

# Rapport R507 - Dispositifs interactifs

**Tom BOULLAY,  
Noah HEINRICH,  
Clément DE ROBERTI**

**MMI3**

**Université Savoie Mont Blanc**

**17 février 2025**

# Table des matières

|  |           |
|--|-----------|
| <b>1 Présentation</b>  | <b>2</b>  |
| 1.1 Rappel du sujet . . . . .  | 2         |
| 1.2 Notre projet . . . . .   | 2         |
| <b>2 Modèle 3D</b>   | <b>4</b>  |
| 2.1 Réflexion . . . . .  | 4         |
| 2.2 Modélisation . . . . .   | 6         |
| <b>3 Arduino</b>   | <b>8</b>  |
| 3.1 Montage . . . . .  | 8         |
| 3.1.1 V1 . . . . .   | 8         |
| 3.1.2 V2 . . . . .   | 9         |
| 3.1.3 V3 . . . . .   | 9         |
| 3.2 Code . . . . .   | 10        |
| <b>4 Application</b>   | <b>15</b> |
| 4.1 Figma . . . . .  | 15        |
| 4.2 Développement . . . . .  | 16        |
| 4.2.1 Back . . . . .   | 16        |
| 4.2.2 Front . . . . .  | 19        |
| <b>5 Amélioration</b>  | <b>22</b> |
| 5.1 Prototype 3D : Ajustement et qualité des éléments . . . . .        | 22        |
| 5.2 Optimisation du code . . . . .                                     | 22        |
| 5.2.1 Rendre le code plus clair et mieux organisé . . . . .            | 22        |
| 5.2.2 Réduire les mises à jour inutiles . . . . .                      | 23        |
| 5.2.3 Simplifier les constantes . . . . .                              | 23        |
| 5.2.4 Ajouter des tests pour vérifier le fonctionnement . . . . .      | 23        |
| 5.2.5 Préparer le code pour une utilisation à grande échelle . . . . . | 23        |
| 5.2.6 Conclusion . . . . .   | 24        |

# 1

# Présentation

---

## 1.1 Rappel du sujet

Dans le cadre de la ressource R507 - Dispositifs Interactifs, il nous a été demandé de réaliser un piège à moustiques connecté. Ce projet repose sur plusieurs axes techniques et fonctionnels essentiels, que nous devons aborder de manière cohérente et intégrée. Notre objectif principal est de développer un dispositif complet qui inclut la création d'un modèle 3D permettant d'attirer, aspirer et piéger les moustiques, ainsi que la mise en œuvre d'un système Arduino qui assurera la gestion technique et le contrôle de l'ensemble du piège. De plus, nous devons concevoir une interface utilisateur fonctionnelle, permettant d'afficher en temps réel les statistiques relatives au fonctionnement du piège et d'interagir avec ce dernier.

Le piège devra être conçu avec l'intégration de plusieurs composants obligatoires. Ces derniers incluent : un ventilateur pour aspirer les moustiques, un capteur de CO<sub>2</sub> afin de savoir si le liquide qui attire les moustiques est toujours présent et actif, des LEDs bleues qui attireront les moustiques, une antenne LORA pour communiquer entre le piège et l'application ainsi qu'un détecteur/compteur de moustiques permettant de suivre en temps réel le nombre de moustiques capturés par le système.

L'interface utilisateur que nous développerons devra répondre à des exigences en termes de fonctionnalité et d'ergonomie. Elle devra afficher au minimum les éléments suivants : le nombre de moustiques capturés, le taux de CO<sub>2</sub> détecté par le capteur, ainsi qu'un bouton permettant de déclencher le ventilateur à distance. Cette interface devra être intuitive, simple à utiliser, et permettre une interaction fluide avec le système.

Ainsi, ce projet combine une approche technique approfondie avec une interface utilisateur fonctionnelle, permettant de développer un dispositif à la fois innovant et performant.

## 1.2 Notre projet

Pour répondre aux exigences du projet, nous avons opté pour une solution simple mais intégrant tous les éléments obligatoires spécifiés. Notre démarche a consisté à concevoir un piège fonctionnel, modulaire et pratique.

Le design du piège repose sur une boîte principale dotée de plusieurs compartiments soigneusement agencés :

- **Un compartiment pour le système électronique** : cet espace abrite les composants électroniques nécessaires à la gestion et au fonctionnement du piège, notamment le microcontrôleur Arduino et les circuits associés.
- **Un espace dédié à un tiroir amovible** : ce tiroir permet d'insérer le capteur de CO<sub>2</sub> ainsi qu'un récipient contenant le liquide ou la poudre servant à attirer les moustiques. Ce système amovible facilite l'entretien et le remplacement des consommables.
- **Un emplacement spécifique pour le ventilateur** : le ventilateur est positionné de manière optimale pour aspirer les moustiques vers l'intérieur du piège. Son installation est pensée pour maximiser son efficacité tout en restant accessible pour un éventuel entretien.
- **Une ouverture adaptée pour insérer un entonnoir** : cette ouverture guide les moustiques vers le piège. L'entonnoir est conçu pour canaliser efficacement les insectes tout en minimisant les risques d'échappement.

Les éléments extérieurs au corps principal comprennent :

- **Les tiroirs amovibles** : ils sont indépendants pour permettre une manipulation facile, notamment lors de la maintenance ou du nettoyage du piège.
- **L'entonnoir** : il intègre un emplacement dédié au détecteur de passage des moustiques. Ce composant est essentiel pour comptabiliser précisément le nombre d'insectes capturés.

En combinant modularité et simplicité, notre solution garantit une efficacité optimale tout en étant facile à assembler, à entretenir et à utiliser.

# 2

# Modèle 3D

---

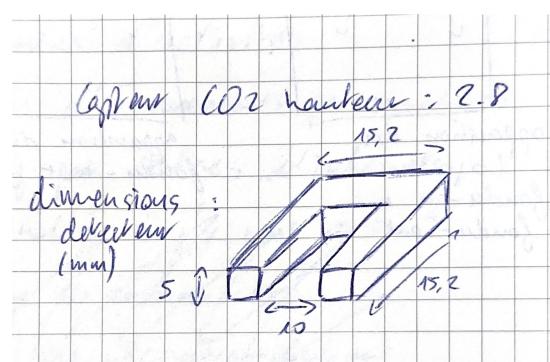
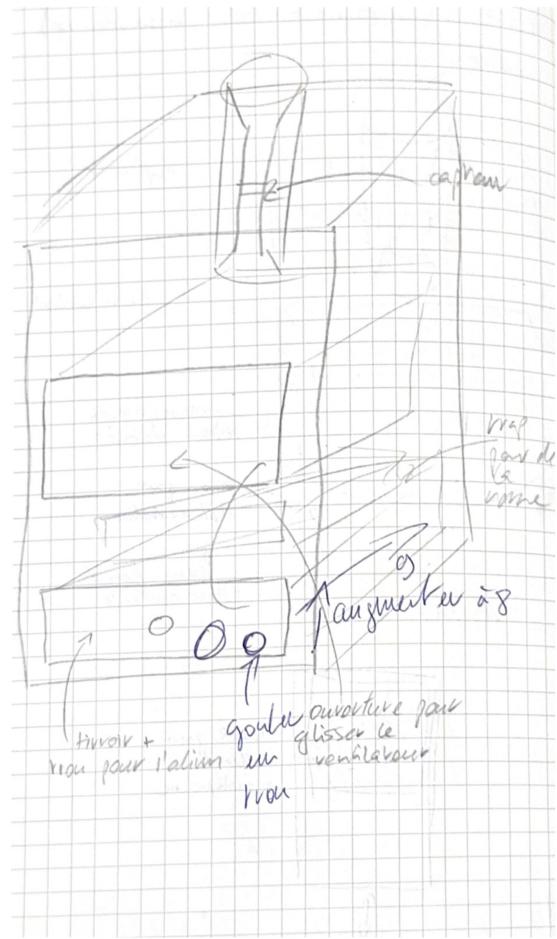
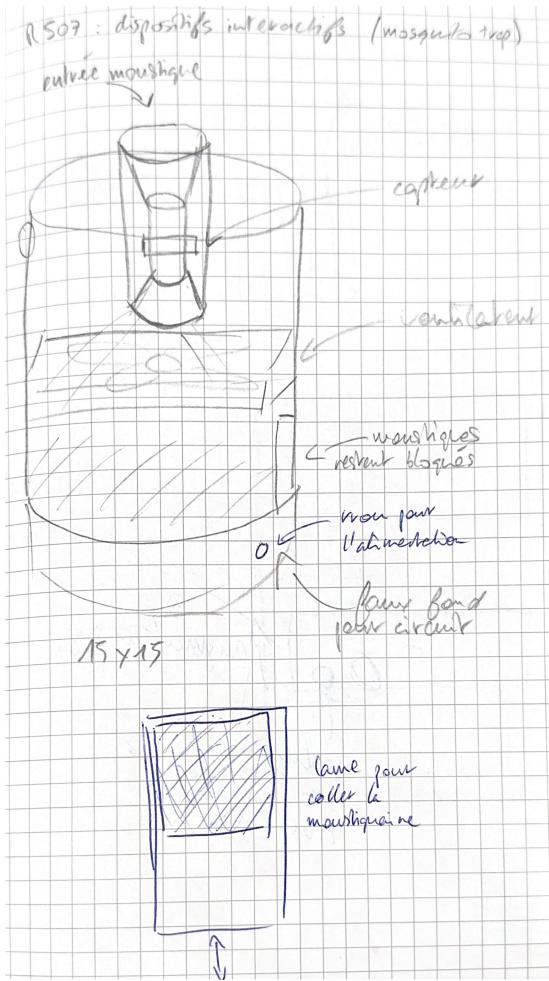
## 2.1 Réflexion

Avant de débuter la création du modèle 3D, il est essentiel de réfléchir à l'emplacement de chaque composant du piège à moustiques, afin d'assurer un fonctionnement optimal et un assemblage cohérent. Cette réflexion préalable permet de déterminer l'agencement des éléments obligatoires (ventilateur, capteur de CO<sub>2</sub>, détecteur de passage, etc.) dans un ordre logique, garantissant que chaque élément interagit efficacement avec les autres tout en optimisant l'espace disponible.

Un autre critère majeur est la rapidité d'impression du modèle 3D. Étant donné que plusieurs itérations et tests seront nécessaires pour ajuster les dimensions et la fonctionnalité du piège, l'impression doit être rapide pour permettre la production de prototypes dans des délais raisonnables. Il convient donc de veiller à un design qui ne surcharge pas le processus d'impression tout en respectant les exigences techniques du projet. Cette démarche permet d'obtenir des prototypes fonctionnels et d'évaluer rapidement les performances du modèle.

Dans cette phase de réflexion, nous avons créé des schémas papier. Ces croquis nous ont permis de visualiser concrètement l'agencement des différents composants et de vérifier leur praticabilité, notamment en termes d'espace et de connectivité. Ces esquisses ont servi à affiner notre concept et à évaluer la faisabilité du projet avant de passer à la modélisation 3D proprement dite.

Enfin, l'objectif principal reste la création d'un piège à moustiques pratique et simple d'utilisation. L'interface utilisateur et l'agencement des composants doivent être intuitifs, permettant à l'utilisateur de manipuler facilement le piège, que ce soit pour l'installer, le configurer ou l'entretenir. Cette simplicité d'utilisation doit s'accompagner d'une conception ergonomique, visant à garantir une expérience optimale pour l'utilisateur final.



## 2.2 Modélisation

Au cours du processus de modélisation, nous avons passé en revue plusieurs versions du modèle 3D, en ajustant progressivement l'emplacement et les dimensions des différents composants afin de répondre aux exigences fonctionnelles tout en optimisant la conception pour l'impression 3D.

- **Prototype 1 :** Dans cette première version, nous avons réfléchi à un système de disquette collante à insérer dans le piège, où les moustiques resteraient collés après avoir été aspirés. Cependant, lors de la conception, nous avons omis de prendre en compte l'emplacement du capteur de CO<sub>2</sub>, un élément essentiel pour attirer les moustiques. Cette omission nous a conduits à revoir la disposition des composants pour intégrer correctement ce capteur.
- **Prototype 2 :** Ce modèle représente une version plus proche de la version finale. Nous avons ajouté un tiroir amovible, remplaçant la disquette collante, pour placer le liquide attirant ainsi que le capteur de CO<sub>2</sub>. Cette modification a permis de mieux organiser l'espace interne du piège. Toutefois, la version comportait trop de pièces à imprimer, ce qui rendait le processus d'impression plus long et complexe. Nous avons alors décidé de simplifier le design pour réduire le nombre de pièces nécessaires tout en conservant la fonctionnalité.
- **Version finale :** La version finale du piège à moustiques a été optimisée pour minimiser la quantité de matière à imprimer. Nous avons réduit le nombre de composants imprimés et créé des emplacements et des trous spécifiques pour recevoir les différents éléments. Cette version finale intègre un agencement efficace, avec des emplacements bien définis pour chaque composant, tout en facilitant l'assemblage et l'entretien du piège.

Concernant les tiroirs, nous avons également ajusté leurs dimensions à plusieurs reprises afin de garantir qu'ils puissent contenir correctement le liquide attirant et le capteur de CO<sub>2</sub>. L'ajustement des tailles a permis d'optimiser l'espace et d'améliorer la praticité du piège.

En ce qui concerne l'entonnoir, une des modifications clés a été l'ajout d'un emplacement pour les LEDs bleues, supposées attirer les moustiques. De même, le détecteur de moustiques a été placé sous l'entonnoir pour assurer une détection efficace des insectes. Cette disposition permet un flux optimal des moustiques à travers le piège tout en maximisant la précision du comptage.

Dans l'ensemble, cette phase de modélisation a été marquée par une série d'ajustements successifs pour perfectionner la conception du piège, réduire le temps et les matériaux nécessaires à l'impression, et garantir une intégration harmonieuse de tous les composants.

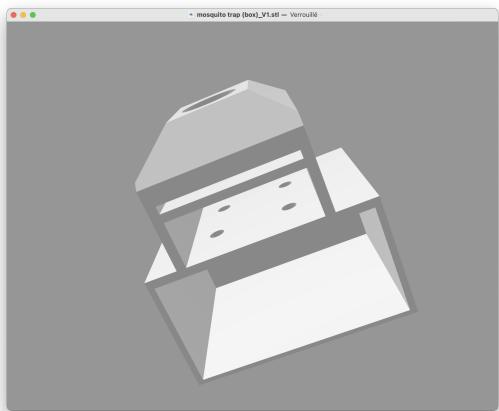


FIGURE 2.4 – Prototype 1

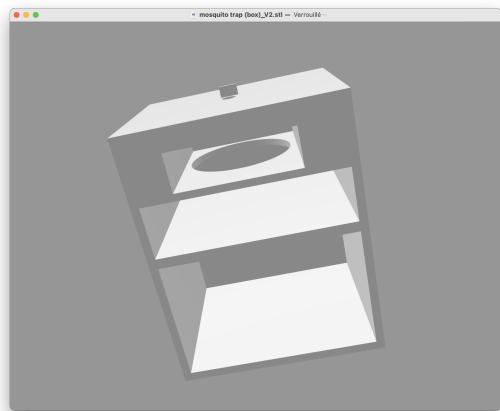


FIGURE 2.5 – Prototype 2

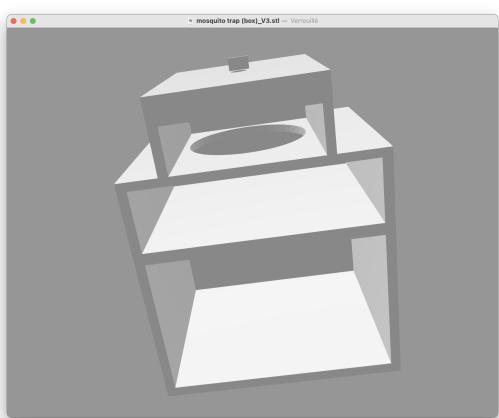


FIGURE 2.6 – Version finale

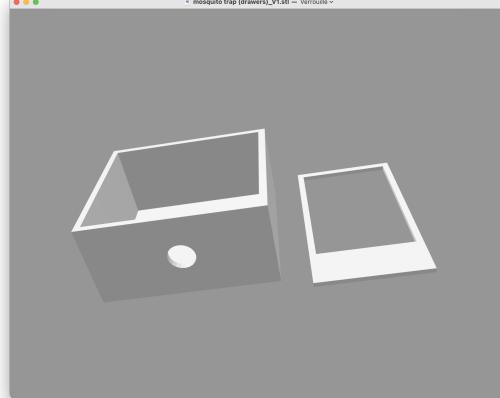


FIGURE 2.7 – Tiroir + disquette version 1

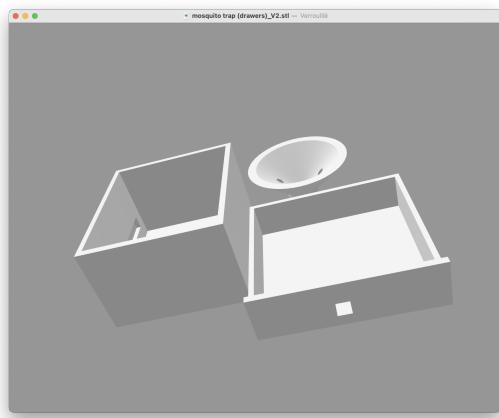


FIGURE 2.8 – Tiroirs versions finale + entonnoir

# 3

# Arduino

## 3.1 Montage

### 3.1.1 V1

Pour le montage de l'Arduino du piège à moustiques, plusieurs versions ont été développées. Au début, nous avons simulé les fourches optiques à l'aide d'un bouton poussoir, d'une LED, et d'un buzzer sonore. Lorsque le bouton était pressé, le buzzer émettait un son (aléatoire à chaque passage), ce qui simulait le passage d'un moustique à travers la fourche optique. Cela allumait également brièvement la LED. Ensuite, à l'aide d'un relais et d'un transistor (le transistor pilotait le relais pour ouvrir ou fermer le circuit en fonction du signal reçu de la pin), une LED restait allumée pendant 5 secondes pour simuler le fonctionnement du ventilateur, garantissant que le moustique soit capturé. Nous avons également ajouté un capteur de CO<sub>2</sub> pour mesurer les niveaux de dioxyde de carbone et transmettre ces données à l'application React. Pour cela, il a suffi de connecter les fils du capteur : l'un sur le 5V, un autre sur le GND, et le dernier sur une pin analogique, afin de recevoir les informations.

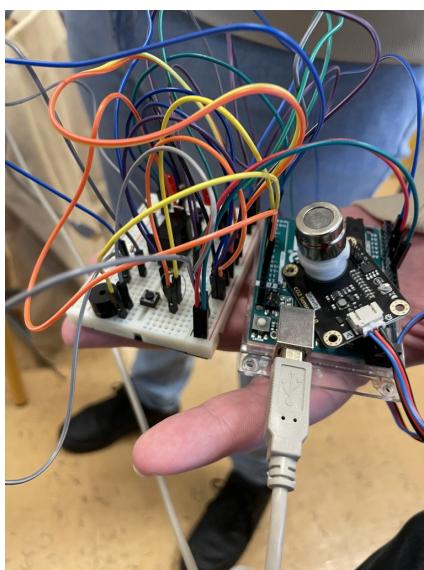


FIGURE 3.1 – Photo1 du mon-  
tage de la V1

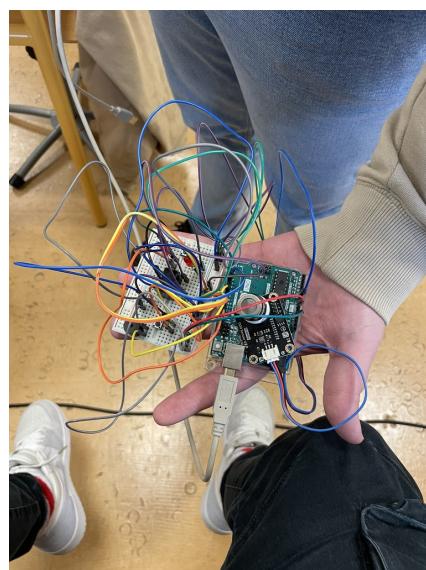


FIGURE 3.2 – Photo2 du mon-  
tage de la V1

### 3.1.2 V2

Dans cette version, nous avons ajouté une fourche optique connectée à une nouvelle pin, sans modifier le système existant avec le buzzer. Le code a été adapté pour que les actions déclenchées par le bouton poussoir soient également réalisées via la fourche optique. Cela nous facilitait les tests, car il était plus pratique d'appuyer sur un bouton que de passer un objet à travers la fourche. Nous avons également remplacé la LED connectée au relais par un ventilateur, sans avoir à modifier le code, le montage étant simplement à ajuster. À cette étape, nous avons également ajouté au montage un capteur LoRa, directement connecté à la carte Arduino. Pour cela, nous avons dû passer d'une carte Uno à une carte Leonardo, car le capteur LoRa nécessite un type de connexion spécifique. Nous lui avons également ajouté un "shield" adapté pour faciliter ce branchement particulier.

### 3.1.3 V3

La version finale consistait à passer de deux breadboards à un circuit imprimé unique, ce qui nécessitait une optimisation du montage. Nous avons retiré le bouton poussoir, la dernière LED (devenue inutile), ainsi que le buzzer pour gagner de la place. Nous avons ajouté deux LEDs bleues destinées à attirer les moustiques à l'intérieur du dispositif. Cependant, n'étant pas experts en soudure, nous avons rencontré de nombreuses difficultés, notamment avec les fourches optiques, ce qui a rendu la réalisation de cette version assez complexe.

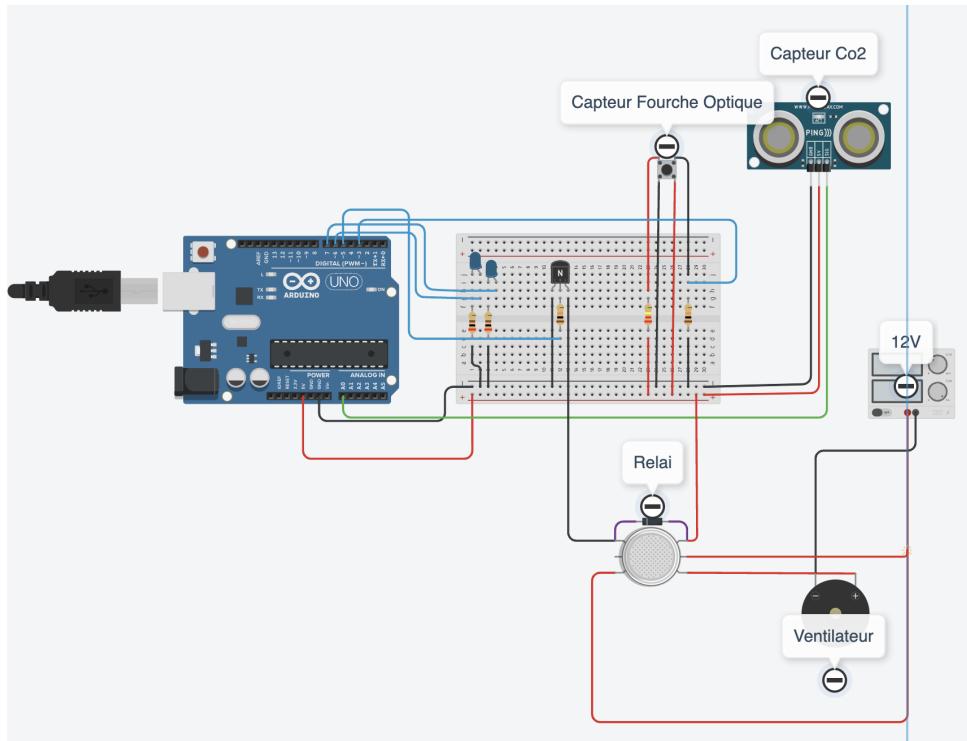


FIGURE 3.3 – Montage de la V3

## 3.2 Code

```
#include <Arduino.h>
#include "lorae5.h"
#include "config_board.h"
#include "config_application.h"
```

Importation des scripts pour gérer la communication via le LoraWan.

```
#define RELAI 5          // Ventilateur contrôlé par le relais
#define LED1 7           // LED1 toujours allumée
#define LED2 6           // LED2 toujours allumée
#define CO2_PIN A0        // Capteur de CO2
#define OPTICAL_SENSOR 3 // Broche pour le capteur optique
```

Définition des différents PIN sur lesquels sont branchés les composants.

```
volatile int mosquitoCount = 0; // Compteur de moustiques

uint8_t sizePayloadUp = 6; // Ajout d'un octet pour l'état du relais et un
→ pour le niveau de CO2
uint8_t payloadUp[20] = {0};

uint8_t sizePayloadDown = 1;
uint8_t payloadDown[20] = {0};

unsigned long lastDetectionTime = 0;
const unsigned long DETECTION_DELAY = 500;

unsigned long relayStartTime = 0; // Moment où le relais est activé
const unsigned long RELAY_ON_DURATION = 5000; // Durée en ms pour garder
→ le relais actif

bool relayActive = false; // État du relais

LORAE5 lorae5(devEUI, appEUI, appKey, devAddr, nwkSKey, appSKey);

// Timer pour les transmissions LoRa
```

```

unsigned long previousMillisLoRa = 0;
unsigned long intervalLoRa = 10000; // 10 secondes pour l'envoi LoRa

```

Définition de toutes les variables nécessaires au bon déroulement du programme.

```

void setup() {
    pinMode(RELAI, OUTPUT);
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    pinMode(OPTICAL_SENSOR, INPUT);
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    // Relais éteint au démarrage
    digitalWrite(RELAI, LOW);

    Serial.begin(9600);

    // Initialisation du matériel LoRa
    lorae5.setup_hardware(&Debug_Serial, &LoRa_Serial);
    lorae5.setup_lorawan(REGION, ACTIVATION_MODE, CLASS, SPREADING_FACTOR,
    ↳ ADAPTIVE_DR, CONFIRMED, PORT_UP, SEND_BY_PUSH_BUTTON, FRAME_DELAY);
    lorae5.printInfo();
    if (ACTIVATION_MODE == OTAA) {
        Serial.println("Join Procedure in progress...");
        while (lorae5.join() == false);
        delay(2000);
    }
}

```

Mise en route de tous les composants connectés et initialisation du LoraWan afin de pouvoir envoyer et recevoir des données.

```

void loop() {
    unsigned long currentMillis = millis();

    // Lire l'état du capteur optique
    if (digitalRead(OPTICAL_SENSOR) == LOW && (currentMillis -
    ↳ lastDetectionTime > DETECTION_DELAY)) {
        mosquitoCount++; // Incrémenter le compteur de moustiques
}

```

```

Serial.print("Moustique détecté ! Compteur : ");
Serial.println(mosquitoCount);

// Activer le relais
digitalWrite(RELAI, HIGH);
relayStartTime = currentMillis;
relayActive = true;

lastDetectionTime = currentMillis; // Mettre à jour le temps de
→ détection
}

// Désactiver le relais après la durée définie
if (relayActive && (currentMillis - relayStartTime >= RELAY_ON_DURATION))
→ {
    digitalWrite(RELAI, LOW); // Éteindre le relais
    relayActive = false;
}

```

Démarrer un compteur afin de savoir depuis quand le programme tourne. Si le capteur optique est déclenché et que la dernière fois qu'il a été déclenché est supérieur à 0.5s, on incrémente le nombre de moustique, on active le relai afin de déclencher le ventilateur pour aspirer le moustique, puis on met à jour les différentes variables. Si le relai est activé et qu'il a été activé pendant plus de 5s, on le désactive et on met à jour la variable.

```

// Envoi LoRa toutes les 30 secondes
if (currentMillis - previousMillisLoRa >= intervalLoRa) {
    previousMillisLoRa = currentMillis;
    sendLoRaData(); // Envoyer les données
}

// Traitement des messages descendus via LoRa
if (lorae5.awaitForDownlinkClass_A(payloadDown, &sizePayloadDown) ==
→ RET_DOWNLINK) {
    processDownlink();
}

lorae5.sleep();
}

```

Si la dernière fois qu'un message à été envoyé depuis le LoraWan, date de plus de 30s, on appelle la fonction qui permet d'en envoyer un nouveau et on met à jour les variables. Si on reçoit un message depuis le React, dans le LoraWan, on appelle la fonction qui permet de gérer ce message. On stop le LoraWan afin de ne pas être bloqué après avoir reçus un message depuis le React.

```
// Fonction pour envoyer les données LoRa
void sendLoRaData() {
    int sensorValue = analogRead(CO2_PIN);
    float voltage = sensorValue * (5.0 / 1023.0);
    float CO2 = 2000 * voltage / 5.0;
    Serial.print("Concentration CO2 : ");
    Serial.print(CO2);
    Serial.println(" ppm");

    uint8_t co2State = 0;
    if (CO2 < 400.0) co2State = 1;

    payloadUp[0] = (uint8_t)((int)CO2 >> 8);
    payloadUp[1] = (uint8_t)((int)CO2 & 0xFF);
    payloadUp[2] = (uint8_t)(mosquitoCount >> 8);
    payloadUp[3] = (uint8_t)(mosquitoCount & 0xFF);
    payloadUp[4] = relayActive ? 1 : 0; // État du relais
    payloadUp[5] = co2State;

    lorae5.sendData(payloadUp, sizePayloadUp);
}
```

Fonction qui gère l'envoie des données du LoraWan vers le React. On récupère le taux de CO2 capté grâce au capteur de CO2. On calcule le taux de ppm grâce à la valeur précédente. On définit le taux de CO2 minimum que l'on souhaite pour l'application React. Le taux de CO2 et le nombre de moustique est séparé en 2 parties, une pour le bit de poids fort et une pour le bit de poids faible. On remplit le message avec toutes les informations utiles pour l'application React, puis on envoie le message.

```
// Fonction pour traiter les données descendantes
void processDownlink() {
    Serial.print("Downlink reçu : ");
    Serial.println(payloadDown[0]);
```

```

if (payloadDown[0] == 49) { // Code 49 : Allumer le ventilateur
    digitalWrite(RELAI, HIGH);
    relayActive = true;
    payloadUp[4] = 1; // Mettre à jour l'état du relais
    Serial.println("Relais activé par downlink.");
} else if (payloadDown[0] == 48) { // Code 48 : Éteindre le ventilateur
    digitalWrite(RELAI, LOW);
    relayActive = false;
    payloadUp[4] = 0; // Mettre à jour l'état du relais
    Serial.println("Relais désactivé par downlink.");
} else {
    Serial.println("Commande inconnue.");
}
}

```

Fonction qui gère la réception des données du React vers le LoraWan. Si la première partie du message est égale à 49, on active le relay et on met à jour le message qui va envoyer les données vers le React. Si la première partie du message est égale à 48, on désactive le relay et on met à jour le message qui va envoyer les données vers le React. 49 et 48 correspondent au code ASCII des chiffres 1 et 0.

# 4

# Application

---

## 4.1 Figma

Afin de concevoir une application dotée d'une interface utilisateur cohérente avec le piège à moustiques, nous avons utilisé Figma pour réaliser une maquette visuelle. Cette maquette avait pour objectif de fournir une expérience utilisateur fluide et intuitive, en prenant en compte les fonctionnalités essentielles à intégrer dans l'application.

Nous avons choisi d'afficher plusieurs éléments clés sur l'interface pour garantir un suivi efficace des données du piège, tout en assurant une utilisation simple et rapide :

- **Un bouton de contrôle à distance du ventilateur** : Ce bouton permet à l'utilisateur d'allumer ou d'éteindre le ventilateur à distance, directement depuis l'application. Il est placé de manière visible et accessible, pour un accès rapide et pratique.
- **Un récapitulatif des données** : Cette section présente des informations détaillées et actualisées sur le fonctionnement du piège. Les utilisateurs peuvent consulter : Le nombre de moustiques attrapés au cours des dernières heures, jours, semaines, mois et années. Ce suivi permet de visualiser l'efficacité du piège sur différents intervalles de temps. Le niveau de CO<sub>2</sub> mesuré tout au long de la semaine. Cette donnée est importante pour évaluer l'attractivité du piège et son efficacité dans l'attraction des moustiques.
- **Le détail du nombre de moustiques attrapés aujourd'hui** : Cette section fournit des informations précises sur l'activité du piège pour la journée en cours, permettant à l'utilisateur de suivre en temps réel le nombre de moustiques capturés.
- **Le détail du niveau de CO<sub>2</sub> en temps réel** : Cette section fournit des informations précises sur le niveau de CO<sub>2</sub> actuel, permettant à l'utilisateur de savoir si le liquide attire toujours autant les moustiques ou non.

Après la création de la maquette, nous avons procédé à la réalisation d'un prototype interactif dans Figma pour simuler l'expérience utilisateur. Cela nous a permis de visualiser l'interface dans un contexte réaliste, sur un appareil mobile. Nous avons testé ce prototype sur un iPhone SE, afin de nous assurer que l'application soit non seulement fonctionnelle, mais également ergonomique et adaptée aux différentes tailles d'écrans. Cette étape nous a permis de valider la conception visuelle, d'ajuster la disposition des éléments et de tester la fluidité de la navigation avant de passer à la phase de développement final.

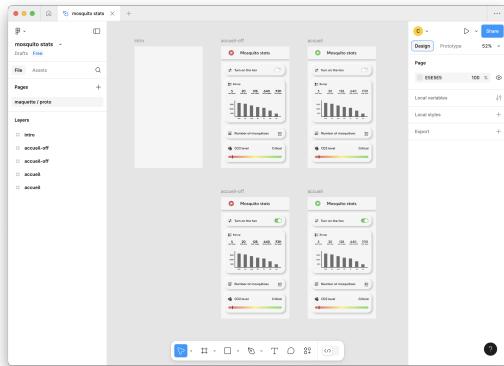


FIGURE 4.1 – Maquette de l’application

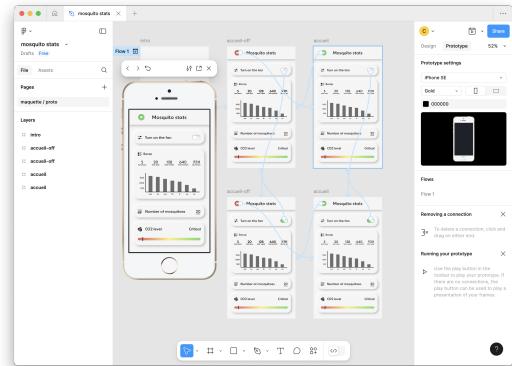


FIGURE 4.2 – Prototype de l’application

## 4.2 Développement

La partie développement de notre projet se divise en deux volets principaux : le back-end et le front-end. Pour la gestion du back-end, nous avons choisi d’utiliser Node.js, une plateforme de développement JavaScript qui nous permet de créer un serveur rapide et efficace. Pour le front-end, nous avons opté pour React, une bibliothèque JavaScript dédiée à la création d’interfaces utilisateurs interactives et dynamiques. Ces deux technologies sont complémentaires et nous ont permis de créer une application performante, scalable et facile à maintenir.

L’utilisation de Node.js pour le back-end nous a permis de tirer parti de JavaScript des deux côtés du développement, ce qui simplifie le processus de développement et rend le code plus cohérent. De son côté, React a facilité la création d’une interface utilisateur dynamique et réactive, avec une gestion de l’état simplifiée et un rendu rapide.

Dans les sous-parties suivantes, nous détaillerons chacune de ces technologies, leurs principes de fonctionnement et les raisons de notre choix, en plus de fournir des exemples de code pour illustrer leur mise en œuvre dans notre projet.

### 4.2.1 Back

Node.js est une plateforme de développement basée sur JavaScript qui permet d’exécuter du code JavaScript côté serveur. Sa principale caractéristique est qu’il repose sur un modèle asynchrone et événementiel, ce qui lui permet de gérer un grand nombre de connexions simultanées de manière extrêmement efficace. Ce modèle non-bloquant rend Node.js particulièrement adapté aux applications nécessitant des mises à jour en temps réel, comme la gestion des données provenant de notre piège à moustiques.

Quelques explications du code :

```

const express = require('express');
const app = express();
const server = http.createServer(app);

```

Utilisation d'Express, un framework minimaliste qui simplifie la création d'un serveur HTTP en Node.js. Express permet une gestion rapide et efficace des routes et des middlewares, ce qui en fait un choix idéal pour un projet nécessitant des API et des intégrations complexes comme MQTT et WebSocket.

```

const mqtt = require('mqtt');

const protocol = 'mqtts';
const host = process.env.MQTT_HOST;
const port = process.env.MQTT_PORT;
const clientId = `mqtt_${Math.random().toString(16).slice(3)}`;
const connectUrl = `${protocol}://${host}:${port}`;

const mqttClient = mqtt.connect(connectUrl, {
  clientId,
  clean: true,
  connectTimeout: 4000,
  username: process.env.MQTT_USER,
  password: process.env.MQTT_KEY,
});

```

Configuration d'une connexion sécurisée (via mqtts) avec un broker MQTT, un service clé pour récupérer les données en temps réel (par exemple, les niveaux de CO2 ou le nombre de moustiques capturés). La flexibilité de MQTT en fait un protocole idéal pour la communication entre dispositifs IoT et serveurs.

```

mqttClient.on('message', (topic, message) => {
  try {
    const parsedMessage = JSON.parse(message.toString());
    if (parsedMessage.uplink_message &&
      parsedMessage.uplink_message.decoded_payload) {
      const bytes =
        parsedMessage.uplink_message.decoded_payload.bytes;
      console.log('Bytes reçus:', bytes);
      io.emit('mqttData', bytes); // Émission des données aux
      clients via WebSocket
    }
  }
})

```

```

    } else {
        console.error('Erreur: Données MQTT incorrectes ou absentes.');
    }
} catch (error) {
    console.error('Erreur de parsing JSON:', error.message);
}
);

```

Cette partie montre comment les messages provenant du broker MQTT sont traités :

- Les messages sont reçus, convertis en JSON, et extraits pour en récupérer les données utiles.
- Ces données sont ensuite transmises en temps réel au front-end via Socket.IO.

```

const { Server } = require('socket.io');
const io = new Server(server, {
    cors: {
        origin: 'http://localhost:3000',
        methods: ['GET', 'POST'],
    },
});

io.on('connection', (socket) => {
    console.log('Client React est connecté');
});

```

Socket.IO est utilisé pour établir une communication bidirectionnelle entre le serveur et les clients (React dans ce cas). Cela permet de pousser les données en temps réel vers le front-end, comme l'état actuel des moustiques attrapés ou les niveaux de CO2.

```

server.listen(3001, () => {
    console.log('Serveur Node.js en écoute sur le port 3001');
});

```

Le serveur est configuré pour écouter sur le port 3001, assurant une séparation claire entre le back-end (port 3001) et le front-end (port 3000). Cette séparation améliore la modularité et la gestion du projet.

#### 4.2.2 Front

React est une bibliothèque JavaScript développée par Facebook pour la création d'interfaces utilisateur dynamiques et réactives. L'un des principaux avantages de React est son approche déclarative : au lieu de manipuler directement le DOM (Document Object Model), React permet de décrire l'interface utilisateur sous forme de composants réutilisables et gère automatiquement les mises à jour de l'interface en fonction des changements d'état.

Quelques explications du code :

```
const socket = io('http://localhost:3001');
```

La connexion à Socket.IO permet de recevoir des données en temps réel depuis le serveur Node.js. Ici, l'application React écoute les événements émis par le serveur.

```
useEffect(() => {
  socket.on('mqttData', (bytes) => {
    console.log('Bytes reçus depuis le serveur:', bytes);
    setMqttBytes(bytes);

    if (bytes[5] === 1) {
      setAlert(true);
    } else {
      setAlert(false);
    }
  });

  return () => {
    socket.off('mqttData');
  };
}, []);
```

Le hook useEffect s'exécute une fois lors du montage du composant pour écouter l'événement mqttData. Les données reçues (sous forme de tableau de bytes) sont enregistrées dans l'état local (setMqttBytes) et si une alerte est détectée (par exemple, bytes[5] === 1), une alerte est activée.

```

const mosquitoCount = mqttBytes ? (mqttBytes[2] << 8) | mqttBytes[3] : 0;
const co2Level = mqttBytes ? (mqttBytes[0] << 8) | mqttBytes[1] : 0;
console.log('Niveau de CO2 calculé:', co2Level);

```

Les bytes reçus contiennent des données combinées. Ici, les valeurs sont calculées en combinant deux octets grâce à un décalage binaire (`<<`). L'avantage étant que le traitement des bytes directement dans React réduit la charge côté serveur et permet une interface réactive.

```

{alert && (
  <div className="fixed inset-0 bg-gray-800 bg-opacity-50 flex
    justify-center items-center">
    <div className='bg-white rounded-3xl shadow-lg p-6 text-center mx-5'>
      <h2 className="text-red-600 text-2xl font-bold mb-4">Alerte CO2
        bas</h2>
      <p className="text-gray-700">Le niveau de CO2 est trop bas. Prenez
        des mesures pour augmenter le taux de CO2.</p>
      <button
        className="mt-4 bg-red-500 text-white px-4 py-2 rounded
          hover:bg-red-600"
        onClick={() => setAlert(false)}
      >
        OK
      </button>
    </div>
  </div>
)}

```

Lorsque alert est activé (par exemple, si bytes[5] === 1), un composant d'alerte s'affiche en superposition, l'utilisateur peut alors fermer l'alerte via un bouton, ce qui réinitialise l'état (setAlert(false)).

```

<Header />
<div className='flex flex-col gap-y-10 mx-5'>
  <Toggle />
  <Recap co2Level={co2Level} mosquitoCount={mosquitoCount} />
  <Counter mosquitoCount={mosquitoCount} />
  <CO2LevelBar level={co2Level} />

```

```
</div>
```

Chaque élément de l'interface utilisateur est encapsulé dans des composants indépendants (Header, Toggle, Recap, etc.), ce qui facilite la gestion et la réutilisation.

# 5

# Amélioration

---

## 5.1 Prototype 3D : Ajustement et qualité des éléments

Bien que le modèle 3D utilisé dans notre projet remplisse son rôle fonctionnel, plusieurs ajustements pourraient améliorer sa précision et sa praticité :

Problème : Certains éléments du modèle ne s'emboîtent pas parfaitement, ce qui peut affecter l'efficacité globale du système (par exemple, une mauvaise isolation de la zone du capteur ou des fuites d'air près du ventilateur).

Améliorations possibles :

- **Refonte partielle des pièces** : Revoir les dimensions des pièces clés pour garantir un ajustement parfait.
- **Utilisation de matériaux de meilleure qualité** : Intégrer des matériaux plus robustes pour augmenter la durabilité du système.
- **Précision accrue** : Améliorer la résolution des fichiers STL pour des impressions 3D plus précises, en tenant compte des tolérances des imprimantes.
- **Tests d'assemblage virtuel** : Simuler l'assemblage en 3D avec un logiciel comme Fusion 360 ou SolidWorks pour identifier les incompatibilités avant l'impression.

Ces modifications garantiraient une meilleure intégration des éléments mécaniques et électroniques, rendant le dispositif plus fiable.

## 5.2 Optimisation du code

L'optimisation du code est une étape essentielle pour garantir une meilleure performance et évolutivité de notre projet. Voici les points à améliorer et pourquoi ils sont importants :

### 5.2.1 Rendre le code plus clair et mieux organisé

Problème : Certaines parties du code sont longues ou mélangeant plusieurs tâches, ce qui les rend difficiles à lire.

Solution : Diviser ces parties en plus petites fonctions. Par exemple, créer une fonction spéciale pour gérer les calculs des données MQTT (comme le nombre de moustiques et le niveau de CO<sub>2</sub>) au lieu de tout faire dans le composant principal.

### 5.2.2 Réduire les mises à jour inutiles

Problème : Les données en temps réel envoyées par Socket.IO peuvent surcharger le système si elles arrivent trop souvent.

Solution : Limiter la fréquence des mises à jour. Par exemple, n'envoyer les nouvelles données que toutes les heures, même si des messages arrivent plus fréquemment.

### 5.2.3 Simplifier les constantes

Problème : Les indices des tableaux MQTT (bytes[0], bytes[5], etc.) sont utilisés directement dans le code. Si ces indices changent ou si on veut ajouter une nouvelle donnée, il faut modifier plusieurs endroits.

Solution : Rassembler toutes ces informations dans un fichier avec des noms clairs, comme :

```
const BYTE_INDEX = {  
    CO2_HIGH: 0,  
    CO2_LOW: 1,  
    MOSQUITO_HIGH: 2,  
    MOSQUITO_LOW: 3,  
    ALERT: 5,  
};
```

Cela permet de remplacer bytes[5] par bytes[BYTE\_INDEX.ALERT], ce qui est beaucoup plus compréhensible.

### 5.2.4 Ajouter des tests pour vérifier le fonctionnement

Problème : Si une partie du code ne fonctionne pas ou si une modification introduit un bug, il est difficile de le détecter rapidement.

Solution : Ajouter des tests simples pour vérifier que le calcul du nombre de moustiques ou du niveau de CO2 donne bien le bon résultat. Ces tests peuvent être faits automatiquement avec des outils comme Jest.

### 5.2.5 Préparer le code pour une utilisation à grande échelle

Problème : Pour l'instant, le système est fait pour fonctionner localement avec un seul utilisateur. Si plusieurs pièges connectés sont utilisés en même temps, cela peut poser des problèmes.

Solution : Adapter le serveur pour gérer plusieurs appareils et utilisateurs, et envisager de l'héberger sur un service en ligne pour plus de stabilité.

### **5.2.6 Conclusion**

Avec ces changements, le code serait plus simple à lire, plus rapide et plus adaptable si nous voulons étendre le projet. Cela faciliterait aussi le travail en équipe, car chaque membre comprendrait mieux les différentes parties du programme.