

howest
hogeschool

Socket.IO

Realtime - drink

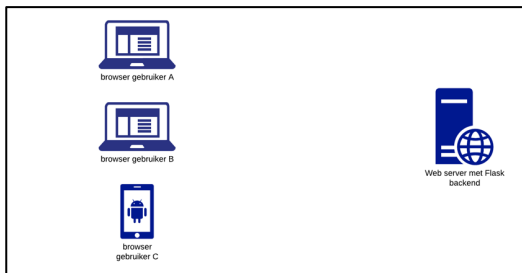
1 Contents

2	LEERDOELEN	2
3	VOORAF	3
3.1	Packages installeren	3
3.2	Concept socket.io	3
3.2.1	Verzenden zonder extra data (send)	3
3.2.2	Verzenden met extra data (emit)	4
3.2.3	Speciaal event	4
3.2.4	Naamgeving	4
4	OEFENING “DRINKS”	5
4.1	Backend klaarmaken	6
4.2	Frontend	7
4.3	Eerste test frontend en backend	10
4.3.1	Socket.io events op de backend en frontend	10
4.3.2	UI Events op de frontend	11
4.4	B2F_connected verder uitwerken (socket event)	11
4.5	Click event verder uitwerken (UI event)	13
4.6	Overview.html	15

2 Leerdoelen



In dit labo gaan we met een realtime verbinding werken. Realtime verbindingen zijn een soort van kleine events die we aan de hand van hun naam kunnen versturen (`send` of `emit`) en ontvangen (`on`). Zowel van de **frontend** naar de **backend** als vice versa.

Voor de eerste keer gaan we in de **frontend** verschillende gebruikers (clients) simuleren die onze website bezoeken. Deze gebruikers moeten een vloeiende UX krijgen als er door 1 client een aanpassing wordt doorgegeven aan de webserver.



Om dit werkend te krijgen, voorzien we zowel in de **frontend** als aan de **backend** extra code, die we krijgen van SocketIO.

Wat we gaan behandelen:

- Werken met socket.io.
 -  [Realtime backend server](#) maken.
 -  [Realtime frontend](#) maken.
- Een “productie” webserver (eventlet) toevoegen aan de packages van Python, zodat deze wordt gebruikt in de plaats van de “development” webserver (Werkzeug) bij het opstarten van een socket.io project.
- Flask beschikbaar maken in een LAN-omgeving.
- Herhaling werken met Flask:
 - MySQL verbinding.
 - Routes.
- Herhaling werken met JavaScript:
 - DOM aanspreken.
 - Events toevoegen en verwerken.
 - Externe library gebruiken.
- Herhaling werken met MySQL:
 - COUNT() gebruiken.
 - INSERT gebruiken.
 - LIKE op een bepaalde datum.
 - Nieuwe queries schrijven.

3 Vooraf

3.1 Packages installeren

Open de folder FDWD-projecten (de folder met je oplossingen van week 4 tem 8 waar ook de virtual environment zich in bevindt).

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

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

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

- Installeer de Eventlet package (de productie webserver)

```
(venv_fswd) pip install eventlet
```

- Installeer de socket.io package

```
(venv_fswd) pip install flask-socketio
```

3.2 Concept socket.io

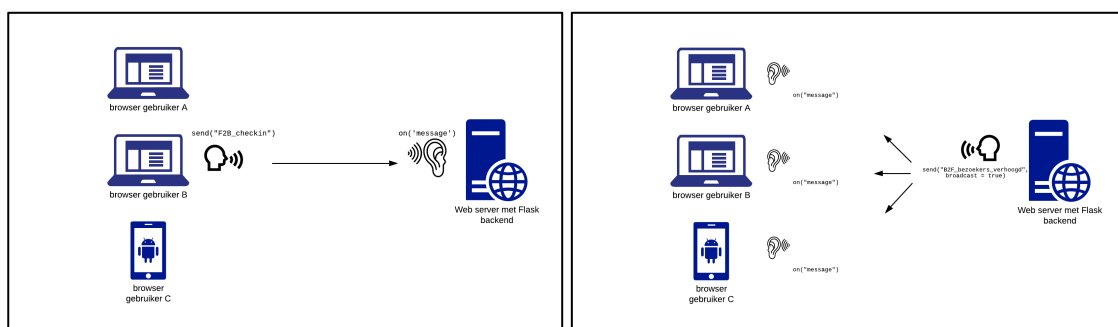
Socket.io draait volledig door boodschappen te versturen en te ontvangen van de **client** (browser) naar de **server** en van de **server** naar de **client(s)** browser(s).

Je kan ervoor kiezen om enkel een bericht (terug) te sturen naar **client** waarvan je het bericht ontving of een bericht te verzenden naar alle **clients** (broadcast) die verbonden zijn.

3.2.1 Verzenden zonder extra data (send)

- Een boodschap **zonder** payload: **verzend je** via `send("omschrijving")`.
- Het **ontvangen** van zo een boodschappen doe je via `on('message')`.

Als voorbeeld nemen we een web applicatie waarbij de gebruiker kan zeggen dat hij is aangekomen. Er wordt een boodschap naar de webserver verstuurd van 1 client dat hij is ingecheckt. De webserver antwoordt door naar alle clients een boodschap te sturen dat het bezoekersaantal is verhoogd.

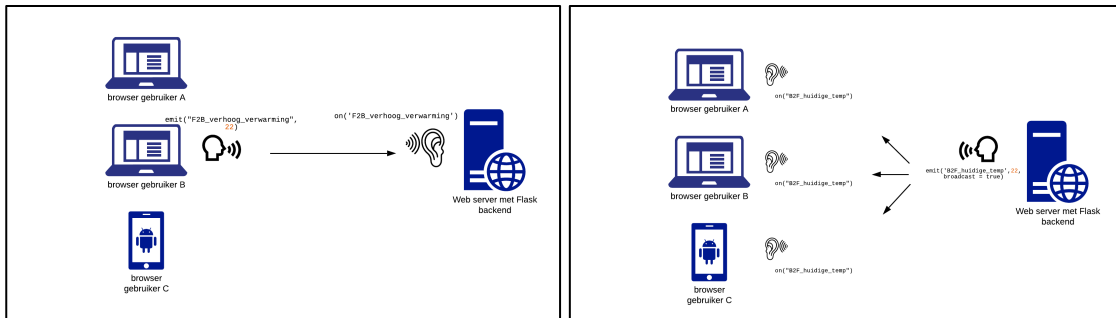


3.2.2 Verzenden met extra data (emit)

- Een boodschap **met** payload: verzend je via `emit("een_omschrijving", payload)`.
- Het ontvangen van de boodschappen die je opnieuw via `on("een_omschrijving")`. Aan dit event koppel je een functie met **1 parameter** die de payload voorstelt.

Als voorbeeld nemen we een web applicatie die de verwarming verhoogt. Er wordt een boodschap naar de webserver verstuurd van 1 client dat de verwarming naar 22°C moet. De webserver antwoordt door naar alle clients een boodschap te versturen dat de huidige temperatuur van de verwarming 22°C is.

In dit voorbeeld is de payload een integer, in de oefening zal de payload een json object worden.



3.2.3 Speciaal event

De eerste maal dat een **client** een verbinding maakt met de webserver wordt er (automatisch) een message/event "connect" verstuurd. Je hoeft – in de javascript code - dit event niet zelf te versturen. Het volstaat om verbinding te maken met de socket.io webserver om deze message te versturen.

```
const lanIP = "192.168.x.x:5000";  
const socket = io(lanIP);
```

Door op de webserver te luisteren naar `on('connect')` weet je wanneer een nieuwe **client** verbinding maakt met de socket.io op de webserver.

3.2.4 Naamgeving

Om eenvoudig socket.io boodschappen te herkennen, gebruiken we volgende prefix als naamgeving:

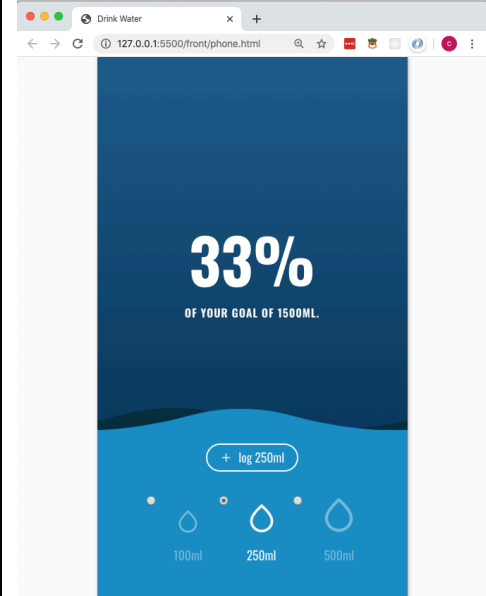
- "F2B_" **F**rontendTo**B**ackend: als het een boodschap is die wordt verzonden van de **client** (**frontend**) naar de **backend**.
- "B2F_" **B**ackendTo**F**rontend: als het een boodschap is die wordt verzonden van de **backend** naar de **client(s)** (**frontend**)

4 Oefening “drinks”

Een kleine webapplicatie om via je telefoon te loggen wat je vandaag al gedronken hebt. Aan de andere kant is er een beheer-gedeelte om te zien hoeveel - en op welk moment - je precies dronk. Als er een nieuwe logging bijkomt, dan wordt de data hier ook live geüpdatet!

Frontend

Web app gedeelte



Beheer gedeelte

Wis gedronken water vandaag

Hoeveel water heb je gedronken?

[Alle data \(live\)](#)

Date	Amount
Fri, 29 Mar 2019 14:55:22 GMT	500 ml
Fri, 29 Mar 2019 14:55:20 GMT	100 ml
Fri, 29 Mar 2019 14:54:48 GMT	250 ml
Fri, 29 Mar 2019 14:52:00 GMT	100 ml
Fri, 29 Mar 2019 14:51:57 GMT	250 ml
Fri, 29 Mar 2019 14:51:00 GMT	100 ml
Fri, 29 Mar 2019 14:50:53 GMT	500 ml
Fri, 29 Mar 2019 14:49:38 GMT	250 ml

4.1 Backend klaarmaken

Data/water.sql

Importeer de database `water` via MySQL Workbench.

backend/app.py

Bereid de backend voor om gebruik te maken van socket.io

Enkele zaken zijn nieuw ten opzichte van de werkwijze van de voorbije weken.

1. Voeg een secret key toe aan de app zodat verzonden data wordt versleuteld.
(https://flask.palletsprojects.com/en/1.1.x/config/#SECRET_KEY)
2. Geef de `app` variabele mee aan de socket.io constructor, zodat de socket.io functionaliteit wordt toegevoegd aan je Flask server. Via de tweede parameter duid je aan dat socket.io CORS aanvaardt.
3. CORS op de gewone API-routes moeten natuurlijk ook gewoon blijven werken.
4. Voeg een eerste socket.io listener `connect` toe.
5. Je stuurt naar de client een boodschap en geeft als payload mee dat er momenteel 0 milliliter is gedronken.
6. Je start de webserver op via de functie `run()` van het object `socketio`
 - Aan deze functie geef je 3 parameters mee
 - De app variabele.
 - Of er al dan niet in debug mode wordt opgestart.
 - De host zet je op `0.0.0.0`.

Op deze manier is de webserver bereikbaar via:

- De localhost (127.0.0.1)
- Maar ook extern via zijn publieke ipadres (192.168.x.x).

Zo kan je straks via je gsm connectie maken met de backend, als je het adres van je laptop weet.

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

# Start app
app = Flask(__name__)
app.config['SECRET_KEY'] = 'Secret!aBcdXYZ' #(1)

# Stel CORS in op Routes en Socket
socketio = SocketIO(app, cors_allowed_origins="*") #(2)
CORS(app) #(3)

# Custom endpoint
endpoint = '/api/v1'

# ROUTES
@app.route('/')
def info():
    return jsonify(info='Please go to the endpoint ' + endpoint)

# # SOCKET.IO EVENTS
@socketio.on('connect') #(4)
def initial_connection():
    print('A new client connect')
    # # Send to the client!
    socketio.emit('B2F_connected', {'currentProgress': 0}) #(5)

# START THE APP
if __name__ == '__main__':
    socketio.run(app, debug=True, host='0.0.0.0') #(6)
```

4.2 Frontend

front/phone.html

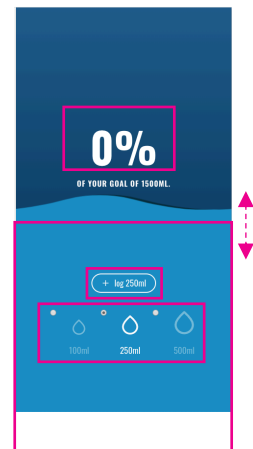
Voor deze oefening krijg je de HTML-structuur al. Pas geen CSS klassen of ID's aan.

- Er is een link naar de socket.io library, datahandler.js en app.js
- Er is een link naar de stylesheet.
- Er is een link naar het webfont dat we gebruiken (Oswald).

We nemen een kijkje in de HTMLcode. Voornamelijk de elementen met een `js-` CSS klasse interesseren ons. Want dit zijn elementen die we straks aanspreken in Javascript.

```
<body class="u-space-center-center">
  <main class="c-app">
    <header class="c-header u-bring-to-front"></header>
    <div class="c-percentage u-bring-to-front">
      <h2 class="c-percentage__current js-percentage"></h2>
      <p class="c-percentage__goal"></p>
    </div>
    <div class="u-bring-to-front">
      <button class="c-btn js-log-water" data-amount="250"></button>
      <div class="c-water-amounts">
        <input class="o-hide-accessible c-water-input js-water-amount" type="radio" name="water" id="water-0" data-amount="100">
        <label for="water-0" class="c-water-amount"></label>
        <input class="o-hide-accessible c-water-input js-water-amount" type="radio" name="water" id="water-1" checked="" data-amount="250">
        <label for="water-1" class="c-water-amount"></label>
        <input class="o-hide-accessible c-water-input js-water-amount" type="radio" name="water" id="water-2" data-amount="500">
        <label for="water-2" class="c-water-amount"></label>
      </div>
    </div>
  </main>
  <div class="c-water js-waves"></div>
</body>
```

- `h2.js-percentage` . Hier wordt het percentage van het gedronken volume in weergegeven. *totale som van wat gedronken is vandaag / 1500 * 100*
- `button.js-log-water`: als er op deze knop wordt geklikt dan verstuurt je via socket.io een message naar de server dat er voor x-aantal milliliter gedronken is. Deze x-waarde wordt bewaard in het attribuut `data-amount` van de knop, zodat deze snel is op te vragen in het clickevent van de knop.
De knop wordt via CSS gestyled, zodat hij mooi is afgerond, hiervoor wordt de CSS klasse `c-btn` gebruikt.
- In `div.c-water-amount`
 - Bevinden zich 3 radiobuttons met steeds de klasse `.js-water-amount`. Deze 3 input elementen hebben hetzelfde `name` attribuut maar natuurlijk wel een andere `id`. Op deze manier kan er slechts 1 radiobutton worden aangeduid.
 - Bevinden zich 3 labels die - via het `for` attribuut - gelinkt zijn met hun overeenkomstig input element (radiobutton). In ieder label staat een SVG afbeelding in de vorm van een waterdruppel.
 - Je leerde reeds: als er geklikt wordt op een label, het bijhorende input element `checked` wordt.
 - Via JS wordt er straks geluisterd naar het veranderen van deze radiobutton. Afhankelijk van de aangeduide radiobutton veranderen we het `data-amount` attribuut van de knop `btn.js-log-water`.
Deze knop zal later op zijn beurt de juiste waarde doorsturen naar de server.
- `js-wave`: dit element bevat een SVG afbeelding in de vorm van een golf. Via zijn CSS klasse `.c-water` krijgt deze SVG een blauwe achtergrondkleur, neemt zijn hoogte en breedte de volledige 100% van de container in.
Via de `top` en `left` property is de startpositie van dit element links bovenaan. Tenslotte laten we de afbeelding "zakken" tot de helft van zijn container via de property `transform: translateY(50%)`. Dit wordt de startpositie van dit element.



```
.c-water {
  position: absolute;
  bottom: 0;
  left: 0;
  height: 100%;
  width: 100%;
  background: #1998c7;
  will-change: transform;
  -webkit-transform: translateY(50%);
  transform: translateY(50%);
  transition: transform 1s ease-in-out;
  z-index: 10;
}
```


front/script/app.js

Analyse van probleemstelling

- Als er een verbinding is met de Socket.io webserver, wordt het totaal gedronken volume uit de database getoond in de UI.
 - => luisteren naar de boodschap van een geslaagde connectie met Socket.io en de payload gebruiken in de UI.
- Als de radiobutton verandert, verandert het data-amount attribuut van de knop zodat er een ander volume kan worden doorgestuurd als er straks op de knop wordt geklikt.
- Als er op de knop wordt geklikt verhoogt het gedronken volume in de database en UI.
 - => klik event van de knop opvangen
 - => de webserver laten weten hoeveel er zonet gedronken is (emit een message). Gebruik het data-amount attribuut van de knop.
 - => wachten op een message van de socket.io webserver met het aantal gedronken milliliters.

DOMContentLoaded

ListenToUI ()
ListenToSocket()

• Opbouw app.js: globale variabelen

In de JS-code staan er al enkele variabelen:

```
let html_addButton, html_wave, html_percentage
const dailyGoal = 1500;
let currentProgress = 0; // in milliliter
const lanIP = '192.168.100.1:5000'; // publiek ip van de webserver
const socket = io(lanIP);
```

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.

Omdat het IP-adres van je laptop (de webserver) kan veranderen, laten we javascript het IP opvragen uit de URL. Op deze manier hoeven we dit niet steeds aan te passen in het script. *Later zal dit 'automatisch' het IP adres worden van de Raspberry PI.*

```
let html_addButton, html_wave, html_percentage
const dailyGoal = 1500;
let currentProgress = 0; // in milliliter
const lanIP = '192.168.100.1:5000'; // publiek ip van de webserver
const lanIP = `${window.location.hostname}:5000`; // publiek ip van de webserver
const socket = io(lanIP);
```

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

Als de DOM geladen is

- Stel de variabele `html_addButton`, `html_wave`, `html_percentage` in

```
html_addButton = document.querySelector('.js-log-water');  
html_wave = document.querySelector('.js-waves');  
html_percentage = document.querySelector('.js-percentage');
```

- Roep de 2 functies op:

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

In de functie `listenToUI()`

- Koppel aan alle input elementen `.js-water-amount` een event als de checkbox verandert.
Binnen dit event:
 - Verander je de tekst in de button `.js-log-water` naar 100, 250 of 500 ml.
 - Verander je het `data-amount` attribuut van de button `.js-log-water` naar 100, 200 of 500 (ml).
- Koppel aan de button `.js-log-water` een click event.
 - Voorlopig schrijf je de string “er wordt x ml gedronken” naar de console van Chrome Dev Tools.

In de functie `listenToSocket()` luister je naar 2 socket.io events.

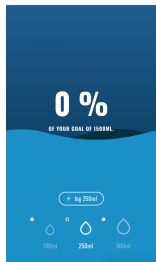
- Luister naar de automatisch verzonden message “connected”
 - Schrijf de string “verbonden met socket webserver” in de console van Chrome.
- Luister naar de message “B2F_connected” die 1 parameter binnen krijgt met een json object `{ 'currentProgress':xx }`
 - Schrijf de string “eerste boodschap server: huidig aantal ml gedronken in DB xx ml” in de console van Chrome. Waar xx de waarde is die binnenkomt via de parameter.

4.3 Eerste test frontend en backend

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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
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:docent-versie2020 christophe$ source /Us
(venv_fswd) Christophe-Howests-MacBook-Pro:docent-versie2020 christophe
* Restarting with stat
* Debugger is active!
* Debugger PIN: 282-832-857
(4315) wsgi starting up on http://0.0.0.0:5000
```

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



4.3.1 Socket.io events op de backend en frontend

Doordat we dit doen, triggeren we reeds enkele events.

In de terminal van VS.Code (**backend**) zien we

```
A new client connect
127.0.0.1 - - [17/Apr/2020 18:02:20] "GET /socket.io/?EIO=3&transport=polling&M68u6Vm HTTP/1.1" 200 446 0.005676
(4315) accepted ["127.0.0.1", 55443]
127.0.0.1 - - [17/Apr/2020 18:02:20] "GET /socket.io/?EIO=3&transport=polling&M68u6X1&sid=c24f1a7524614c788c34bd1d6e64cfd HTTP/1.1" 200 235 0.000196
127.0.0.1 - - [17/Apr/2020 18:02:20] "GET /socket.io/?EIO=3&transport=polling&M68u6Y6&sid=c24f1a7524614c788c34bd1d6e64cfd HTTP/1.1" 200 235 0.000190
127.0.0.1 - - [17/Apr/2020 18:02:20] "GET /socket.io/?EIO=3&transport=polling&M68u6ZV&sid=c24f1a7524614c788c34bd1d6e64cfd HTTP/1.1" 200 235 0.000283
```

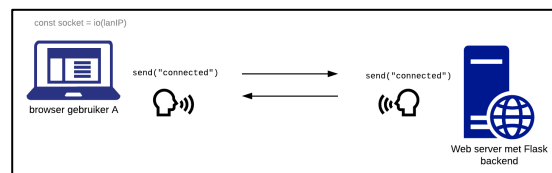
In de console van Chrome Dev Tools (**frontend**) zien we

```
verbonden met socket webserver
eerste boodschap server: huidig aantal ml gedronken in DB 0 ml
```

Dit komt omdat er reeds enkele messages zijn doorgezonden via socket.io.

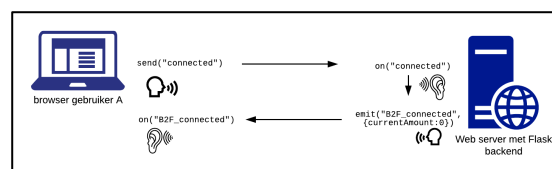
- **Automatische messages**

- De client (**frontend**) verzendt een “connected” message als hij connectie maakt via de regel `const socket = io(lanIP)`.
- De **backend** verstuurt ook een automatisch “connected” message naar de client (frontend) als hij deze verbinding binnenkrijgt.



- **Eigen gemaakte messages**

- De **backend** luistert naar een “connected” message van een client en verstuurt op zijn beurt een message “B2F_connected” met als payload het json object `{currentAmount: 0}`.
- De **client** luistert naar het event “B2F_connected” en verwerkt de payload door deze te tonen in de console van Chrome Dev Tools



4.3.2 UI Events op de frontend

Verander de radiobutton naar een ander volume (bijvoorbeeld naar 500 ml). Controleer of het `data-amount` attribuut van de knop mee is veranderd, en of de `innerHTML` van `span.js-log` is veranderd.



Klik op de button, in de console van Chrome Dev Tools verschijnt het volume dat je wil loggen.

verbonden met socket webserver
eerste boodschap server: huidige aantal ml gedronken in DB 0 ml
er wordt 500 ml gedronken

4.4 B2F_connected verder uitwerken (socket event)

- **Frontend**

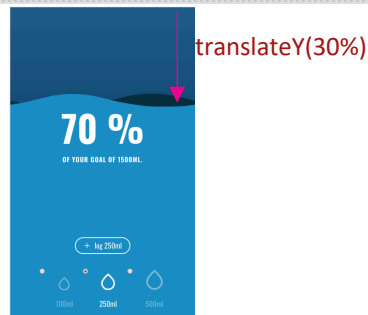
Momenteel log je enkel het binnengekomen volume in de console

```
socket.on('B2F_connected', function (value) {  
  console.log(`eerste boodschap server: huidige aantal ml gedronken in  
  DB ${value.currentProgress} ml`);  
});
```

Voeg extra javascript code toe aan dit event

- Bewaar de binnengekomen payload `value.currentProgress` waarde in de globale variabele `currentProgress`.
- Bereken het percentage dat er momenteel is gedronken en rond dit af naar boven.
 - Bv: $700 \text{ (ml)} / 1500 \text{ (ml)} * 100 = 46,6666 \text{ (}\%) \rightarrow 47 \text{ (}\%)$
- Roep de functie `updateView(progressInPercentage)` op.
- Schrijf de functie `updateView(progressInPercentage)`
 - Toon het binnengekomen percentage in `html_percentage`.
 - Verander de positie van `html_wave`.
 - Stel dat je reeds 70% dronk van je dagelijks doel:
Dan heb je volgende berekening nodig

```
html_wave.style.transform = `translateY(${100 - 70}%)`;
```



- Heb je 0 % gedronken dan zal de uitkomst $100-0 = 100\%$ zijn, en zal de golf helemaal gedaald zijn tot onderaan de container.
- Heb je 100 % gedronken dan zal de uitkomst $100-100 = 0\%$ zijn, en zal de golf helemaal bovenaan de container staan.
- Hou er rekening mee dat je meer kan drinken dan de daily goal. In zo een situatie zal de uitkomst $100-150 = -50\%$ zijn, en zal de golf boven/buiten de container vallen.
 - Corrigeer dus - door middel van een if structuur - alle negatieve getallen naar 0.

- **Backend**

Als je gaat kijken naar de backend, zie je dat er altijd gestart wordt met 0 (ml) als currentProgress. Dit zal natuurlijk niet altijd het geval zijn, misschien is er eerder op de dag al water gelogd in de database.

backend/repositories/DataRepository.py: er is een functie `read_total_progress(date)`, deze functie neemt de som van alle amounts van een bepaalde dag (like '2020-04-25%').

backend/app.py: pas de code in `@socket.on('connect')` aan. Zodat je niet altijd `{'currentProgress': 0}` terugstuurt maar de huidige totale progress van vandaag. Dit doe je door aan de functie `read_total_progress(date)`, de huidige dag mee te geven.

Je moet wel een extra controle doen, want als er vandaag nog niets is logd in de database zal `data['amount']`, None zijn.

Omdat er geen record voldoet aan de WHERE clause. In deze situatie stellen we `previous_progress` gelijk aan 0.

```
@socketio.on('connect')
def initial_connection():
    print('A new client connect')
    # socketio.emit('B2F_connected', {'currentProgress': 0})
    data = DataRepository.read_total_progress(date.today())
    if data['amount']:
        previous_progress = data['amount']
    else:
        previous_progress = 0
    # Send to the client!
    socketio.emit('B2F_connected', {'currentProgress': previous_progress})
```

4.5 Click event verder uitwerken (UI event)

Als er op de knop geklikt wordt, verzend je via socket.io een message naar de backend met het volume dat er zojuist is gelogd.

Nadat de backend deze hoeveelheid heeft gelogd, verstuurt hij in een bericht naar de client(s) hoeveel er net is gedronken.

- **Frontend**

Momenteel ziet het click-event van de knop er als volgt uit.

```
html_addButton.addEventListener('click', function () {  
  const newAmount = html_addButton.dataset.amount; // What amount has been  
  clicked?  
  console.log(`er wordt ${newAmount} ml gedronken`);  
});
```

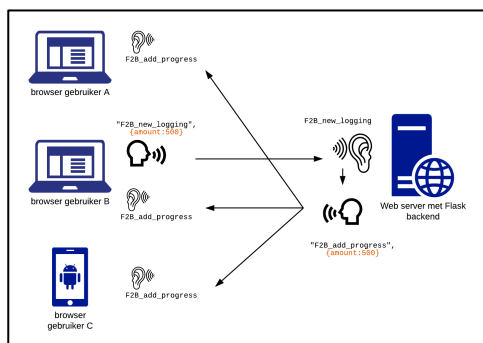
Vervolledig het click event met de code om een message te versturen naar de backend.

- Geef de message de naam "F2B_new_logging"
- De payload is een json object {amount: newAmount}

- **Backend**

Backend/app.py

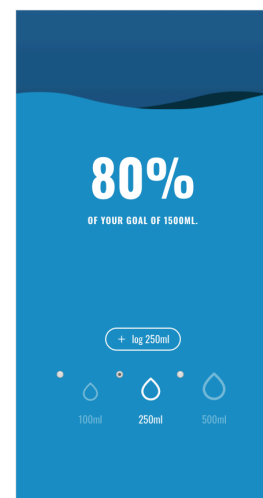
- Luister naar de message "F2B_new_logging"
- Schrijf de binnengekomen amount weg naar de database. Gebruik hiervoor de functie `create_log()` van de `datarepository`.
 - Deze functie verwacht als eerste parameter het huidige tijdstip. Deze kan je opvragen via `datetime.now()`
- Als het toevoegen gelukt is verzend je een message 'B2F_addProgress' naar de client(s). De payload is opnieuw een jsonobject {'amount': amount}



- **Frontend**

Luister in de frontend naar de message `F2B_add_progress`

- Als er zo een message binnenkomt:
 - Verhoog je de waarde van de globale variabele `currentProgress` met de binnengekomen waarde.
 - Via de functie `parseInt("500")` kan je een stringwaarde omzetten naar een geheel getal.
 - Bereken hoeveel procent je moment dronk van de daily amount.
 - Pas je de UI aan via de functie `UpdateView()`. Als parameter geef je het nieuwe percentage mee.



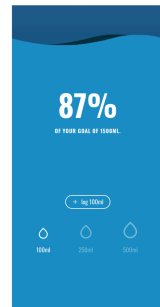
Open de website in verschillende browserstabs (of op je gsm), je zal zien dat alle schermen geupdate worden. Dit komt omdat er standaard gebroadcast wordt naar alle clients.

Op je gsm gebruik je het publieke adres van de Live Server <http://192.168.x.x:5500/front/phone.html>

- **Afwerking**

Verberg de radiobuttons - via CSS - door volgende css-klasse uit commentaar te halen in front/style/screen.css. Deze CSS-klasse is reeds gekoppeld aan alle input elementen.

```
.o-hide-accessible {  
  position: absolute;  
  width: 1px;  
  height: 1px;  
  padding: 0;  
  margin: -1px;  
  overflow: hidden;  
  clip: rect(0, 0, 0, 0);  
  border: 0;  
}
```



4.6 Overview.html


- **Backend**

 **backend/app.py**

Schrijf 2 normale API routes zoals je reeds deed in week 4,5,6,7.

/api/v1/progress/today	DELETE	DataRepository.delete_total_progress(date.today()) Verwijder alle records die vandaag zijn gelogd.
/api/v1/progress	GET	DataRepository.read_logging() Geef alle records terug.

- **Frontend**

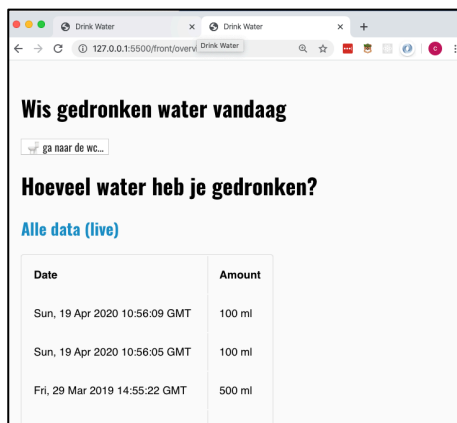
 **front/overview.html + front/script/overview.js**


Voor deze oefening krijg je de HTML-structuur al. Pas geen classes of ID's aan.


- Er is een link naar de socket.io library, datahandler.js en overview.js
- Er is een link naar de stylesheet.

Je volgt de klassieke werkwijze van REST API:

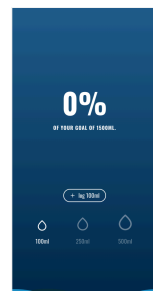
- Bij het laden van de pagina vul je de table `.js-overview` op met het resultaat van een fetch call naar het endpoint `GET /api/v1/progress`.
- Als er op de knop `.js-clear-amount-today` wordt geklikt, doe je een fetch call naar het endpoint `DELETE /api/v1/progress/today`.
 - Schenk aandacht dat je dit via de DELETE method doet van het endpoint!
 - Bij een geslaagde fetch call. Schrijf je een boodschap weg naar de console van Chrome Dev Tools.



 **front/phone.html + front/script/app.js**

Als er een phone.html pagina openstaat, merk je dat deze onveranderlijk blijft. Een goede UX zou het percentage onmiddellijk naar 0% brengen indien er in overview.html op “ ga naar wc” wordt geklikt.

Daarom verzend je in de **backend**code van het endpoint `DELETE /api/v1/progress/today`, een extra `emit('B2F_clear', {'amount': 0})`. In de **frontend** van phone.html, luister je naar deze message, en pas je de UI aan, door `UpdateView(0)` op te roepen met 0 (%) als parameterwaarde.



front/overview.html + front/script/overview.js

Eenmaal de tabel is geladen, blijft de data in de tabel ongewijzigd.

Bij een socket.io message 'B2F_clear' of 'B2F_addProgress' moet de tabel echter aangepast worden.

Luister daarom naar deze messages in de **frontend**. Treden deze messages op, dan doe je een nieuwe API call naar het endpoint `GET /api/v1/progress` zodat de nieuwe data wordt opgevraagd en wordt getoond in de tabel.

Hoeveel water heb je gedronken?	
Alle data (live)	
Date	Amount
Sun, 19 Apr 2020 10:56:09 GMT	100 ml
Sun, 19 Apr 2020 10:56:05 GMT	100 ml
Fri, 29 Mar 2019 14:55:22 GMT	500 ml
Fri, 29 Mar 2019 14:55:20 GMT	100 ml

howest
hogeschool