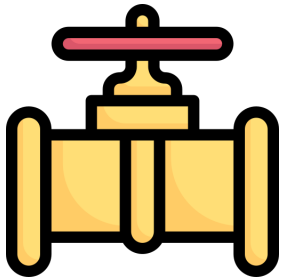
Pilotage d'une vanne avec MQTT, une interface web sur un Raspberry Pi

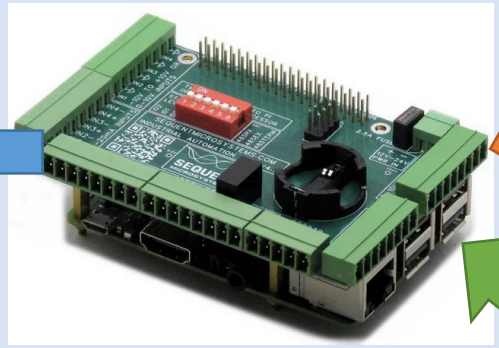
Sommaire

- Introduction
  - Schéma générale du dispositif
  - Qu’est ce que MQTT?
  - Qu’est ce qu’il faut pour une interface web?
- Comment faire?
  - Technologie choisie
  - Architecture logiciel choisie
- Aperçu
  - Aperçu de Mongoose 
  - Aperçu de Boostrap 5 
  - Aperçu de VueJS 3 
  - Aperçu sans framework JS

Schéma générale du dispositif



4 – 20mA



Système autonome sans écran dans un coffret en Wifi

WIFI

Interface Web

WIFI

Broker MQTT

WIFI

- 2 capteurs d'O_2
- 1 capteur de niveau d'N_2 liquide

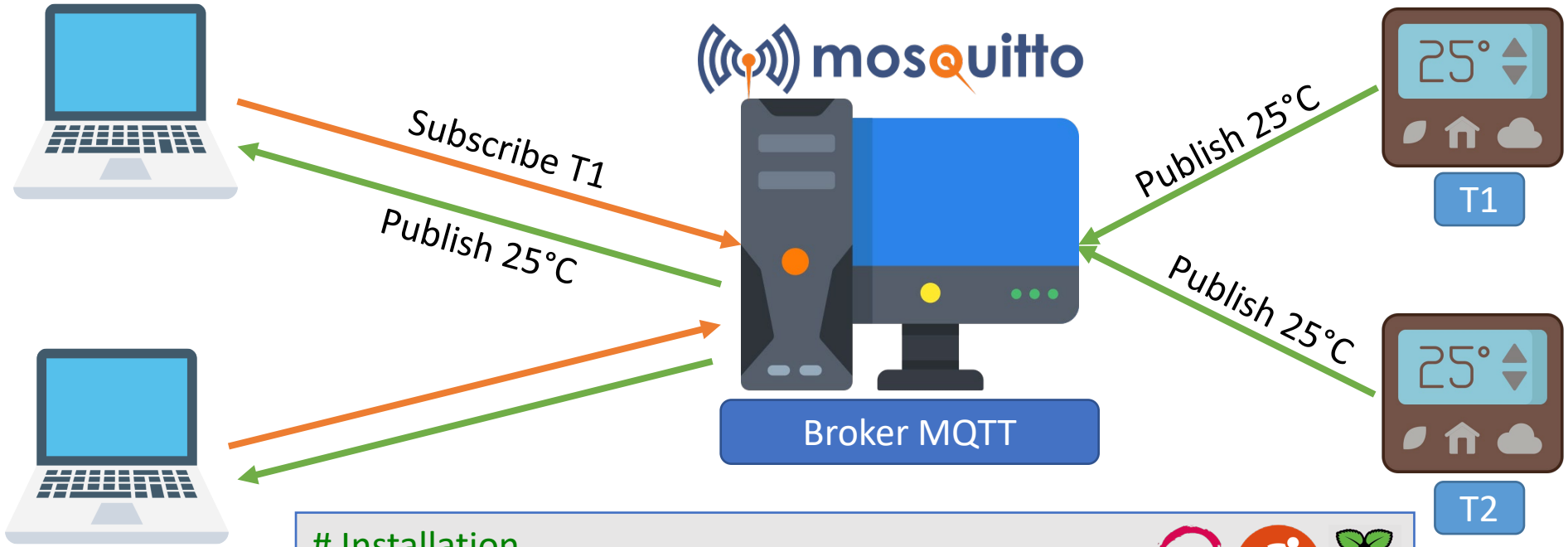
MQTT

HTTP

Pour Information:
SM-I-001 de Sequent Microsystems

Qu’est ce que MQTT?

MQTT = Message Queuing Telemetry Transport




```
# Installation
sudo apt install mosquitto mosquitto-clients
# Publish
mosquitto_pub -h 127.0.0.1 -p 1883 -t T1 -m 25.0
# Subscribe
mosquitto_sub -h 127.0.0.1 -p 1883 -t T1
```

Qu'est ce qu'il faut pour une interface web?

- Node JS: Nuxt, NEXT.js, …
- Python: Streamlit, Dash, …

Backend - Serveur

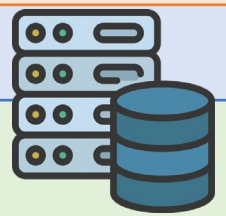
Serveur HTTP (Apache, Ngnix, …)

- Symfony
- Laravel
- CakePHP
- …

- Flask
- Bottle
- Starlette
- Django
- …

- Ruby on Rails (Ruby)
- Express (Node JS)
- …

Base de donnée (MySQL, PostgreSQL, MongoDB, InfluxDB,…)
Object Relational Mapper: SQLAlchemy (Python), Doctrine (PHP), …



Frontend - Client

- **Bootstrap**
- Tailwind CSS
- Materialize
- Foundation
- …

Sass → CSS

- **VueJS**
- React
- Angular
- Svelte
- jQuery
- …

TS → JS

Que choisir comme technologie? (1/2)

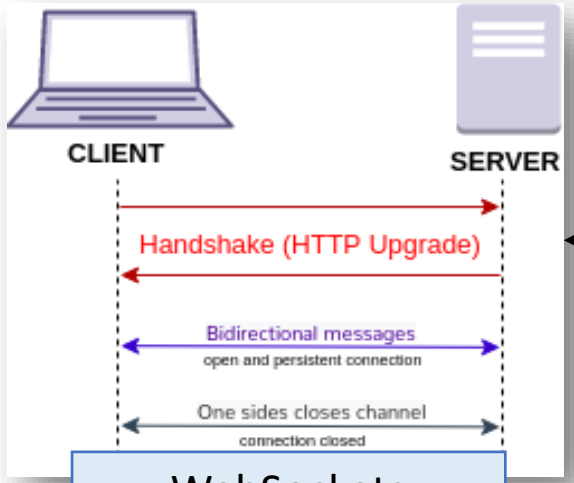
- Le code doit fonctionner sur un Rasberry Pi et donc avec Linux
- L'interface doit être déportée et sera aussi sur un ordinateur Linux
- J'ai un code C prêt à l'emploie pour MQTT (paho)
- Un code pas trop lourd
- Reprise de connexion (Wifi pas très stable)
- Le code doit démarrer automatiquement quand on alimente le Rasberry Pi
- Le "Stackable HAT" fonctionne en Python, C ou Node-RED

Que choisir comme technologie? (2/2)

Langage	Compatible avec la carte	MQTT?	Dashboard Web	Expérience avec ce langage
LabVIEW	NON	Librairie externe dans le VIPM	Ni G Web Développement Software	OUI
Node-RED	OUI	Inclus	OUI	NON
	OUI	OUI avec une librairie externe	A coder	Un peu d'expérience
	OUI	OUI avec une librairie externe (j'ai déjà le code)	A coder	Un peu d'expérience

Choix du langage C

Architecture logiciel (1/2)



WebSockets

Supervisor pour la gestion des "*services*"

Qu'est ce qu'une WebSocket?

Mongoose – Embedded web server

- MQTT: Paho
- Configuration: Libconfig
- Librairie de la carte
- lecture du JSON: Mongoose

Sur le Rasberry Pi

WebSocket

HTTP

VueJS + Bootstrap

- highlight.js
- moment.js

Dans le navigateur

***IP*:8181** => Web Interface
***IP*:8181/websocket** => WebSocket
***IP*:9001** => Supervisor

Pour information, à ma connaissance:

- Le Dashboard Node-RED "*1.0*" utilise Angular + WebSockets
- Le Dashboard Node-RED "*2.0*" (flowfuse) utilise VueJS 3 + Socket.io ("*wrapper pour les WebSockets*")

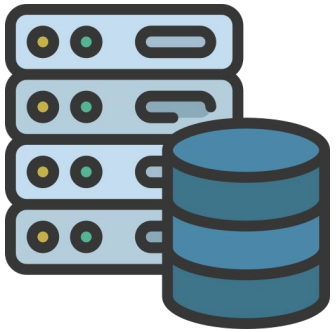
Architecture logiciel (2/2)

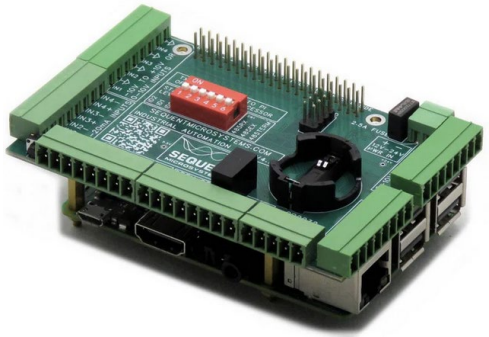
Connexion MQTT Broker → MQTT Subscribe → Attendre des données → **Ecriture** des données dans une variable → Vérification des seuils → Mettre la valeur sur la vanne et Publish MQTT

⚠ C'est une exécution par Callback donc pas vraiment en parallèle

Lecture des données dans la variable → Envoie des données dans la WebSocket en JSON → Attendre 600 ms

Attendre connexion HTTP → Envoie des fichiers statiques au client





Mongoose (1/2)

```c
static void ev_handler(struct mg_connection *c, int ev, void *ev_data)
{
    if (ev == MG_EV_HTTP_MSG)
    {
        // New http request receive
        struct mg_http_serve_opts opts = {.root_dir = "web_root"};
        mg_http_serve_dir(c, ev_data, &opts);
    }
}
```

```c
struct mg_mgr mgr; // hold all connection
mg_mgr_init(&mgr); // initialisation
const char *web_host = "http://0.0.0.0:8181";
mg_http_listen(&mgr, web_host, ev_handler, NULL);

while(keepRunning == 1)
{
    mg_mgr_poll(&mgr, 250);
}

mg_mgr_free(&mgr);
```

```
.
├── lib
│   ├── mongoose.c
│   └── mongoose.h
├── main.c
├── Makefile
└── web_root
    ├── bootstrap.min.css
    ├── bootstrap.min.css.map
    └── index.html
```

Librairie Mongoose

Fichier "statique"

```c
struct mg_mgr {
  struct mg_connection *conns;  // List of active con
  struct mg_dns dns4;           // DNS for IPv4
  struct mg_dns dns6;           // DNS for IPv6
  int dnstimeout;               // DNS resolve timeou
  unsigned long nextid;         // Next connection ID
  void *userdata;               // Arbitrary user data pointer
  ...
  struct mg_tcpip_if *ifp;      // Builtin TCP/IP stack only. Interface pointer
};
```

```c
struct mg_connection {
  struct mg_connection *next;  // Linkage in struct mg_mgr :: connections
  struct mg_mgr *mgr;          // Our container
  struct mg_addr loc;          // Local address
  struct mg_addr rem;          // Remote address
  void *fd;                    // Connected socket, or LWIP data
```

Mongoose (2/2)

```
enum {
  MG_EV_ERROR,      // Error                        char *error_message
  MG_EV_OPEN,       // Connection created           NULL
  MG_EV_POLL,       // mg_mgr_poll iteration        uint64_t *uptime_millis
  MG_EV_RESOLVE,    // Host name is resolved        NULL
  MG_EV_CONNECT,    // Connection established       NULL
  MG_EV_ACCEPT,     // Connection accepted          NULL
  MG_EV_TLS_HS,     // TLS handshake succeeded      NULL
  MG_EV_READ,       // Data received from socket    long *bytes_read
  MG_EV_WRITE,      // Data written to socket       long *bytes_written
  MG_EV_CLOSE,      // Connection closed            NULL
  MG_EV_HTTP_HDRS,  // HTTP headers                 struct mg_http_message *
  MG_EV_HTTP_MSG,   // Full HTTP request/response   struct mg_http_message *
  MG_EV_WS_OPEN,    // Websocket handshake done     struct mg_http_message *
  MG_EV_WS_MSG,     // Websocket msg, text or bin   struct mg_ws_message *
  MG_EV_WS_CTL,     // Websocket control msg        struct mg_ws_message *
  MG_EV_MQTT_CMD,   // MQTT low-level command       struct mg_mqtt_message *
  MG_EV_MQTT_MSG,   // MQTT PUBLISH received        struct mg_mqtt_message *
  MG_EV_MQTT_OPEN,  // MQTT CONNACK received        int *connack_status_code
  MG_EV_SNTP_TIME,  // SNTP time received           uint64_t *epoch_millis
  MG_EV_WAKEUP,     // mg_wakeup() data received    struct mg_str *data
  MG_EV_USER        // Starting ID for user events
};
```

```
CC = gcc
CFLAGS  = -Wall
ifeq ($(OS),Windows_NT)
    LIBS = -lws2_32 -lbcrypt
endif
SRC = main.c lib/mongoose.c

all:
    $(CC) $(SRC) $(CFLAGS) $(LIBS) -o ServerHTTP
clean:
    rm ServerHTTP
```

Makefile

https://github.com/cesanta/mongoose/tree/master/tutorials
https://mongoose.ws/documentation/

- MQTT
- HTTP
- WebSockets
- Sockets
- Wizard génération automatique Web UI

Bootstrap 5 (1/3)

CSS en local ou distant (Content Delivery Network)

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Ma page</title>
    <link href="bootstrap.min.css" rel="stylesheet" />
  </head>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

HTML pour Boostrap

Qu'est ce que le CSS?

```css
.maClass
{
    background-color: blue;
}

#monId
{
    border: 1px solid #83d351;
}

div.MyClass > ul > li
{
    color: red;
}
```

https://developer.mozilla.org/fr/docs/Web/CSS
https://developer.mozilla.org/fr/docs/Web/HTML

Bootstrap 5 (2/3)

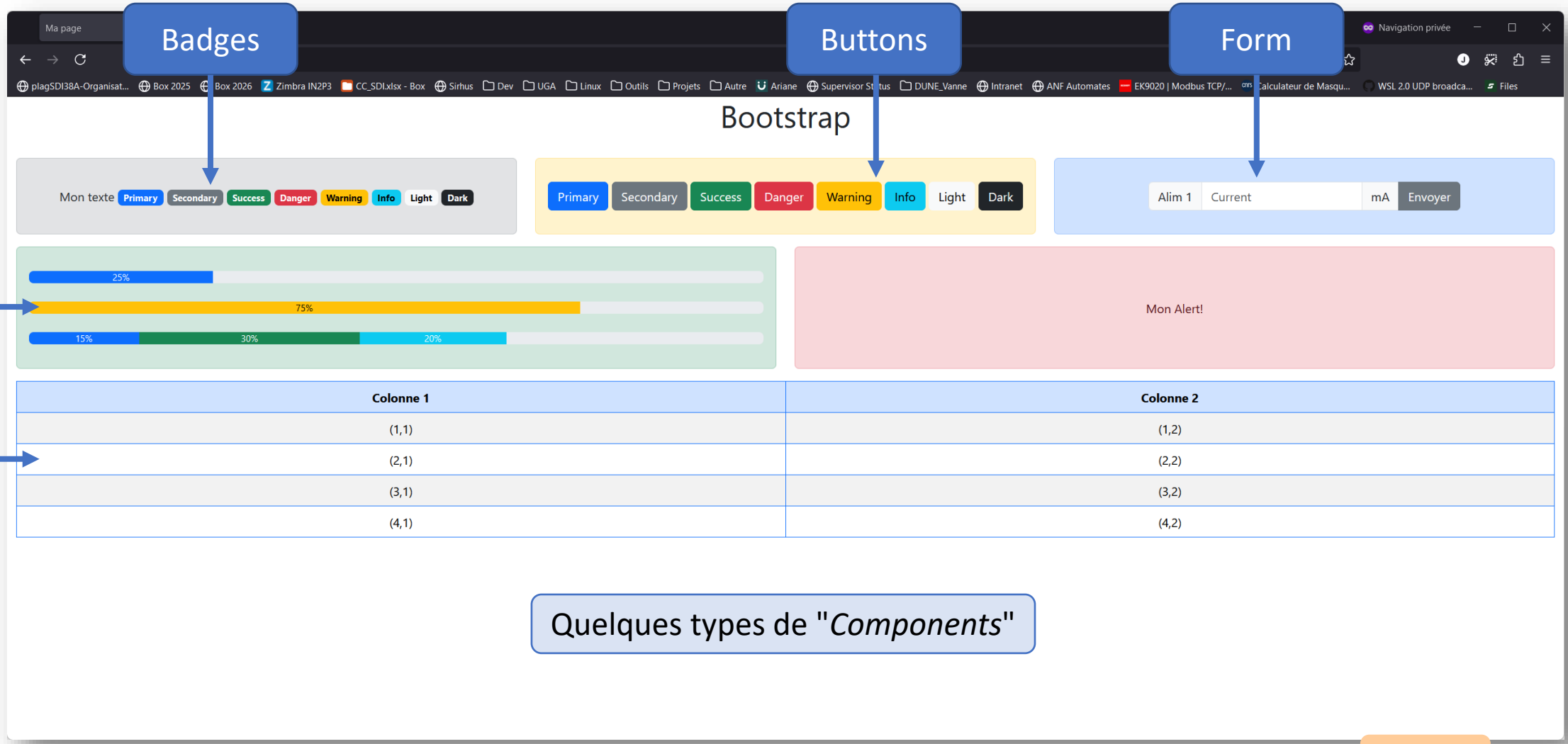
	xs <576px	sm ≥576px	md ≥768px	lg ≥992px	xl ≥1200px	xxl ≥1400px
Container `max-width`	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`	`.col-xxl-`
# of columns	12					

```
<div class="container-fluid text-center">
    <div class="row">
        <div class="col">
            Ma grille
        </div>
    </div>
    <div class="row">
        <div class="col">
            Column 1,1
        </div>
        <div class="col">
            Column 1,2
        </div>
    </div>
    <div class="row">
        <div class="col-3">
            Column 2,1
        </div>
        <div class="col-9">
            Column 2,2
        </div>
    </div>
</div>
```

Grille responsive à mettre entre les balises "*body*"

Bootstrap 5 (3/3)

https://getbootstrap.com/docs/5.3/getting-started/introduction/



Progress Bar

Table

Colonne 1	Colonne 2
(1,1)	(1,2)
(2,1)	(2,2)
(3,1)	(3,2)
(4,1)	(4,2)

Quelques types de "*Components*"

VueJS 3 (1/4)

npm create vue@latest

```
.
├── index.html
├── jsconfig.json
├── package.json
├── public
│   └── favicon.ico
├── README.md
├── src
│   ├── App.vue
│   ├── assets
│   ├── components
│   ├── composables
│   └── main.js
└── vite.config.js
```

- public — Fichier statique
- App.vue — Application VueJS
- assets — CSS
- components — Composants réutilisables
- composables — Fonctions JS

https://fr.vuejs.org/guide/introduction.html
https://developer.mozilla.org/fr/docs/Web/API/WebSocket
https://nodejs.org/fr

```js
import { onMounted, onUnmounted, ref } from 'vue'

export function useWebSocket(address)
{
    const socket = ref(null)
    const WebSocketData = ref(null)

    function startWebSocket()
    {
        socket.value = new WebSocket(address)

        socket.value.onmessage = (e) => {
            // il y a des datas dans la WebSocket
            WebSocketData.value = JSON.parse(e.data)
        }

        socket.value.onopen = (e) => {
            console.log("WebSocket connection opened: ", e)
        }

        socket.value.onclose = (e) => {
            console.log("WebSocket close: ", e)
        }

        socket.value.onerror = (e) => {
            console.log("WebSocket error", e)
        }
    }

    onMounted(() => {
        // composant Mounted
        startWebSocket()
    })
    onUnmounted(() => {
        // composant Unmounted
    })
    return {WebSocketData, socket}
}
```

useWebSocket.js

VueJS 3 (2/4)

```
.
├── index.html
├── jsconfig.json
├── package.json
├── public
│   └── favicon.ico
├── README.md
├── src
│   ├── App.vue
│   ├── assets
│   ├── components
│   ├── composables
│   └── main.js
└── vite.config.js
```

Fichier statique

Application VueJS

CSS

Composants réutilisables

Fonctions JS

```
<DisplayData :data="WebSocketData"></DisplayData>
```

Template

```
<template>
    <div class="alert alert-light">
        <code><pre v-html="highlight_code"></pre></code>
    </div>
</template>
```

Script

```
<script setup>
    import { computed } from 'vue'

    const props = defineProps({data: Object})

    const highlight_code = computed(() => {
      return JSON.stringify(props.data, null, 2)
    })
</script>
```

Style

```
<style scoped>
    /* Mon style CSS */
</style>
```

DisplayData.**vue**

VueJS 3 (3/4)

```
.
├── index.html
├── jsconfig.json
├── package.json
├── public                    Fichier statique
│   └── favicon.ico
├── README.md
├── src
│   ├── App.vue               Application VueJS
│   ├── assets                CSS
│   ├── components            Composants réutilisables
│   ├── composables           Fonctions JS
│   └── main.js
└── vite.config.js
```

Script

Template

Style

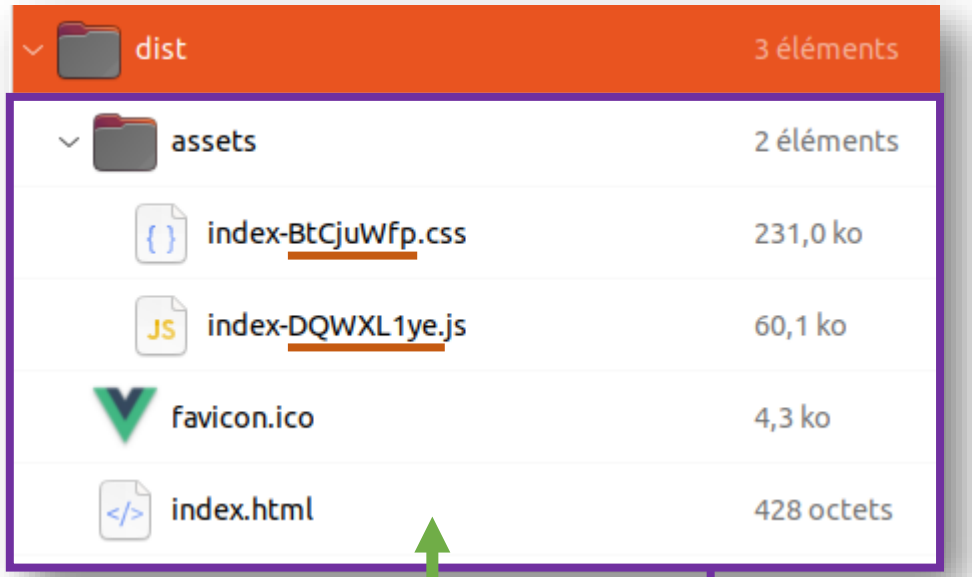
```vue
<script setup>
  import { useWebSocket } from './composables/useWebSocket.js'
  import DisplayData from './components/DisplayData.vue';
  var ws = "ws://127.0.0.1:8181/websocket"
  const {WebSocketData, socket} = useWebSocket(ws)
</script>

<template>
  <div class="container-fluid">
    <div class="row">
      <div class="col">
        <DisplayData :data="WebSocketData"></DisplayData>
      </div>
    </div>
  </div>
</template>

<style scoped>

</style>
```

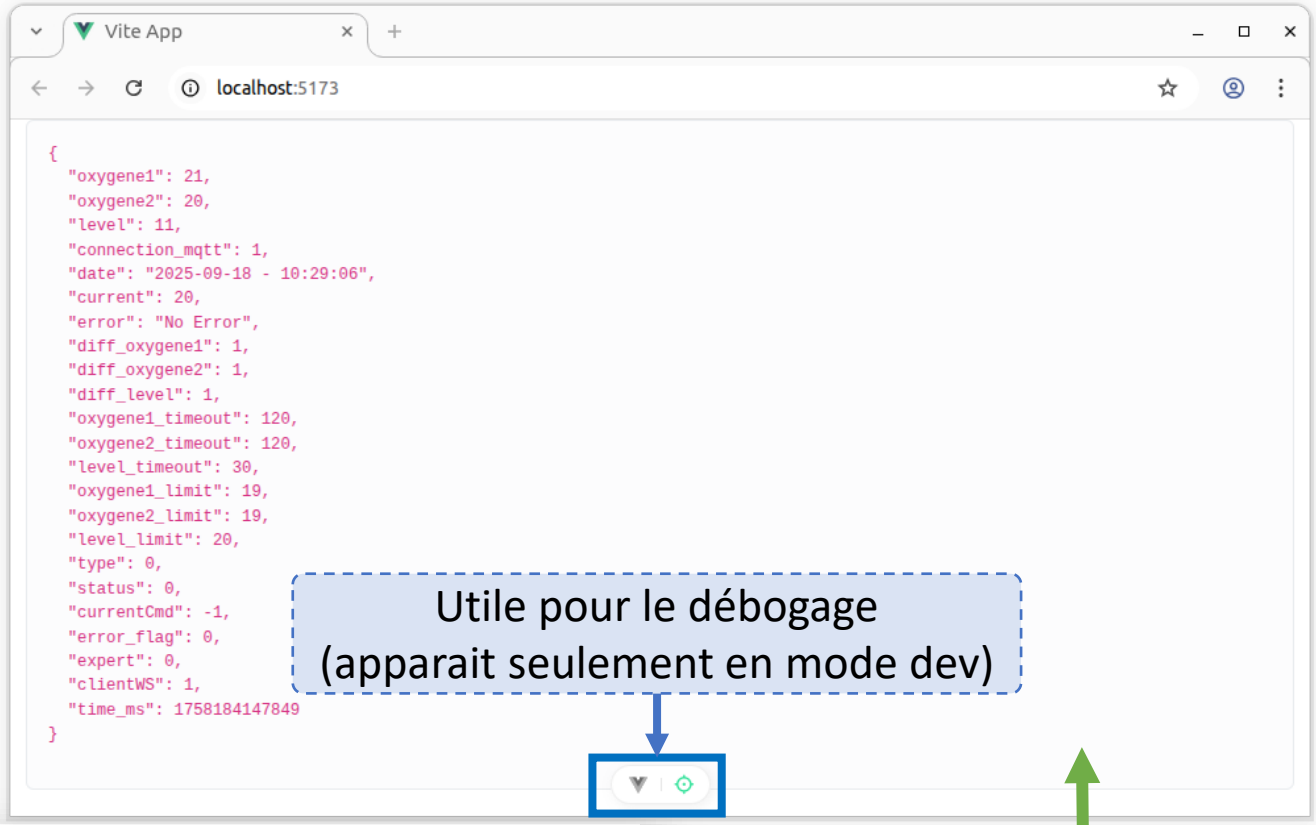
App.vue

```
import './assets/bootstrap.min.css'

import { createApp } from 'vue'
import App from './App.vue'

createApp(App).mount('#app')
```

main.js

VueJS 3 (4/4)

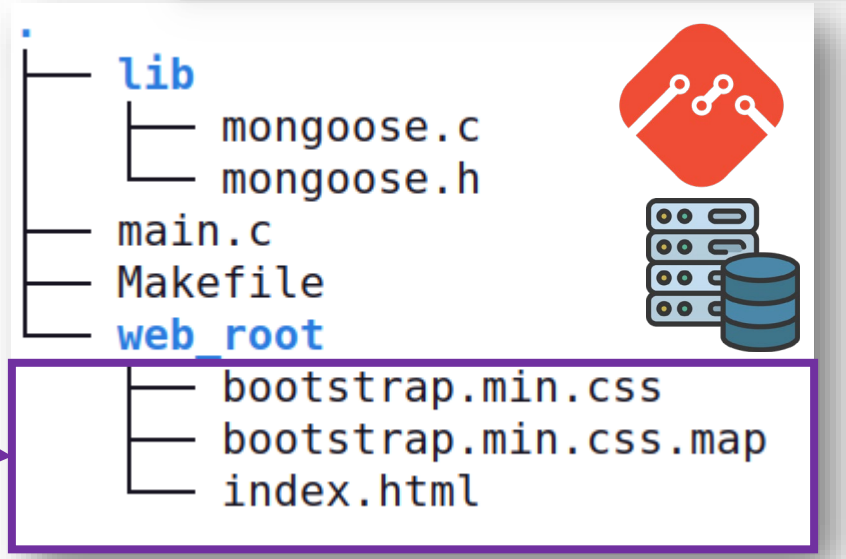




Utile pour le débogage
(apparait seulement en mode dev)

npm run **build**

npm run **dev**



Sans framework JS

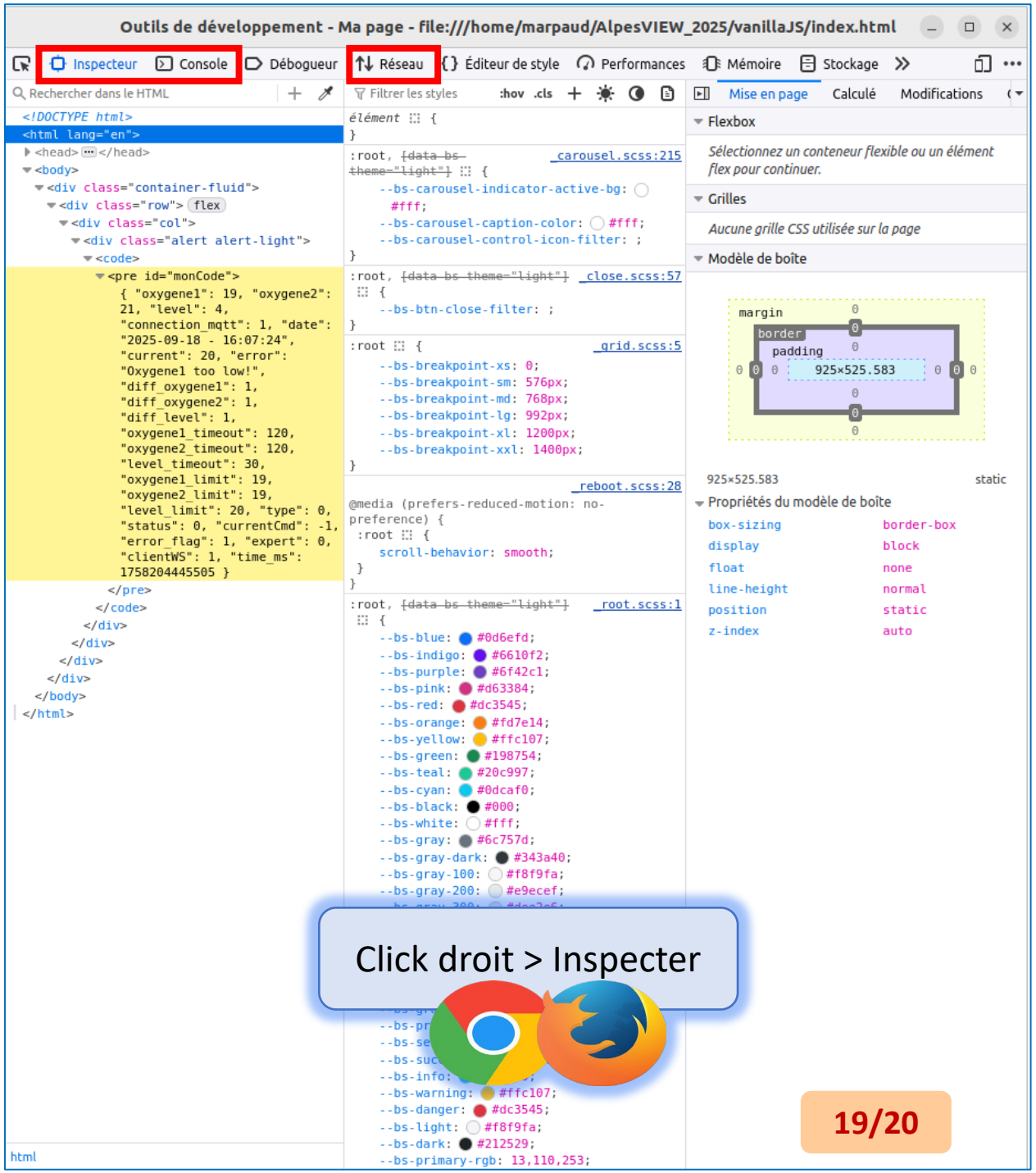
```
var ws = "ws://127.0.0.1:8181/websocket"
const socket = new WebSocket(ws)

socket.onmessage = (e) => {
    var data
    data = JSON.parse(e.data)
    data = JSON.stringify(data, null, 2)
    document.getElementById("Code").innerHTML = data
}
```

main.js

```
<div class="container-fluid">
    <div class="row">
        <div class="col">
            <div class="alert alert-light">
                <code><pre id="Code"></pre></code>
            </div>
        </div>
    </div>
</div>
```

Extrait de index.html

⚠ Cross-origin resource sharing (CORS)



Conclusion

svg.js

chart.js

Status WebSocket | Manuel | 2025-09-19T12:34:22.903+02:00 - diff: 0.603 s

0 % Send | Start Auto | Expert mode

Current: 4.00 mA

Channels	Value	Last Data (s)
Oxygene 1	25%	14s
Oxygene 2	24%	14s
Level N2	19 cm	14s
Current	4.00 mA (0.0%)	
Error	No Error	
Last Data	a few seconds ago	
Timestamp	a few seconds ago	

JSON


```
{
  "oxygene1": 25,
  "oxygene2": 24,
  "level": 19,
  "connection_mqtt": 1,
  "date": "2025-09-19 - 12:34:08",
  "current": 4,
  "error": "No Error",
  "diff_oxygene1": 14,
  "diff_oxygene2": 14,
  "diff_level": 14,
  "oxygene1_timeout": 120,
  "oxygene2_timeout": 120,
  "level_timeout": 30,
  "oxygene1_limit": 19,
  "oxygene2_limit": 19,
  "level_limit": 20,
  "type": 0,
  "status": 0,
  "currentCmd": -1,
```


23/09/2025

- Possibilité de faire énormément de chose en terme de personnalisation
- Demande une maitrise de différents langages (JS/CSS/HML, Python/C/…)
- Possibilité d'utiliser Ni G Web Développement Software

(mais peu de personnalisation)

- Pour VueJS, il y a des librairies de components/composables (vueuse, primevue, …)
- Demande plus de retour pour savoir si ça peut fonctionner un peu de partout

20/20