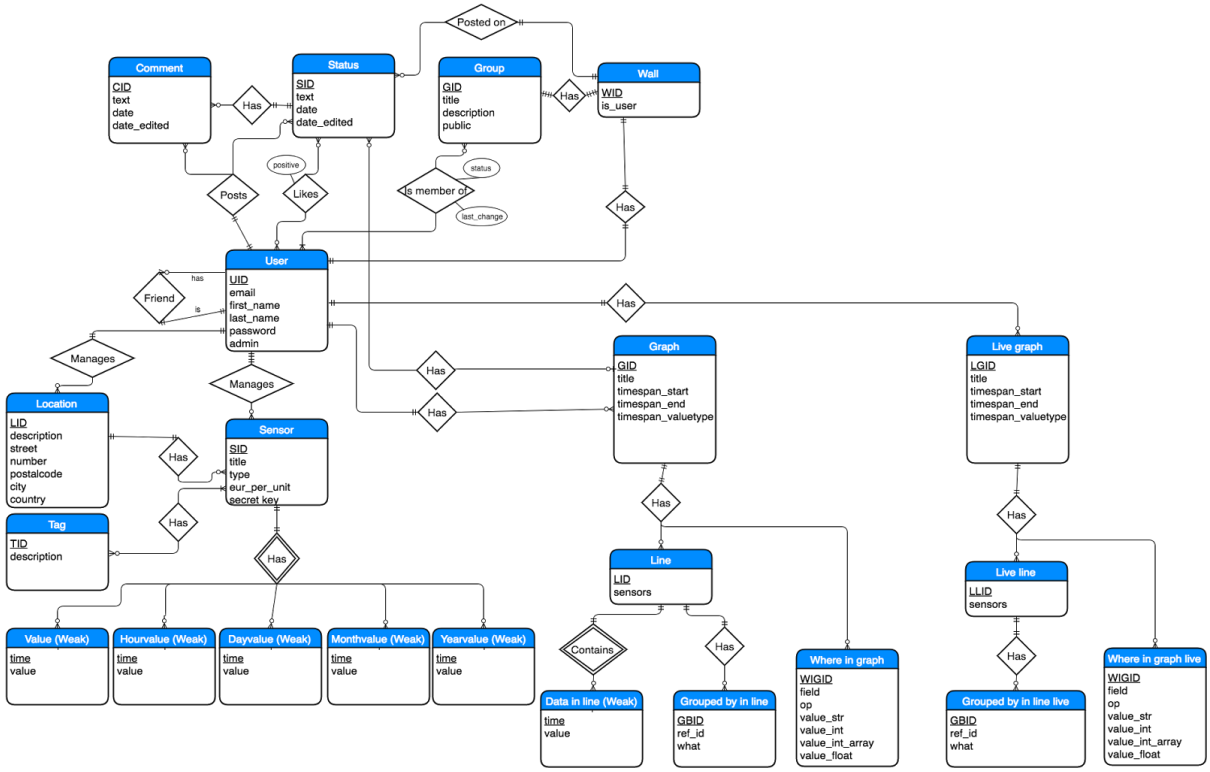


Database

Database schema

Het database schema is opgebouwd uit 4 delen:

1. Basisvereisten (user, sensor, location, tag, values, etc…)
2. Sociale tak (status, wall, comment, group)
3. Grafiek (graph, line, data in line, grouped by in line, where in graph)
4. Live grafiek (live graph, live line, grouped by in line live, where in graph live)



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:

User:

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 user en location (“Manages”):
    a. Elke user beheert 0 of meerdere locaties (0-many)

   b. Een locatie wordt beheerd door 1 user (referentiële integriteit)
3. 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
4. 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
5. Relatie user en wall (“Has”):
   a. Elke user heeft 1 wall (referentiële integriteit)
   b. Een wall behoort tot 1 user (referentiële integriteit)
6. 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
7. Relatie user en graph (“Has”):
   a. Een user kan 0 of meerdere graphs hebben (0-many)
   b. Een graph behoort tot 1 user (referentiële integriteit)
8. Relatie user en live graph (“Has”):
   a. Een user kan 0 of meerdere graphs hebben (0-many)
   b. Een graph behoort tot 1 user (referentiële integriteit)
9. 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)

Sensor:

10. 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
11. Relatie sensor en location (“Has”):
    a. Elke sensor heeft 1 locatie (referentiële integriteit)
    b. Een locatie kan 0 of meerdere sensoren bevatten (0-many)
12. Relatie sensor en tag (“Has”,weak relationship):
    a. Een sensor kan 0 of meer tags hebben (0-many)
    b. Een tag behoort tot 1 of meerdere sensoren (1-many)

Status:

13. 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)

14. Relatie status en comment (“Has”):
    a. Een status kan 0 of meerdere comments hebben (0-many)
    b. Een comment behoort tot 1 status (referentiële integriteit)
15. 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)

Group:

16. Relatie group en wall(“Has”):
    a. Een group heeft 1 wall (referentiële integriteit)
    b. Een wall behoort tot 1 group (referentiële integriteit)

Graphs - Live graphs:

17. Relatie graph en line (“Has”):
    a. Een grafiek kan 0 of meerdere lijnen hebben (0-many)
    b. Een lijn behoort tot 1 grafiek (referentiële integriteit)
18. Relatie graph en where in graph(“Has”):
    a. Een grafiek kan 0 of meerdere where in graphs hebben (0-many)
    b. Een where in graph behoort tot 1 grafiek (referentiële integriteit)
19. Relatie line en data in line (“Contains”, weak relationship):
    a. Een lijn kan 0 of meerdere data in lines bevatten (0-many)
    b. Data in line behoort tot maar 1 lijn (referentiële integriteit)
20. Relatie line en grouped by in line (“Has”):
    a. Een lijn kan 0 of meerdere data in lines bevatten (0-many)
    b. Grouped by in line behoort tot maar 1 lijn (referentiële integriteit)
21. Relatie live graph en live line (“Has”):
    a. Een live grafiek kan 0 of meerdere live lijnen hebben (0-many)
    b. Een live lijn behoort tot 1 live grafiek (referentiële integriteit)
22. Relatie live graph en where in graph live(“Has”):
    a. Een live grafiek kan 0 of meerdere where in graphs live hebben (0-many)
    b. Een where in graph behoort tot 1 grafiek (referentiële integriteit)
23. Relatie live line en grouped by in line live(“Has”):
    a. Een live lijn kan 0 of meerdere data in lines live bevatten (0-many)
    b. Grouped by in line live behoort tot maar 1 live lijn (referentiële integriteit)

3. Product

Basisvereisten

We hebben de applicatie zelf opgedeeld in 7 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. Om de gebruiker te ondersteunen hebben we ook een help-pagina voorzien.

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 wordt nu wel gebruikt voor het tonen van live grafieken. Dit zijn echter niet gewone live grafieken maar live grafieken die de gebruiker zelf als belangrijk gemarkeerd heeft.

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), comments plaatsen op statussen ...

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. Hierin kan men ook weer die grafieken sharen.

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.

7. Help pagina

Het doel van deze pagina spreekt voor zich. Het bevat een FAQ-sectie en daarnaast wordt de gebruiker ook geholpen aan de hand van nuttige tips en screenshots.

Extra features

1. Live grafieken

Naast het opslaan en delen van grafieken (die als snapshots werken van de huidige situatie), ondersteunen we ook het maken van grafieken die live geupdated worden. Dit werkt voor elk type, en voor elke mogelijke groepering van sensoren. Meer zelfs, de groepering van sensoren zelf wordt geupdated. Op deze manier kan bijvoorbeeld heel nauwkeurig de data van één sensor live bekijken, maar ook bijvoorbeeld de gemiddelde waardes per uur bekijken van de voorbije dag voor alle sensoren die bij een bepaalde locatie horen en een bepaald type hebben.

2. Ondersteuning voor 'echte' sensors

Aansluitend op de live grafieken, hebben we ook een manier om 'echte' sensoren aan onze website te hangen, met behulp van een python script en bijhorende API. Het python script is te vinden in 'varia/ow-client/client.py'. Dit script is bedoeld om op een Raspberry Pi of vergelijkbare computers te draaien. Het kan makkelijk worden uitgebreid om een meting te doen (standaard geeft deze nog random waardes).

3. Live updates van andere objecten

Naast het gebruik van live grafieken, trekken we deze live updates ook door naar andere objecten. Zo verschijnen bijvoorbeeld comments al direct op het scherm zonder dat de gebruiker de website moet vernieuwen.

4. Fullscreen grafieken

Voor de data in meer detail te bekijken, kan een gebruiker een grafiek vergroten. Dit werkt ook met live grafieken.

5. Help pagina

Het doel van deze extra feature spreekt voor zich. De gebruiker wordt geholpen aan de hand van nuttige tips en screenshots.

5. Appendix: SQL code

SQL statements van de entiteiten

Deze sectie bevat enkel de essentiële SQL statements zoals `CREATE TABLE, CREATE TYPE`. Voor de overige statements kan men ‘`python3 overwatch.py sql_info`’ uitvoeren in de terminal of in het bijgeleverd bestand ‘sql_statements.txt’ kijken.

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 BIGINT NOT NULL,
	FOREIGN KEY (wall_WID) REFERENCES table_Wall ON DELETE CASCADE,
	PRIMARY KEY (UID)
);
```

Location

```
CREATE TABLE table_Location (
	LID SERIAL,
	description VARCHAR NOT NULL,
	number BIGINT NOT NULL,
	street VARCHAR NOT NULL,
	city VARCHAR NOT NULL,
	postalcode BIGINT NOT NULL,
	country VARCHAR NOT NULL,
	user_UID BIGINT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User ON DELETE CASCADE,
	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,
	secret_key VARCHAR,
	user_UID BIGINT NOT NULL,
	location_LID BIGINT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User ON DELETE CASCADE,
	FOREIGN KEY (location_LID) REFERENCES table_Location ON DELETE CASCADE,
	PRIMARY KEY (SID)
);
```

Value (General)

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

YearValue

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

MonthValue

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

DayValue

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

HourValue

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

Tag

```
CREATE TABLE table_Tag (
	description VARCHAR NOT NULL,
	TID SERIAL,
	PRIMARY KEY (TID)
);
```

Tagged (relationship between sensor and tag)

```
CREATE TABLE table_Tagged (
	sensor_SID BIGINT NOT NULL,
	tag_TID BIGINT NOT NULL,
	FOREIGN KEY (sensor_SID) REFERENCES table_Sensor ON DELETE CASCADE,
	FOREIGN KEY (tag_TID) REFERENCES table_Tag ON DELETE CASCADE,
	PRIMARY KEY (sensor_SID, tag_TID)
);
```

Graph

```
CREATE TYPE timespan_valuetype_type AS ENUM ('Value', 'HourValue', 'DayValue',
'MonthValue', 'YearValue')

CREATE TABLE table_Graph (
	GID SERIAL,
	timespan_start BIGINT NOT NULL,
	timespan_end BIGINT NOT NULL,
	timespan_valuetype timespan_valuetype_type NOT NULL,
	title VARCHAR NOT NULL,
```

```
	user_UID BIGINT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User ON DELETE CASCADE,
	PRIMARY KEY (GID)
);
```

Where in graph

```
CREATE TYPE op_type AS ENUM ('eq', 'in', 'lt', 'ge', 'le', 'gt')

CREATE TABLE table_WhereInGraph (
	WIGID SERIAL,
	field VARCHAR NOT NULL,
	op op_type NOT NULL,
	value_str VARCHAR,
	value_int BIGINT,
	value_float DOUBLE PRECISION,
	value_int_array BIGINT[],
	graph_GID BIGINT NOT NULL,
	FOREIGN KEY (graph_GID) REFERENCES table_Graph ON DELETE CASCADE,
	PRIMARY KEY (WIGID)
);
```

Line

```
CREATE TABLE table_Line (
	LID SERIAL,
	sensors BIGINT[] NOT NULL,
	graph_GID BIGINT NOT NULL,
	FOREIGN KEY (graph_GID) REFERENCES table_Graph ON DELETE CASCADE,
	PRIMARY KEY (LID)
);
```

Data in line

```
CREATE TABLE table_DataInLine (
	time BIGINT NOT NULL,
	value DOUBLE PRECISION NOT NULL,
	line_LID BIGINT NOT NULL,
	FOREIGN KEY (line_LID) REFERENCES table_Line ON DELETE CASCADE,
	PRIMARY KEY (line_LID, time)
);
```

Grouped by in line

```
CREATE TABLE table_GroupedByInLine (
	GBID SERIAL,
	what VARCHAR NOT NULL,
	ref_ID BIGINT NOT NULL,
	line_LID BIGINT NOT NULL,
```

```
	FOREIGN KEY (line_LID) REFERENCES table_Line ON DELETE CASCADE,
	PRIMARY KEY (GBID)
);
```

Live graph

```
CREATE TABLE table_LiveGraph (
	LGID SERIAL,
	timespan_start BIGINT NOT NULL,
	timespan_end BIGINT NOT NULL,
	timespan_valuetype timespan_valuetype_type NOT NULL,
	title VARCHAR NOT NULL,
	user_UID BIGINT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User ON DELETE CASCADE,
	PRIMARY KEY (LGID)
);
```

Where in graph live

```
CREATE TABLE table_WhereInGraphLive (
	value_float DOUBLE PRECISION,
	value_str VARCHAR,
	WIGID SERIAL,
	field VARCHAR NOT NULL,
	op op_type NOT NULL,
	value_int BIGINT,
	value_int_array BIGINT[],
	graph_LGID BIGINT NOT NULL,
	FOREIGN KEY (graph_LGID) REFERENCES table_LiveGraph ON DELETE CASCADE,
	PRIMARY KEY (WIGID)
);
```

Live line

```
CREATE TABLE table_LiveLine (
	LLID SERIAL,
	sensors BIGINT[] NOT NULL,
	graph_LGID BIGINT NOT NULL,
	FOREIGN KEY (graph_LGID) REFERENCES table_LiveGraph ON DELETE CASCADE,
	PRIMARY KEY (LLID)
);
```

Grouped by in line live

```
CREATE TABLE table_GroupedByInLineLive (
	what VARCHAR NOT NULL,
```

```
	GBID SERIAL,
	ref_ID BIGINT NOT NULL,
	line_LLID BIGINT NOT NULL,
	FOREIGN KEY (line_LLID) REFERENCES table_LiveLine ON DELETE CASCADE,
	PRIMARY KEY (GBID)
);
```

Status

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

Like

```
CREATE TABLE table_Like (
	positive BOOL NOT NULL,
	status_SID BIGINT NOT NULL,
	user_UID BIGINT NOT NULL,
	FOREIGN KEY (status_SID) REFERENCES table_Status ON DELETE CASCADE,
	FOREIGN KEY (user_UID) REFERENCES table_User ON DELETE CASCADE,
	PRIMARY KEY (status_SID, user_UID)
);
```

Friendship

```
CREATE TABLE table_Friendship (
	user1_UID BIGINT NOT NULL,
	user2_UID BIGINT NOT NULL,
	FOREIGN KEY (user1_UID) REFERENCES table_User ON DELETE CASCADE,
	FOREIGN KEY (user2_UID) REFERENCES table_User ON DELETE CASCADE,
	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 BIGINT NOT NULL,
	FOREIGN KEY (wall_WID) REFERENCES table_Wall ON DELETE CASCADE,
	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 BIGINT NOT NULL,
	user_UID BIGINT NOT NULL,
	group_GID BIGINT NOT NULL,
	FOREIGN KEY (user_UID) REFERENCES table_User ON DELETE CASCADE,
	FOREIGN KEY (group_GID) REFERENCES table_Group ON DELETE CASCADE,
	PRIMARY KEY (user_UID, group_GID)
);
```

Comment

```
CREATE TABLE table_Comment (
	CID SERIAL,
	date BIGINT NOT NULL,
	date_edited BIGINT NOT NULL,
	text VARCHAR NOT NULL,
	status_SID BIGINT NOT NULL,
	author_UID BIGINT NOT NULL,
	FOREIGN KEY (status_SID) REFERENCES table_Status ON DELETE CASCADE,
	FOREIGN KEY (author_UID) REFERENCES table_User ON DELETE CASCADE,
	PRIMARY KEY (CID)
);
```

Niet triviale SQL queries

- Vind alle groepen van een bepaalde user

```
SELECT * FROM table_Group WHERE table_Group.gid IN (SELECT
table_Membership.group_gid FROM table_Membership WHERE table_Membership.user_uid
```

```
= {given UID})
```

- Selecteer gemiddelde value van bepaalde sensoren en in een bepaald tijdsinterval:

```
SELECT avg(value), time AS value FROM {g.cls._table_name} WHERE sensor_SID IN
{sensors} GROUP BY time HAVING time >= %(start)s AND time < %(end)s ORDER BY
time
```

- Vind alle tags van een bepaalde sensor

```
SELECT * FROM table_Tag WHERE table_Tag.tid IN (SELECT table_Tagged.tag_tid FROM
table_Tagged WHERE table_Tagged.sensor_sid = {given SID})
```