



Haute École de la Province de Liège

Master en sciences de l'ingénieur industriel, orientation informatique

SmartCities & IoT

Rapport Nichoir connecté

Rapport réalisé par :

MARTIN Maximilien

AL-ZUBAIDI Mohammed

Année académique 2025-2026

Table des matières

1	INTRODUCTION ET ANALYSE DU BESOIN	5
1.1	Contexte	5
1.1.1	Enjeux du projet	5
1.2	Architecture globale	5
1.2.1	Le nœud hardware (Capture)	5
1.2.2	La passerelle et le stockage (Traitement)	5
1.2.3	Interface utilisateur (Consultation)	6
1.2.4	Schéma bloc fonctionnel	6
2	CONCEPTION HARDWARE ET INTÉGRATION	7
2.1	Choix des composants	7
2.1.1	M5Stack TimerCAM	7
2.1.2	PIR (BS612)	7
2.1.3	LED IR (BIR-HO033A-TRB)	7
2.1.4	Batterie LiPo 2000	8
2.1.5	Régulateur de tension (APD150AUZ-3.3-R7)	8
2.1.6	Raspberry Pi	9
2.2	Schémas électriques	10
2.2.1	Schéma bloc	10
Le schéma ci-dessous illustre l'architecture électrique du nichoir, conçue pour maximiser l'autonomie tout en garantissant la stabilité des composants.	10	
• Gestion de l'énergie : La source principale est une batterie LiPo de 2000 mAh. Comme sa tension varie entre 4,2 V et 3,0 V, l'utilisation du régulateur APD150AUZ-3.3-R7 est indispensable pour fournir un 3,3 V constant à la TimerCAM et au capteur PIR.	10	
• Flux logique de réveil : Le système est optimisé pour rester en mode Deep Sleep afin de ne consommer que 2µA au repos. Le PIR (BS612) agit comme l'élément déclencheur : dès qu'une présence est détectée, il envoie une impulsion de 3,3 V vers la broche de réveil de la TimerCAM.	10	
• Transmission : Une fois réveillée, la TimerCAM capture l'image et l'envoie via Wi-Fi au Raspberry Pi avant de se remettre instantanément en veille pour économiser l'énergie.	10	
2.2.2	Schéma de câblage	11
2.3	Intégration physique (Le nichoir)	11
3	PARTIE EMBARQUÉE (ESP32 & TIMERCAM)	12
3.1	Algorigramme	12
3.2	Gestion de la batterie	13
3.3	Gestion de l'énergie	13
3.3.1	Stratégie de mise en veille	13
3.3.2	Réveil par interruption vs réveil temporel	13
3.4	La communication	14
4	PARTIE SERVEUR (RASPBERRY PI, BDD & WEB)	15
4.1	Architecture logicielle du Pi	15

4.2	Base de données.....	15
4.3	Interface web.....	16
4.3.1	Fonctionnement technique et accès aux images.....	16
4.3.2	Galerie et Responsivité	16
5	TESTS, CARACTÉRISATION ET FIABILITÉ (C2)	17
5.1	Bilan énergétique	17
5.1.1	Calcul théorique de l'autonomie	17
5.1.2	Mesures réelles	17
5.1.3	Conclusion	17
5.2	Fiabilité de la détection	18
5.3	Fiabilité du code	18
6	RÉFLEXIVITÉ ET GESTION DE PROJET (C3).....	18
6.1	Gestion du travail et collaboration (Binôme).....	18
6.2	Auto-formation et acquis.....	19
6.3	Difficultés et Améliorations (V2)	19
6.3.1	Difficultés techniques :	19
6.3.2	Axes d'amélioration pour une version 2 :	19
7	Annexes	21
7.1	Annexe A : Captures d'écran de l'interface	21
7.2	Annexe B : Consommation en usage intensif	23

Tables des illustrations

Figure 1 : Schéma bloc fonctionnel	6
Figure 2 : Évolution de Vout en fonction de Vin pour les différents courants de charge (Iout)	9
Figure 3 : Schéma bloc (électrique).....	10
Figure 4 : Schéma de câblage (Gerber).....	11
Figure 5 : Algorigramme principe de fonctionnement	12
Figure 6 : ERD	15
Figure 7 : Photothèque	21
Figure 8 : Albums	21
Figure 9 : Favoris.....	22
Figure 10 : Affichage d'une image (détails).....	22
Figure 11 : Photothèque (toutes les photos)	22
Figure 12 : Avertissement batterie.....	23
Figure 13 : Consommation en usage intensif	23

1 INTRODUCTION ET ANALYSE DU BESOIN

1.1 Contexte

Ce projet a été réalisé dans le cadre du cours de **SmartCities**. L'idée est de concevoir un nichoir connecté capable de surveiller la nidification des oiseaux de manière autonome, tout en restant sur une solution à bas coût.

1.1.1 Enjeux du projet

Le développement s'est concentré sur trois contraintes majeures :

- **Accessibilité** : Le coût total du matériel doit rester sous la barre des 50 €.
- **Autonomie** : Le système doit pouvoir tenir environ 6 mois sur batterie grâce à une gestion optimisée de la mise en veille.
- **Évolutivité** : La conception permet d'ajouter un panneau solaire par la suite pour augmenter la durée de vie.

1.2 Architecture globale

Le système est structuré autour de trois pôles principaux qui communiquent entre eux pour assurer la capture, la transmission et la consultation des données.

1.2.1 Le nœud hardware (Capture)

Situé à l'intérieur du nichoir, ce bloc est responsable de l'acquisition des données :

- **Unité centrale** : Module TimerCAM (ESP32) gérant la prise de vue et la connectivité Wi-Fi.
- **Détection** : Un capteur PIR (BS-612) réveille le système lors d'un mouvement.
- **Éclairage** : Une LED IR permet de capturer des images claires sans déranger les oiseaux, même dans le noir.

1.2.2 La passerelle et le stockage (Traitement)

Le Raspberry Pi centralise les informations envoyées par le nichoir :

- **Communication** : Un Broker MQTT reçoit les messages contenant les images et le niveau de batterie.
- **Stockage** : Un script Python traite ces messages pour les archiver dans une base de données MariaDB.

1.2.3 Interface utilisateur (Consultation)

La restitution des données se fait via une interface accessible sur smartphone ou ordinateur :

- **Serveur Web** : Développé avec ASP C# .NET Core sur le Raspberry Pi.
- **Affichage** : Une galerie permet de visualiser les photos prises et d'être averti en cas de batterie faible.

1.2.4 Schéma bloc fonctionnel

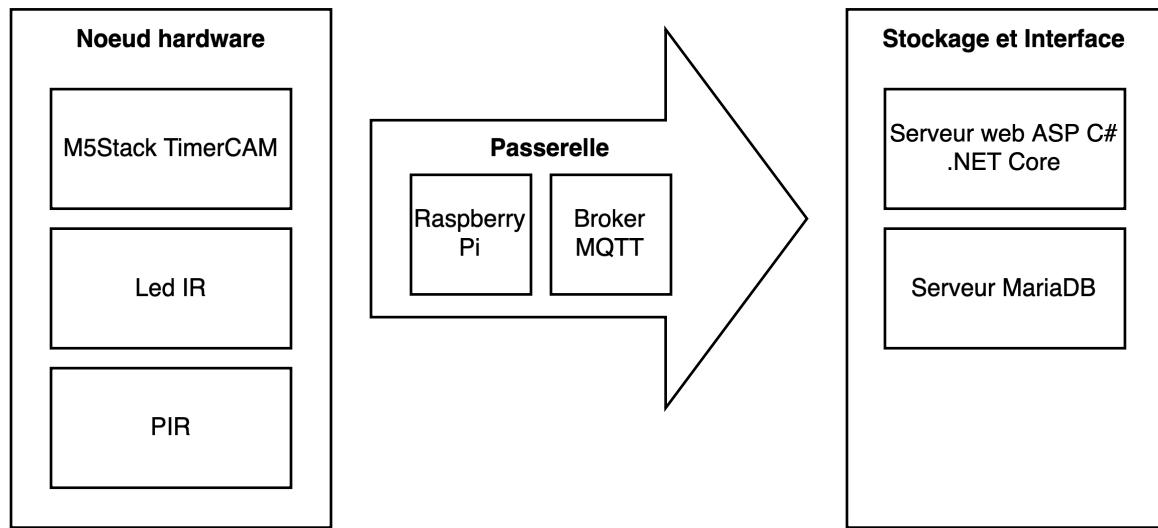


Figure 1 : Schéma bloc fonctionnel

2 CONCEPTION HARDWARE ET INTÉGRATION

2.1 Choix des composants

2.1.1 M5Stack TimerCAM

On peut dire que c'est le composant au centre du projet. Ce module est basé sur un ESP32 qui a été choisi pour sa gestion ultra-basse consommation.

- **Atout technique :** Il possède une puce RTC intégrée (BM8563) capable de réveiller le système, ce qui réduit le courant de veille à seulement 2 μ A.
- **Capteur d'image :** Sa caméra OV3660 (3MP) offre un angle de vue de 66.5°, idéal pour l'espace restreint d'un nichoir.

2.1.2 PIR (BS612)

Le capteur de mouvement infrarouge passif (PIR) agit comme le déclencheur principal du système. Il permet de maintenir la TimerCAM en mode veille prolongée tant qu'aucune activité n'est détectée, optimisant ainsi l'autonomie de la batterie.

Configuration technique et sensibilité :

- **Réglage de la sensibilité :** Nous avons intégré un circuit de réglage sur la broche « SENS » en combinant une résistance fixe et un potentiomètre (100 k Ω et 1 M Ω). Cela permet d'ajuster la distance de détection selon l'environnement du nichoir.
- **Gestion du timing (Broche OEN) :** Nous avons choisi de relier la broche OEN à la masse (GND). Cette configuration matérielle fixe la durée de l'impulsion de sortie à 2 secondes lorsqu'un mouvement est détecté.
- **Traitements logiciels :** Pour éviter les captures en boucle ou les rebonds, cette impulsion de 2 secondes est gérée dans le code de l'ESP32. Une fois le signal reçu et la photo prise, le script ignore les nouvelles impulsions pendant un intervalle défini (timer de 300s) avant de repasser en mode écoute.

2.1.3 LED IR (BIR-HO033A-TRB)

Pour observer les oiseaux sans perturber leur cycle naturel, nous avons intégré une LED infrarouge à la conception. Ce type d'éclairage permet d'obtenir des images nettes dans l'obscurité du nichoir, le capteur de la TimerCAM étant sensible à cette longueur d'onde, sans pour autant émettre de lumière visible gênante.

Validation technique :

- **Test de polarité** : Lors du montage, nous avons utilisé le mode test de diode d'un multimètre pour identifier l'anode et la cathode et valider le sens de branchement.
- **Diagnostic** : Malgré les tests de continuité et l'application d'une tension nominale, le composant n'émettait aucun signal.
- **Décision** : La LED a été diagnostiquée comme défectueuse. Par souci de fiabilité et pour éviter tout court-circuit sur la carte à pastilles, nous avons choisi de ne pas l'implanter sur le prototype final tout en conservant l'architecture logicielle prête à l'accueillir.

2.1.4 Batterie LiPo 2000

Une batterie lithium-polymère de 2000 mAh a été sélectionnée pour son rapport capacité/taille. Grâce au courant de veille extrêmement faible du projet, elle permet d'atteindre l'objectif d'autonomie de 6 mois.

Caractéristiques de fonctionnement :

- **Tension nominale (3,7 V)** : Valeur de référence utilisée pour le dimensionnement électrique et le calcul de l'autonomie du nichoir.
- **Plage de charge (4,2 V - 3,0 V)** : La tension varie de 4,2 V à pleine charge jusqu'à un seuil de décharge recommandé de 3,0 V pour préserver la chimie de la batterie.
- **Sécurité et coupure** : En dessous de 2,7 V, la batterie entre en zone critique. Le système a été conçu pour transmettre le niveau de batterie via MQTT afin que l'utilisateur puisse anticiper la recharge avant d'atteindre ce seuil de dégradation.

2.1.5 Régulateur de tension (APD150AUZ-3.3-R7)

Le régulateur assure une tension stable de 3.3V vers les composants, même lorsque la tension de la batterie diminue. C'est une sécurité indispensable pour protéger l'ESP32 et garantir la qualité du signal Wi-Fi.

Justification technique :

- **Stabilité de la tension** : Il garantit une sortie constante de 3.3 V, indispensable pour le bon fonctionnement de l'ESP32 et la stabilité du signal Wi-Fi, même lorsque la tension de la batterie LiPo chute avec la décharge.
- **Faible chute de tension (LDO)** : Ce régulateur possède une très faible tension de déchet (dropout), ce qui permet d'extraire un maximum d'énergie de la batterie avant que le système ne s'éteigne.

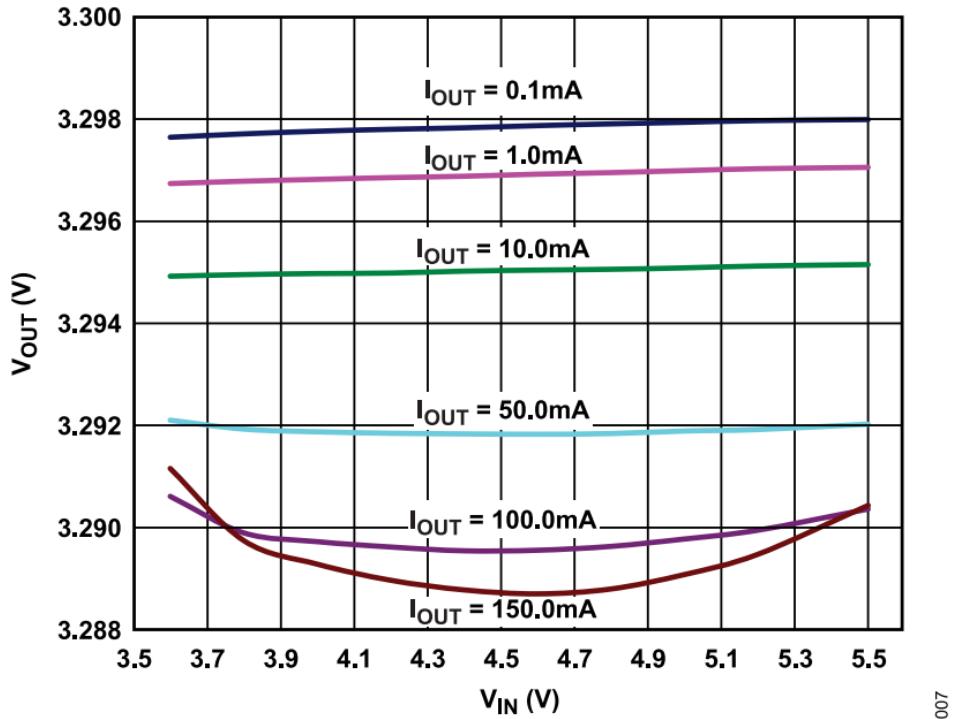


Figure 2 : Évolution de V_{out} en fonction de V_{in} pour les différents courants de charge (I_{out})

Ce graphique montre l'évolution de la tension de sortie (V_{out}) en fonction de la tension d'entrée (V_{in}) pour différents courants de charge (I_{out}).

On observe que, pour une large plage de tensions d'entrée comprise entre environ 3,6 V et 5,5 V, la tension de sortie reste quasi constante autour de 3,3 V. Cela indique que le régulateur assure correctement sa fonction de régulation de tension, indépendamment des variations de V_{in} .

Lorsque le courant de sortie augmente (jusqu'à 150 mA), une légère chute de V_{out} apparaît. Cette baisse reste toutefois très limitée (quelques millivolts).

2.1.6 Raspberry Pi

Utilisé comme **passerelle (gateway)**, il reste allumé 24h/24 en dehors du nichoir. Il héberge le broker MQTT pour recevoir les messages, la base de données MariaDB pour le stockage, et le serveur ASP C# .NET Core pour l'interface web.

2.2 Schémas électriques

2.2.1 Schéma bloc

Le schéma ci-dessous illustre l'architecture électrique du nichoir, conçue pour maximiser l'autonomie tout en garantissant la stabilité des composants.

- **Gestion de l'énergie :** La source principale est une batterie LiPo de 2000 mAh. Comme sa tension varie entre 4,2 V et 3,0 V, l'utilisation du régulateur APD150AUZ-3.3-R7 est indispensable pour fournir un 3,3 V constant à la TimerCAM et au capteur PIR.
- **Flux logique de réveil :** Le système est optimisé pour rester en mode Deep Sleep afin de ne consommer que $2\mu\text{A}$ au repos. Le PIR (BS612) agit comme l'élément déclencheur : dès qu'une présence est détectée, il envoie une impulsion de 3,3 V vers la broche de réveil de la TimerCAM.
- **Transmission :** Une fois réveillée, la TimerCAM capture l'image et l'envoie via Wi-Fi au Raspberry Pi avant de se remettre instantanément en veille pour économiser l'énergie.

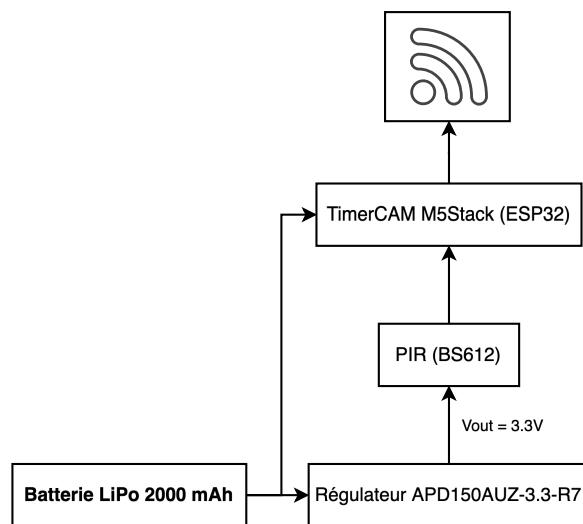


Figure 3 : Schéma bloc (électrique)

2.2.2 Schéma de câblage

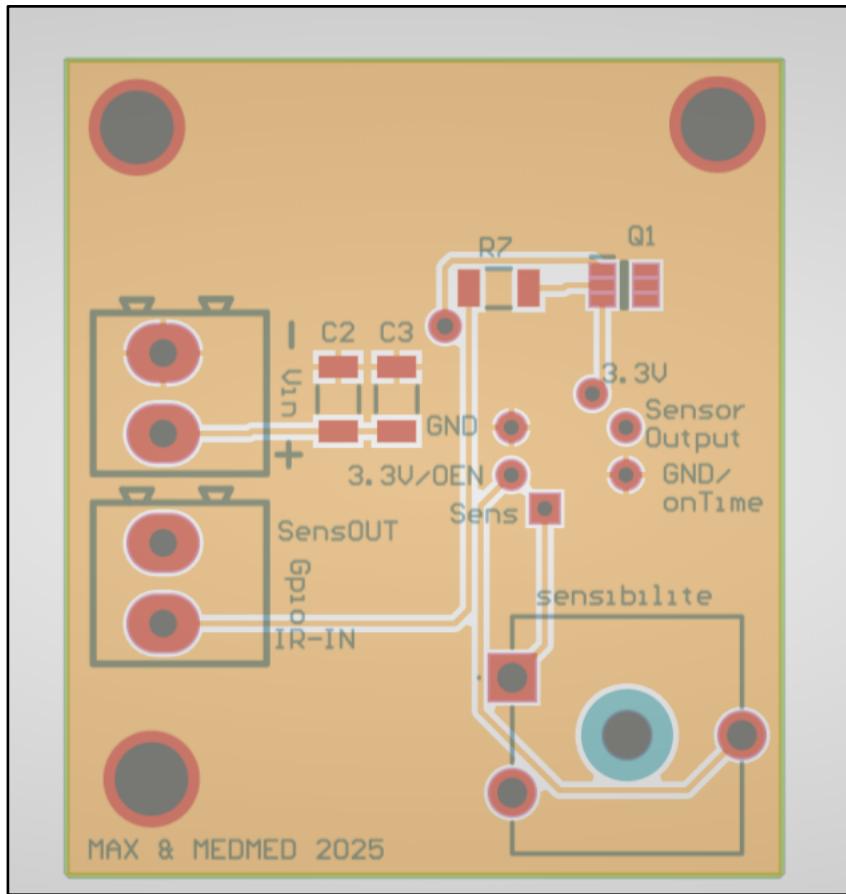


Figure 4 : Schéma de câblage (Gerber)

2.3 Intégration physique (Le nichoir)

L'intégration a été optimisée par une approche « plafonnier » pour maximiser la protection et la visibilité.

- **Fixation et Boîtier :** Nous avons utilisé un boîtier acheté sur internet, dimensionné selon les mesures prises sur le nichoir de classe. L'électronique (PCB et batterie) y est fixée par des colsons et des attaches bananes, garantissant la stabilité de l'ensemble.
- **Positionnement zénithal :** Le boîtier est fixé sous le toit, parallèlement au sol. La caméra, le PIR et la LED IR sont orientés verticalement vers le bas.
- **Champ de vision :** Bien que la caméra soit à plat, son angle d'ouverture de 66.5° permet de couvrir toute la surface au sol sans inclinaison physique, capturant l'oiseau dès son entrée.

3 PARTIE EMBARQUÉE (ESP32 & TIMERCAM)

3.1 Algorigramme

L'intelligence du nichoir repose sur un cycle optimisé pour minimiser le temps d'éveil de l'ESP32. Le programme ne suit pas une boucle infinie classique, mais fonctionne par cycles de « réveil - action – sommeil ».

- **Phase d'initialisation :** À chaque réveil, l'ESP32 initialise les périphériques (Caméra, Wi-Fi) et vérifie la source du réveil (PIR ou Timer interne).
- **Logique de décision :** Si le PIR a détecté un mouvement, la capture d'image est lancée.
- **Fin de cycle :** Le système commande à la puce RTC de couper l'alimentation pour effectuer un arrêt matériel.

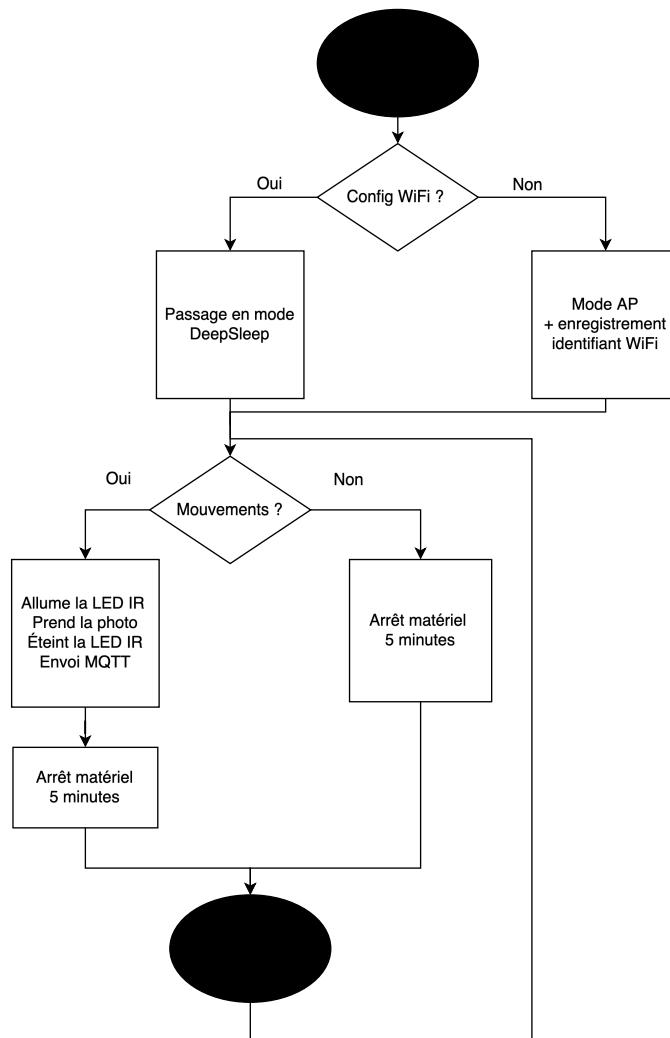


Figure 5 : Algorigramme principe de fonctionnement

3.2 Gestion de la batterie

Pour déterminer le niveau d'énergie restant, nous mesurons la tension de la batterie LiPo.

- **Lecture analogique :** La tension est lue via un pont diviseur de tension interne relié à un ADC (convertisseur analogique-numérique) de l'ESP32.
- **Conversion :** La valeur brute est convertie en tension (V) puis en pourcentage (%) en suivant la courbe de décharge de la batterie (de 4.2V à 3.0V).
- **Transmission :** Cette donnée est systématiquement concaténée aux messages MQTT pour permettre un suivi de l'autonomie sur l'interface Web.

3.3 Gestion de l'énergie

3.3.1 Stratégie de mise en veille

La survie du projet sur 6 mois repose sur l'utilisation du mode Deep Sleep.

- **Consommation ultra-basse :** En veille, la quasi-totalité de l'ESP32 est hors tension. Seule la puce RTC (BM8563) reste active pour surveiller les interruptions.
- **Courant de veille :** Grâce à cette architecture, nous atteignons une consommation d'environ 2 μ A, ce qui rend l'autodécharge de la batterie plus significative que la consommation du circuit lui-même.

3.3.2 Réveil par interruption vs réveil temporel

Nous avons implémenté deux modes de réveil distincts pour équilibrer surveillance et diagnostic :

- **Réveil par interruption (PIR) :** C'est le mode prioritaire. Le capteur PIR envoie un signal sur une broche GPIO configurée en « Wake-up ». Cela déclenche une prise de photo immédiate dès qu'un oiseau entre dans le nichoir.
- **Réveil temporel (Heartbeat) :** Lorsqu'un mouvement est détecté, les TimerCAM capture l'image et transmet l'ensemble des données (visuel + batterie) au serveur. Une fois l'envoi confirmé, le système initie une coupure matérielle complète de 5 minutes. Cette temporisation est stratégique : elle permet d'éviter les déclenchements redondants pour un même événement, économisant ainsi l'énergie et l'espace de stockage sur le serveur, tout en garantissant une période de repos au système avant la prochaine détection.

3.4 La communication

La communication avec le Raspberry Pi est standardisée via le protocole MQTT pour sa légèreté.

- **Structure des messages :** Les données sont envoyées sous forme de messages contenant :
 - L'image capturée (format binaire ou base64).
 - Le niveau de batterie (en Volts ou %).
 - Le timestamp (géré côté broker lors de la réception).
- **Topics :** Nous utilisons des topics distincts pour séparer les flux d'images des flux de téléémétrie (batterie), facilitant ainsi le traitement par le script Python sur la passerelle.

4 PARTIE SERVEUR (RASPBERRY PI, BDD & WEB)

4.1 Architecture logicielle du Pi

Le Raspberry Pi fait office de passerelle (gateway) et de serveur d'application. Son architecture repose sur trois briques logicielles qui communiquent de manière asynchrone :

- **Le Broker MQTT (Mosquitto)** : Il reçoit les messages envoyés par la TimerCAM en temps réel.
- **Le Listener MQTT (Python)** : Un script Python tourne en arrière-plan. Il écoute les topics spécifiques, extrait l'image et le niveau de batterie, enregistre le fichier image dans un dossier local et insère les métadonnées dans la base de données.
- **Le Serveur Web (ASP.NET Core C#)** : Il interroge la base de données pour afficher l'historique des détections et sert les images via une interface utilisateur dynamique.

4.2 Base de données

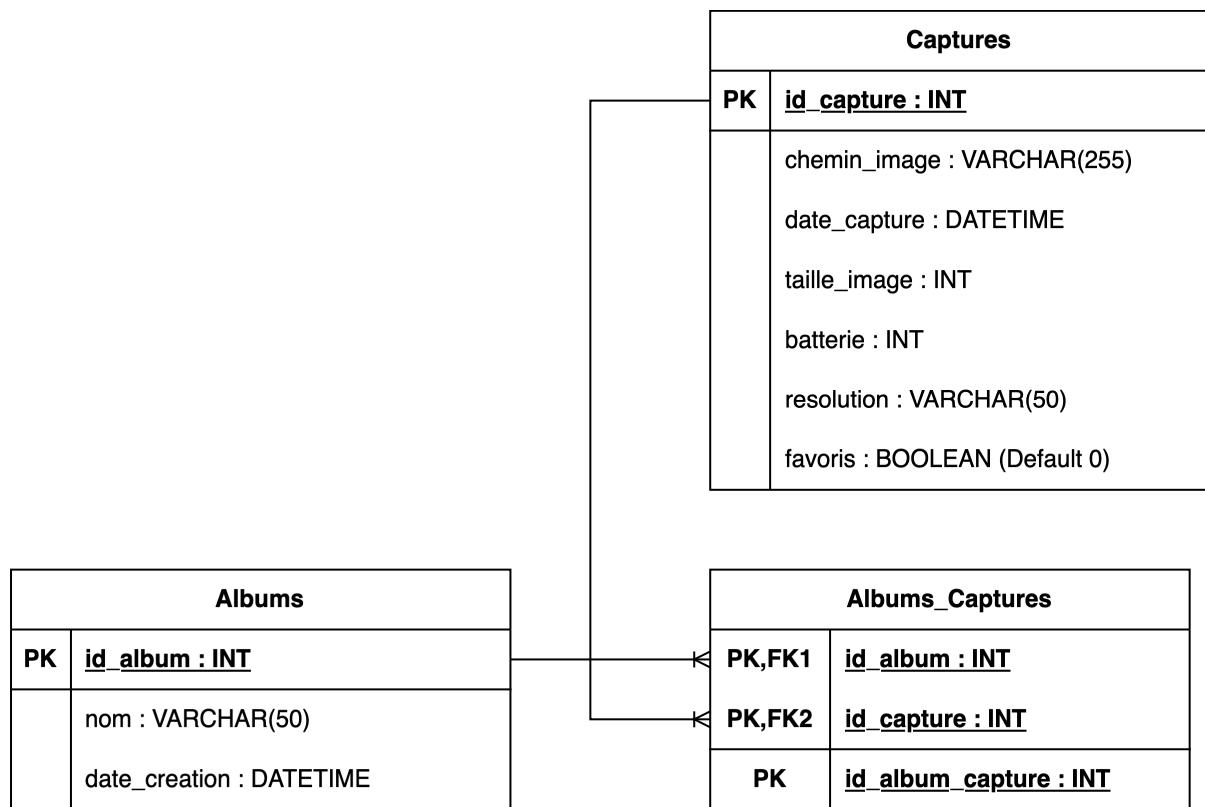


Figure 6 : ERD

4.3 Interface web

L'interface utilisateur a été développée avec le framework ASP.NET Core (C#) en utilisant le patron d'architecture MVC (Modèle-Vue-Contrôleur). Ce choix nous a permis de séparer la logique d'accès aux données de la présentation visuelle. (**voir Annexe A : Captures d'écran de l'interface**).

4.3.1 Fonctionnement technique et accès aux images

Le processus d'affichage d'une image suit un cycle précis lors de chaque requête utilisateur :

- **Le Contrôleur (Logic)** : Lorsqu'un utilisateur accède à la page de la galerie, le contrôleur intercepte la requête. Il utilise Entity Framework Core pour exécuter une requête SQL SELECT sur la table des captures.
- **Récupération des données** : La requête récupère les métadonnées de chaque capture : l'horodatage, le niveau de batterie reçu via MQTT, et surtout le chemin d'accès (URL/Path) de l'image stockée sur le disque du Raspberry Pi.
- **Le Modèle (Data)** : Ces informations sont encapsulées dans un objet « Model » qui est ensuite transmis à la vue.
- **La Vue (Display)** : La vue, rédigée en Razor (.cshtml), parcourt cette liste pour générer dynamiquement le code HTML.

4.3.2 Galerie et Responsivité

- **Affichage dynamique** : La galerie présente les photos sous forme de « cards » élégantes, affichant pour chaque prise de vue l'heure exacte et l'état de santé de la batterie au moment de l'événement.
- **Accès distant** : Grâce à l'hébergement sur le Raspberry Pi, l'interface est accessible depuis n'importe quel appareil connecté au réseau local, permettant une surveillance en temps réel sans intervention manuelle sur le nichoir.

5 TESTS, CARACTÉRISATION ET FIABILITÉ (C2)

5.1 Bilan énergétique

5.1.1 Calcul théorique de l'autonomie

Ce calcul utilise les valeurs maximales des fiches techniques pour garantir un scénario "pire cas".

- **Consommation en mode Veille (Deep Sleep) :**
 - ESP32 TimerCAM : 150µA.
 - Capteur PIR (veille) : $\approx 100\mu\text{A}$ (valeur moyenne de repos pour ce type de capteur).
 - Total Veille : 0,25mA, soit 6mAh/jour.
- **Consommation en mode Actif (10 captures de 10s) :**
 - ESP32 (Mode Normal) : 240mA.
 - PIR (Max) : 100mA.
 - LED IR (Max) : 50mA.
 - Total Actif : 390mA.
 - Consommation journalière active :
 $(10 \text{ détections} \times 10\text{s} \times 390\text{mA}) / 3600 \approx 10,83\text{mAh/jour.}$

Bilan total : La consommation cumulée est de $6 + 10,83 = 16,83\text{mAh/jour}$. L'autonomie théorique est de $3000\text{mAh} / 16,83\text{mAh} \approx 178$ jours.

5.1.2 Mesures réelles

Pour affiner ce bilan, nous avons effectué des relevés sur le prototype :

- **Modes basse consommation :** Les mesures au multimètre indiquent un courant de $4\mu\text{A}$ en mode Shutdown et de $800\mu\text{A}$ en mode Deep Sleep. Bien que le Deep Sleep réel soit supérieur à la théorie, le mode Shutdown permet une économie drastique entre deux réveils.
- **Mode Actif (Oscilloscope) :** Nous avons inséré une résistance de shunt de 1Ω en série avec l'alimentation. La tension mesurée à l'oscilloscope présente un plateau à 150mV , ce qui correspond à un courant de 150mA lors des phases de calcul intensif. (**voir annexe B : Consommation en usage intensif**)

5.1.3 Conclusion

L'objectif de 6 mois (180 jours) est parfaitement tenable. Si le calcul théorique « pire cas » donne 178 jours, nos mesures réelles montrent que le système consomme souvent moins (150 mA en calcul contre 240 mA théorique). En optimisant le passage en mode Shutdown et en limitant les faux positifs du PIR, l'autonomie réelle dépassera largement les 200 jours, sécurisant ainsi toute une saison de nidification.

5.2 Fiabilité de la détection

Le couplage entre le PIR et la TimerCAM a été le point le plus critique du projet :

- **Faux positifs** : Nous avons observé des déclenchements intempestifs. L'analyse a montré que le câblage manuel sur carte à trous favorisait les parasites électriques. Nous sommes donc passé à un PCB.
- **Optimisation** : La fiabilité a été améliorée par deux leviers : le réglage physique de la broche SENS (potentiomètre) et l'implémentation d'un code permettant une coupure matériel de 5 minutes pour ignorer les détections successives trop proches.

5.3 Fiabilité du code

Le firmware a été conçu pour être résilient face aux imprévus du monde réel.

- **Modularité** : Le code est structuré en bibliothèques logiques (Wi-Fi, MQTT, Caméra), facilitant la maintenance et évitant les dépendances bloquantes.
- **Architecture non-bloquante** : L'utilisation de fonctions asynchrones empêche le système de « crasher » si un module (comme le Wi-Fi) est temporairement indisponible.
- **Gestion des erreurs (Fail-safe)** :
 - **Wi-Fi** : En cas de perte de signal, le système tente une reconnexion puis bascule en mode AP (Access Point) pour maintenir un accès de secours.
 - **MQTT** : Si le broker est injoignable, les autres fonctionnalités restent opérationnelles sans interruption du cycle de veille.
- **Diagnostic** : L'intégration de logs détaillés via le moniteur série permet un débogage rapide et un suivi précis de l'état du matériel.

6 RÉFLEXIVITÉ ET GESTION DE PROJET (C3)

6.1 Gestion du travail et collaboration (Binôme)

Pour ce projet, nous avons privilégié une approche collaborative plutôt qu'une simple division des tâches, afin de garantir que chacun maîtrise l'intégralité de la chaîne technique.

- **Méthodologie Agile** : Nous avons utilisé l'outil Projets (Kanban) de GitHub. Grâce au système d'Issues, nous avons listé chaque fonctionnalité (To-Do, In Progress, Done). Cela nous a permis d'avoir une vision claire de l'avancement et de centraliser les priorités.
- **Utilisation de GitHub (Versioning)** : Workflow : Nous avons travaillé avec un dépôt centralisé comprenant une branche principale (main) et deux branches de développement spécifiques (Maximilien et Mohammed).

- **Collaboration** : Une fois les concepts de base (MQTT, Raspberry Pi) validés ensemble, nous avons utilisé des « Pull Requests » pour fusionner nos travaux, ce qui nous a permis de réviser le code de l'autre avant l'intégration finale.
- **Répartition des rôles** : Dès que la compréhension commune était acquise, nous nous sommes répartis les tâches spécifiques. Par exemple, Maximilien s'est concentré sur le design du Frontend et la modélisation des diagrammes ERD, tandis que Mohammed a assuré la liaison entre le Frontend et le Backend.

6.2 Auto-formation et acquis

La réussite de ce projet a reposé sur un mélange de connaissances académiques et de curiosité personnelle :

- **Auto-apprentissage (Hors cours)** : Bien que le cours proposait d'autres pistes technologiques, nous avons pris l'initiative d'implémenter notre solution web en ASP.NET Core (C#). Maîtrisant déjà le langage C# pour le développement Backend grâce à notre cursus précédent, nous avons profité de ce projet pour approfondir par nous-mêmes la gestion du Frontend via le système de vues ASP.NET. Cela nous a permis de livrer une interface robuste tout en renforçant notre polyvalence sur cette stack technologique.
- **Compétences mobilisées** : Nous avons réutilisé nos bases en SQL pour la gestion de la base de données MariaDB et nos notions de réseaux pour configurer la communication entre l'ESP32 et la passerelle.

6.3 Difficultés et Améliorations (V2)

Le passage au prototype matériel a révélé plusieurs défis.

6.3.1 Difficultés techniques :

- **Prototypage** : L'usage d'une carte à pastilles a complexifié les soudures et favorisé l'instabilité du signal.
- **Instabilité du PIR** : Des déclenchements aléatoires ont été observés, liés soit à la sensibilité du capteur bas coût, soit à des interférences de câblage.
- **Composants défectueux** : Des difficultés de diagnostic, notamment sur le sens de la **LED IR** au multimètre, ont mené à l'identification d'un composant défectueux lors de l'assemblage.

6.3.2 Axes d'amélioration pour une version 2 :

Pour une future itération, deux axes majeurs sont envisagés pour accroître la fiabilité et l'ergonomie :

- **Optimisation Hardware :**
 - **Autonomie illimitée** : Intégration d'un panneau solaire pour rendre le système totalement autonome.

- **Gestion intelligente de la lumière** : Ajout d'une photorésistance pour n'activer la LED IR que si la luminosité est insuffisante, économisant ainsi l'énergie.
- **Optimisation Software :**
 - **Interface Web** : Amélioration du design et de la responsivité du site pour une consultation plus fluide de la galerie sur mobile.

7 Annexes

7.1 Annexe A : Captures d'écran de l'interface

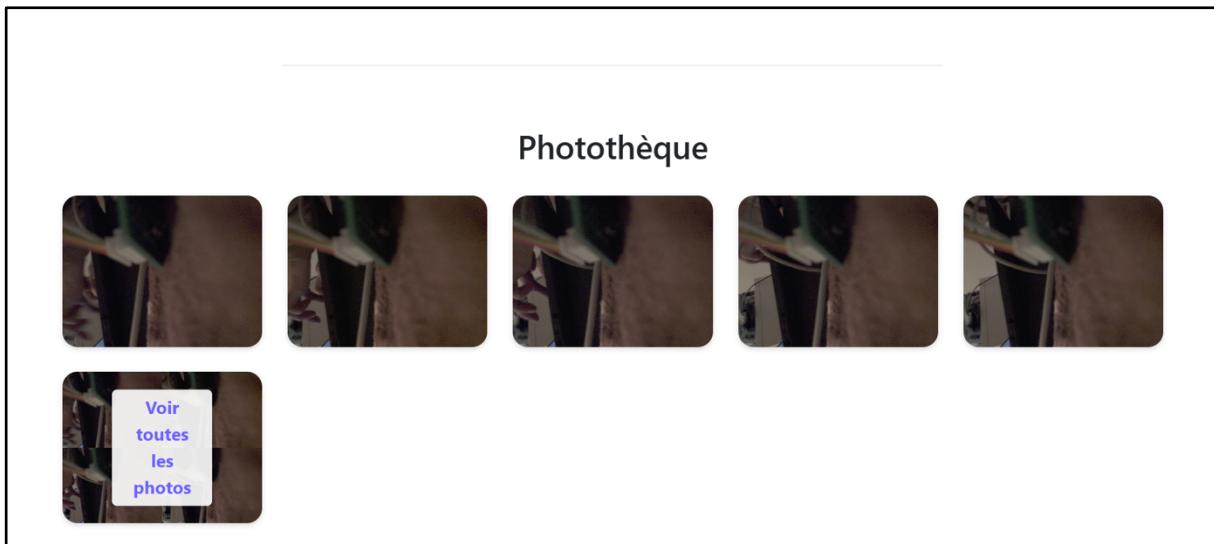


Figure 7 : Photothèque

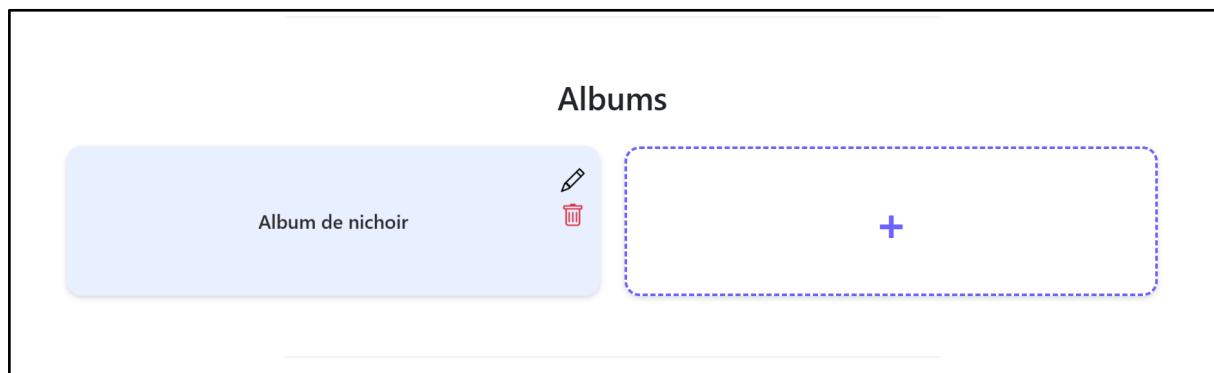
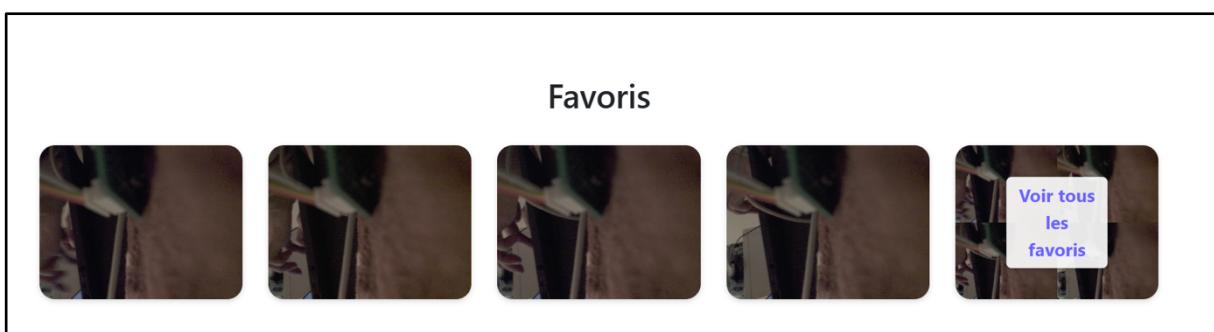


Figure 8 : Albums



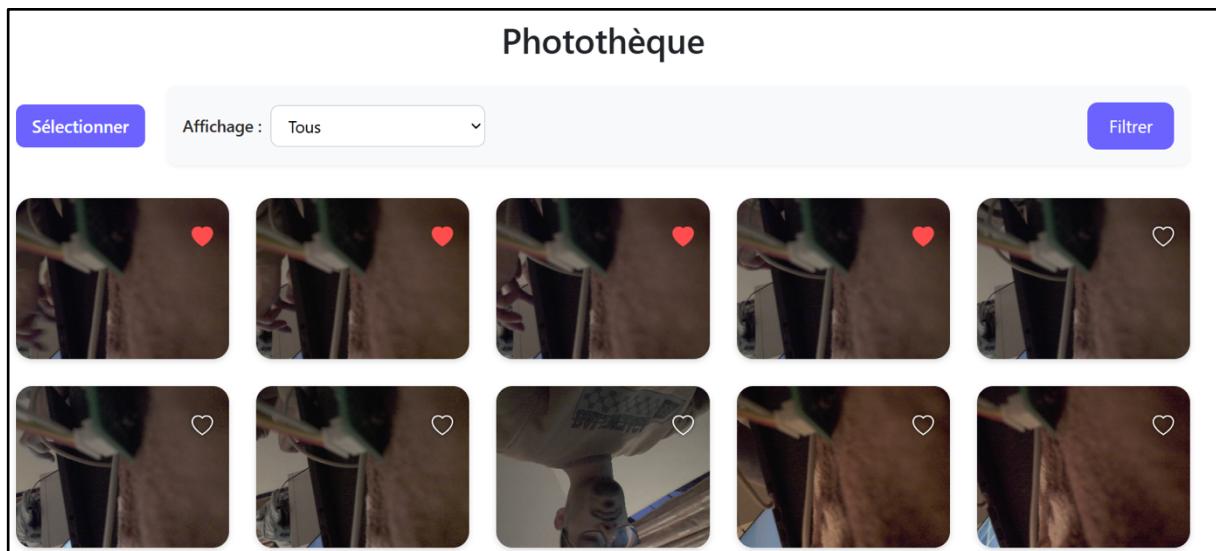


Figure 11 : Photothèque (toutes les photos)

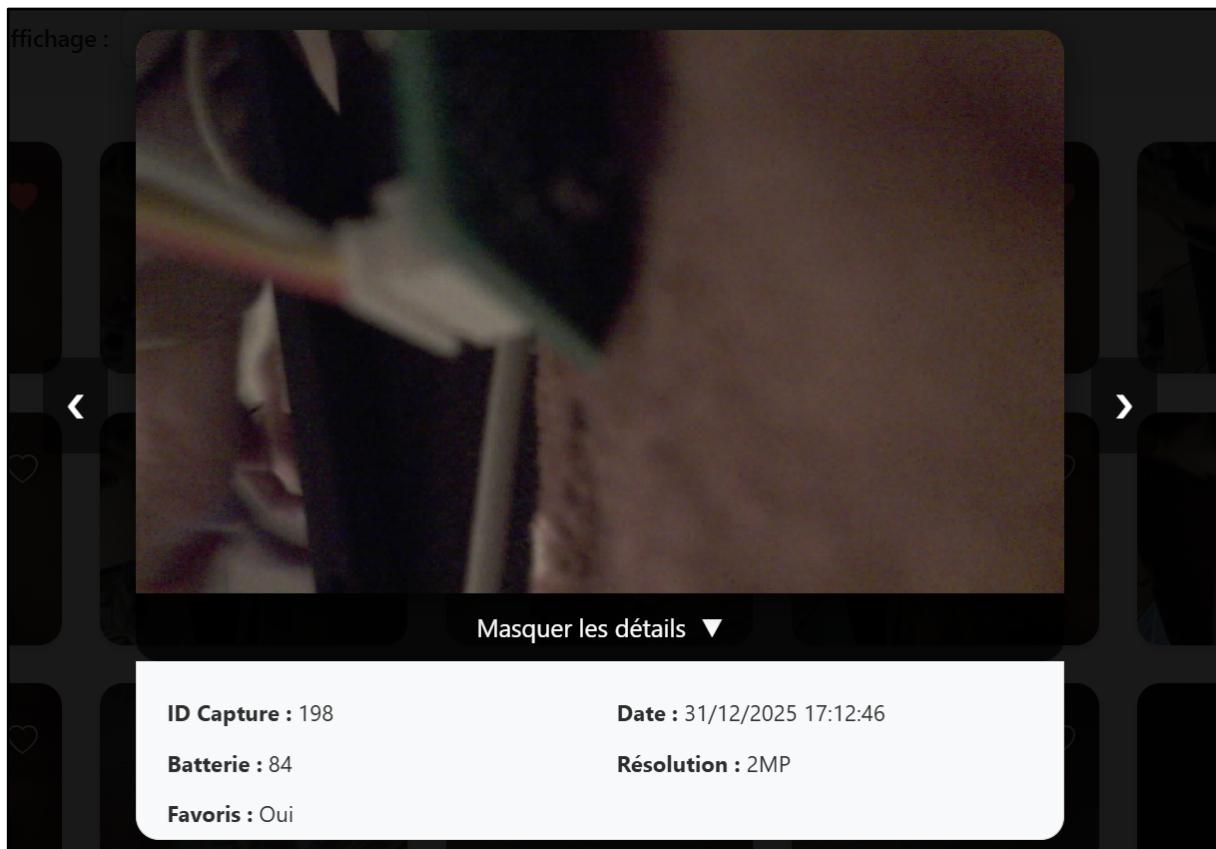


Figure 10 : Affichage d'une image (détails)

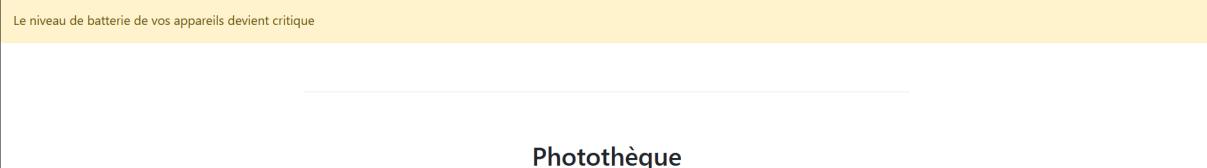


Figure 12 : Avertissement batterie

7.2 Annexe B : Consommation en usage intensif

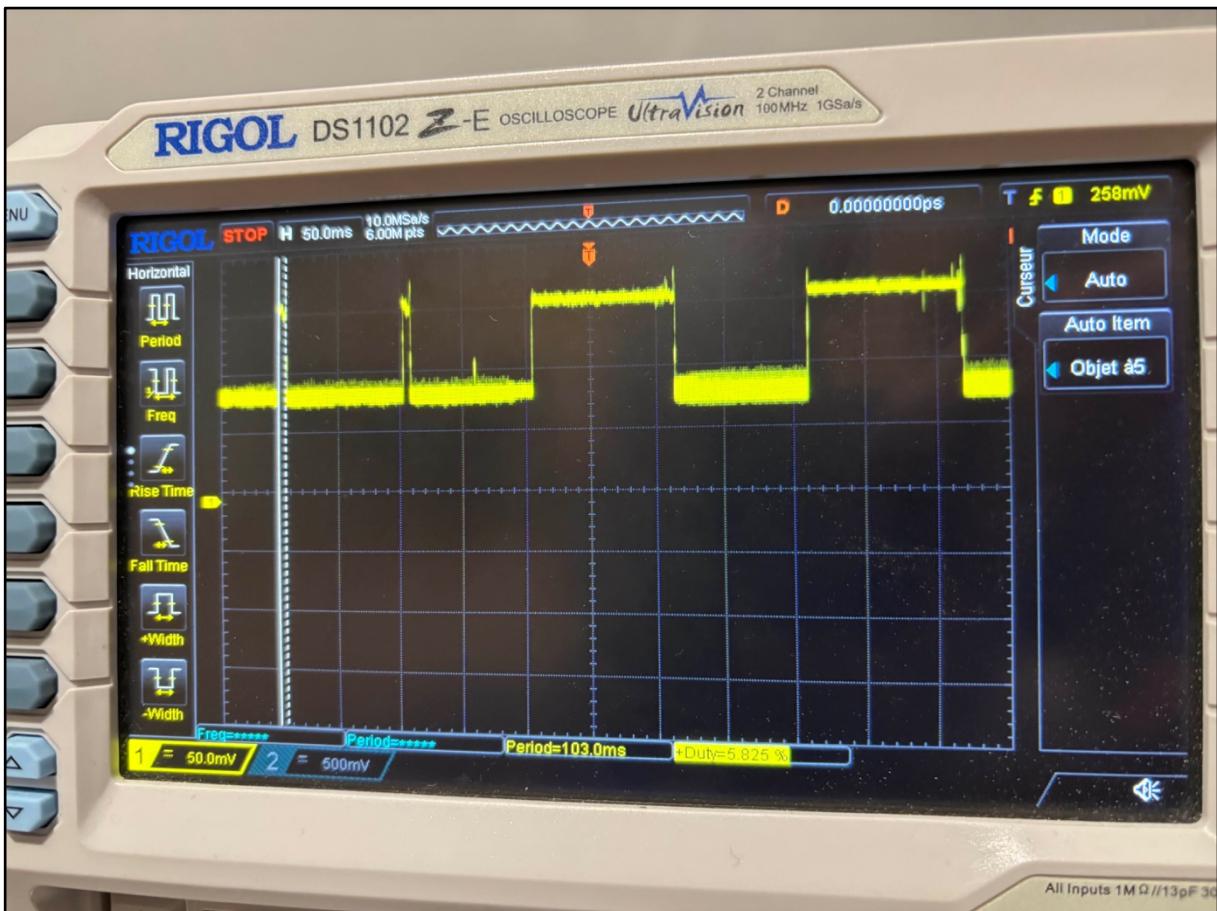


Figure 13 : Consommation en usage intensif