

Monitoring en temps réel, Température & humidité

The screenshot displays the SensorHub mobile application interface. On the left, a sidebar contains navigation items: SensorHub logo, Recherche (Search) bar, Maison (Home) dropdown, Tableau de bord (Dashboard) dropdown, Aperçu (Preview), Notifications (10), Historique des transactions (Transaction history), and Utilisateurs (Users). The main content area features a welcome message "Bon retour, Olivia" and a subtitle "Suivez et gérez le climat des salles de classes". It includes a "Gérer le seuil par défaut" button and an "Ajouter" button. Below this, there are four temperature status buttons: Froid (<18°C), Optimal (18-24°C), Chaud (24-28°C), and Très chaud (>28°C). Further down are buttons for En hausse (Up), En baisse (Down), and Abonnements (Subscriptions). A search bar is also present. The bottom section shows three rooms with their current temperatures: A35 (13°), A44 (12°), and C36 (23°). Each room card includes a percentage change indicator (↑ 4% for A35, ↑ 20% for C36) and a small line graph.

Chambre	Température	Change (%)	Dernière tendance
A35	13°	↑ 4%	Stable
A44	12°	↓ 10%	En baisse
C36	23°	↑ 20%	En hausse

Le problème :

Conditions climatiques non suivies et plaintes récurrentes concernant les temp. désagréables.

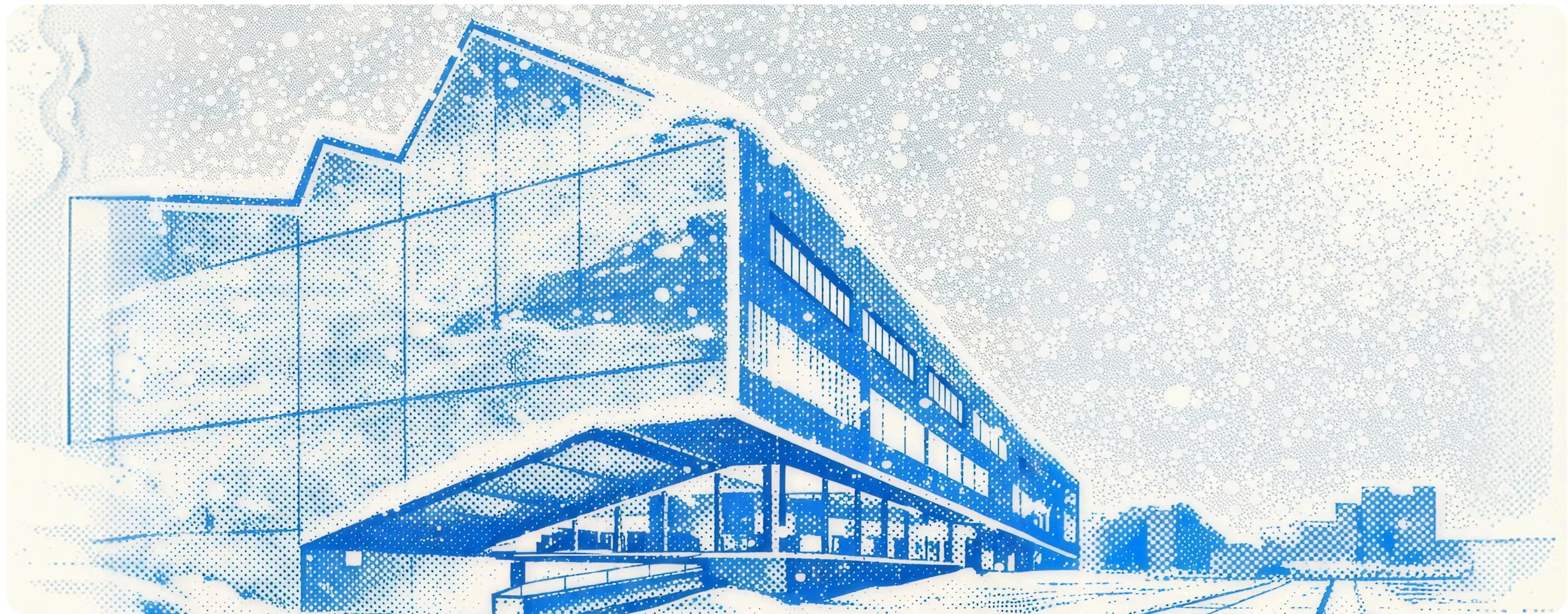
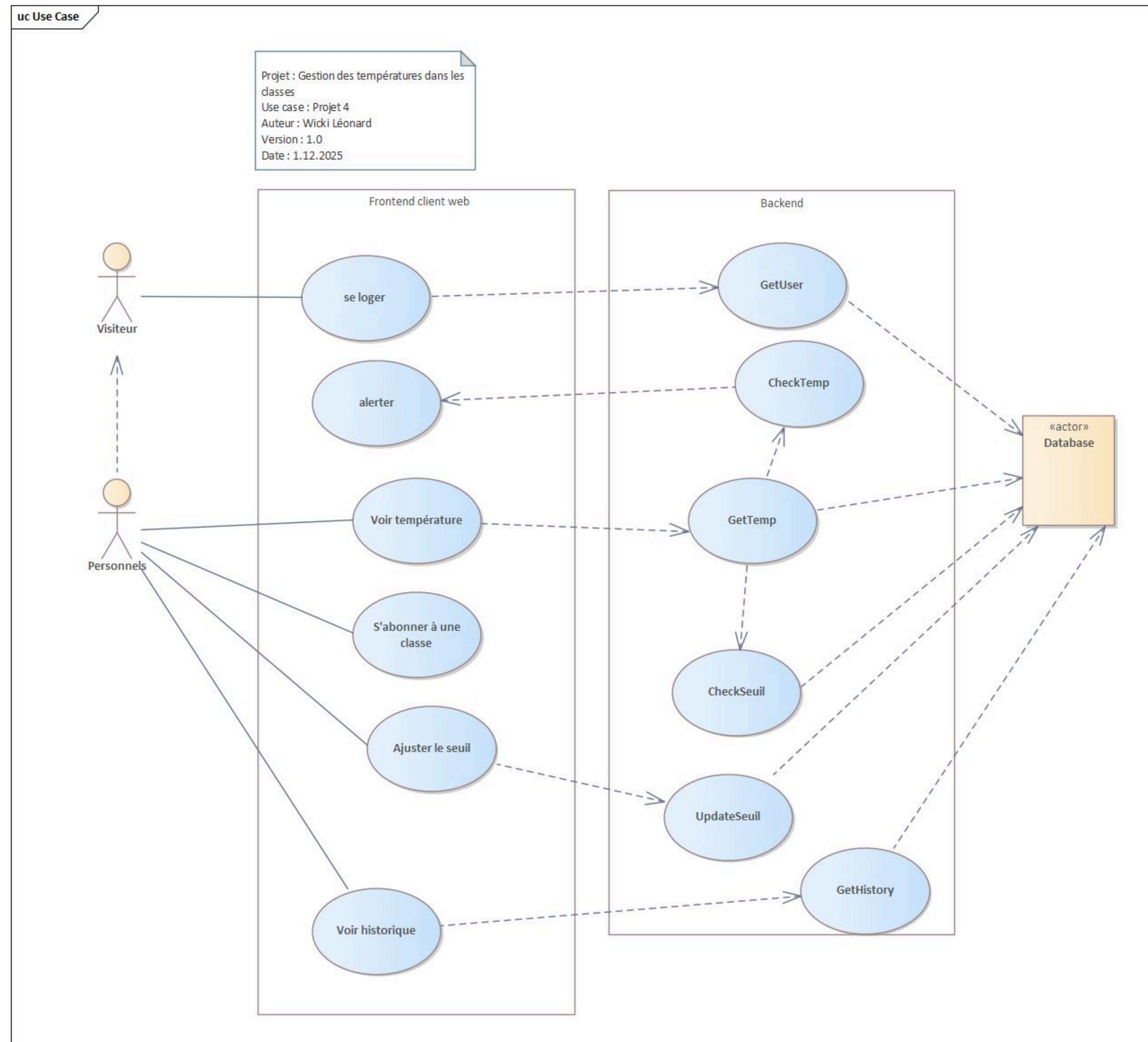
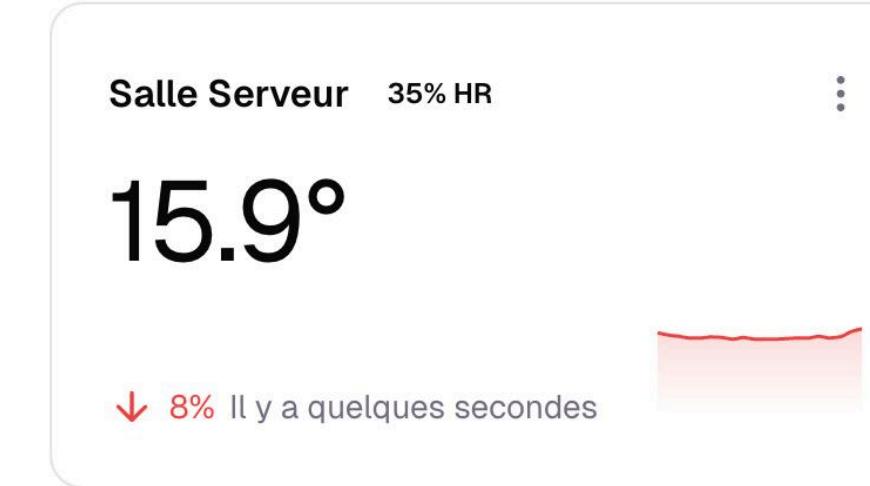


Diagramme Use Case



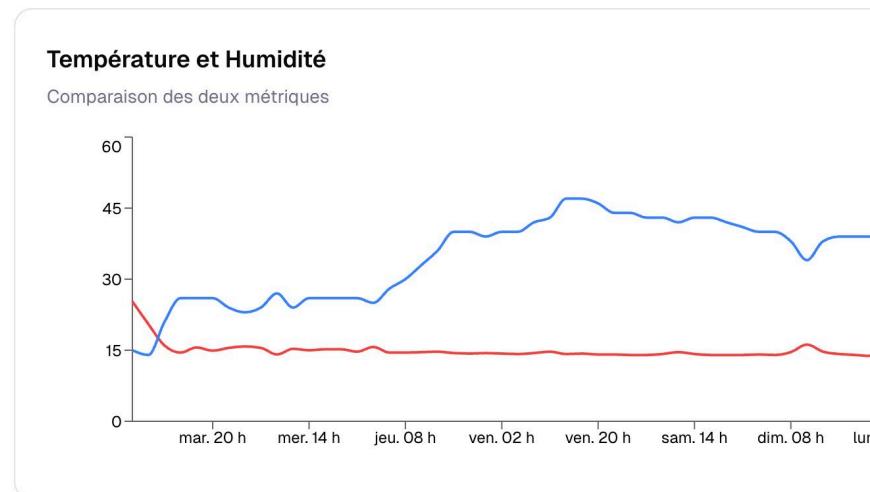
features clés

Du problème à l'objectif



Monitoring en temps réel

Affichage instantané température & humidité

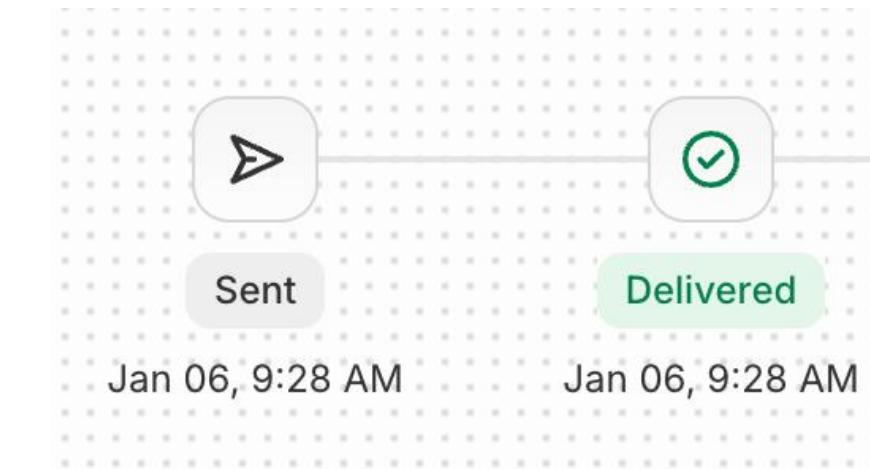


Historique 1 mois

Consultation et analyse des données passées

Configuration form for alert thresholds:

- Temperature minimum (°C): 18
- Temperature maximum (°C): 24
- Alert delay (minutes): 30
- Time before alert triggers if threshold is exceeded
- Humidity minimum (%):
- Humidity maximum (%):



Seuils configurables

Définition d'alertes par mails personnalisées par salle

Système d'alertes

Notifications automatiques en cas de dépassement



SensorHub

🏠 Maison

📊 Tableau de bord

👤 Utilisateurs

📄 Documentation

⚙️ Paramètres

Bon retour leowicki !

Suivez et gérez le climat des salles de classes .

Dernière mise à jour : 11:43:20

☀️ Mode sombre

⟳ Rafraîchir

🌡️ Gérer le seuil par défaut

+ Ajouter

Froid (<18°C)

Optimal (18-24°C)

Chaud (24-28°C)

Très chaud (>28°C)

En hausse

En baisse

Abonnements

Recherche

A42 18% HR

24.1°

⬇️ 2% Il y a quelques secondes

Salle Serveur 35% HR

16.4°

⬇️ 9% Il y a quelques secondes

A45 16% HR

24.1°

⬇️ 4% Il y a quelques secondes



leowicki



SensorHub

- Maison
- Tableau de bord
- Utilisateurs
- Documentation
- Paramètres

Dernières 24h

Dernières 24h

Retard actuel

Dernières 24h

4 heures 1 jour 1 semaine 1 mois

Graphiques de température

Évolution de la température

Données de la dernière semaine

mar. 17 h Température (°C) 15,5

Température et Humidité

Comparaison des deux métriques

Descente de code

Création de graphiques : Récupération de l'historique

```
1  /**
2   * GET /api/rooms/:roomId/history?period=4h
3   * Récupère l'historique des données d'une salle
4   * Paramètres query: period (4h, 1d, 1w, 1m)
5   */
6  export const getRoomHistory = async (req, res) => {
7    const { roomId } = req.params;
8    const { period = '4h' } = req.query;
9
10   // Calculer la date de début selon la période
11   switch (period) {
12     case '4h':
13       startDate.setHours(now.getHours() - 4);
14       intervalMinutes = 5; // Un point toutes les 5 minutes
15       break;
16     case '1d':
17       startDate.setDate(now.getDate() - 1);
18       intervalMinutes = 30; // Un point toutes les 30 minutes
19       break;
20     case '1w':
21       startDate.setDate(now.getDate() - 7);
22       intervalMinutes = 180; // Un point toutes les 3 heures
23       break;
24     case '1m':
25       startDate.setMonth(now.getMonth() - 1);
26       intervalMinutes = 720; // Un point toutes les 12 heures
27       break;
28     default:
29       startDate.setHours(now.getHours() - 4);
30       intervalMinutes = 5;
31   } // ...
```

Descente de code

Création de graphiques : intégration du backend et du frontend.

```
1 // Récupérer les capteurs de température et humidité
2 const tempSensor = room.sensors.find(s => s.type ===
3   'TEMPERATURE');
4 const humiditySensor = room.sensors.find(s => s.type ===
5   'HUMIDITY');
6
7 // Récupérer les readings depuis startDate
8 const [tempReadings, humidityReadings] = await Promise.all([
9   tempSensor ? prisma.sensorReading.findMany({
10     where: {
11       sensorId: tempSensor.id,
12       timestamp: { gte: startDate }
13     },
14     orderBy: { timestamp: 'asc' }
15   }) : [],
16   humiditySensor ? prisma.sensorReading.findMany({
17     where: {
18       sensorId: humiditySensor.id,
19       timestamp: { gte: startDate }
20     },
21     orderBy: { timestamp: 'asc' }
22   }) : []
23 ]);
```

Descente de code

Récupérer les données par intervalle en base de données

```
1 // Regrouper les données par intervalles
2 const groupedData = [];
3 const intervalMs = intervalMinutes * 60 * 1000;
4 let currentTime = startDate.getTime();
5
6 while (currentTime <= now.getTime()) {
7     const nextTime = currentTime + intervalMs;
8
9     // Trouver les readings dans cet intervalle
10    const tempInInterval = tempReadings.filter(r => {
11        const t = r.timestamp.getTime();
12        return t >= currentTime && t < nextTime;
13    });
14
15    const humidityInInterval = humidityReadings.filter(r => {
16        const t = r.timestamp.getTime();
17        return t >= currentTime && t < nextTime;
18    });
}
```

Descente de code

Calculer la moyenne pour chaque intervalle

```
1 // Calculer la moyenne pour cet intervalle
2 const avgTemp = tempInInterval.length > 0
3   ? tempInInterval.reduce((sum, r) => sum + r.value, 0) /
tempInInterval.length
4   : null;
5
6 const avgHumidity = // same as before
7
8 // grouper les data par moyenne d'intervalle
9 if (avgTemp !== null || avgHumidity !== null) {
10   groupedData.push({
11     timestamp: new Date(currentTime),
12     temperature: avgTemp !== null ? Math.round(avgTemp *
13       10) / 10 : null,
14     humidity: avgHumidity !== null ?
Math.round(avgHumidity) : null
15   });
16
17 // format de la réponse .json
18 res.json({
19   roomId,
20   period,
21   data: groupedData
22 });
23 } catch (error) {
24   res.status(500).json({ error: 'Erreur lors de la
récupération de l\'historique' });
25 }
```

Descente de code

Depuis le frontend : fetch les données via la route API

```
1 const response = await fetch(  
2   `${API_BASE_URL}/api/rooms/${room.id}/history?  
3   period=${selectedPeriod}`  
4 )  
5 const result = await response.json()  
6
```

Descente de code

Formatter et afficher les données et mettre les données en cache (State React)

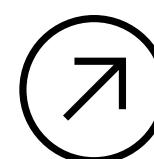
```
1  // Format data for chart
2  const formattedData = result.data.map((item: any) => {
3    switch (selectedPeriod) {
4      case '4h':
5      case '1d':
6        // ...
7        break
8      case '1w':
9        // ...
10     break
11    case '1m':
12      // ...
13      break
14    default:
15      timeFormat = date.toLocaleTimeString('fr-FR', {
16        hour: '2-digit',
17        minute: '2-digit'
18      })
19    }
20  }
21
22  return {
23    time: timeFormat,
24    temperature: item.temperature ?? 0,
25    humidity: item.humidity ?? 0
26  }
27})
28 setHistoricalData(formattedData)
29 // Mettre en cache les données
30 setDataCache(prev => ({ ...prev, [selectedPeriod]: formattedData }))
```

Descente de code

En cas d'erreur, afficher un Mockup de données

```
1 } catch (error) {
2   console.error("Erreur lors de la récupération de l'historique
: ", error)
3
4   // Fallback to simulated data if API fails
5   const now = new Date()
6
7   const fallbackData = Array.from({ length: 48 }, (_, index) =>
8     {
9       const time = new Date(
10         now.getTime() - (48 - index - 1) * 5 * 60 * 1000
11       )
12
13       return {
14         time: time.toLocaleTimeString('fr-FR', {
15           hour: '2-digit',
16           minute: '2-digit'
17         }),
18         temperature: room.temperature + (Math.random() - 0.5) * 4,
19         humidity: room.humidity + (Math.random() - 0.5) * 15
20       }
21
22     setHistoricalData(fallbackData)
23   } finally {
24     setIsLoadingHistory(false)
25   }
```

Roadmap les prochaines releases



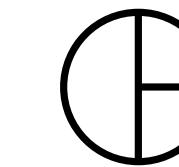
Version 3.1

- Update de sécurité avancé
- gestion de compte, paramètres configurable



Version 3.5

- Notification par l'application
- Gestion de seuils par utilisateurs et par salle



Version 4.0

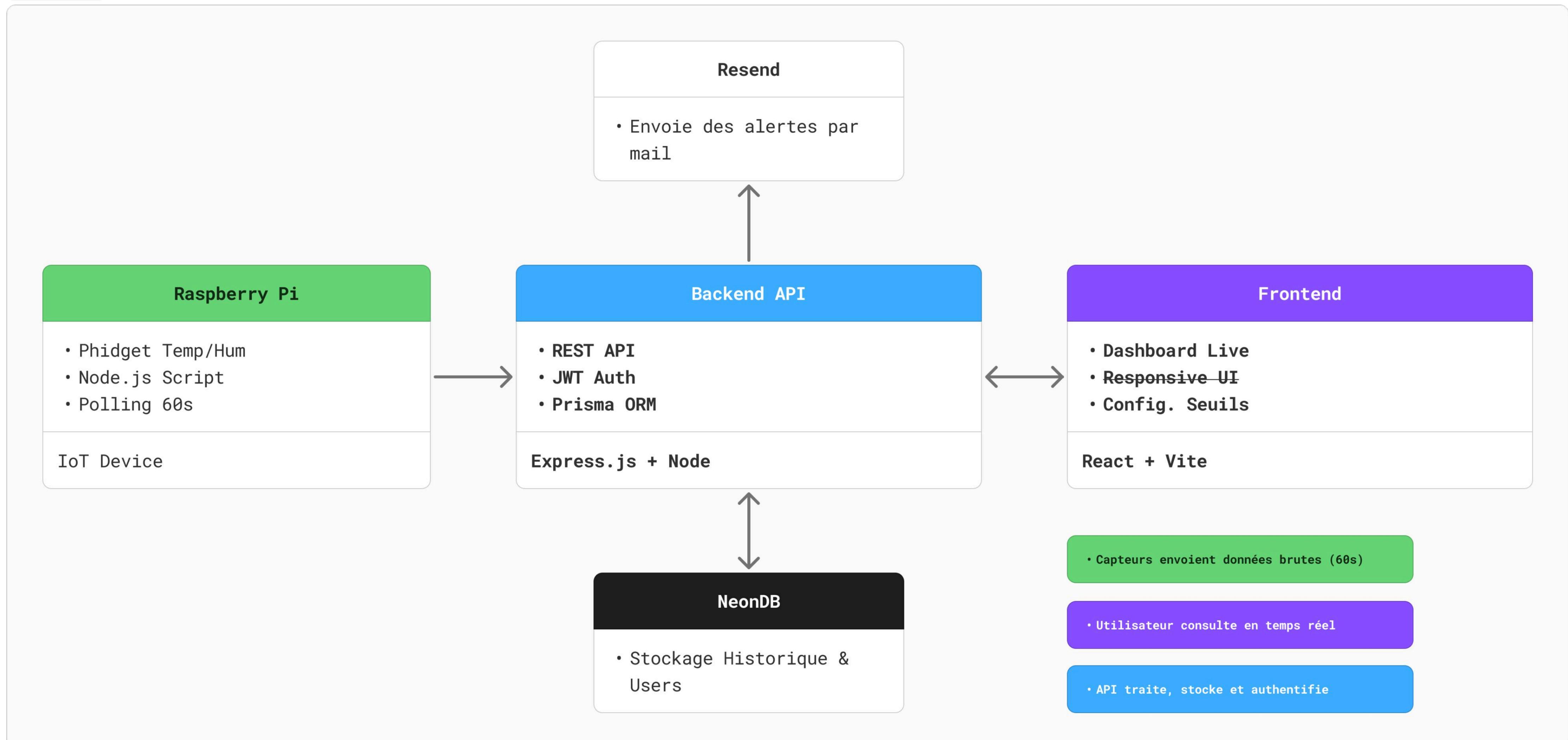
- Interface Web responsive
- Contrôle HVAC automatique
- Application native mobile

API REST Endpoints

Méthode	Endpoint	Auth	Description
GET	/rooms	Public	lister les salles
POST	/rooms/:roomId/readings	Public	intégré des lectures (RPI)
GET	/sensors	API-Key	Lister les capteurs
POST	/auth/login	Public	Connexion (JWT)
POST	/subscriptions/rooms/:roomId	Public	S'abonner
DELETE	/subscriptions/rooms/:roomId	Public	Se désabonner

```
1 curl -X POST "http://localhost:3000/api/rooms/
<roomId>/readings" \
2   -H "Content-Type: application/json" \
3   -H "X-API-Key: <RPI_API_KEY>" \
4   -d '{ "temperature": 22.4, "humidity": 41.0 }'
5
6 // Lecture capteur :
7 curl -X POST "http://localhost:3000/api/sensors/
8   readings" \
9   -H "Content-Type: application/json" \
10  -H "X-API-Key: <RPI_API_KEY>" \
11  -d '{ "serialNumber": "PHIDGET-123", "value": 
12  22.4 }'
13
14 // Réponse attendue (201 Created)
15 HTTP/1.1 201 Created
16 Content-Type: application/json
17 {
18   "success": true,
19   "reading": {
20     "id": "reading_123456789",
21     "timestamp": "2026-01-12T10:30:00.000Z"
22   }
23 }
```

Architecture Globale



Technologies

Les outils utilisé dans le projet Sensorhub



Vercel

Déployer, héberger et scaler des applications web modernes.



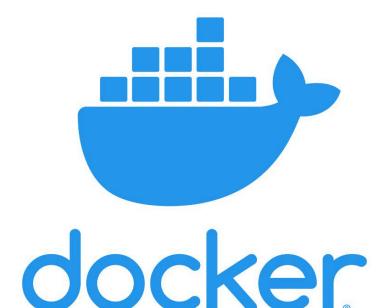
Resend

Envoi d'emails transactionnels fiable pour les applications.



Prisma + NeonDB

Prisma fournit un ORM typé pour interagir avec la base de données, tandis que NeonDB offre une base PostgreSQL serverless, rapide et élastique.



Docker

Environnements de développement et de production standardisés.

Installation et Intégration d'un nouveau Phidget dans le système

- 1 ✓ Docker installé.
- 2 ✓ Plugin docker compose déjà présent.
- 3 🛣 Configuration de la salle et des capteurs...
- 4 📁 Nom de la **salle** (ex: **C114**): testPourDemo
- 5 🖊 Description de la **salle** (optionnel):
- 6 🌡 TEMP_SENSOR_ID (ex: PHIDGET-TEMP-001): PHIDGET-TEMP-004
- 7 💧 HUMIDITY_SENSOR_ID (ex: PHIDGET-HUM-001): PHIDGET-HUM-004
- 8 🕒 Création de la **salle** et des capteurs dans la base de données...
- 9 ✓ Salle créée avec succès ! ID: mn6Bop
- 10 📄 Fichier .env généré.
- 11 🛠 Génération docker-compose.yml...
- 12 🚀 Déploiement avec docker **compose** (image gabrielbeer15/phidget-client:latest)...

Nom de la salle

001

Définissez l'espace physique où les capteurs seront installés.

Configuration des Phidget

002

Connectez les capteurs de température et d'humidité.

Création de la salle en base de données

003

La **salle** et ses capteurs sont enregistrés dans le système.

Docker Compose

004

Le client est lancé et commence à collecter les données.

Démonstration

SENSORHUB