

PROJET DE Développement Personnel

Conception d'un système de surveillance de la qualité de l'air



ENCADRÉ PAR :

- Pr ENNOUAARY Abdeslam



REALISÉ PAR :

- LAAZIZ Fatima Zahrae
- OUSSEINI HASSANE Malika
- TAMLALTI Maryam

Table des matières

I.	RESUME.....	4
II.	ABSTRACT	4
III.	PROBLEMATIQUE.....	5
IV.	SOLUTION PROPOSEE	5
V.	TECHNOLOGIES UTILISEES	5
A.	FRONTEND	5
B.	BACKEND	6
C.	BASE DE DONNEES.....	7
VI.	DESCRIPTION DU SYSTEME IOT.....	7
A.	LES COMPOSANTS NECESSAIRES.....	7
1.	<i>Le module capteur MQ-2.....</i>	7
2.	<i>La carte NodeNCU ESP8266</i>	8
3.	<i>Le capteur DHT11</i>	9
4.	<i>Breadboard.....</i>	9
B.	MONTAGE DU SYSTEME.....	10
C.	CODE	10
VII.	FONCTIONNALITES.....	13
A.	LE DASHBOARD.....	13
1.	<i>Fonctionnalités de la page.....</i>	14
B.	LE BREAKDOWN.....	22
C.	GESTION DES ADMINS.....	24
D.	LA PAGE DE RECOMMANDATION	27
E.	SUPPORT ET AIDE	28
VIII.	LIEN GITHUB.....	29
IX.	CONCLUSION	29

INTRODUCTION

L'importance de la qualité de l'air dans les environnements hospitaliers ne peut être sous-estimée. La santé et le bien-être des patients, ainsi que du personnel médical, dépendent en grande partie de l'air qu'ils respirent. Cependant, la surveillance et la capture de données liées à la qualité de l'air dans un hôpital peuvent s'avérer complexes et cruciales pour garantir un environnement sûr et sain. La conception et la mise en œuvre d'une solution efficace pour surveiller et capturer ces données représentent un défi de taille. Il est essentiel de développer une approche intégrée qui permette de collecter en temps réel des informations précises sur la qualité de l'air, notamment en ce qui concerne l'humidité, la température et la présence de substances potentiellement dangereuses.

Une surveillance constante permettrait de détecter les dépassements de seuils et de déclencher des alertes pour une intervention immédiate, réduisant ainsi les risques de complications ou de maladies associées à une mauvaise qualité de l'air. Dans cette étude, nous aborderons les différentes étapes de la conception et de la mise en œuvre d'une telle solution, en mettant l'accent sur les technologies et les méthodologies les plus appropriées. Nous explorerons également les avantages potentiels d'une surveillance continue de la qualité de l'air dans un environnement hospitalier, tant du point de vue de la sécurité des patients que de l'efficacité des opérations médicales.



I. Résumé :

Ce rapport présente la conception et la réalisation d'une plateforme web intégrée à un système IoT destiné à surveiller et capturer des données liées à la qualité de l'air dans un hôpital. La plateforme et le système sont spécifiquement conçus pour être installés au sein de l'hôpital afin de collecter des informations essentielles telles que l'humidité, la température et la présence de gaz tels que le butane, le propane, le méthane, les émanations d'alcool et l'hydrogène.

L'objectif principal de cette plateforme est de détecter les dépassements de seuils de ces données afin de prévenir les risques d'incendie et d'autres problèmes potentiels liés à la qualité de l'air. Lorsque les seuils prédéfinis sont dépassés, un système de notification est automatiquement déclenché, alertant ainsi le personnel hospitalier concerné pour prendre des mesures immédiates.

La plateforme web offre une interface conviviale permettant aux utilisateurs autorisés d'accéder aux données en temps réel. Elle présente des visualisations claires, telles que des graphiques et des tableaux, pour faciliter l'interprétation des données et surveiller efficacement la qualité de l'air dans différentes zones de l'hôpital. De plus, une page d'instructions détaillées est mise à disposition pour guider le personnel sur les mesures à prendre en cas de dépassement des seuils.

La solution repose sur un réseau IoT comprenant des capteurs spécifiques installés dans différentes zones de l'hôpital pour collecter les données. L'intégration du système IoT avec la plateforme permet une surveillance en temps réel, une analyse des données historiques et une prise de décision proactive pour garantir un environnement sûr et sain pour les patients, le personnel médical et les installations hospitalières.

II. Abstract :

This report presents the design and implementation of a web platform integrated with an IoT system aimed at monitoring and capturing air quality data in a hospital. The platform and system are specifically designed to be installed within the hospital to collect crucial information such as humidity, temperature, and the presence of gases such as butane, propane, methane, alcohol vapors, and hydrogen.

The main objective of this platform is to detect threshold exceedances of these data to prevent fire hazards and other potential issues related to air quality. When predefined thresholds are exceeded, a notification system is automatically triggered, alerting the relevant hospital staff to take immediate action.

The web platform provides a user-friendly interface that allows authorized users to access real-time data. It presents clear visualizations, such as graphs and tables, to facilitate data interpretation and effectively monitor air quality in different areas of the hospital. Additionally,

a detailed instruction page is provided to guide staff on the necessary measures to be taken in case of threshold exceedances.

The solution relies on an IoT network consisting of specific sensors installed in different areas of the hospital to collect data. The integration of the IoT system with the platform enables real-time monitoring, historical data analysis, and proactive decision-making to ensure a safe and healthy environment for patients, medical staff, and hospital facilities.

III. Problématique :

Comment concevoir et mettre en œuvre une solution pour surveiller et capturer des données liées à la qualité de l'air dans un hôpital, afin d'améliorer la sécurité et le bien-être des patients et du personnel médical ?

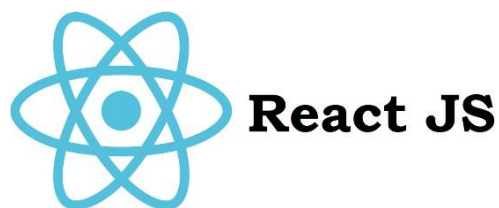
IV. Solution proposée :

Notre projet « **BESAFE** » représente la solution à la problématique de surveillance et de captation des données liées à la qualité de l'air dans un hôpital. En concevant et en réalisant une plateforme web intégrée à un système IoT, nous offrons aux professionnels de la santé un outil puissant pour surveiller en temps réel les paramètres de la qualité de l'air, tels que la température, l'humidité et les niveaux de contaminants. Cette solution leur permettra de prendre des mesures préventives, d'identifier rapidement les problèmes potentiels et de mettre en place des actions correctives pour garantir un environnement sain et sûr pour les patients et le personnel médical. Grâce à notre plateforme intuitive, les données sont visualisées de manière claire et compréhensible, offrant ainsi une base solide pour une prise de décision éclairée et une amélioration continue de la qualité de l'air dans l'hôpital.

V. Technologies utilisées :

A. Frontend :

- **React JS**



React.js est une bibliothèque JavaScript open-source utilisée pour la création d'interfaces utilisateur interactives et réactives. Elle a été développée par Facebook et est largement utilisée dans le développement web moderne. React utilise une approche basée sur les composants, ce qui signifie que vous pouvez créer des morceaux réutilisables de code appelés "composants" qui encapsulent le comportement et l'apparence d'une partie spécifique de votre interface utilisateur. Ces composants peuvent être imbriqués les uns dans les autres pour construire des interfaces complexes.

- **CSS et Javascript**



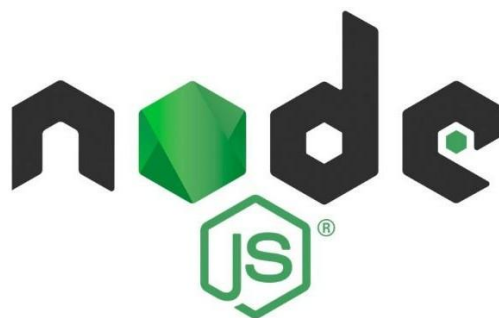
CSS (Cascading Style Sheets) est un langage de feuille de style utilisé pour décrire la présentation d'un document écrit en HTML ou XML (y compris les dialectes XML tels que SVG, MathML ou XHTML). CSS décrit comment les éléments doivent être rendus à l'écran, sur papier, dans la parole ou sur d'autres supports. Il permet de contrôler les couleurs, les polices, les marges, les espacements, les bordures, les animations.

JavaScript est un langage de programmation léger, interprété ou compilé juste à temps. Bien qu'il soit le plus connu comme langage de script pour les pages Web, de nombreux environnements autres que les navigateurs l'utilisent également, tels que Node.js, Apache CouchDB et Adobe Acrobat. JavaScript est un langage dynamique basé sur des prototypes, multi-paradigmes, à thread unique, prenant en charge les styles orientés objet, impératifs et déclaratifs (par exemple, la programmation fonctionnelle).

B. Backend :

- **Node.js**

Node.js est un environnement d'exécution JavaScript côté serveur, basé sur le moteur JavaScript V8 de Google Chrome. Il permet d'exécuter du code JavaScript en dehors du navigateur, ce qui ouvre de nombreuses possibilités pour développer des applications web, des API, des outils en ligne de commande, des serveurs et bien plus encore.



- **Express.js**

Express.js est un Framework web rapide, minimaliste et flexible pour Node.js. Il est conçu pour construire des applications web et des API de manière simple et efficace. Express.js offre une multitude de fonctionnalités et de middleware qui facilitent le développement d'applications web. Il fournit des routes pour gérer les différentes requêtes HTTP (GET, POST, PUT, DELETE, etc.), permettant ainsi de définir facilement les points d'entrée de l'application. Il permet également de gérer les paramètres de requête, les corps de requête, les en-têtes, les cookies, etc.



C. Base de données

MongoDB Atlas est une plateforme cloud entièrement gérée qui offre un service de base de données MongoDB. MongoDB est une base de données NoSQL populaire, orientée document, qui permet de stocker et de récupérer des données de manière flexible et évolutive. MongoDB Atlas facilite la gestion et le déploiement de vos bases de données MongoDB. Il vous permet de créer et de configurer facilement des clusters de bases de données MongoDB sur le cloud, sans avoir à vous soucier de la mise en place de l'infrastructure sous-jacente

VI. Description du système IoT :

A. Les composants nécessaires :

1. Le module capteur MQ-2 :

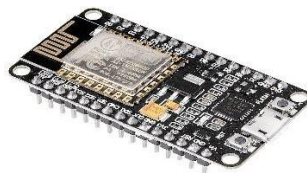


Le capteur de gaz **MQ2** est un capteur électronique utilisé pour détecter la **concentration de gaz** dans l'air tels que le GPL, le propane, le méthane, l'hydrogène, l'alcool, la fumée et le monoxyde de carbone.

Le capteur de gaz MQ2 est également connu sous le nom de chimirésistance. Il contient un matériau de détection dont la résistance change lorsqu'il entre en contact avec le gaz. Ce changement de valeur de résistance est utilisé pour la détection de gaz.

MQ2 est un **semi-conducteur** à oxyde métallique capteur de gaz de type. Les concentrations de gaz dans le gaz sont mesurées à l'aide d'un diviseur de tension réseau présent dans le capteur. Ce capteur fonctionne sur une tension continue de 5V. Il peut détecter des gaz à une concentration de **200 à 10000 ppm**.

2. La carte NodeNCU ESP8266 :



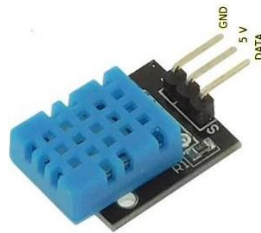
Module basé sur un **ESP8266** cadencé à 80 MHz et exécutant le firmware open source **NodeMCU**. Cette carte se programme via l'IDE Arduino et est compatible avec les scripts **LUA**.

Ce microcontrôleur dispose d'une interface **WiFi** idéale pour les objets connectés. Des connecteurs latéraux mâles et femelles permettent d'enfiler le module sur une plaque de montage rapide.

L'interface sans fil Wifi permet la création de point d'accès sans fil, l'hébergement d'un serveur, la connexion à internet et le partage des données par exemple.

Le module se programme directement à partir de l'IDE Arduino (installation d'une extension nécessaire) et nécessite un cordon microUSB (non inclus). Son implantation le rend compatible avec les plaques de connexions rapides.

3. Le capteur DHT11 :



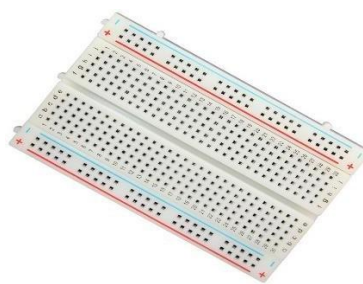
Le capteur **DHT11** mesure la **température** et l'**humidité**. Il a 4 broches, mais il est souvent vendu sur une carte support qui possède 3 broches. Il communique avec l'Arduino très simplement au travers d'une de ses entrées numériques. Les 2 autres broches sont pour son alimentation 5 V et la masse (GND).

DHT11 est un capteur compact qui se connecte facilement à une plage de **microcontrôleurs** tels que les microcontrôleurs Arduino et Raspberry Pi. Il utilise une méthode de mesure numérique pour mesurer la température et l'humidité ambiantes avec une précision élevée.

Ce module est équipé d'une interface de communication numérique qui permet une **transmission rapide** et **fiable** des données. De plus, il est doté d'un boîtier en plastique robuste qui le protège des dommages environnementaux tels que la poussière et l'eau.

DHT11 est un choix idéal pour les applications nécessitant une surveillance en **temps réel** de la température et de l'humidité, telles que les systèmes de contrôle de l'environnement pour les serres, les systèmes de climatisation et de ventilation, les systèmes de surveillance de la santé pour les animaux et les systèmes de contrôle de la **qualité de l'air** (c'est le cas de notre système).

4. Breadboard :



Un **breadboard** est une plaque utilisée pour prototyper ou construire des circuits sans souder. Il permet de placer des composants et des connexions sur la plaque pour réaliser des circuits **rapidement** et **facilement**. Les trous dans le breadboard servent à maintenir les composants et les fils en place et à les connecter électriquement à l'intérieur de la plaque. Les breadboards sont pratiques pour l'apprentissage et le prototypage de circuits simples, mais ils ne conviennent pas aux circuits complexes ou haute fréquence. Contrairement aux circuits construits sur protoboard

ou PCB, les circuits réalisés sur breadboard ne sont pas adaptés à une utilisation à long terme, mais ils évitent les soudures et les coûts de conception et de fabrication.

Les breadboards fonctionnent grâce à des **bandes métalliques** situées sous les rangées de petits trous, qui sont en contact avec les broches des composants. Les rails d'alimentation positive et négative sur les côtés du breadboard permettent d'acheminer l'alimentation externe vers le circuit. Il est également possible de modifier la connexion des rails d'alimentation en utilisant des fils de connexion détachables.

B. Montage du système :

Les connexions nécessaires :

- ✓ Le capteur MQ-2 est connecté à la broche analogique A0 du microcontrôleur.
- ✓ Le capteur DHT11 est connecté au broche numérique 2 (D2) du microcontrôleur.



C. Code :

Le code qu'on a écrit permet de mesurer la qualité de l'air, l'humidité et la température à l'aide de capteurs connectés à un microcontrôleur ESP8266. Les données mesurées sont ensuite envoyées au serveur de notre backend distant via une requête HTTP POST au format JSON. Le code établit une connexion Wi-Fi, lit les valeurs des capteurs, prépare les données et les envoie au serveur. Le processus se répète en boucle avec un délai de 5 secondes entre chaque envoi. En voici une brève explication de chaque bout de code :

- Les bibliothèques ESP8266WiFi, ESP8266HTTPClient et DHT sont tout d'abord incluses.
- Puis les variables ssid, password et serverAddress sont définies pour la connexion Wi-Fi et l'adresse du serveur.

```
#include <ESP8266WiFi.h>
```

```
#include <ESP8266HTTPClient.h>
#include <DHT.h>

const char* ssid = "";
const char* password = "";
const char* serverAddress = "http://192.168.145.99:5003/api/airquality";
```

- Après avoir défini la broche et le type de DHTPIN, on a défini la fonction setup(), qui initialise la communication série, se connecte au réseau Wi-Fi et démarre le capteur DHT.

```
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Pin utilisé pour le capteur MQ-2
#define MQ_PIN A0

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    Serial.println("");

    Serial.print("Connecting");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(250);
    }
    Serial.println("");
    Serial.print("Successfully connected to: ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    dht.begin();
}
```

- La fonction loop() est ensuite définie. Elle lit la qualité de l'air à partir du capteur MQ-2, lit l'humidité et la température à partir du capteur DHTet prépare les données à envoyer sous forme de chaîne JSON.

```
void loop() {
    int aq = readAirQuality();
    float humidity = dht.readHumidity();
```

```

float temperature = dht.readTemperature();

// Prepare data to send
String postData = "[" + "{\"aq\":" + String(aq) + ",\"humidity\":" +
String(humidity) + ",\"temperature\":" + String(temperature) + "}]" +

HTTPClient http;
http.begin(serverAddress);
http.addHeader("Content-Type", "application/json");

int httpCode = http.POST(postData);
String payload = http.getString();

Serial.println(httpCode);
Serial.println(payload);

http.end();

delay(5000); // Wait for 5 seconds before sending the next data
}

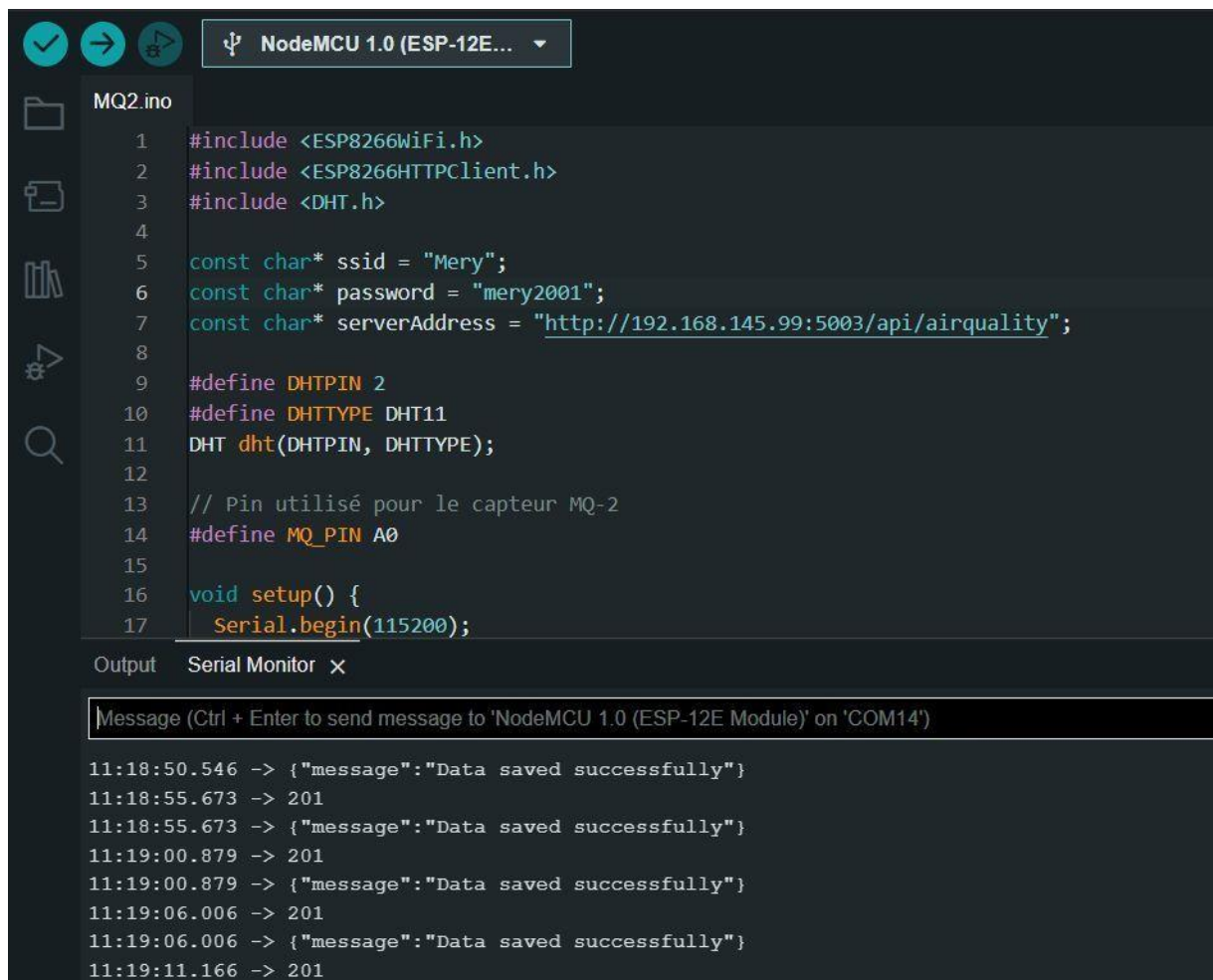
int readAirQuality() {
// Lire la valeur de qualité de l'air à partir du capteur MQ-2
int airQuality = analogRead(MQ_PIN);

// Effectuer tout calcul ou traitement supplémentaire nécessaire

return airQuality;
}

```

Après avoir téléchargé les bibliothèques requises, il est temps de le téléverser le code dans notre Arduino IDE.



The screenshot displays the Arduino IDE interface. At the top, the board is set to 'NodeMCU 1.0 (ESP-12E...)'. The file 'MQ2.ino' is open, showing the following code:

```
1 #include <ESP8266WiFi.h>
2 #include <ESP8266HTTPClient.h>
3 #include <DHT.h>
4
5 const char* ssid = "Mery";
6 const char* password = "mery2001";
7 const char* serverAddress = "http://192.168.145.99:5003/api/airquality";
8
9 #define DHTPIN 2
10 #define DHTTYPE DHT11
11 DHT dht(DHTPIN, DHTTYPE);
12
13 // Pin utilisé pour le capteur MQ-2
14 #define MQ_PIN A0
15
16 void setup() {
17     Serial.begin(115200);
```

Below the code editor, the 'Serial Monitor' is open, showing the following output:

```
Message (Ctrl + Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM14')
11:18:50.546 -> {"message":"Data saved successfully"}
11:18:55.673 -> 201
11:18:55.673 -> {"message":"Data saved successfully"}
11:19:00.879 -> 201
11:19:00.879 -> {"message":"Data saved successfully"}
11:19:06.006 -> 201
11:19:06.006 -> {"message":"Data saved successfully"}
11:19:11.166 -> 201
```

Comme montré ci-dessus, les données de la qualité de l'air, de l'humidité et de la température sont capturées et envoyées avec succès à notre serveur distant du backend.

VII. Fonctionnalités :

A. Le dashboard :

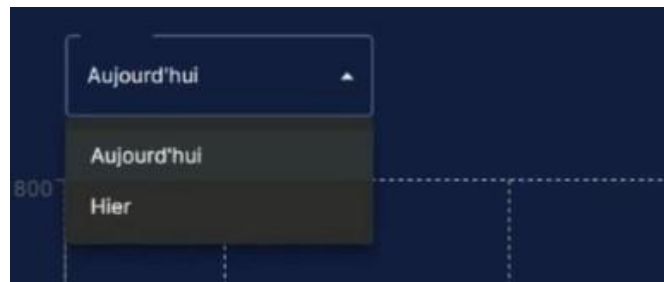
Concernant la page "Dashboard", elle nous permet de sélectionner une période spécifique pour visualiser les données de qualité de l'air, de les exporter et d'obtenir des statistiques clés.



Toutes les fonctionnalités de ce dashboard, sont implémentées au niveau du fichier client/src/scenes/dashboard/index.jsx.

1. Fonctionnalités de la page :

Sélection de la période : La page comprend une liste déroulante permettant à l'utilisateur de choisir la période pour laquelle il souhaite afficher les données. Les options de période incluent "Aujourd'hui" et "Hier".



Le code suivant permet de filtrer des données en fonction d'une période spécifique et de réagir aux changements de filtre sélectionnés par l'utilisateur.

```
const filterDataByPeriod = (data, period) => {
  let filteredData = [];
  const currentDate = new Date().toISOString().split('T')[0]; // Get current date in YYYY-MM-DD format

  switch (period) {
    case 'today':
      filteredData = data.filter((item) => item.timestamp.split('T')[0] === currentDate);
      break;
    case 'yesterday':
      const yesterday = new Date();
      yesterday.setDate(yesterday.getDate() - 1);
      const yesterdayDate = yesterday.toISOString().split('T')[0]; // Get yesterday's date in YYYY-MM-DD format
      filteredData = data.filter((item) => item.timestamp.split('T')[0] === yesterdayDate);
      break;
    // Add more cases for other periods
    default:
      filteredData = data;
      break;
  }

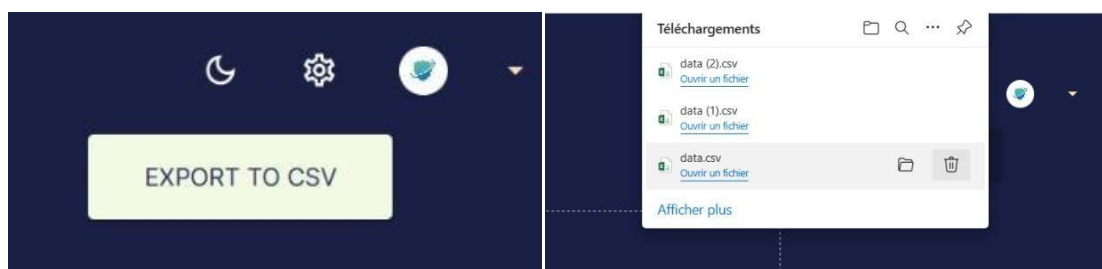
  return filteredData;
};

const handleFilterChange = (event) => {
  setSelectedPeriod(event.target.value);
};
```

Ce code contient deux parties :

- La fonction `filterDataByPeriod` est utilisée pour filtrer des données en fonction d'une période spécifique. Elle prend un tableau de données et une période en entrée, puis renvoie un nouveau tableau contenant les éléments filtrés. Les périodes prises en charge sont "today" (aujourd'hui), "yesterday" (hier) et par défaut, aucune période de filtrage n'est appliquée.
- Le gestionnaire d'événements `handleFilterChange` est utilisé pour gérer les changements de valeur dans un élément de formulaire (par exemple, une liste déroulante). Lorsqu'un changement de valeur se produit, la fonction `setSelectedPeriod` est appelée avec la nouvelle valeur sélectionnée. Cela permet de mettre à jour la période de filtrage utilisée par la fonction `filterDataByPeriod`.

Exportation vers CSV : Un bouton "Export to CSV" permet à l'utilisateur d'exporter les données filtrées au format CSV. Les données exportées sont filtrées en fonction du mois en cours.



Ainsi, les données exportées sont filtrées en fonction du mois en cours.

data (3)

	A	B	C	D	E	
1	_id	aq	humidity	temperature	timestamp	_v
2	646e83b2225fd	907	76	24.3	2023-05-24T21:37:54.940Z	
3	646e83b8225fd	906	76	24.3	2023-05-24T21:38:00.289Z	
4	646e83bd225fd	906	76	24.3	2023-05-24T21:38:05.429Z	
5	646e83c2225fd	906	76	24.3	2023-05-24T21:38:10.572Z	
6	646e83c7225fd	904	76	24.3	2023-05-24T21:38:15.754Z	
7	646e83cc225fd	904	76	24.3	2023-05-24T21:38:20.898Z	
8	646e83d2225fd	904	76	24.3	2023-05-24T21:38:26.045Z	
9	646e83d7225fd	903	76	24.3	2023-05-24T21:38:31.221Z	
10	646e83dc225fd	902	76	24.3	2023-05-24T21:38:36.368Z	
11	646e87ef225fd	726	76	24.6	2023-05-24T21:55:59.426Z	
12	646e87f4225fd	866	76	24.6	2023-05-24T21:56:04.624Z	
13	646e87fa225fd	903	76	24.6	2023-05-24T21:56:10.040Z	
14	646e87ff225fd	916	76	24.6	2023-05-24T21:56:15.178Z	
15	646e8804225fd	921	76	24.6	2023-05-24T21:56:20.335Z	
16	646e8809225fd	921	76	24.6	2023-05-24T21:56:25.504Z	
17	646e880e225fd	921	76	25	2023-05-24T21:56:30.649Z	
18	646e8813225fd	919	75	24.6	2023-05-24T21:56:35.790Z	
19	646e8818225fd	918	76	24.6	2023-05-24T21:56:40.933Z	
20	646e881e225fd	916	75	24.6	2023-05-24T21:56:46.090Z	

Cette fonctionnalité a été implémentée à l'aide de la fonction `exportToCSV` qui exporte des données filtrées au format CSV.

```
const exportToCSV = () => {
  const currentDate = new Date();
  const startDate = startOfMonth(currentDate);
  const endDate = endOfMonth(currentDate);

  const filteredData = data.filter((item) => {
    const itemDate = new Date(item.timestamp);
    return isWithinInterval(itemDate, { start: startDate, end: endDate });
  });

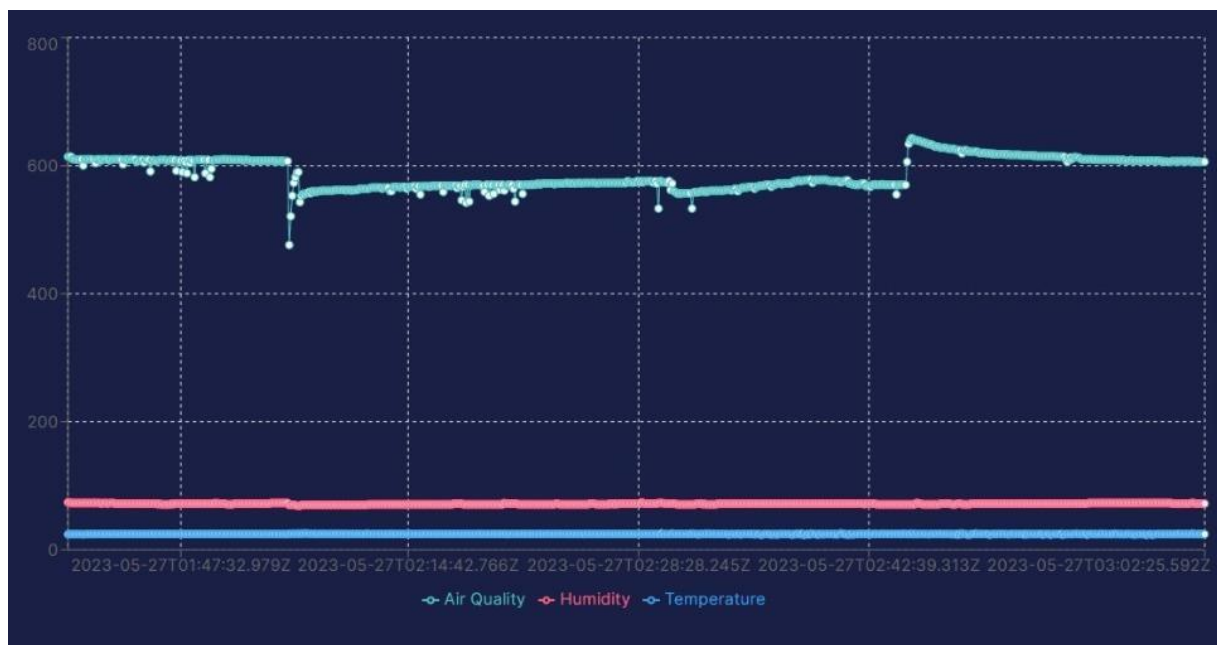
  const csvData = Papa.unparse(filteredData);
  const csvBlob = new Blob([csvData], { type: 'text/csv;charset=utf-8;' });
  saveAs(csvBlob, 'data.csv');
};
```

- La fonction récupère la date et l'heure actuelles dans la variable `currentDate`.
- Les variables `startDate` et `endDate` sont définies en utilisant des fonctions externes

``startOfMonth`` et ``endOfMonth``, qui renvoient respectivement le début et la fin du mois correspondant à ``currentDate``.

- Les données existantes sont filtrées en utilisant la méthode ``filter`` pour ne conserver que les éléments dont la date (``item.timestamp``) se situe entre ``startDate`` et ``endDate``.
- Les données filtrées sont converties en une chaîne CSV à l'aide de la fonction ``Papa.unparse``.
- Une ``Blob`` est créée en encapsulant la chaîne CSV, spécifiant le type de contenu comme `"text/csv;charset=utf-8;"`.
- La fonction ``saveAs`` est utilisée pour télécharger le fichier CSV en utilisant le ``Blob`` créé précédemment. Le fichier est enregistré avec le nom `'data.csv'`.

Graphique en ligne : Le graphique en ligne affiche les données de la qualité de l'air (AQI), de l'humidité et de la température. Chaque série de données est représentée par une ligne de couleur différente. Le graphique est interactif et affiche des informations supplémentaires lors du survol chaque 5 secondes, tout en précisant la date exacte de la capture des données.



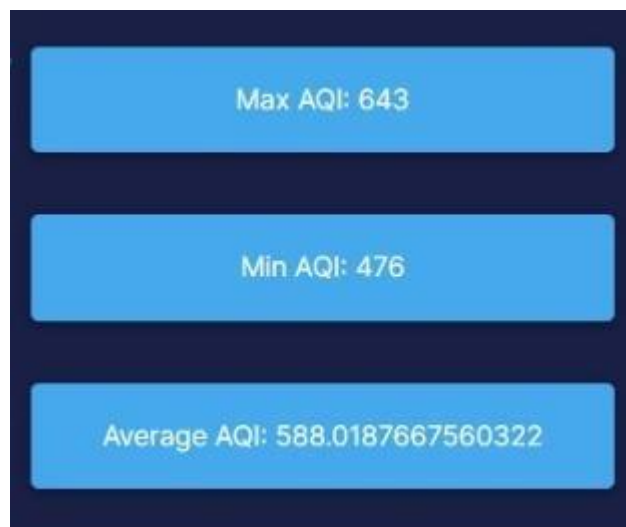
Le code crée un graphique en ligne avec des lignes utilisant la bibliothèque de visualisation de données `recharts` et représentant différentes séries de données (la qualité de l'air, l'humidité et la température). Il utilise des composants spécifiques pour définir les axes, la grille, la légende et les couleurs des lignes. Les données à afficher sont fournies via la propriété ``data``.

```

<Grid item>
  <LineChart width={1000} height={480} data={data}>
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis dataKey="timestamp" />
    <YAxis />
    <Tooltip />
    <Legend />
    <Line type="monotone" dataKey="aq" stroke="rgba(75, 192, 192, 1)" name="Air Quality" />
    <Line type="monotone" dataKey="humidity" stroke="rgba(255, 99, 132, 1)" name="Humidity" />
    <Line type="monotone" dataKey="temperature" stroke="rgba(54, 162, 235, 1)" name="Temperature" />
  </LineChart>
</Grid>

```

Statistiques : Trois blocs affichent les statistiques relatives à la qualité de l'air. Il s'agit de la valeur maximale (Max AQI), de la valeur minimale (Min AQI) et de la valeur moyenne (Average AQI). Ces statistiques sont calculées à partir des données filtrées pour la période sélectionnée.



Afin de les calculer, nous avons défini une fonction appelée `calculateStatistics`.

```

const calculateStatistics = (data) => {
  const aqValues = data.map((item) => parseInt(item.aq));

  const minAq = Math.min(...aqValues);
  const maxAq = Math.max(...aqValues);
  const averageAq = aqValues.reduce((sum, value) => sum + value, 0) / aqValues.length;

  setMinValue(minAq);
  setMaxValue(maxAq);
  setAverageValue(averageAq);
};

```

Voici un résumé du fonctionnement de ce code:

- ✚ La fonction prend un paramètre `data`, qui représente un tableau de données.
- ✚ Les valeurs de la propriété "aq" de chaque élément de `data` sont extraites et stockées dans le tableau `aqValues`.
- ✚ La variable `minAq` est calculée en utilisant la fonction `Math.min` avec l'opérateur de

décomposition (...) pour étaler les valeurs de `aqValues` en tant qu'arguments.

- De même, la variable `maxAq` est calculée en utilisant la fonction `Math.max`.
- La variable `averageAq` est calculée en faisant la somme des valeurs de `aqValues` à l'aide de la méthode `reduce`, puis en divisant cette somme par la longueur de `aqValues`.
- Les valeurs calculées (`minAq`, `maxAq` et `averageAq`) sont utilisées pour mettre à jour l'état de l'application en utilisant des fonctions de modification de l'état (`setMinValue`, `setMaxValue` et `setAverageValue`).

Fonctionnalité d'Alerte : Pour ce qui est de la qualité de l'air lorsqu'elle est mauvaise une notification le prouvant sera envoyée sur le Dashboard pour donner l'état de la qualité de l'air. Cette notification se base sur les valeurs retournées par les capteurs d'humidité, de température et de qualité de l'air afin de pouvoir afficher si la qualité de l'air est bonne ou pas. Le dashboard intègre aussi un menu convivial, si on clique sur les 3 barres à côté de l'espace de recherche ça va agrandir la vue sur les statistiques et la vue sera comme présentée ci bas.



Pour le code source de cette fonctionnalité, les imports en début de fichier importent les différentes dépendances nécessaires au composant, telles que React, des composants MUI (Material-UI) et des icônes.

Le composant `AlertComponent` est une fonction qui prend deux props : `"data"` (les données liées à la qualité de l'air) et `"sumValue"` (la somme des valeurs).

```
import React, { useState, useEffect } from 'react';
import { IconButton } from '@mui/material';
import { MdAddAlert } from 'react-icons/md';
import Dialog from '@mui/material/Dialog';
import DialogTitle from '@mui/material/DialogTitle';
import DialogContent from '@mui/material/DialogContent';
import DialogContentText from '@mui/material/DialogContentText';
import DialogActions from '@mui/material/DialogActions';
import Button from '@mui/material/Button';

const AlertComponent = ({ data, sumValue }) => {
  const [recommendation, setRecommendation] = useState('');
  const [recommendationList, setRecommendationList] = useState([]);
  const [openDialog, setOpenDialog] = useState(false);
```

Le composant utilise les hooks `useState` et `useEffect` de React. `useState` est utilisé pour gérer les états "recommendation" (recommandation) et "recommendationList" (liste de recommandations), ainsi que "openDialog" (ouverture/fermeture de la boîte de dialogue). `useEffect` est utilisé pour effectuer des actions lorsque les données changent. Dans le `useEffect`, lorsque les données "data" changent, le composant extrait les dernières données disponibles (`latestData`) et vérifie la valeur de la qualité de l'air (`value`). En fonction de cette valeur, il met à jour les états "recommendation" et "recommendationList" avec les recommandations appropriées.

```
15
16   useEffect(() => {
17     if (data) {
18       const latestData = data[data.length - 1];
19       const value = latestData.aq;
20
21       if (value >= 20000) {
22         setRecommendation("Mauvaise qualité de l'air");
23         setRecommendationList([
24           "Évitez les activités extérieures.",
25           "Portez un masque facial pour réduire l'inhalation de polluants.",
26           "Restez dans des espaces intérieurs bien ventilés."
27         ]);
28       } else if (value > 500 && value < 20000) {
29         setRecommendation("Qualité de l'air modérée");
30         setRecommendationList([
31           "Évitez les activités extérieures intenses.",
32           "Restez à l'intérieur si vous ressentez des symptômes d'inconfort.",
33           "Consultez votre médecin si les symptômes persistent."
34         ]);
35       } else {
36         setRecommendation("Bonne qualité de l'air");
37         setRecommendationList([]);
38       }
39     }
40   }, [data]);
```

Les fonctions `handleDialogOpen` et `handleDialogClose` sont utilisées pour ouvrir et fermer la boîte de dialogue. Le rendu du composant se fait à l'intérieur de la balise `<>`, qui est une syntaxe raccourcie pour `React.Fragment`. Il affiche la recommandation et un bouton d'alerte (`IconButton`) s'il y a une recommandation à afficher.

```

12  const handleDialogOpen = () => {
13    setOpenDialog(true);
14  };
15
16  const handleDialogClose = () => {
17    setOpenDialog(false);
18  };
19
20  return (
21    <>
22      {recommendation !== '' && (
23        <div>
24          <h2>{recommendation}</h2>
25          {recommendation !== "Bonne qualité de l'air" && (
26            <IconButton onClick={handleDialogOpen} color="primary">
27              <MdAddAlert />
28            </IconButton>
29          )}
30        </div>
31      )}
32    </>
33  )

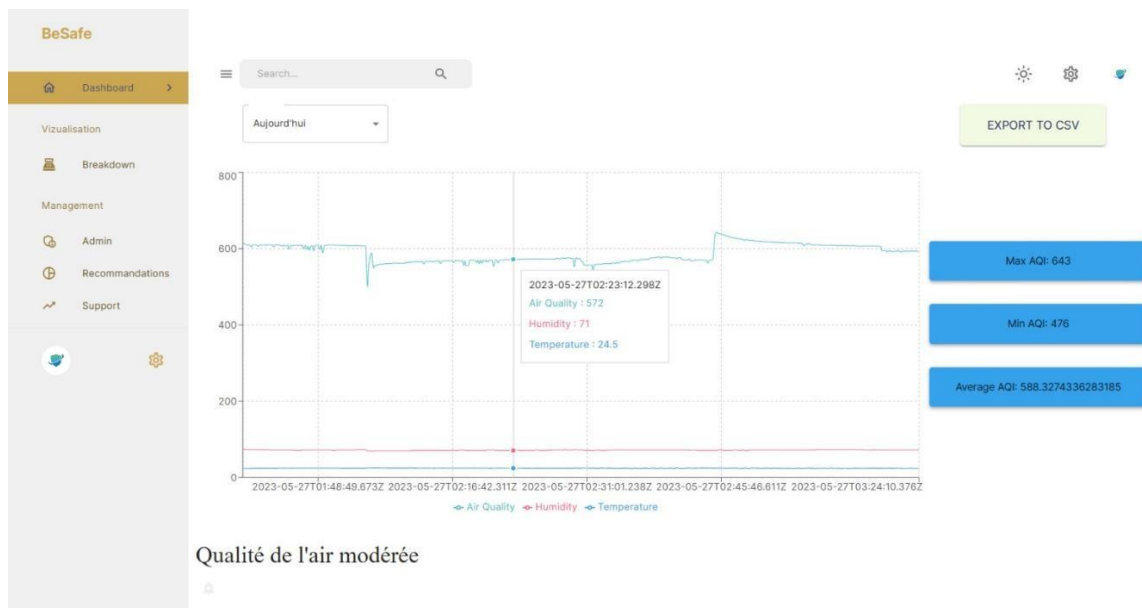
```

```

client > src > scenes > recommendations > index.jsx > ...
63  <Dialog open={openDialog} onClose={handleDialogClose}>
64    <DialogTitle>Recommendations</DialogTitle>
65    <DialogContent>
66      <DialogContentText>
67        <h2>{recommendation}</h2>
68        {recommendationList.length > 0 ? (
69          <ul>
70            {recommendationList.map((item, index) => (
71              <li key={index}>{item}</li>
72            ))}
73          </ul>
74        ) : (
75          <p>Aucune recommandation disponible.</p>
76        )}
77        {sumValue !== null && (
78          <p>La somme des valeurs est de : {sumValue}</p>
79        )}
80      </DialogContentText>
81    </DialogContent>
82    <DialogActions>
83      <Button onClick={handleDialogClose}>Fermer</Button>
84    </DialogActions>
85  </Dialog>
86 </>
87 );
88 };
89
90 export default AlertComponent;

```


Light Mode :en cliquant sur l'icône de « lune » sur le Dashboard en haut à droite on a accès à une autre fonctionnalité de notre application web,il s'agit du « **light mode** » que nous avons ajouté afin que l'application soit plus un outil conviviale et agréable. Ce mode a été ajouté surtout pour les utilisateurs sensibles à la luminosité. Certains utilisateurs peuvent être sensibles à la luminosité élevée des écrans, ce qui peut entraîner une fatigue oculaire ou des maux de tête. Le mode clair offre une interface plus douce pour leurs yeux, en réduisant la luminosité globale de l'application



B. Le breakdown :

Dans cette partie on va créer et afficher un graphique de décomposition circulaire des mesures de la qualité de l'air, de l'humidité et de la température en valeurs moyennes, ainsi que l'estimation de la qualité de l'air en se basant sur ces valeurs moyennes.



```

EXPLORER
  > OPEN EDITORS
  > DASHBOARD
    > node_modules
    > public
    > src
      > assets
      > components
      > scenes
        > admin
        > breakdown
        > index.jsx
      > dashboard
      > layout
      > recommendations
      > support
      > state
    JS App.js
    # index.css
    JS index.js
  JS theme.js
  NavBar.jsx client\src\components
  # style.css

client > src > scenes > breakdown > index.jsx
1 import React from "react";
2 import { Box } from "@mui/material";
3 import Header from "components/Header";
4 import BreakdownChart from "components/BreakdownChart";
5
6
7 const Breakdown = () => {
8   return (
9     <Box m="1.5rem 2.5rem">
10       <Header title="BREAKDOWN" subtitle="Breakdown of AirQuality By measures" />
11       <Box mt="40px" height="75vh">
12         <BreakdownChart />
13       </Box>
14     </Box>
15   );
16
17   export default Breakdown;

```

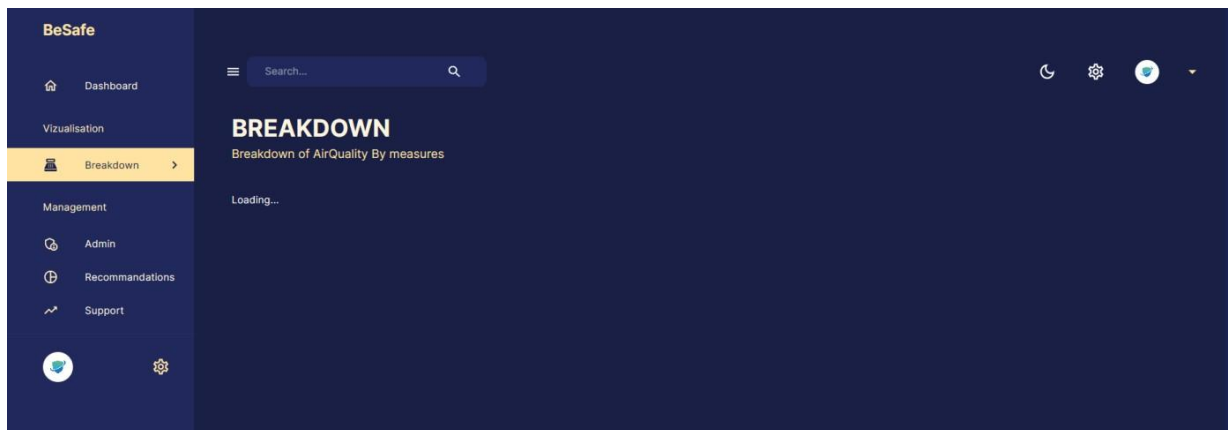
Au niveau du scenes/breakdown/index.jsx on va définir le squelette de notre page breakdown tel qu'on va faire appel à un component BreakdownChart qu'on va expliquer par la suite.

```

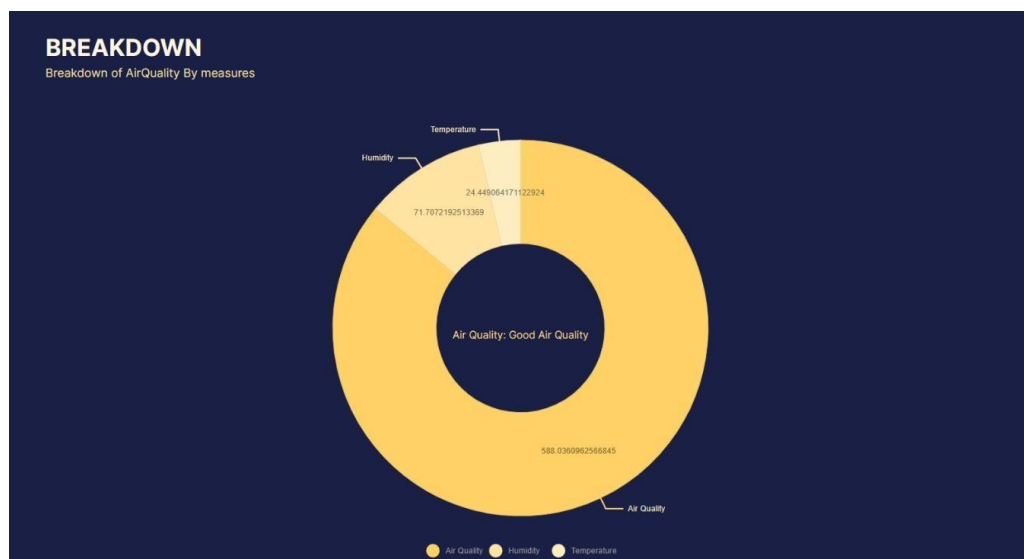
8   const { data, isLoading } = useGetAirQuery();
9   const theme = useTheme();
10  console.log(data)
11  if (!data || isLoading) return "Loading...";
12
13  const colors = [
14    theme.palette.secondary[500],
15    theme.palette.secondary[300],
16    theme.palette.secondary[200],
17    theme.palette.secondary[350],
18  ];
19  const airQualityAvg = data.reduce((acc, { aq }) => acc + aq, 0) / data.length;
20  const HumidityAvg = data.reduce((acc, { humidity }) => acc + humidity, 0) / data.length;
21  const TemperatureAvg = data.reduce((acc, { temperature }) => acc + temperature, 0) / data.length;
22
23  let airQualityLabel = "Good Air Quality";
24
25  if (airQualityAvg > 700 || airQualityAvg < 430 || HumidityAvg > 80 || TemperatureAvg > 40) {
26    airQualityLabel = "Poor Air Quality";
27  }

```

Le composant utilise le hook "useGetAirQuery" pour récupérer les données de l'API de l'air, telles que les mesures de qualité de l'air, d'humidité et de température. Les données et le statut de chargement sont stockés dans les variables "data" et "isLoading ». Une fois que les données sont disponibles et que le chargement est terminé, le composant calcule la valeur moyenne de chaque mesure à partir des données. En fonction de ces moyennes, il détermine une étiquette pour la qualité de l'air. Si la qualité de l'air moyenne est supérieure à 700 ou inférieure à 430, ou si l'humidité moyenne est supérieure à 80 ou si la température moyenne est supérieure à 40, l'étiquette est définie sur "Poor Air Quality". Sinon, elle est définie sur "Good Air Quality".



Le composant formate ensuite les données dans un tableau avec des objets correspondant à chaque mesure. Chaque objet contient un identifiant, une étiquette, une valeur et une couleur correspondante. Ces données formatées sont ensuite utilisées pour rendre le graphique de décomposition à l'aide du composant ResponsivePie de la bibliothèque "@nivo/pie".



C. Gestion des Admins

Au niveau de la partie Admin on va gérer et afficher es administrateurs dans notre interface utilisateur comme le montre la figure ci-dessous

The screenshot shows the BeSafe Admin interface. On the left is a sidebar with navigation links: Dashboard, Visualisation, Breakdown, Management, Admin (highlighted), Recommendations, and Support. The main area is titled 'ADMINS' with the subtitle 'Managing admins and the list of Admins'. It contains a table with the following data:

ID	Name	Email	Phone Number	Occupation	Role
63701cc1f03239b7f700001e	Maryam TAMLALTI	mery2001@techhealth.com	(634)631-5674	Responsable bloc A	admin
63701cc1f03239b7f700002e	Malika ousseini	lika@techhealth.com	83463156	Responsable bloc B	admin
63701cc1f03239b7f700003e	Fatimazahrae laaziz	faty@techhealth.com	834631567	Responsable bloc C	admin
63701cc1f03239b7f700004e	Oussakel khadja	duja@techhealth.com	83465674	Responsable bloc D	admin

D'abord, on va utiliser Mongoose pour définir un schéma de modèle de données représentant un utilisateur dans une application. Le schéma spécifie les champs tels que le nom, l'e-mail, le mot de passe, l'occupation, le numéro de téléphone et le rôle de l'utilisateur.

The screenshot shows a code editor with the following JavaScript code defining a Mongoose schema for a user:

```

server > models > JS User.js > [0] default
1 import mongoose from "mongoose";
2
3 const UserSchema = new mongoose.Schema(
4   {
5     name: {
6       type: String,
7       required: true,
8       min: 2,
9       max: 100,
10    },
11    email: {
12      type: String,
13      required: true,
14      max: 50,
15      unique: true,
16    },
17    password: {
18      type: String,
19      required: true,
20      min: 5,
21    },
22    occupation: String,
23    phoneNumber: String,
24    role: {
25      type: String,
26    },
27  },
28  { timestamps: true },
29);
30 export default mongoose.model('User', UserSchema);
  
```

Ensuite, on va se permettre de définir un tableau de données représentant des utilisateurs en respectant le schéma prédéfini, afin de le pusher à la base de données MongoDB Atlas.

```

export const dataUser = [
  {
    _id: "63701cc1f03239b7f700001e",
    name: "Maryam TAMLALTI",
    email: "mery2001@techhealth.com",
    password: "meryyy",
    occupation: "Responsable blocA",
    phoneNumber: "8346315874",
    role: "admin",
  },
  {
    _id: "63701cc1f03239b7f700002e",
    name: "Malika ousseini",
    email: "lika@techhealth.com",
    password: "likaa",
    occupation: "Responsable bloc B",
    phoneNumber: "83463158",
    role: "admin",
  },
  {
    _id: "63701cc1f03239b7f700003e",
    name: "Fatimazahrae laaziz",
    email: "faty@techhealth.com",
    password: "fatyyy",
    occupation: "Responsable bloc C",
  }
]

```

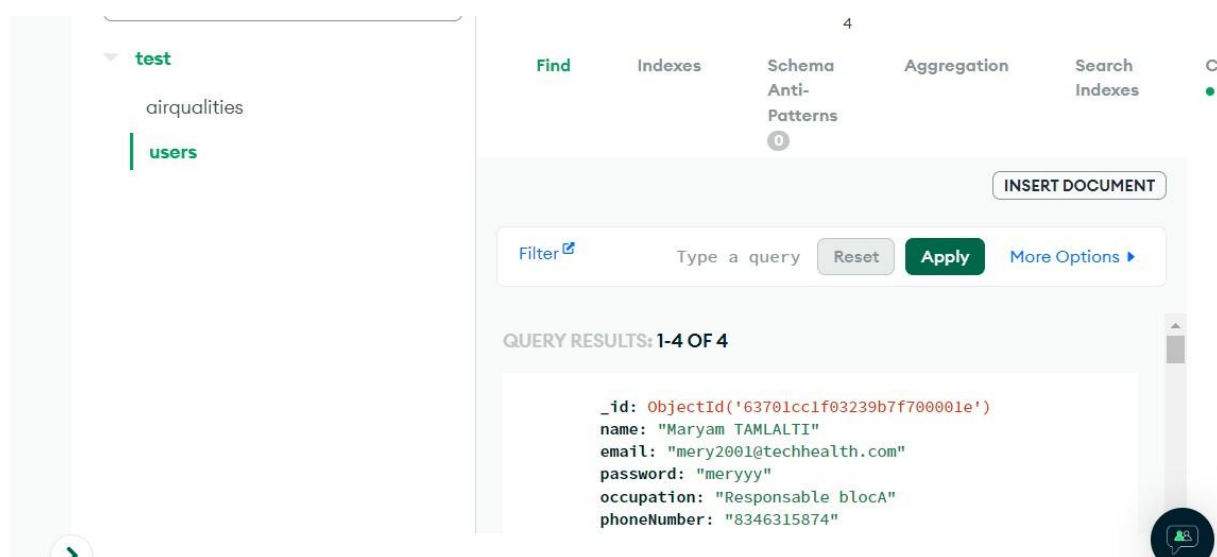
Afin de pusher on va utiliser la ligne de commande `User.insertMany(dataUser)`

NB : Dès qu'on insère les données, on va commenter cette ligne afin de ne pas avoir un problème de l'unicité de l'ID.

```

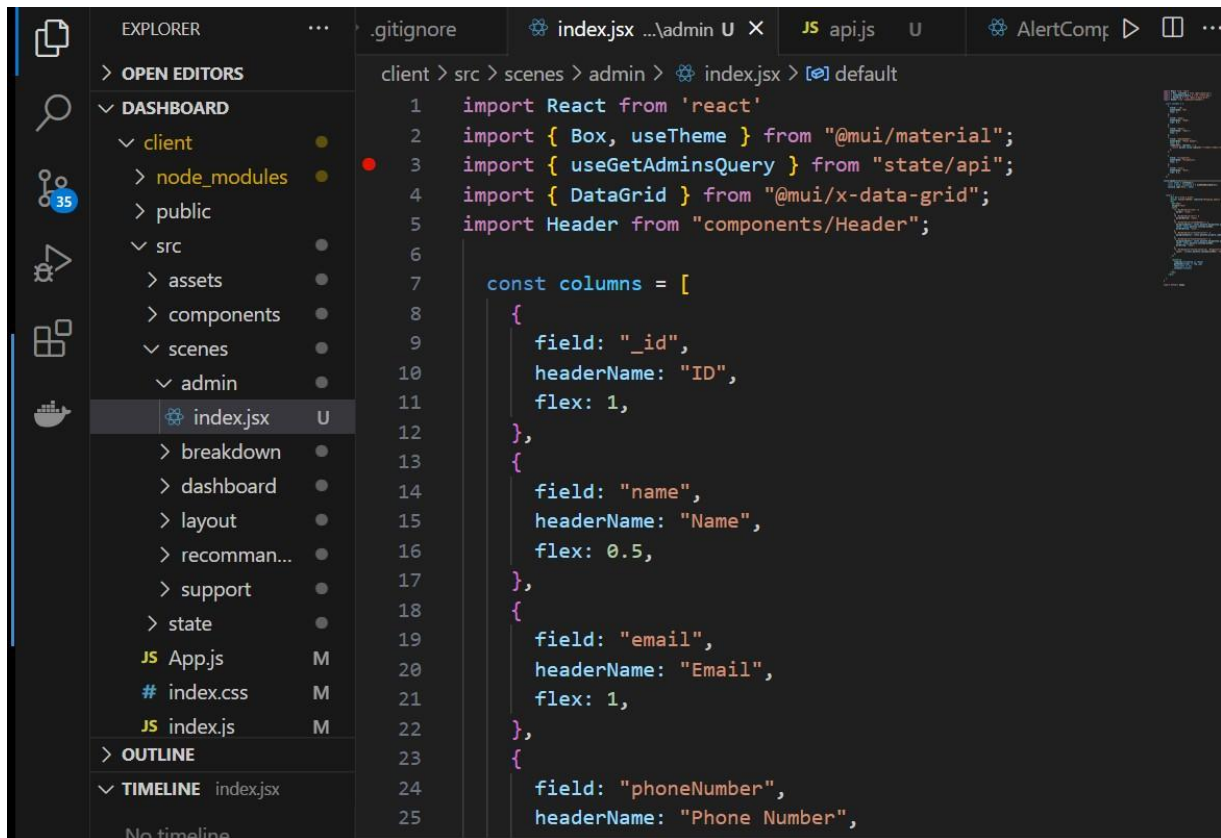
/* ONLY ADD DATA ONE TIME */
User.insertMany(dataUser);
})
.catch((error) => console.log(`${error} did not connect`));

```

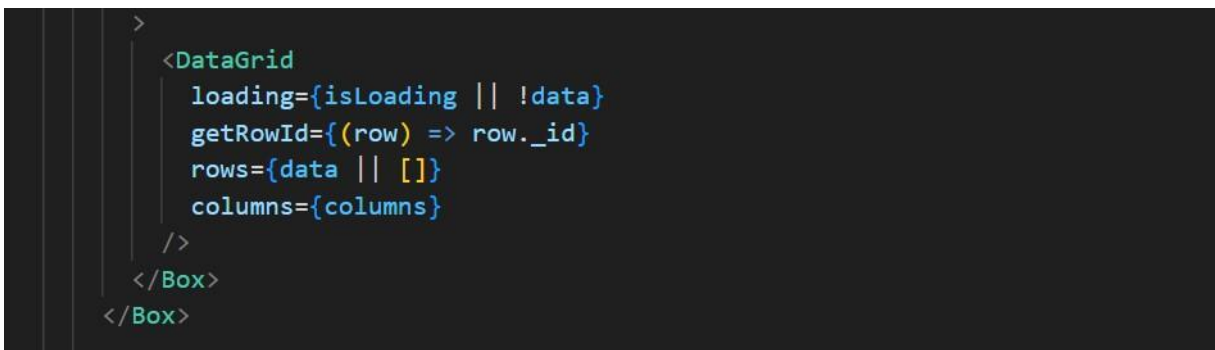


Les données ont été bien ajoutées au niveau de la base de données test.users.

Pour afficher les adamins dans l'interface on va procéder comme suit :



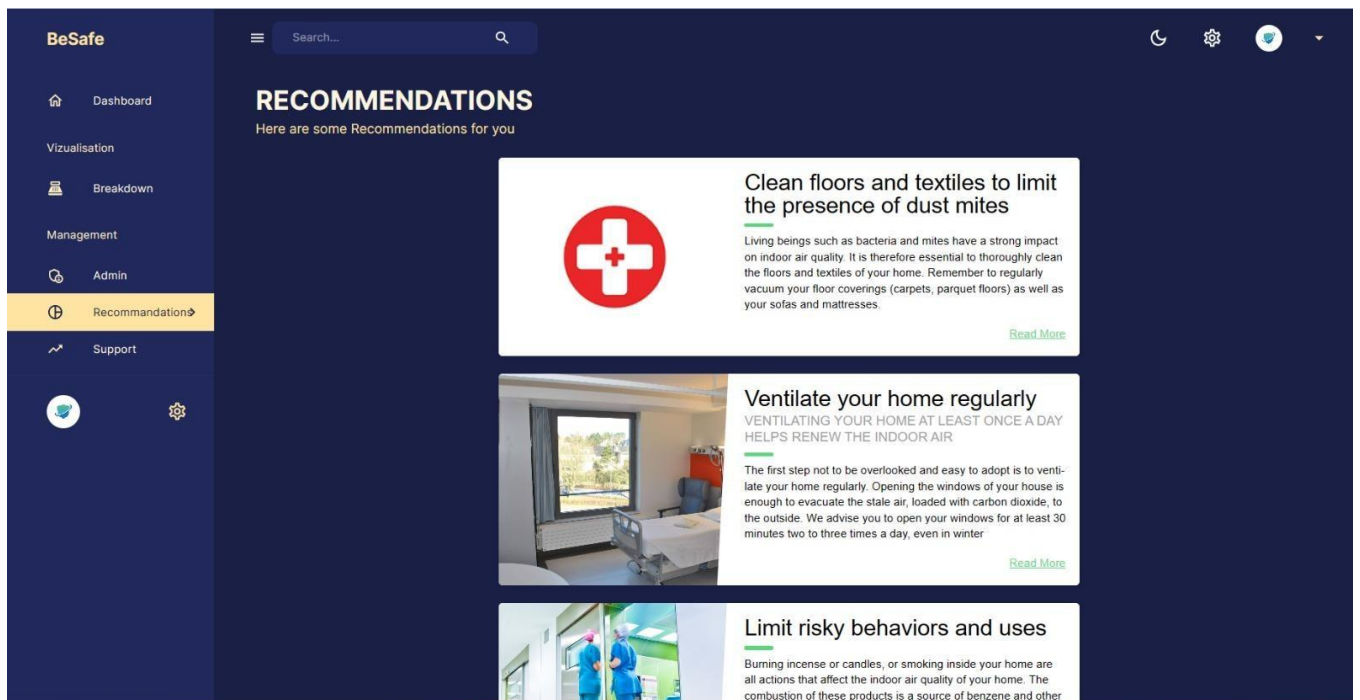
Au niveau du dossier client/src/scenes/admin on va créer le composant « Admin » pour afficher la liste des administrateurs, en utilisant également le hook useGetAdminsQuery pour obtenir les données des administrateurs à partir de l'API.



A ce niveau, on va utiliser le composant DataGrid de Material-UI pour afficher un tableau de données des administrateurs. Il gère le chargement asynchrone des données, configure les colonnes du tableau et les propriétés d'apparence, et utilise les données récupérées pour afficher les lignes correspondantes. Cela permet d'afficher facilement et de manière conviviale les informations des administrateurs dans notre interface utilisateur.

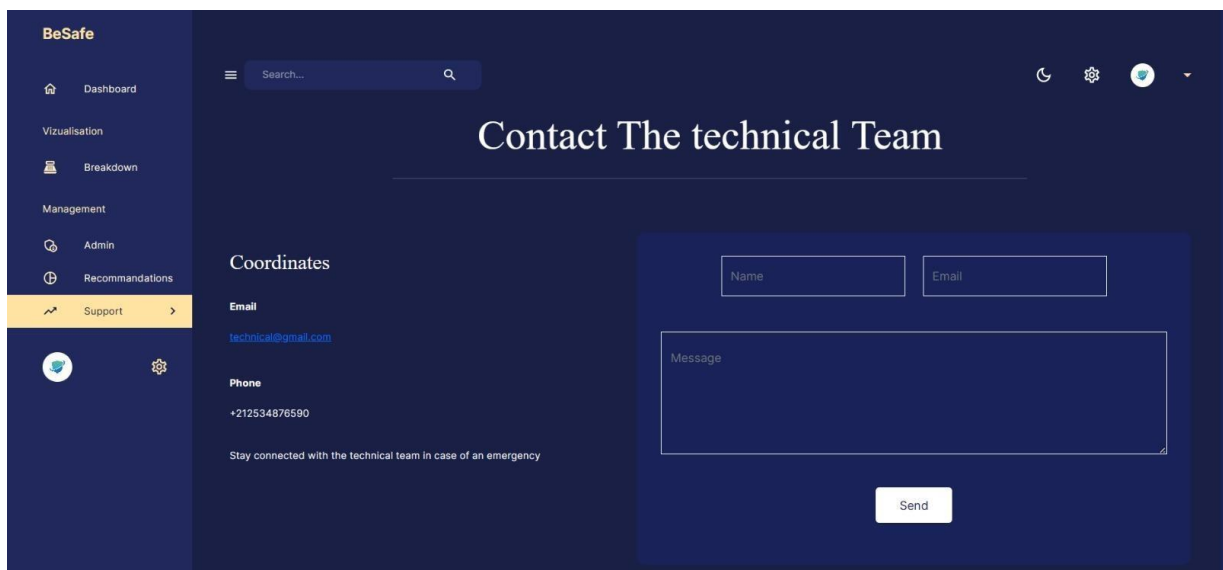
D. La page de recommandation

Cette page est destinée à un usage instructif. Il y a différentes instructions qui y sont inscrites en vue de guider les personnes à préserver encore plus leur santé lorsque les conditions atmosphériques ne sont pas bonnes.



E. Support et aide :

Cette page est conçue pour pouvoir envoyer des messages à l'équipe technique en cas de problème. Elle contient ainsi les coordonnées nécessaires et un formulaire à remplir en cas d'urgence.



VIII. Lien github :

<https://github.com/maryamtamlalti2001/BeSafe/tree/master>

IX. Conclusion :

En conclusion, la conception et la réalisation de ce système a été couronné de succès. La plateforme offre une solution complète et efficace pour collecter, analyser et surveiller en temps réel les données essentielles liées à la qualité de l'air, notamment l'humidité, la température et la présence de gaz potentiellement dangereux.

L'objectif principal de cette plateforme est d'assurer la sécurité des patients, du personnel médical et des installations hospitalières en détectant les dépassements de seuils et en déclenchant des notifications pour une intervention immédiate. Grâce à une interface conviviale et des visualisations claires, les utilisateurs autorisés peuvent accéder aux données en temps réel, interpréter facilement les informations et prendre des mesures préventives appropriées.

La collaboration entre le système IoT et la plateforme web permet une surveillance proactive de la qualité de l'air, une analyse des données historiques et une prise de décision éclairée. Les capteurs spécifiques installés dans différentes zones de l'hôpital assurent une collecte précise des données, fournissant ainsi une vision globale de la qualité de l'air dans l'établissement.

En somme, cette plateforme offre une solution complète et fiable pour la surveillance de la qualité de l'air dans un environnement hospitalier. Elle contribue à prévenir les risques d'incendie et à garantir un environnement sûr et sain pour tous les acteurs impliqués. Cette innovation représente un pas en avant important dans l'amélioration de la sécurité et du bien-être des patients, ainsi que dans l'efficacité des opérations hospitalières.