Fiche Technique

Sommaire

II. Capteurs et emplacement

1 .Capteurs utilisé

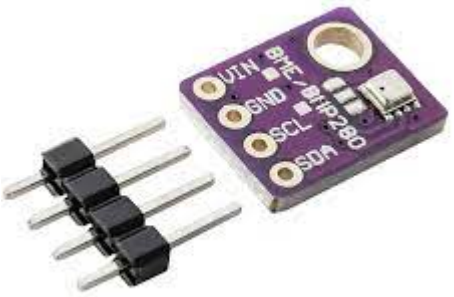
a. MQ135:

Le capteur MQ-135 permet d' évaluer la qualité de l'air ambiant. Il mesure la concentration de différents gaz tels que le dioxyde de carbone (CO2), l'ammoniac (NH3), Son utilisation permet d'obtenir des données précieuses sur la santé environnementale.

b.BME280:

Le BME280 est un capteur polyvalent qui possède une grande précision et qui est capable de mesurer la pression atmosphérique, la température, l'humidité et l'altitude .

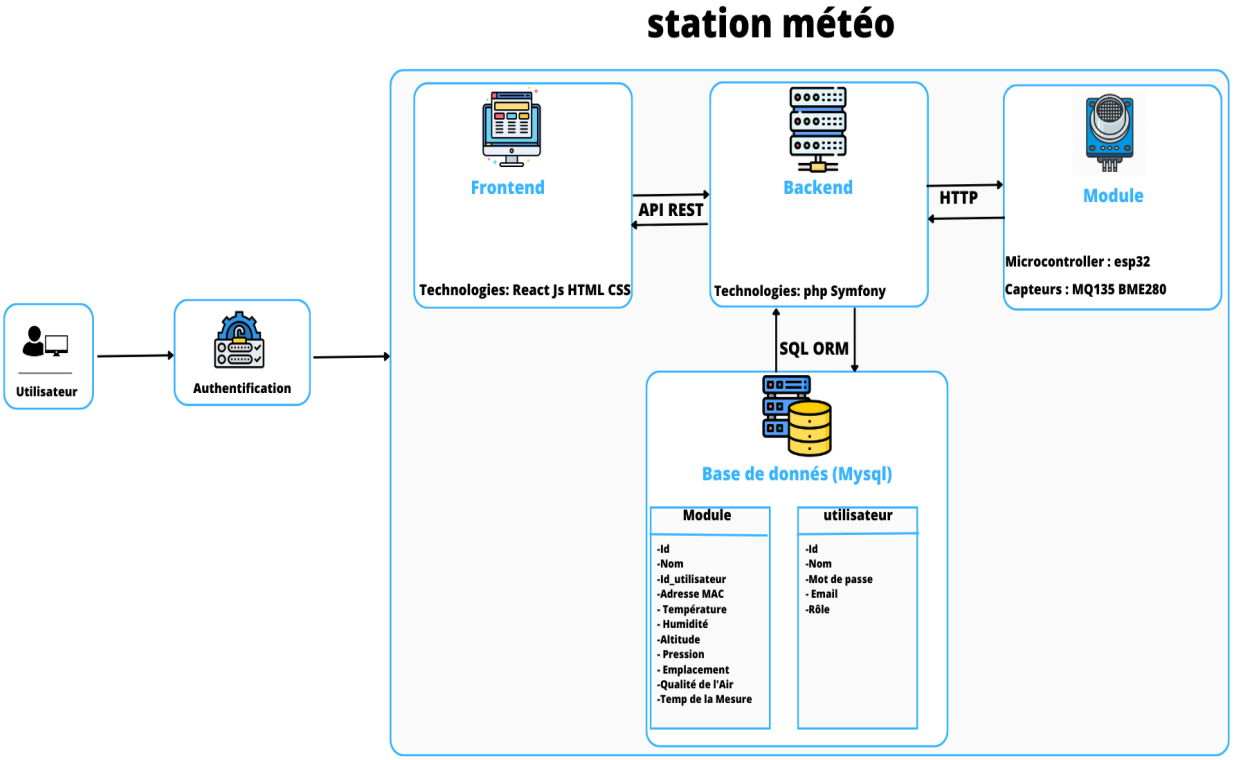
2 .Emplacement



Chaque module, composé de l'ESP 32 et des capteurs mentionnés, sera déployé dans différentes salles et zones du campus. Cette répartition permettra une collecte diversifiée de données, couvrant ainsi l'ensemble de l'environnement du campus

III. Infrastructure et Gestion Système

1 .Architecture générale



2 .Module

a. Gestion du wifi:

Le code intègre la bibliothèque WiFiManager, offrant une gestion dynamique des connexions WiFi. En cas d'échec de connexion, le dispositif bascule automatiquement en mode de configuration pour permettre la saisie des informations de connexion WiFi. Cette approche simplifie la mise en réseau des capteurs.

```
// Try to connect to WiFi or configure if not connected
WiFiManager wifiManager;
if (!wifiManager.autoConnect("ESP32-MQ135-BME280"))
{
  Serial.println("Failed to connect to WiFi and entered configuration mode.");
}
else
{
  Serial.println("Connected to WiFi!");
}
}
```

b. Récupération des valeurs:

Le code utilise les bibliothèques Adafruit pour interagir avec les capteurs. Les valeurs de température, pression, humidité et altitude du BME280 et la valeur de qualité de l'air du MQ135

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <ArduinoJson.h>
#include <WiFiManager.h>
#include <HTTPClient.h>

// Create an instance of the BME280 sensor
Adafruit_BME280 bme;

const int MQ135_ANALOG_PIN = A0; // Analog pin for MQ135
```

-Bibliothèques utilisées

```
void setup()
{
  // Set up Serial to print data
  Serial.begin(9600);

  // Set up BME280 sensor
  Wire.begin();
  if (!bme.begin(0x76))
  {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1)
      ;
  }
```

-Instanciation des Capteurs et Configuration Initiale

```
void loop()
{
  // Read BME280 sensor values
  float altitude = bme.readAltitude(1020);
  float temperatureBME = bme.readTemperature();
  float humidity = bme.readHumidity();
  float pressure = bme.readPressure() / 100.0;

  // Read MQ135 sensor value (example)
  int mq135AnalogReading = analogRead(MQ135_ANALOG_PIN) * 100 / 4095;

  // Calculate average temperature (using only BME280 temperature)
  float averageTemperature = temperatureBME;

  // Get MAC address
  String macAddress = WiFi.macAddress();
```

-Récupération des Valeurs des Capteurs

c. Envoi des Données via HTTP

Les données des capteurs sont stockées dans un objet JSON et envoyées via une requête HTTP POST vers le serveur symfony. Les valeurs de chaque capteur, ainsi que l'adresse MAC du dispositif, sont incluses dans le corps de la requête

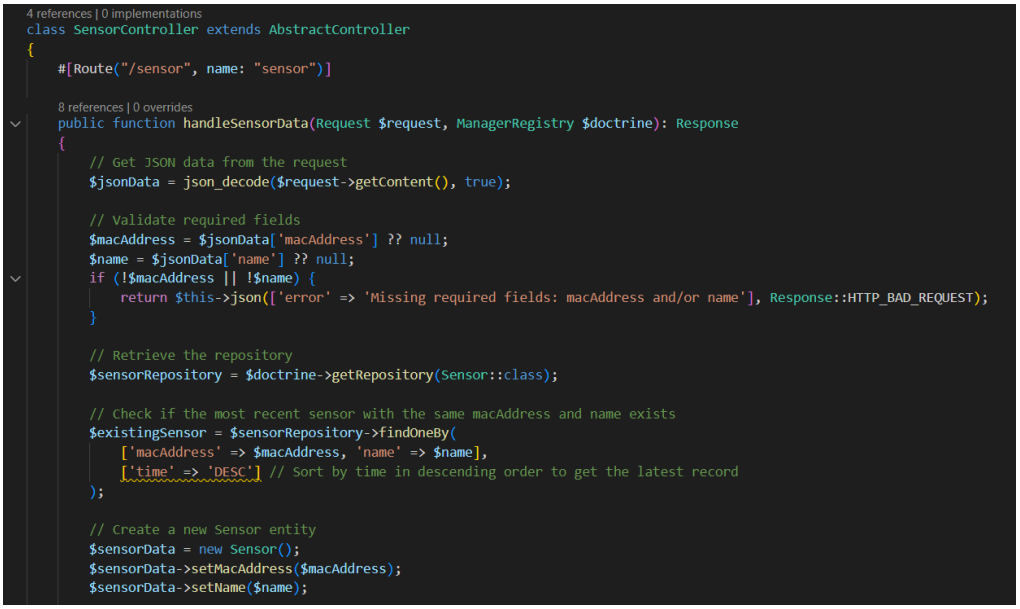
```
  // JSON data
  DynamicJsonDocument doc(200);

  doc["pressure"] = pressure;
  doc["humidity"] = humidity;
  doc["altitude"] = altitude;
  doc["airQuality"] = mq135AnalogReading;
  doc["temperature"] = averageTemperature;
  doc["macAddress"] = macAddress;
  doc["name"] = "station_2";

  String jsonString;
  serializeJson(doc, jsonString);
  Serial.println("Sensor Data:");
  Serial.println(jsonString);

  // Make HTTP POST request
  HTTPClient http;
  const char *serverUrl = "http://10.1.5.140:8000/sensor";
  http.begin(serverUrl);
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST(jsonString);

  if (httpResponseCode > 0)
  {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
```

```
    String response = http.getString();
    Serial.println("Response: " + response);
  }
  else
  {
    Serial.print("Error in HTTP request: ");
    Serial.println(httpResponseCode);
  }

  http.end();

  delay(60000); // Delay for 20 seconds before sending next request
}
```

3 .Serveur

a. Enregistrer les données du capteur

La fonction `handelSensorData` permet d'enregistrer les données d'un nouveau capteur dans la base de données. Elle commence par récupérer et valider les données JSON. Ensuite, elle vérifie si un capteur avec la même adresse MAC et le même nom existe déjà. Si tel est le cas, elle met à jour les données du capteur existant avec les nouvelles valeurs, sinon elle crée une nouvelle entrée pour ce capteur. Elle associe également l'utilisateur et le lieu au capteur si ces informations sont disponibles. Enfin, elle persiste les données dans la base de données.

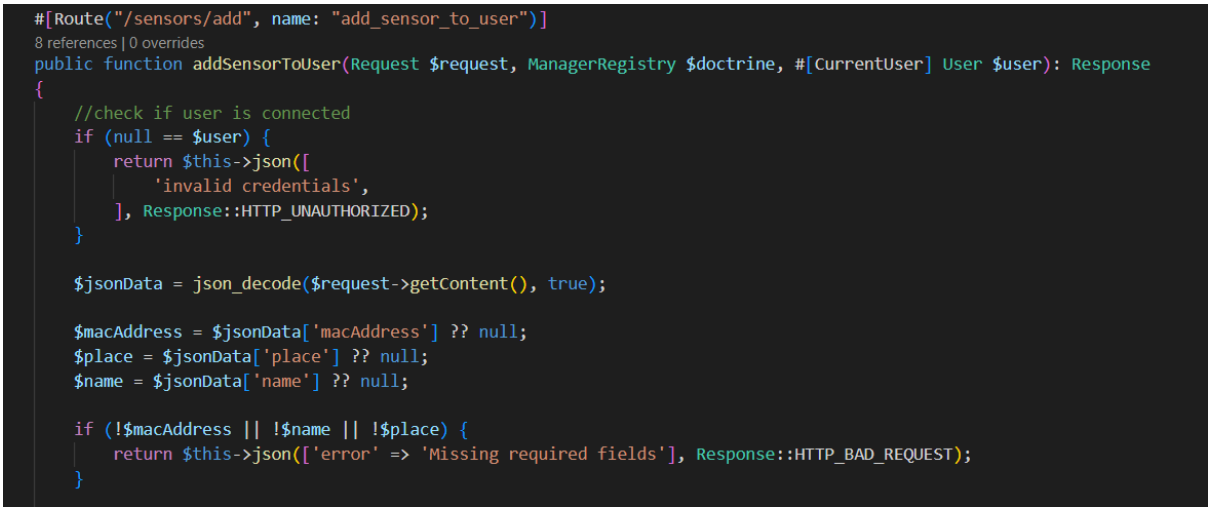


```
4 references | 0 implementations
class SensorController extends AbstractController
{
    #[Route("/sensor", name: "sensor")]

    8 references | 0 overrides
    public function handleSensorData(Request $request, ManagerRegistry $doctrine): Response
    {
        // Get JSON data from the request
        $jsonData = json_decode($request->getContent(), true);

        // Validate required fields
        $macAddress = $jsonData['macAddress'] ?? null;
        $name = $jsonData['name'] ?? null;
        if (!$macAddress || !$name) {
            return $this->json(['error' => 'Missing required fields: macAddress and/or name'], Response::HTTP_BAD_REQUEST);
        }

        // Retrieve the repository
        $sensorRepository = $doctrine->getRepository(Sensor::class);

        // Check if the most recent sensor with the same macAddress and name exists
        $existingSensor = $sensorRepository->findOneBy(
            ['macAddress' => $macAddress, 'name' => $name],
            ['time' => 'DESC'] // Sort by time in descending order to get the latest record
        );

        // Create a new Sensor entity
        $sensorData = new Sensor();
        $sensorData->setMacAddress($macAddress);
        $sensorData->setName($name);
```

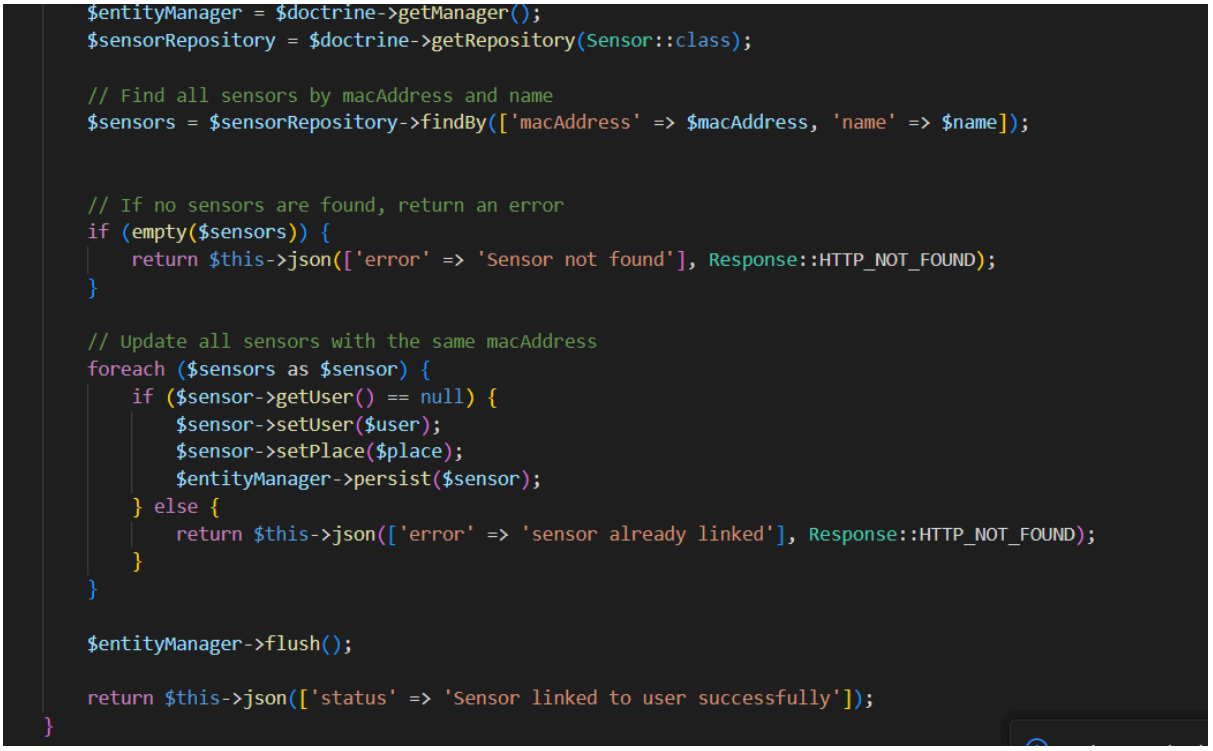
```php
        // Handle the JSON data
        $sensorData->setPressure($jsonData['pressure'] ?? null);
        $sensorData->setHumidity($jsonData['humidity'] ?? null);
        $sensorData->setAltitude($jsonData['altitude'] ?? null);
        $sensorData->setAirQuality($jsonData['airQuality'] ?? null);


        $currentTime = new \DateTime();
        $currentTime->modify('+2 hours');
        $sensorData->setTime($currentTime);

        $sensorData->setTemperature($jsonData['temperature'] ?? null);

        if ($existingSensor) {
            $user = $existingSensor->getUser();
            $place = $existingSensor->getPlace();

            // Check if user and place are not null before setting them
            if ($user !== null) {
                $sensorData->setUser($user);
            }

            if ($place !== null) {
                $sensorData->setPlace($place);
            }
        }

        // Use the Doctrine entity manager to persist and flush the data
        $entityManager = $doctrine->getManager();
        $entityManager->persist($sensorData);
        $entityManager->flush();

        // Return a response
        return $this->json(['status' => 'Sensor data saved successfully']);
    }
```

b. Ajouter un capteur à un utilisateur

La fonction `addSensorToUser` permet d'associer un capteur existant à l'utilisateur actuellement connecté. Elle récupère les données JSON de la requête pour obtenir l'adresse MAC du capteur, le nom du capteur et le lieu auquel il est associé. Si ces champs sont complets, elle recherche tous les capteurs correspondant à l'adresse MAC et au nom fournis , associe l'utilisateur et le lieu aux capteurs trouvés puis persister ces modifications dans la base de données



```php
#[Route("/sensors/add", name: "add_sensor_to_user")]
8 references | 0 overrides
public function addSensorToUser(Request $request, ManagerRegistry $doctrine, #[CurrentUser] User $user): Response
{
    //check if user is connected
    if (null == $user) {
        return $this->json([
            'invalid credentials',
        ], Response::HTTP_UNAUTHORIZED);
    }

    $jsonData = json_decode($request->getContent(), true);

    $macAddress = $jsonData['macAddress'] ?? null;
    $place = $jsonData['place'] ?? null;
    $name = $jsonData['name'] ?? null;

    if (!$macAddress || !$name || !$place) {
        return $this->json(['error' => 'Missing required fields'], Response::HTTP_BAD_REQUEST);
    }
```



```php
    $entityManager = $doctrine->getManager();
    $sensorRepository = $doctrine->getRepository(Sensor::class);

    // Find all sensors by macAddress and name
    $sensors = $sensorRepository->findBy(['macAddress' => $macAddress, 'name' => $name]);


    // If no sensors are found, return an error
    if (empty($sensors)) {
        return $this->json(['error' => 'Sensor not found'], Response::HTTP_NOT_FOUND);
    }

    // Update all sensors with the same macAddress
    foreach ($sensors as $sensor) {
        if ($sensor->getUser() == null) {
            $sensor->setUser($user);
            $sensor->setPlace($place);
            $entityManager->persist($sensor);
        } else {
            return $this->json(['error' => 'sensor already linked'], Response::HTTP_NOT_FOUND);
        }
    }

    $entityManager->flush();

    return $this->json(['status' => 'Sensor linked to user successfully']);
}
```

c. Dissocier les capteurs d'un utilisateur

La fonction `unlinkSensorsFromUser` permet de supprimer l'association d'un utilisateur avec tous les capteurs ayant une adresse MAC spécifique.Elle récupère l'adresse MAC du capteur à dissocier à partir des données JSON de la requête. Elle recherche ensuite tous les capteurs ayant cette adresse MAC dans la base de données. Pour chaque capteur trouvé, elle supprime l'association avec l'utilisateur et le lieu et persiste ces modifications dans la base de données

```php
#[Route('/sensors/delete', name: 'unlink_sensors_from_user', methods: ['DELETE'])]
8 references | 0 overrides
public function unlinkSensorsFromUser(Request $request, ManagerRegistry $doctrine, #[CurrentUser] User $user): Response
{
    //check if user is connected
    if (null == $user) {
        return $this->json([
            'invalid credentials',
        ], Response::HTTP_UNAUTHORIZED);
    }

    $jsonData = json_decode($request->getContent(), true);

    $macAddress = $jsonData['macAddress'] ?? null;

    if (!$macAddress) {
        return $this->json(['error' => 'Missing macAddress field'], Response::HTTP_BAD_REQUEST);
    }

    $entityManager = $doctrine->getManager();
    $sensorRepository = $entityManager->getRepository(Sensor::class);
```

```php
    // Find all sensors by macAddress
    $sensors = $sensorRepository->findBy(['macAddress' => $macAddress]);

    // If no sensors are found, return a not found response
    if (empty($sensors)) {
        return $this->json(['error' => 'No sensors found with the provided macAddress'], Response::HTTP_NOT_FOUND);
    }

    // Unlink the user from all found sensors and check that
    foreach ($sensors as $sensor) {

        $sensor->setUser(null);
        $sensor->setPlace(null);
        $entityManager->persist($sensor);
    }
    $entityManager->flush();

    return $this->json(['message' => 'User unlinked from all sensors successfully']);
}
```

d. Obtenir les capteurs de l'utilisateur

La fonction `getUserSensors` récupère tous les capteurs associés à l'utilisateur actuellement connecté. Elle commence par vérifier si l'utilisateur est authentifié. Ensuite, elle interroge le référentiel des capteurs pour obtenir tous les capteurs associés à cet utilisateur puis retourne ces données structurées sous forme de réponse HTTP JSON.

```php
#[Route("/user/sensors", name: "user_sensors", methods: ["GET"])]
8 references | 0 overrides
public function getUserSensors(ManagerRegistry $doctrine, #[CurrentUser] User $user): Response
{
    if (null == $user) {
        return $this->json([
            'error' => 'Invalid credentials',
        ], Response::HTTP_UNAUTHORIZED);
    }

    $sensorRepository = $doctrine->getRepository(Sensor::class);
    $userSensors = $sensorRepository->findBy(['user' => $user]);

    $groupedSensors = [];
    foreach ($userSensors as $sensor) {
        $key = $sensor->getMacAddress() . '_' . $sensor->getName();
        if (!isset($groupedSensors[$key])) {
            $groupedSensors[$key] = [
                'macAddress' => $sensor->getMacAddress(),
                'name' => $sensor->getName(),
                'place' => $sensor->getPlace(),

            ];
        }
    }

    return $this->json(['sensorData' => array_values($groupedSensors)]);
}
```

f. Détails du dernier enregistrement du capteur

La fonction `getLastSensorDetails` récupère les détails du dernier enregistrement d'un capteur spécifique pour l'utilisateur actuellement connecté. Elle commence par vérifier si l'utilisateur est authentifié. Ensuite, elle récupère l'adresse MAC et le nom du capteur à partir des paramètres de requête. En interrogeant la base de données, elle récupère les détails du dernier enregistrement de ce capteur

```php
#[Route("/sensors/details/last", name: "sensor_details_last")]
8 references | 0 overrides
public function getLastSensorDetails(Request $request, ManagerRegistry $doctrine, #[CurrentUser] User $user): Response
{
    if (null == $user) {
        return $this->json([
            'error' => 'Invalid credentials',
        ], Response::HTTP_UNAUTHORIZED);
    }
    $macAddress = $request->query->get('macAddress');
    $name = $request->query->get('name');

    if (!$macAddress || !$name) {
        return $this->json(['error' => 'Missing macAddress or name'], Response::HTTP_BAD_REQUEST);
    }

    // Get the current user
    $currentUser = $this->getUser();

    // Retrieve the sensor from the database
    $sensorRepository = $doctrine->getRepository(Sensor::class);
    $sensor = $sensorRepository->findOneBy(
        ['macAddress' => $macAddress, 'name' => $name],
        ['time' => 'DESC'] // Sort by time in descending order to get the latest record
    );
```

```php
    // If sensor not found, return 404 Not Found
    if (!$sensor) {
        return $this->json(['error' => 'Sensor not found'], Response::HTTP_NOT_FOUND);
    }

    // Check if the current user is authorized to access this sensor
    if ($sensor->getUser() !== $currentUser) {
        return $this->json([
            'error' => 'unothorized',
        ], Response::HTTP_UNAUTHORIZED);
    }

    // Serialize sensor data for response
    $sensorData = [

        'pressure' => $sensor->getPressure(),
        'humidity' => $sensor->getHumidity(),
        'altitude' => $sensor->getAltitude(),
        'airQuality' => $sensor->getAirQuality(),
        'temperature' => $sensor->getTemperature(),
        'time' => $sensor->getTime() ? $sensor->getTime()->format('Y-m-d H:i:s') : null,
        // Add more fields as needed
    ];

    return $this->json($sensorData);
}
```

g. Détails de tous les enregistrements des capteurs

La fonction `getAllSensorDetails` récupère tous les détails des enregistrements de capteurs pour l'utilisateur actuellement connecté et une adresse MAC spécifique. Elle interroge le référentiel des capteurs pour récupérer tous les enregistrements de capteurs associés à cet utilisateur et à l'adresse MAC spécifiée. Elle structure les données des capteurspuis retourne ces détails sous forme de réponse HTTP JSON.

```php
#[Route("/sensors/details/all", name: "sensor_details_all")]
6 references | 0 overrides
public function getAllSensorDetails(Request $request, ManagerRegistry $doctrine, #[CurrentUser] User $user): Response
{
    $macAddress = $request->query->get('macAddress');
    $name = $request->query->get('name');
    if (null == $user) {
        return $this->json([
            'error' => 'Invalid credentials',
        ], Response::HTTP_UNAUTHORIZED);
    }

    // Retrieve the sensor repository
    $sensorRepository = $doctrine->getRepository(Sensor::class);

    // Retrieve all sensors belonging to the current user
    $userSensors = $sensorRepository->findBy(
        ['user' => $user, 'macAddress' => $macAddress],
        ['time' => 'DESC']
    );
```

```php
        // If no sensors found for the user, return an appropriate response
        if (empty($userSensors)) {
            return $this->json(['message' => 'No sensors found for the user'], Response::HTTP_NOT_FOUND);
        }

        // Initialize an array to store all sensor details
        $sensorDetails = [];

        // Iterate through each sensor to extract details
        foreach ($userSensors as $sensor) {
            $sensorData = [

                'pressure' => $sensor->getPressure(),
                'humidity' => $sensor->getHumidity(),
                'altitude' => $sensor->getAltitude(),
                'airQuality' => $sensor->getAirQuality(),
                'temperature' => $sensor->getTemperature(),
                'time' => $sensor->getTime() ? $sensor->getTime()->format('Y-m-d H:i:s') : null,
                // Add more fields as needed
            ];

            // Push sensor data into the sensorDetails array
            $sensorDetails[] = $sensorData;
        }

        // Return JSON response with all sensor details
        return $this->json(['sensorDetails' => $sensorDetails]);
    }
}
```

h. Gestion des utilisateurs

Pour gérer les comptes utilisateurs, nous avons créé un objet utilisateur :

```php
#[ORM\Entity(repositoryClass: UserRepository::class)]
52 references | 1 implementation
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    2 references
    private $passwordHasher;

    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
```

Celui-ci est ensuite utilisé dans les fichiers SecurityController et UserController. SecurityController sert à connecter un utilisateur déjà

inscrit ainsi qu'à se déconnecter :

```php
class SecurityController extends AbstractController
{
    #[Route("/login", name: "app_login")]
    8 references | 0 overrides
    public function login(#[CurrentUser] User $user): Response
    {
        if (null == $user) {
            return $this->json([
                'invalid credentials',
            ], Response::HTTP_UNAUTHORIZED);
            # code...
        }
        return $this->json([

            "name" => $user->getName(),
            "email" => $user->getEmail(),
            "roles" => $user->getRoles()
        ]);
    }
    #[Route("logout", name: "app_logout")]
    4 references | 0 overrides
    public function logout()
    {
    }
}
```

UserController permet quant à lui de créer un utilisateur, mais aussi de vérifier que le compte est déjà existant ainsi qu'a lister les utilisateurs :

```php
class UserController extends AbstractController
{
    //function to verify user email for firest react registration form
    #[Route('/users/verify', name: 'users/verify', methods: "Post")]
    8 references | 0 overrides
    public function verifyUser(Request $request, ManagerRegistry $doctrine): Response
    {
        $repository = $doctrine->getRepository(User::class);

        // Get data from the request
        $data = json_decode($request->getContent(), true);

        // Check if the email already exists
        $existingUser = $repository->findOneBy(['email' => $data['email']]);
        if ($existingUser) {
            return $this->json(['Email already exists'], Response::HTTP_CONFLICT);
        }

        return $this->json(["user doesnt not exist"]);
    }
```

```php
#[Route('/users', name: "add_users", methods: "Post")]
8 references | 0 overrides
public function addUser(Request $request, ManagerRegistry $doctrine, UserPasswordHasherInterface $passwordHasher): Response
{
    $repository = $doctrine->getRepository(User::class);
    //get data from user
    $data = json_decode($request->getContent(), true);
    // Check if the email already exists
    $existingUser = $repository->findOneBy(['email' => $data['email']]);
    if ($existingUser) {
        return $this->json(['Email already exists'], Response::HTTP_CONFLICT);
    }
    $user = new User($passwordHasher);
    $user->setEmail($data['email']);
    $user->setName($data['name']);
    $user->setPassword($data['password']);
    $user->setRoles(["client"]);
    $entityManger = $doctrine->getManager();
    $entityManger->persist($user);
    $entityManger->flush();

    return $this->json(['user added succesfully ']);
}
```

```php
#[Route("/users", name: "get_users", methods: "Get")]
8 references | 0 overrides
public function getUsers(Request $request, ManagerRegistry $doctrine): Response
{
    $repository = $doctrine->getRepository(User::class);
    $users = $repository->findAll();
    return $this->json($users);
}

#[Route("/user", name: "get_user", methods: "GET")]
8 references | 0 overrides
public function getUserData(#[CurrentUser] User $user): JsonResponse
{
    if (null == $user) {
        return $this->json([
            'invalid credentials',
        ], Response::HTTP_UNAUTHORIZED);
        # code...
    }

    // Serialize user data to send back in the response
    $userData = [
        'id' => $user->getId(),
        'email' => $user->getEmail(),
        'name' => $user->getName(),
        'roles' => $user->getRoles(), // Assuming roles are stored as an array
        // Include any other relevant user information
    ];

    return $this->json($userData);
}
}
```

Enfin cette fonction permet de récupérer les information de l’utilisateur connecté :

```php
    #[Route("/user", name: "get_user", methods: "GET")]
    8 references | 0 overrides
    public function getUserData(#[CurrentUser] User $user): JsonResponse
    {
        if (null == $user) {
            return $this->json([
                'invalid credentials',
            ], Response::HTTP_UNAUTHORIZED);
            # code...
        }

        // Serialize user data to send back in the response
        $userData = [
            'id' => $user->getId(),
            'email' => $user->getEmail(),
            'name' => $user->getName(),
            'roles' => $user->getRoles(), // Assuming roles are stored as an array
            // Include any other relevant user information
        ];

        return $this->json($userData);
    }
}
```

4 .Base des données

L'entité Symfony Sensor est créée pour stocker les informations dans la base de données. Elle comporte des attributs tels que pression, humidité, altitude, qualité de l'air, date et heure, température, emplacement, et adresse MAC du capteur. Ces attributs correspondent aux données collectées par les capteurs.

```php
#[ORM\Entity(repositoryClass: SensorRepository::class)]
27 references | 0 implementations
class Sensor
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]

    2 references
    private ?int $id = null;
    #[ORM\ManyToOne(targetEntity: "App\Entity\User", inversedBy: "sensors")]

    2 references
    private ?User $user = null; // Update to allow null

    #[ORM\Column(type: "float", nullable: true)]

    2 references
    private $pressure;

    #[ORM\Column(type: "float", nullable: true)]
    2 references
    private $humidity;

    #[ORM\Column(type: "float", nullable: true)]
    2 references
    private $altitude;

    #[ORM\Column(type: "integer", nullable: true)]
    2 references
    private $airQuality;
```

```php
    #[ORM\Column(type: "datetime", nullable: true)]
    2 references
    private $time;

    #[ORM\Column(type: "float", nullable: true)]
    2 references
    private $temperature;
    #[ORM\Column(type: "string", nullable: true)]
    2 references
    private $place;
    #[ORM\Column(type: "string", nullable: true)]
    2 references
    private $macAddress;
    #[ORM\Column(type: "string", nullable: true)]
    2 references
    private $name;
```

L'entité Symfony User est créée pour stocker les informations des utilisateurs dans la base de données. Elle comporte des attributs tels que le nom, l'adresse e-mail, les rôles, le mot de passe, et les capteurs associés à l'utilisateur. Ces attributs permettent de gérer les informations d'identification et les autorisations des utilisateurs, ainsi que de lier chaque utilisateur aux données collectées par leurs capteurs respectifs.

```
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]


Code Suggestions

    private ?int $id = null;
    #[ORM\OneToMany(targetEntity: "App\Entity\Sensor", mappedBy: "user")]
    private Collection $sensors;
    public function __construct(UserPasswordHasher $passwordHasher)
    {
        $this->sensors = new ArrayCollection();
        $this->passwordHasher = $passwordHasher;
    }
    #[Orm\Column(length: 255, unique: true)]
    private ?string $email = null;


    #[ORM\Column(length: 55)]
    private ?string $name = null;

    #[ORM\Column]
    private array $roles = [];

    /**
     * @var string The hashed password
     */
    #[ORM\Column]
    private ?string $password = null;
```

Schéma de la base de données

```
1 • SELECT * FROM `station-meteo`.sensor;
```

id	user_id	pressure	humidity	altitude	air_quality	time	temperature	place	mac_address	name
321	2	996.9874878	39.37988281	192.0896759	1	2024-07-03 14:35:49	25	Salle video	B4:E6:2D:D9:08:55	station_1
322	2	996.9498291	39.359375	192.4067383	1	2024-07-03 14:36:50	25.03000069	Salle video	B4:E6:2D:D9:08:55	station_1
323	2	996.9007568	39.10742188	192.8201141	1	2024-07-03 14:37:51	25.12999916	Salle video	B4:E6:2D:D9:08:55	station_1
324	2	996.9046021	38.90332031	192.7878418	1	2024-07-03 14:38:52	25.12000084	Salle video	B4:E6:2D:D9:08:55	station_1
325	2	996.9198608	38.86132813	192.6592712	1	2024-07-03 14:39:53	25.04000092	Salle video	B4:E6:2D:D9:08:55	station_1
326	2	996.8723145	38.453125	193.059845	1	2024-07-03 14:40:54	25.03000069	Salle video	B4:E6:2D:D9:08:55	station_1
327	2	996.8804932	38.59082031	192.9906921	1	2024-07-03 14:41:55	25.04999924	Salle video	B4:E6:2D:D9:08:55	station_1
328	1	995.111084	38.93066406	207.9098206	100	2024-07-03 15:50:35	25.44000053	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
329	1	995.114502	37.89941406	207.8811035	100	2024-07-03 15:51:36	25.59000015	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
330	1	995.1468506	37.74414063	207.6082001	100	2024-07-03 15:52:37	25.62999916	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
331	1	995.1004639	37.67675781	207.9996033	100	2024-07-03 15:53:38	25.63999939	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
332	1	995.1300659	38.22851563	207.7497711	100	2024-07-03 15:54:39	25.71999931	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
333	1	995.0906372	37.51953125	208.0821838	100	2024-07-03 15:55:40	25.79999924	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
334	1	995.062561	37.56152344	208.3191986	100	2024-07-03 15:56:41	25.75	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
335	1	994.991272	37.58300781	208.9209595	100	2024-07-03 15:57:42	25.72999954	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
336	1	994.9466553	37.63671875	209.2975464	100	2024-07-03 15:58:43	25.70999908	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
337	1	994.9868774	37.93847656	208.957901	100	2024-07-03 15:59:44	25.78000069	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
338	1	994.9996948	37.31738281	208.8496552	100	2024-07-03 16:00:45	25.82999992	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
339	1	995.0101318	37.24414063	208.7619324	100	2024-07-03 16:01:46	25.88999939	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
340	1	995.0253296	37.17578125	208.633667	100	2024-07-03 16:02:47	25.88999939	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
341	1	996.9043579	41.47460938	192.7898865	100	2024-07-04 13:14:42	27.36000061	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
342	1	996.8373413	39.27246094	193.3543854	100	2024-07-04 13:15:43	26.89999962	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
343	1	996.8494263	39.06933594	193.2524414	100	2024-07-04 13:16:44	26.64999962	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
344	1	996.7876587	39.78710938	193.7729187	100	2024-07-04 13:17:45	26.54000092	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
345	1	996.7518921	40.23828125	194.0746613	100	2024-07-04 13:18:46	26.5	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
346	1	996.8671875	40.25292969	193.1028748	100	2024-07-04 13:19:47	26.54000092	Impression 3d	D4:8A:FC:A4:E1:B8	station_2
347	1	996.80896	40.96386719	193.5936127	100	2024-07-04 13:20:48	26.61000061	Impression 3d	D4:8A:FC:A4:E1:B8	station_2

```
1 • SELECT * FROM `station-meteo`.user;
```

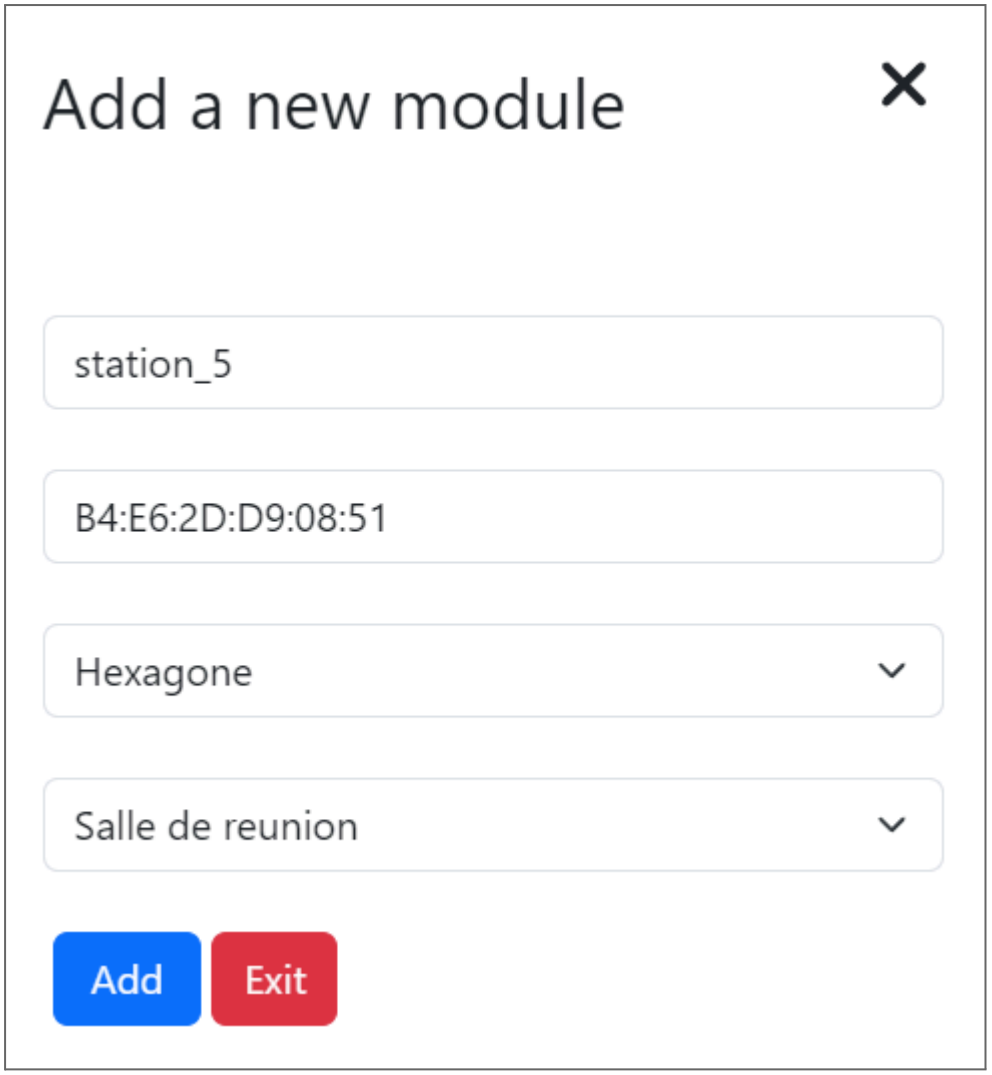
id	email	name	roles	password
1	mohamedmaqhzaoui53@gmail.com	med	["client"]	$2y$13$tZF7./sJ1/B94zPVBkoGTeeV...
2	luk@gmail.com	luk	["client"]	$2y$13$5YOqYrMX3WolaWACq3TZI...
3	loukas.poirier@ecole-hexagone.com	Loukas	["client"]	$2y$13$vwVlDwoOIHzq35Hlouj/veI...
4	wow@gmail.com	test	["client"]	$2y$13$Xk9iypvd3ujoPHU829LWSu...

5 .Interface Graphique

a. Formulaire de Capteur

Le composant `SensorForm` permet à l'utilisateur d'ajouter un nouveau capteur. Il utilise le hook `useForm` pour gérer le formulaire et `yup` pour valider les entrées de l'utilisateur, comme le nom, l'adresse MAC et l'emplacement. Les données sont envoyées au serveur Symfony, et l'utilisateur peut sélectionner une entreprise et un emplacement associé.

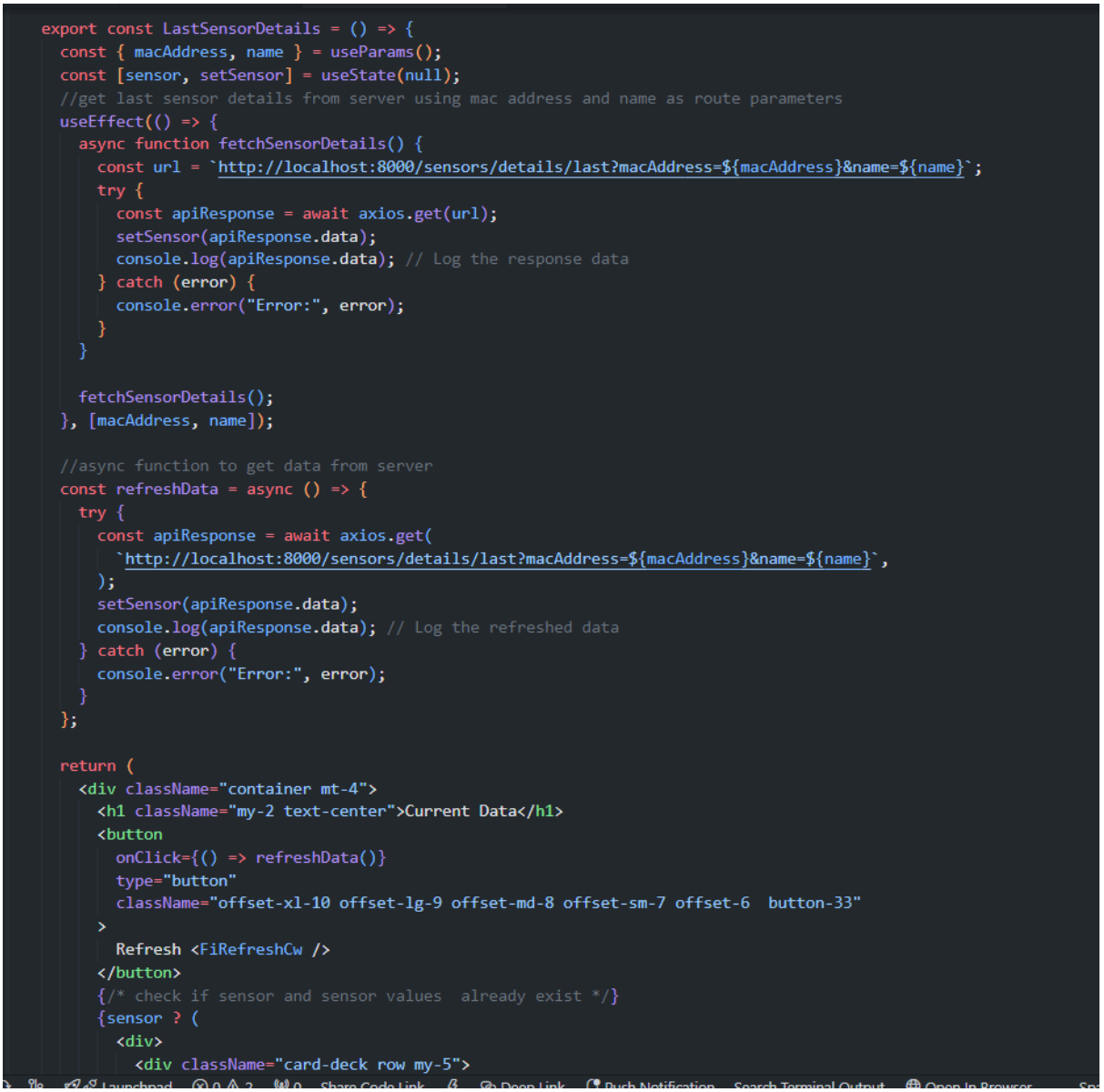
```
export const SensorForm = (props) => {
  const [apiError, setApiError] = useState("");
  const [company, setCompany] = useState(2); // state for company 2=> no company selected and 1=> hexagone
  const moduleSchema = yup.object().shape({
    name: yup.string().required("name is required"),
    //mac address format
    macAddress: yup
      .string()
      .required("MAC address is required")
      .matches(
        /^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$/,
        "Invalid MAC address",
      ),
    place: yup.string().required("place is required"),
  });
  //use the useForm hook for sending data
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({
    resolver: yupResolver(moduleSchema),
  });
  //function to exit form and do animation
  const exitForm = () => {
    setFormClass("hidden");
    setTimeout(() => props.setForm("hidden"), 1000);
  };
  //yup schema for cliend validation

  //function to send data to symfony server
  const submitData = async (sensorData) => {
    try {
      let url = "http://localhost:8000/sensors/add";
      setApiError(
        <div class="spinner-grow text-primary" role="status">
          <span class="visually-hidden">Loading...</span>
        </div>,
      );
      const response = await axios.post(url, sensorData);
      console.log(response);
      exitForm();
      window.location.reload();
```



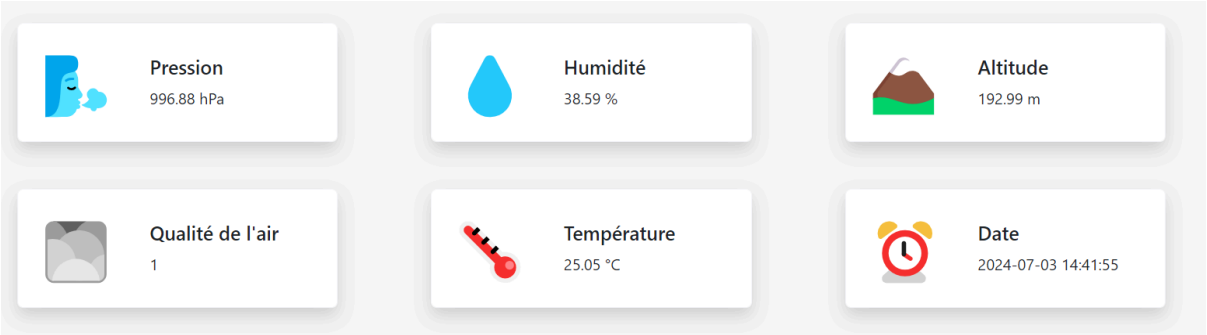
Formulaire pour ajouter un module

b. Détails du Dernier Capteur

Le composant `LastSensorDetails` récupère et affiche les dernières données d'un capteur spécifique, telles que la pression, l'humidité, l'altitude, la qualité de l'air, la température et l'heure. Il permet également de rafraîchir les données en les récupérant à nouveau depuis le serveur.



```jsx
export const LastSensorDetails = () => {
  const { macAddress, name } = useParams();
  const [sensor, setSensor] = useState(null);
  //get last sensor details from server using mac address and name as route parameters
  useEffect(() => {
    async function fetchSensorDetails() {
      const url = `http://localhost:8000/sensors/details/last?macAddress=${macAddress}&name=${name}`;
      try {
        const apiResponse = await axios.get(url);
        setSensor(apiResponse.data);
        console.log(apiResponse.data); // Log the response data
      } catch (error) {
        console.error("Error:", error);
      }
    }

    fetchSensorDetails();
  }, [macAddress, name]);

  //async function to get data from server
  const refreshData = async () => {
    try {
      const apiResponse = await axios.get(
        `http://localhost:8000/sensors/details/last?macAddress=${macAddress}&name=${name}`,
      );
      setSensor(apiResponse.data);
      console.log(apiResponse.data); // Log the refreshed data
    } catch (error) {
      console.error("Error:", error);
    }
  };

  return (
    <div className="container mt-4">
      <h1 className="my-2 text-center">Current Data</h1>
      <button
        onClick={() => refreshData()}
        type="button"
        className="offset-xl-10 offset-lg-9 offset-md-8 offset-sm-7 offset-6  button-33"
      >
        Refresh <FiRefreshCw />
      </button>
      {/* check if sensor and sensor values  already exist */}
      {sensor ? (
        <div>
          <div className="card-deck row my-5">
```

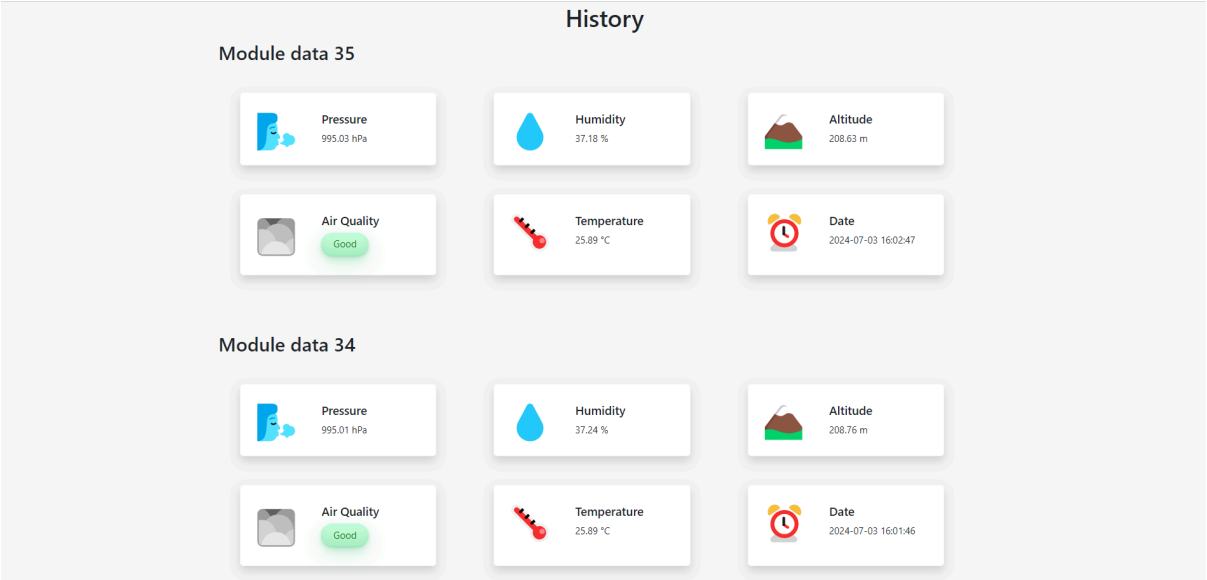


Affichage des dernières données

c. Historique des Données de Capteur

Le composant `SensorDetails` affiche l'historique complet des données pour un capteur donné, en récupérant toutes les valeurs stockées dans le serveur Symfony. Les informations affichées incluent la pression, l'humidité, l'altitude, la qualité de l'air, la température et l'heure pour chaque enregistrement.

```
export const SensorDetails = () => {
  const { macAddress, name } = useParams();
  const [sensorDetails, setSensorDetails] = useState([]);
  // get all sensor details from symfony server using mac address and name as parameters
  useEffect(() => {
    async function fetchAllSensorDetails() {
      const url = `http://localhost:8000/sensors/details/all/?macAddress=${macAddress}&name=${name}`;
      try {
        const apiResponse = await axios.get(url);
        setSensorDetails(apiResponse.data.sensorDetails);
        console.log(apiResponse.data.sensorDetails); // Log the response data
      } catch (error) {
        console.error("Error:", error);
      }
    }

    fetchAllSensorDetails();
  }, [macAddress, name]);
  // get last index to use it in module number
  const lastIndex = sensorDetails.length;

  return (
    <div className="container mt-4">
      <h1 className="text-center">History</h1>
      {sensorDetails.length > 0 ? (
        <div>
          {sensorDetails.map((sensor, index) => (
            <div key={index} className="mt-3">
              <h2>Module data {lastIndex - index}</h2>
              <div className="card-deck row my-5">
                {/* Pressure */}
                {sensor.pressure !== null && sensor.pressure !== undefined && (
                  <div className="card border-light shadow p-3 mb-5 bg-body rounded col-xl-3 col-lg-4 col-md-5 col-sm-5 col-9 mx-xl-5 mx-lg-5 mx-md-4 mx-3">
                    <div className="d-flex align-items-center">
                      <span className="display-3 me-3">🌬️</span>
                      <div className="card-body">
                        <h5 className="card-title">Pressure</h5>
                        <p className="card-text">
                          {sensor.pressure.toFixed(2)} hPa
                        </p>
                      </div>
                    </div>
                  </div>
                )}
```

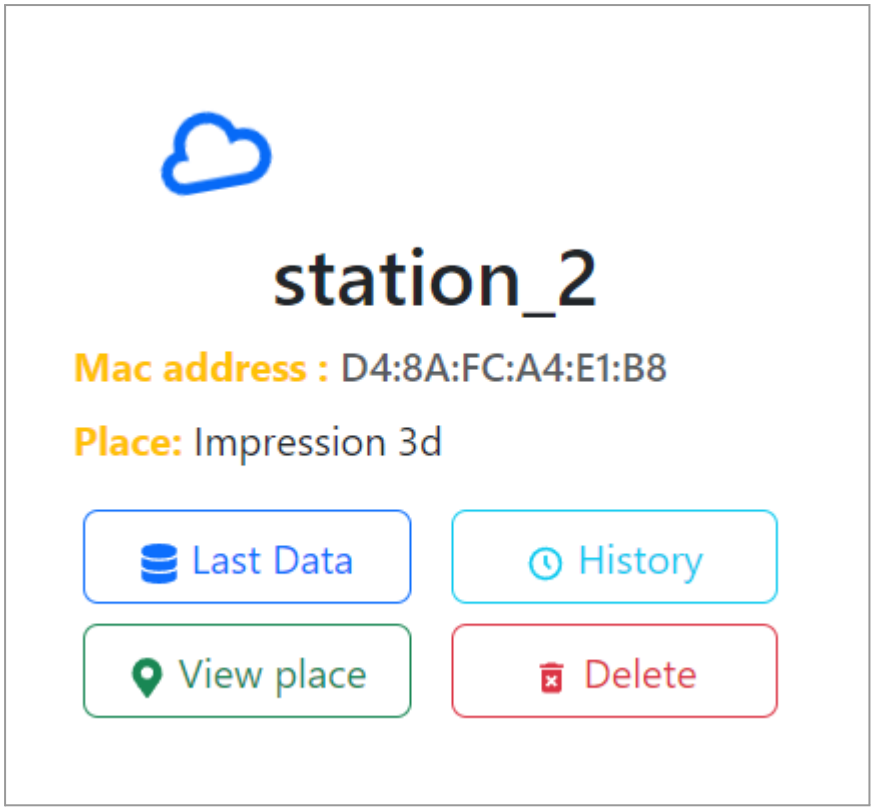


Affichage des tous les données

d. Gestion de la Suppression de Capteur

La fonction `unlinkSensor` est une fonction asynchrone qui gère la suppression d'un capteur en envoyant une requête HTTP DELETE

```
}, []);

async function unlinkSensor(macAddress) {
  setActionStatus("deleting"); // Set action status to deleting
  const url = "http://localhost:8000/sensors/delete";
  try {
    const response = await axios.delete(url, {
      data: { macAddress },
    });
    console.log(response.data); // Log the response data
    fetchData();
  } catch (err) {
    console.log(err);
  } finally {
    setActionStatus("idle"); // Reset action status to idle
  }
}
```



Gestion des module

Lorsqu'un utilisateur est connecté, il peut facilement améliorer la sécurité en ajoutant un module avec la spécification de son emplacement via l'adresse MAC qui serait intégrée dans l'impression 3D de chaque module. Cela garantit que chaque utilisateur ne peut visualiser que les données de son propre module