

howest
hogeschool

Socket.IO

Realtime voorbereiding RPI – doe het licht maar uit

Contents

LEERDOELEN	2
1 VOORAF	3
1.1 Packages installeren	3
1.2 Concept threading.....	4
1.2.1 Demo theading.....	4
2 OEFENING DOE HET LICHT MAAR UIT	7
2.1 Backend klaarmaken	8
2.2 Frontend.....	11
2.2.1 HTML structuur	11
2.2.2 CSS structuur	12
2.2.3 Selector	13
2.3 Eerste test frontend en backend.....	20
2.4 Een extra thread om de lichten iedere 10 seconden te doven	21
2.4.1 Backend	21
2.4.2 Frontend.....	22

Leerdoelen

Dit labo is een herhaling van:

- Messages met payload verzenden en opvangen via Socket.io aan de **client** en **webserver** zijde.
- Zelf de DataRepository schrijven voor de **backend**.

Daarnaast kan je op het einde van het labo

- Werken met een extra thread in de Flask **backend**.
- Een `querySelector()` schrijven die een element selecteert op basis van de waarde van zijn `data-` attribuut.

Dit labo is tevens de voorbereiding voor een workshop bij de start van Project 1. Tijdens die workshop in Project 1, zal je leren hoe je onze “website lamp” aan een “LED” koppelt van je Raspberry Pi en hoe je deze applicatie op je Raspberry Pi plaatst.

Je zal tijdens Project 1 zien dat dit slechts enkele extra lijnen Python code zijn.

Door deze aanpassing aan de leerstof van de module FSWD kunnen we ons 100% concentreren - tijdens de at home lessen - op de leerstof voor het examen.

Op deze manier krijgen jullie deze week een extra oefenkans op Javascript en Python, zonder dat de extra moeilijkheid wordt geïntroduceerd van de hardware aan te spreken op de Raspberry Pi

1 Vooraf

1.1 Packages installeren

De productie webserver **Eventlet**, geeft problemen met threads. Dit is een probleem want we hebben threads nodig in dit labo.

Daarom installeren we een andere productie webserver, namelijk **gevent** op onze venv. Deze server biedt ook een goede ondersteuning voor websockets indien we een extra package **gevent-websocket** installeren.

Open de folder FDWD-projecten (de folder met je oplossingen van week 4 tem 9 waar ook de virtual environment zich in bevindt).

Run een (oud) *.py bestand, zodat VS Code de venv gebruikt in zijn terminal.

```
(venv_fsfd) frederik.waeyaert@PCNAAM $
```

Ga enkel verder als je (venv_fsfd) in de terminal van VS Code ziet staan! Op deze manier installeert pip de packages op de juiste interpreter.

- **un**installeer de Eventlet package (de oude productie webserver)

```
(venv_fsfd) pip uninstall eventlet
```

- installeer de **twee** Gevent package voor de nieuwe productiewebserver

```
(venv_fsfd) pip install gevent
(venv_fsfd) pip install gevent-websocket
```

1.2 Concept threading

 Bekijk eerst het filmpje op Leho over threading

We zijn gewoon om code te schrijven die synchroon van boven naar beneden wordt uitgevoerd. De volgende lijn van je programmeercode kan slechts worden uitgevoerd als de bovenstaande lijn code is afgerond.

Dit kan voor een problemen zorgen.

We gaan straks een programma schrijven dat een (oneindige) lus heeft die iedere 10 seconden alle lichten in het huis uitzet. Deze lus staat in je Flask code boven het opstarten van je webserver.

Dit zou willen zeggen dat de webserver nooit opstart, want boven de `run()` staat een oneindige lus, die nooit zal aflopen en dus de webserver zou “blokkeren”

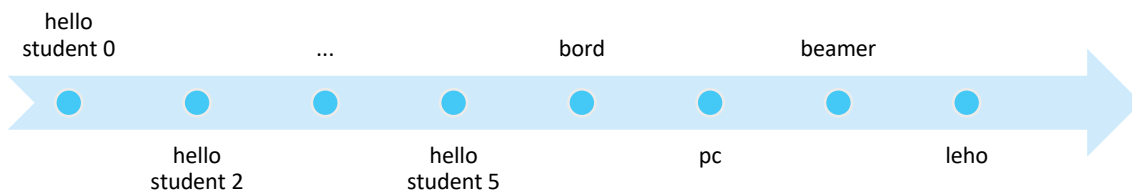
We gaan een techniek moeten vinden om deze processen “naast” elkaar, in plaats van “na” elkaar, uit te voeren.

1.2.1 Demo threading

- **Gewone synchrone programmeercode**

In volgend voorbeeldje komen studenten binnen in het lokaal en moet de docent aan elke student “hello” zeggen. Pas als de for-lus ten einde is, kan hij starten met het afvegen van het bord, de pc op te starten,...

We werken hier in 1 thread. De lijn `time.sleep(2)` zorgt dat er tussen elke verwelkoming 2 seconden wordt gepauzeerd.



```
import time

def zeg_hello():
    #zeg 6 maal hello student maar wacht 2 seconden tussen elke boodschap
    for i in range(6):
        print(f"Hello student {i}")
        time.sleep(2)

zeg_hello()
print('veeg het bord af')
print('start de pc op')
print('start de beamer op')
print('ga naar leho')
```

```
Hello student 0
Hello student 1
Hello student 2
Hello student 3
Hello student 4
Hello student 5
veeg het bord af
start de pc op
start de beamer op
ga naar leho
```

- **Asynchrone programmeercode – werken met meerdere threads (processen)**

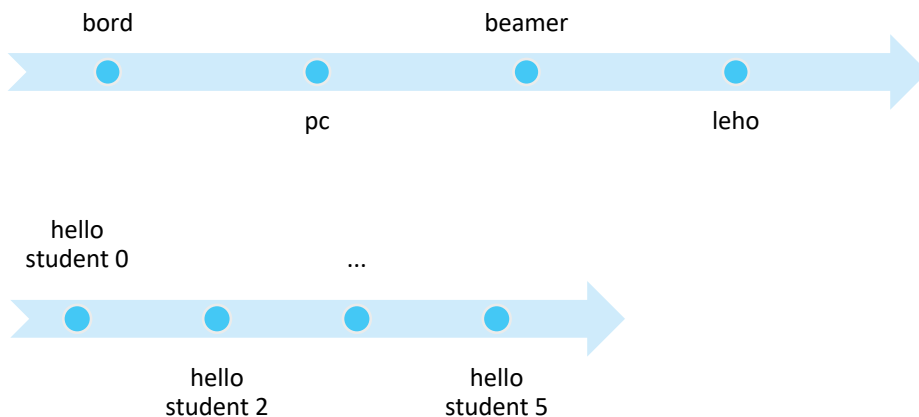
Het vorige codevoorbeeld is niet hoe het “echte” leven werkt. Gelukkig kunnen we als docent “multi-tasken”.

Als een groep studenten binnenkomt: Beginnen we als docent de studenten te verwelkomen.

En - terwijl we dit doen - beginnen we alvast het bord af te vegen, onze laptop op te starten, enz.

Het verwelkomen “blokkeert” - in dit voorbeeld - het voorbereiden van ons lokaal niet. We werken in 2 afzonderlijke threads. We voeren de twee processen “naast” elkaar uit in plaats van “na” elkaar.

Je roept de constructor van de klasse `Thread` op, uit de module `threading`. Als parameter geef je mee welke functie als extra thread moet worden opgestart.



```
import time
import threading

def zeg_hello():
    #zeg 6 maal hello maar wacht 2 seconden tussen elke boodschap
    for i in range(6):
        print(f"Hello student {i}")
        time.sleep(2)

mijn_ander_proces = threading.Thread(target=zeg_hello)
mijn_ander_proces.start()

print('veeg het bord af')
print('start de pc op')
print('start de beamer op')
print('ga naar leho')
```

```
Hello student 0
veeg het bord af
start de pc op
start de beamer op
ga naar leho
Hello student 1
Hello student 2
Hello student 3
Hello student 4
Hello student 5
```

- **Asynchrone programmeercode – de andere thread met vertraging laten starten.**

Soms wil je de andere thread met enkele seconden vertraging laten starten. Zo wil de docent alvast starten met zijn lokaal voor te bereiden én pas na 3 seconden te starten met de eerste student te verwelkomen.

Dit kan je bekomen door de constructor van de klasse `Timer` op te roepen, uit de module `threading`. Als parameters geef je het aantal seconden door dat moet gewacht worden en de naam van de functie die in de tweede thread moet worden opgestart.

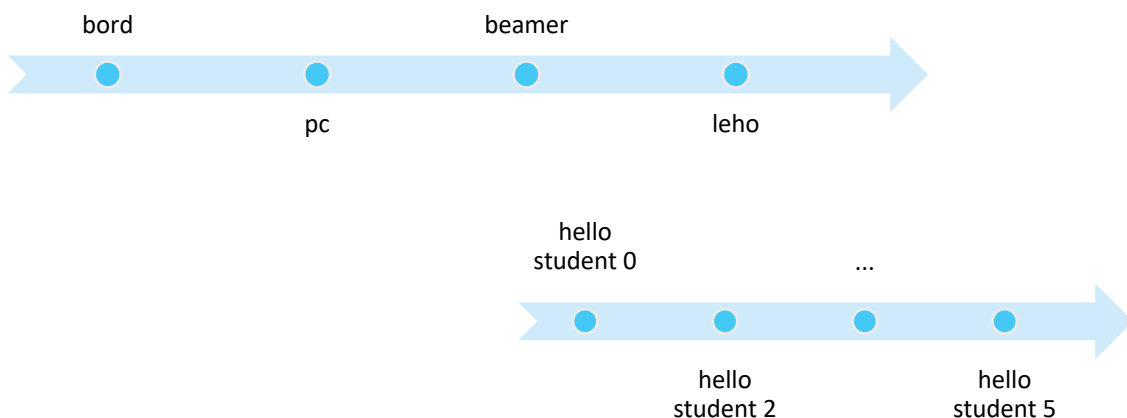
```
import time
import threading

def zeg_hello():
    #zeg 6 maal hello maar wacht 2 seconden tussen elke boodschap
    for i in range(6):
        print(f"Hello student {i}")
        time.sleep(2)

mijn_andere_proces = threading.Timer(3,zeg_hello)
mijn_andere_proces.start()

print('veeg het bord af')
time.sleep(1)
print('start de pc op')
time.sleep(1)
print('start de beamer op')
time.sleep(1)
print('ga naar leho')
time.sleep(1)
```

```
veeg het bord af
start de pc op
start de beamer op
Hello student 0
ga naar leho
Hello student 1
Hello student 2
Hello student 3
Hello student 4
Hello student 5
```



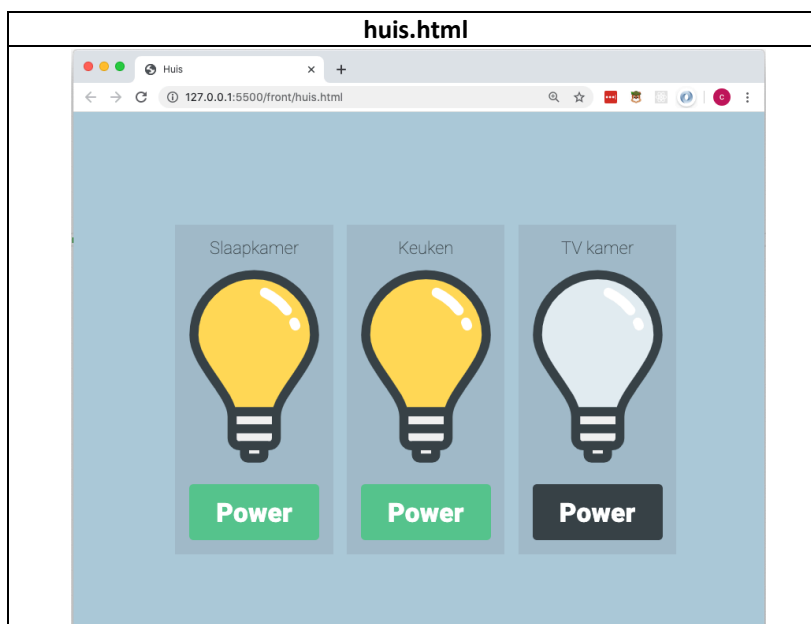
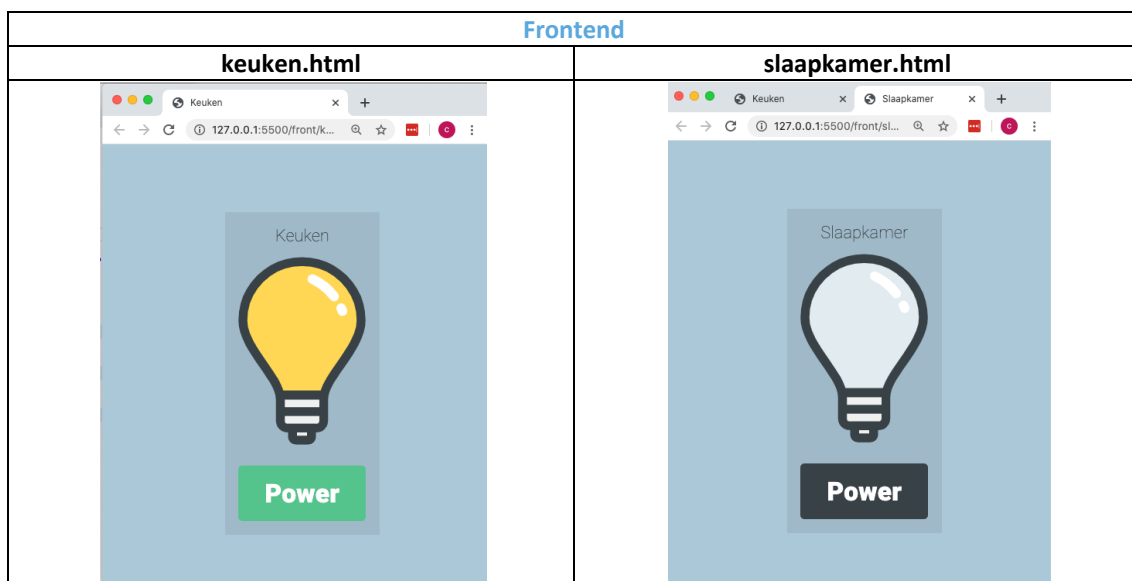
2 Oefening doe het licht maar uit

Je maakt een applicatie die lichten in je huis kan bedienen. Elk licht heeft zijn id en status in de database.

Elk licht heeft zijn eigen webpagina. Tenslotte is er een extra webpagina die alle lichten in je huis verzamelt.

Iedere 10 seconden loopt er een script (thread) dat alle lichten in het huis dooft.

id	omschrijving	status
1	slaapkamer	0
2	keuken	1
3	TV-kamer	0



2.1 Backend klaarmaken

data/homecontrol.sql

Importeer de database `homecontrol` via MySQL Workbench.

backend/repositories/DataRepository.py

Voorzie in de `DataRepository` volgende static functies met bijhorende SQL.

<code>read_status_lampen()</code>	Vraag alle info van alle lampen op.
<code>read_status_lamp_by_id(id)</code>	Vraag alle info op, van de lamp die hoort bij een bepaald id.
<code>update_status_lamp(id, status)</code>	Pas de status van de lamp aan, die hoort bij een bepaald id. De status die binnenkomt zal de waarde 0 of 1 zijn.
<code>update_status_all_lampen(status)</code>	Pas de status van alle lampen aan. De status die binnenkomt zal de waarde 0 of 1 zijn.

backend/repositories/app.py

Zoals je kan zien wordt opnieuw de Flask / Socket.io applicatie opgestart. Ons Flask backend programma zal geen API endpoints hebben maar zuiver werken op Socket.io.

```
from repositories.DataRepository import DataRepository
from flask import Flask, jsonify
from flask_socketio import SocketIO
from flask_cors import CORS

app = Flask(__name__)
app.config['SECRET_KEY'] = 'Hier mag je om het even wat schrijven, zolang het maar
geheim blijft en een string is'

socketio = SocketIO(app, cors_allowed_origins="*")
CORS(app)

# THREAD

# API ENDPOINTS
@app.route('/')
def hallo():
    return "Server is running, er zijn momenteel geen API endpoints beschikbaar."

# SOCKET IO

# START SERVER
if __name__ == '__main__':
    socketio.run(app, debug=True, host='0.0.0.0')
```

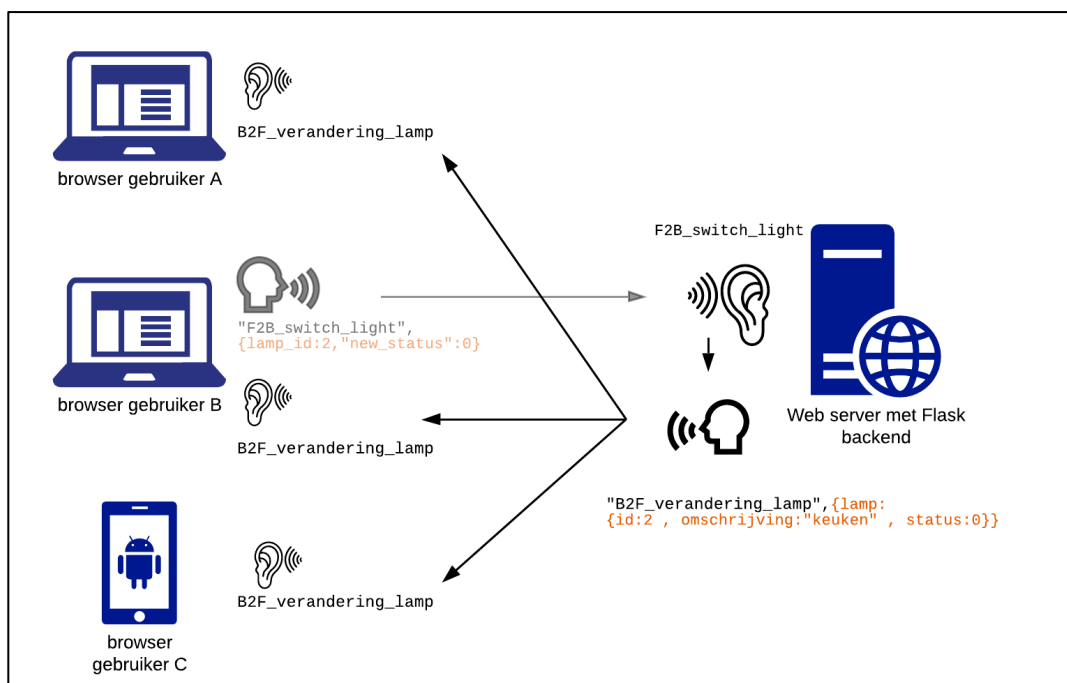
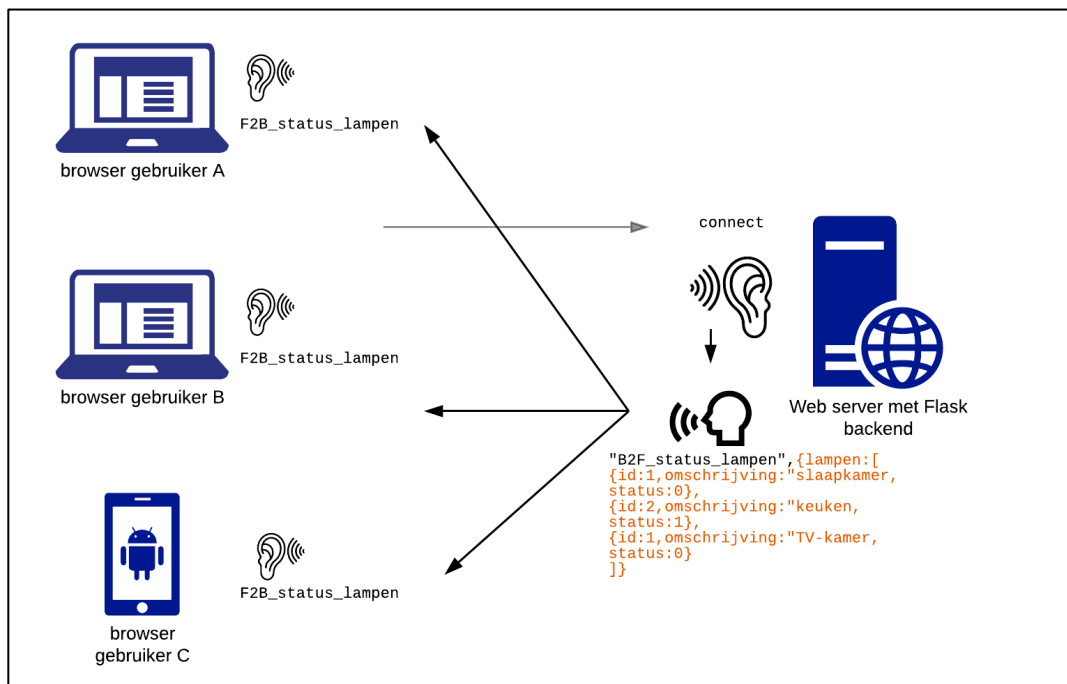
Vul de code verder aan zodat de webserver luistert naar 2 Sockets.io messages.

<code>.on('connect')</code>	<p>Als er een client connecteert met de server.</p> <ul style="list-style-type: none">• Print je het bericht “A new client connect” naar de terminal van VS Code.• Vraag je aan de database de huidige status van alle lampen op, bewaar je het resultaat van deze functie in de variabele <code>status</code><ul style="list-style-type: none">◦ Gebruik hiervoor de functie <code>read_status_lampen()</code> uit de <code>DataRepository</code>.• Je stuurt een message ‘B2F_status_lampen’ via socket.io naar de client.<ul style="list-style-type: none">◦ De payload zal er als volgt uitzien <code>{ 'lampen': status }</code>◦ Waarbij de variabele <code>status</code> een array/list van objecten is, want het resultaat van de functie <code>read_status_lampen()</code> is een list van objecten die records uit de database voorstellen.
<code>.on('F2B_switch_light')</code>	<p>Een client zal altijd een payload meegeven aan dit bericht. De payload ziet er als volgt uit <code>{ 'lamp_id'=3, 'new_status'=0 }</code></p> <ul style="list-style-type: none">• In het voorbeeld met payload <code>{ 'lamp_id'=3, 'new_status'=0 }</code> moet de lamp met id 3, als status 0 krijgen. De lamp zal dus worden uitgeschakeld.• Print een bericht naar de terminal van VS Code “Licht gaat aan/uit”• Pas de status van de lamp aan in de database.<ul style="list-style-type: none">◦ Gebruik hiervoor de functie <code>update_status_lamp(id,status)</code> uit de <code>DataRepository</code>.<ul style="list-style-type: none">▪ Deze functie verwacht een id en een status als parameter.• LATER: zal je hier de code schrijven om de hardware LED aan/uit te zetten.• Vraag aan de database de huidige status van de aangepaste lamp op, bewaar het resultaat van de functie in de variabele <code>status</code><ul style="list-style-type: none">◦ Gebruik hiervoor de functie <code>read_status_lamp_by_id(id)</code>• Je stuurt een message ‘B2F_verandering_lamp’ via socket.io naar de client.<ul style="list-style-type: none">◦ De payload zal er als volgt uitzien <code>{ 'lamp': status }</code>◦ Waarbij de variabele <code>status</code> een object is, die 1 record uit de database voorstelt.

Ter info: Waarom kiezen we ervoor om in het event `F2B_switch_light` én de lamp status aan te passen én de info van diezelfde lamp op te vragen uit de database?

De database is de Single Source of Truth in deze applicatie, we vertrouwen dus niet op het data-status attribuut van de webpagina om te zeggen wat de “echte” status is van de lamp. Want misschien is er ooit een socket.io message niet aangekomen of niet verzonden. En loopt de status van de webpagina “achter” ten opzichte van eigenlijke situatie.

Door na het aanpassen van de status alsnog eens de status op te vragen uit de database, ben je 100% zeker dat de webpagina straks weer de status heeft van de lamp.



2.2 Frontend

front/keuken.html

We gaan straks de html op deze pagina via Javascript aanspreken. Het is daarom belangrijk dat je deze pagina eerst goed verkent, en de “werking” beheerst zonder javascript.

2.2.1 HTML structuur

Verken via Chrome Dev Tools de structuur.

- Er is 1 `article` dat we kunnen aanspreken via zijn css klasse `js-room`. Dit `article` heeft een `data-idlamp` attribuut. Dit komt overeen met de id's (primary keys) in de database.
- Verder is er een `div` die een knop voorstelt. Deze is aan te spreken via de css klasse `js-power-btn`. Deze `div` heeft 2 data attributen.
 - Opnieuw `data-idlamp` die overeenkomt met het id uit de database.
 - `data-statuslamp` waarbij de waarde 0 betekent dat de lamp is uitgeschakeld en de waarde 1 betekent dat de lamp brandt.

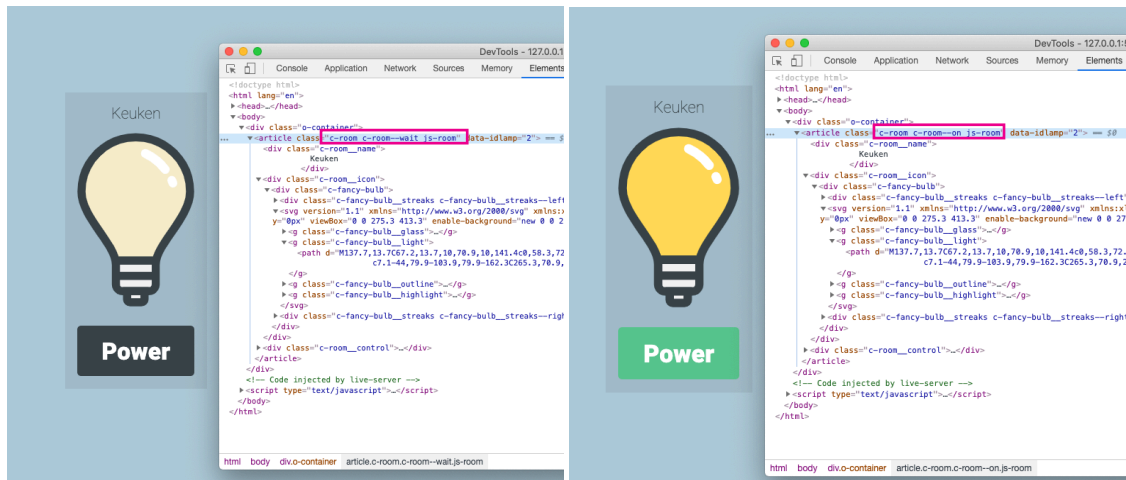
```
<article class="c-room js-room" data-idlamp="2"> == $0
  <div class="c-room__name">
    Keuken
  </div>
  <div class="c-room__icon">
    <div class="c-fancy-bulb">
      <div class="c-fancy-bulb__streaks c-fancy-bulb__streaks--left"></div>
      <svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/
xlink" x="0px" y="0px" viewBox="0 0 275.3 413.3" enable-background="new 0 0 275.3 413.3"
xml:space="preserve">
        <g class="c-fancy-bulb__glass"></g>
        <g class="c-fancy-bulb__light"></g>
        <g class="c-fancy-bulb__outline"></g>
        <g class="c-fancy-bulb__highlight"></g>
      </svg>
      <div class="c-fancy-bulb__streaks c-fancy-bulb__streaks--right"></div>
    </div>
  </div>
  <div class="c-room__control">
    <div class="c-power-btn js-power-btn" data-idlamp="2" data-statuslamp="0">
      Power
    </div>
  </div>
</article>
```

2.2.2 CSS structuur

Verder bekijken we even enkele css regels in screen.css

<pre>.c-fancy-bulb__light { transform: translate(50%, 50%) scale(0); opacity: 0; fill: #ffdb55; }</pre>	<p>De opvulling van de svg vector:</p> <ul style="list-style-type: none">• wordt met de helft verplaatst en verkleind naar 0% (dus verborgen)• zijn opacity wordt 0% (dus verborgen)• de kleur van de vector wordt donker geel. <p>Door deze settings lijkt het alsof de lamp niet brandt. (want de gele kleur wordt volledig verborgen omdat de opacity op 0% staat en de grootte 0% is)</p>
<pre>.c-room--wait .c-fancy-bulb__light { opacity: 1; transform: translate(0) scale(1); transition: all 0.14s ease-in; fill: rgb(248, 238, 203); }</pre>	<p>Als een element <code>.c-fancy-bulb__light</code> een child element is van een parent element met de klasse <code>.c-room--wait</code></p> <ul style="list-style-type: none">• Dan wordt de opacity 100%• Wordt het niet meer naar het midden verplaatst en wordt 100% van de grootte getoond.• Gebeurt de aanpassing binnen 0,14sec• Wordt de vector opgevuld met een heel licht geel. <p>Door deze settings lijkt het alsof het licht reeds héél licht brandt.</p>
<pre>.c-room--on .c-fancy-bulb__light { opacity: 1; transform: translate(0) scale(1); transition: all 0.14s ease-in; fill: #ffdb55</pre>	<p>Als een element <code>.c-fancy-bulb__light</code> een child element is van een parent element met de klasse <code>.c-room--on</code></p> <ul style="list-style-type: none">• Dan wordt de opacity 100%• Wordt het niet meer naar het midden verplaatst en wordt 100% van de grootte getoond.• Gebeurt de aanpassing binnen 0,14sec• Wordt de vector opgevuld met een donker geel. <p>Door deze settings lijkt het alsof het licht op volle sterkte brandt.</p>
<pre>.c-room--on .c-power-btn { background: #58c48d; }</pre>	<p>Als een element <code>.c-power-btn</code> een child element is van een parent element met de klasse <code>.c-room--on</code></p> <ul style="list-style-type: none">• Dan wordt de achtergrondkleur van de knop groen.

Probeer maar eens de BEM modifier (`c-room--wait` of `c-room--on`) extra in de classList bij een `article` te schrijven.

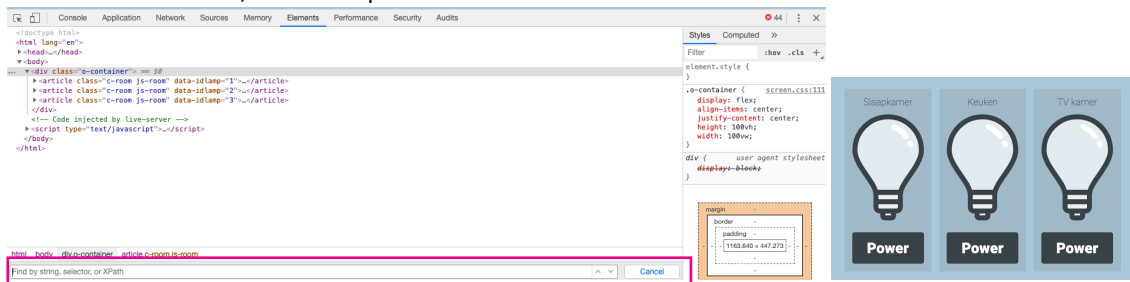


2.2.3 Selector

- **Selectors testen in Chrome Dev tools**

Wist je dat je een `querySelector` ook kan testen/uitvoeren in Chrome Dev Tools? Open `huis.html`. Je zal 3x hetzelfde `article` element herkennen. Elk met zijn eigen uniek `data-idlamp` attribuut, die iedere keer een andere lamp (room) in het huis voorstelt.

Open de tab **Elementen** van Chrome Dev Tools (klik effectief in dit venster zodat het actief komt). Geef de shortcut `ctrl+f` in, hierdoor opent er een zoekvenster onderaan de Dev Tools.



In dit zoekvenster kan je een selector ingeven.

Geef je bijvoorbeeld `.js-room` in, dan zie je dat er 3 elementen voldoen aan deze selector. Je kan dit herkennen doordat er 1 of 3 staat, via de pijltjes spring je naar het volgend element dat voldoet aan de selector.



- **Selector op basis van data-attribuut**

Straks gaan we één bepaalde room/lamp moeten aanspreken, als we 1 bepaalde lamp willen laten branden. Daarom leren we nu reeds een nieuwe selector die je kan gebruiken in de `querySelector()`.

Je kan opgeven dat je een element wil selecteren dat **voldoet aan een CSS klasse** én die een bepaalde **data-attribuut** waarde heeft.

Voorbeeld


Stel dat je het element `.js-room` wil selecteren waarvan het `data-idlamp` attribuut ingesteld staat op de waarde 2.

Als we bovenstaande zin vertalen naar de juiste selector krijgen we:

```
.js-room[data-idlamp="2"]
```

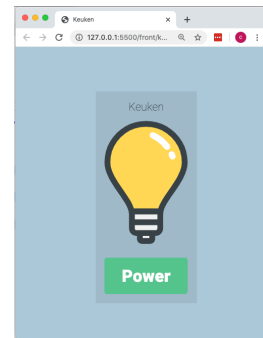
Probeer dit maar even in te geven in de zoekbalk van Chrome Dev Tools. Zoals verwacht voldoet er 1 element aan deze voorwaarde. Vergeet zeker de aanhalingsteken niet rond de waarde. (ook al gaat het over een integer)



 Wil je meer weten over deze selector? Lees dan zeker eens: https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors

- Analyse van de probleemstelling

- Als er een verbinding is met de Socket.io webserver, wordt de huidige status van de lamp getoond.
 - => luisteren naar de boodschap van een geslaagde connectie (B2F_status_lampen) met Socket.io en gebruik de payload (id en status) om de UI aan te passen.
- Als er op de knop geklikt wordt
 - => klik event van de knop opvangen.
 - => het data-idlamp attribuut opvragen van de knop, zodat je weet over welke lamp het gaat.
 - => eventuele css klassen .c-room--wait en .c-room--on verwijderen uit de classList van het element .js-room (dat een data-idlamp attribuut heeft gelijk aan het id dat we zonet ophaalden uit de knop). Op deze manier werkt straks onze code ook als er meerdere lampen/rooms op onze webpagina staan.
 - => aan de classList van het element .js-room (dat een data-idlamp attribuut heeft gelijk aan het id dat we zonet ophaalden uit de knop), de .c-room--wait css klasse toevoegen
 - Op deze manier gaat de lamp een beetje branden en krijgt de website bezoeker feedback dat de aanvraag geregistreerd is. (UX)
 - => aan de webserver doorgeven dat de status van de lamp moet veranderen.
 - Gebruik het data-idlamp en data-statuslamp attribuut van de knop. Natuurlijk “draaien” we de huidige status om zodat we de nieuwe status van de lamp kunnen bepalen, een 0 wordt een 1 én een 1 een 0 als status.
- Wachten op een message van de socket.io webserver met de nieuwe status van de lamp. Deze heeft een payload die er als volgt uitziet {lamp: {id:2 , omschrijving:"keuken" , status:0}}
- Eventuele css klassen .c-room--wait en .c-room--on verwijderen uit de classList van het element .js-room (dat een data-idlamp attribuut heeft dat gelijk is aan het id dat is binnengekomen als payload). Op deze manier werkt straks onze code ook als er meerdere lampen/rooms op onze webpagina staan.
- Als de status in de payload 1 is:
 - Aan de classList van het element .js-room (dat een data-idlamp attribuut heeft dat gelijk is aan het id dat is binnengekomen in de payload), de .c-room--on css klasse toevoegen. De lamp zal nu op volle sterkte beginnen branden.
- Als de status in de payload 0 is:
 - Doen we niets extra meer. De lamp heeft geen extra CSS klassen en zal dus niet branden. (--wait en --on css modifier klassen waren in de vorige stap verwijderd)



DOMContentLoaded

ListenToUI ()
ListenToSocket()

Achtergrond info

Zoals je kan lezen in de analyse gaan we steeds onze knop of onze room heel specifiek gaan aanduiden via het data-lampid attribuut én gaan we er steeds van uitgaan dat er meerdere “lampen”/“rooms” op onze webpagina staan, ook al is dat niet altijd het geval.

Op deze manier moeten we de javascript maar 1x schrijven en zal het werken voor de keuken pagina, slaapkamer pagina, en de pagina voor het huis met meerdere lampen.

Een for-lus over het resultaat van een `querySelectorAll()` met 1 item, werkt ook perfect. In deze situatie halen we hier ons voordeel uit.

- **Opbouw app.js:**

In de JS-code staan er al enkele variabelen en een “helper” functie die bepaalde css klassen verwijdert uit de classList van het element dat binnenkomt als parameter.

Maak verbinding met de socket.io webserver via de functie `io()`, deze `io()` functie is hier gekend omdat het HTML-bestand is gelinkt met de javascript library van Socket.io.

```
const lanIP = `${window.location.hostname}:5000`;
const socket = io(`http://${lanIP}`);

const clearClassList = function (el) {
  el.classList.remove('c-room--wait');
  el.classList.remove('c-room--on');
};

const listenToUI = function () {
};

const listenToSocket = function () {
};

document.addEventListener('DOMContentLoaded', function () {
  console.info('DOM geladen');
  listenToUI();
  listenToSocket();
});
```

- **Opbouw app.js: Als de DOM is geladen**

Als de DOM geladen is

- Roep de 2 functies op:

```
listenToUI();
listenToSocket();
```

In de functie `listenToSocket()` luister je naar 3 socket.io events.

- Luister naar de automatisch verzonden message “connected”
 - Schrijf de string “verbonden met socket webserver” in de console van Chrome.
- Luister naar de message “B2F_status_lampen” die 1 parameter binnenkrijgt in json formaat. De parameter van deze functie geef je de naam **jsonObject**

```
{lampen : [ {id:1,omschrijving:"slaapkamer, status:0"},  
            {id:2,omschrijving:"keuken, status:1"},  
            {id:1,omschrijving:"TV-kamer, status:0"}  
          ]  
}
```

 - Schrijf de **string** “Dit is de status van de lampen” naar de console van Chrome.
 - Schrijf de inhoud van de binnengekomen parameter `jsonObject` weg naar de console van Chrome.
 - Je ziet dat **de binnengekomen parameter** een object met een array is. Je zal dus een for-lus nodig hebben.
 - Overloop alle binnengekomen lampen in de variabele `jsonObject`.
 - Haal per lamp het id op.
 - Zoek per lamp naar een html element in het document met als css klasse `.js-room` en als `data-lampid` attribuut het id dat je zonet ophaalde.
 - Controleer of het html element effectief bestaat in de DOM (je krijgt een array binnen met 3 lampen (en id’s), maar er staat maar 1 lamp in de DOM.
 - Vraag vervolgens de knop `.js-power-btn` op (die zich bevindt in het DOM element `room`). We gebruiken dus `room.querySelector()` ipv `document.querySelector()`
 - Verander het `data-lamp-status` attribuut van deze knop.
 - Verwijder `.c-room--wait` en `.c-room--on` css klassen van het `room` DOM element.
 - Als de binnengekomen status 1 was voeg je de css klasse `c-room--on` toe aan het `room` DOM element. Als de binnengekomen status 0 was, doe je niets extra.

```
socket.on('B2F_status_lampen', function (jsonObject) {  
  console.log('Dit is de status van de lampen');  
  console.log(jsonObject);  
  for (const lamp of jsonObject.lampen) {  
    const room = document.querySelector(`.js-room[data-idlamp="${lamp.id}"]`);  
    if (room) {  
      const knop = room.querySelector('.js-power-btn');  
      # Vul de code hier verder aan  
    }  
  }  
});
```

- Luister naar de message “B2F_verandering_lamp” die 1 parameter binnenkrijgt in het json formaat. De parameter van de functie geef je de naam `jsonObject`
`{lamp : {id:2,omschrijving:"keuken, status:1} }`
 - Schrijf de **string** “Dit is de status van de lampen” naar de console van Chrome.
 - Schrijf de inhoud van de binnengekomen parameter `jsonObject` weg naar de console van Chrome.
 - Je ziet dat **de binnengekomen parameter** een object met een object is. Je zal dus geen for-lus nodig hebben.
 - Zoek naar een html element in het document met als css klasse `.js-room` en als `data-lampid` attribuut het id dat binnenkomt als attribuut.
 - Controleer of het html element effectief bestaat in de DOM
 - *Via Socket io word je verwittigd van elke wijziging van elke lamp (slaapkamer, keuken, tv-room,...). Als de lamp van de TV-room wordt aangepast. Zal de QuerySelector het id niet terug vinden in keuken.html, je wilt geen error krijgen.*
 - Vraag vervolgens de knop `.js-power-btn` op (die zich bevindt in het DOM element `room`). We gebruiken dus `room.querySelector()` ipv `document.querySelector()`
 - Verander het `data-lamp-status` attribuut van deze knop.
 - Verwijder `.c-room--wait` en `.c-room--on` css klassen van het `room` DOM element.
 - Als de binnengekomen status 1 was voeg je de css klasse `c-room--on` toe aan het `room` DOM element. Als de binnengekomen status 0 was, doe je niets extra.

In de functie `listenToUI()`

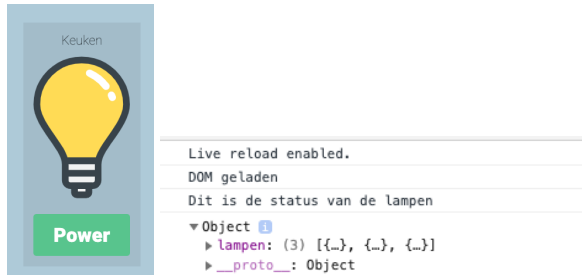
- Koppel aan alle elementen `.js-btn-on` een click event.
 We gebruiken nu reeds een `querySelectorAll()` en for-lus. Op deze manier is de code ook bruikbaar als er straks meerdere knoppen op het scherm staan
 - Haal het `data-idlamp` en `data-statuslamp` attribuut op van de aangeklikte knop.
 - Maak een extra lokale variabele `nieuweStatus`
 - Deze variabele is 0 als `data-statuslamp` 1 was
 - Deze variabele is 1 als `data-statuslamp` 0 was
 - Verwijder de `.c-room--wait` en `.c-room--on` css klassen van `.js-room` (met dezelfde `data-lampid` attribuut als de knop die is aangeklikt)
 - Voeg de `.c-room--wait` css klassen toe aan `.js-room` (met dezelfde `data-lampid` attribuut als de knop die is aangeklikt)
 - Verstuur een socket.io message `F2B_switch_light` naar de backend. Als payload geef je volgend jsonobject mee.
`{lamp_id: 2 , "new_status" : 0}`
 Waarbij de waarde van `lamp_id`, het `data-idlamp` attribuut van de knop was, en de waarde van `new_status` de variabele `nieuweStatus` is.

2.3 Eerste test frontend en backend

Start de backend op via VS Code > app.py > run

```
The default interactive shell is now zsh.  
To update your account to use zsh, please run 'chsh -s /bin/zsh'.  
For more details, please visit https://support.apple.com/4b1828858.  
Christophe-Howests-MacBook-Pro:docent2020 christophe$ source "/Users/christophe/Hogeschool West-Vlaanderen/OHK.MCT - 2019-2020/IMCT/Full Stack Web Development/Labo/10 RP1 gebruiken/docent2020/venv_fswd/bin/activate"  
(venv_fswd) Christophe-Howests-MacBook-Pro:docent2020 christophe$ "/Users/christophe/Hogeschool West-Vlaanderen/OHK.MCT - 2019-2020/IMCT/Full Stack Web Development/Labo/10 RP1 gebruiken/docent2020/venv_fswd/bin/python3" "/Users/christophe/Hogeschool West-Vlaanderen/OHK.MCT - 2019-2020/IMCT/Full Stack Web Development/Labo/10 RP1 gebruiken/docent2020/back/app.py"  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 113-353-078
```

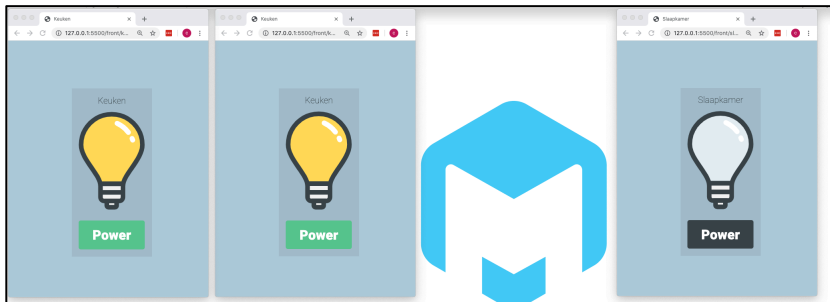
Start de frontend op via VS Code > keuken.html > Open with Live Server, doe dit 2x



Als je alles goed programmeerde, kan je de lamp aan en uit zetten en reageert de andere pagina direct mee.

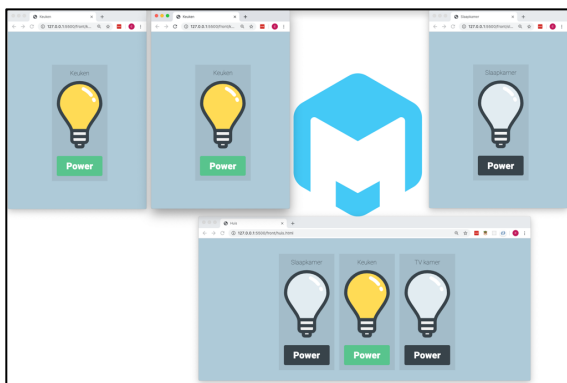
Start de frontend op via VS Code > slaapkamer.html > Open with Live Server

Je merkt op dat de slaapkamer pagina, volledig autonoom werkt van de keuken webpagina. Hoe zou dit komen?



Start de frontend op via VS Code > huis.html > Open with Live Server

Je zal opmerken dat ook deze pagina onmiddellijk goed werkt, doordat we onmiddellijk de goede programmeercode schreven. Kan je in app.js nog aanduiden welke lijnen ervoor zorgen dat het tonen van de status zowel lukt voor 1 lamp als voor 3 lampen?



Je zal al hebben opgemerkt dat de `.js-room` bijna nooit in de `.c-room--wait` status komt, dit komt omdat de socket.io messages zo snel heen en weer worden verstuurd dat er binnen de milliseconde reeds een antwoord terug is gekomen en de lamp al aan of uit staat. Eventueel kan je de **backend** eens uitzetten om dit te testen.

2.4 Een extra thread om de lichten iedere 10 seconden te doven

2.4.1 Backend

back/app.py

Voeg een functie `all_out()` toe aan de backend.

- Deze zet de status van alle lampen in de database op 0 (dus uit).
- Vraagt de status van alle lampen op.
- Verstuurt een socket.io message `B2F_alles_uit`.
- Verstuurt de huidige status van de lampen via de socket.io message `B2F_status_lampen`.
- Nadien wacht hij 10 seconden.
- Omdat dit binnen een oneindige lus (`while True`) staat, zal hij dit iedere 10 seconden blijven herhalen.

De functie starten we de eerste keer op na 10 seconden via `threading.Timer(10, all_out).start()` in een tweede Thread.

```
import time
import threading
```

```
# THREAD
# START een thread op. Belangrijk!!! Debugging moet UIT staan op start van de
server, anders start de thread dubbel op
# werk enkel met de packages gevent en gevent-websocket. uninstall eerst eventlet en
installeer gevent en gevent-socket
def all_out():
    while True:
        print('*** We zetten alles uit **')
        DataRepository.update_status_alle_lampen(0)
        status = DataRepository.read_status_lampen()
        socketio.emit('B2F_alles_uit', {
            'status': "lampen uit"}, broadcast=True)
        socketio.emit('B2F_status_lampen', {'lampen': status}, broadcast=True)
        time.sleep(10)

threading.Timer(10, all_out).start()
```

Threading werkt niet 100% goed in de debug modus van de webserver. Stel daarom de debugging van de server uit door de parameter op `False` te zetten.

```
if __name__ == '__main__':
    socketio.run(app, debug=False, host='0.0.0.0')
```

Als de webserver niet in debug modus staat, moet je na elke wijziging de server afsluiten en opnieuw opstarten.

Stop de server nu manueel, en herstart je server, je zal nu zien dat elke 10 seconden de lichten doven.

2.4.2 Frontend

Dit werkt onmiddellijk omdat onze frontend reeds luisterde naar de message `B2F_status_lampen`. We hebben wel ontdekt dat er soms wat vertraging zit op een emit die wordt verstuurd vanuit een Thread. Soms moet je even wachten tot `B2F_status_lampen` “aankomt” in je frontend.

```
The default interactive shell is now zsh.  
To update your account to use zsh, please run 'chsh -s /bin/zsh'.  
For more details, please visit https://support.apple.com/kb/HT208050.  
Christophe-Howests-MacBook-Pro:docent2020 christophe$ source "/Users/christophe/Hogeschool West-Vlaanderen  
9-2020/IMCT/Full Stack Web Development/Labo/10 RPi gebruiken/docent2020/venv_fswd/bin/activate"  
(venv_fswd) Christophe-Howests-MacBook-Pro:docent2020 christophe$ "/Users/christophe/Hogeschool West-Vlaan  
- 2019-2020/IMCT/Full Stack Web Development/Labo/10 RPi gebruiken/docent2020/venv_fswd/bin/python3" "/User  
ogeschool West-Vlaanderen/OHK.MCT - 2019-2020/IMCT/Full Stack Web Development/Labo/10 RPi gebruiken/docent  
py"  
A new client connect  
A new client connect  
A new client connect  
A new client connect  
A new client connect  
*** We zetten alles uit **  
licht gaat aan/uit  
1  
licht gaat aan/uit  
1  
*** We zetten alles uit **
```

front/script/app.js

Luister naar de extra message “`B2F_alles_uit`” in de frontend.

Schrijf het bericht “alle lampen zijn automatisch uitgezet” weg naar de console van Chrome.

```
DOM geladen  
Dit is de status van de lampen  
▶ {lampen: Array(3)}  
Er is een status van een lamp veranderd  
1  
1  
Er is een status van een lamp veranderd  
2  
1  
alle lampen zijn automatisch uitgezet  
Dit is de status van de lampen  
▶ {lampen: Array(3)}
```

howest
hogeschool