

Bureau d'Études C++ et POO

Compte rendu et documentation technique

Projet Danseuse Capybara

Établissement : INSA Toulouse
Cours : C++
Année : 2025–2026
Plateforme : ESP8266

Auteurs : Roberto Ibanez Mingarro
Kevin Felipe Visbal Pinzón

GitHub :

https://github.com/robertoibanezmingarro-lang/BE_POO_Template/tree/feature/dev-roberto

Table des matières

1	Introduction	2
2	Objectifs et périmètre	2
2.1	Objectifs	2
2.2	Périmètre fonctionnel	2
3	Contexte matériel et logiciel	2
3.1	Matériel	2
3.2	Environnement de développement	2
3.3	Organisation du dépôt	2
4	Analyse fonctionnelle	3
4.1	Cas d'utilisation	3
4.2	Scénario nominal	3
5	Architecture logicielle (POO)	4
5.1	Principes de conception	4
5.2	Diagramme de classes	4
5.3	Description des modules	4
5.3.1	Interfaces Sensor et Actuator	4
5.3.2	Capteurs	4
5.3.3	Actionneurs	5
5.3.4	Musique	5
5.3.5	Service WiFi / HTTP	5
6	Description détaillée du fonctionnement	5
6.1	Initialisation (begin/setup)	5
6.2	Boucle principale (update/loop)	5
6.3	Gestion des états et conditions	5
7	API HTTP	6
7.1	Résumé des endpoints	6
7.2	Exemples de requêtes	6
8	Procédure de compilation et déploiement	6
8.1	Téléversement sur ESP8266	6
8.2	Vérifications	6
9	Gestion de projet et travail collaboratif (Git)	6
9.1	Organisation	6
9.2	Justification	7
10	Conclusion	7
A	Annexe A – Captures du projet (PNG)	7

1 Introduction

Ce document constitue le **compte rendu** et la **documentation technique** du projet Danseuse Capybara, réalisé dans le cadre du **BE de Programmation Orientée Objet (POO) à l'INSA Toulouse**.

Le projet met en œuvre un système embarqué interactif basé sur un **ESP8266** et développé en **C++ (Arduino)**. L'objectif principal est de structurer un code modulaire et maintenable en appliquant les principes de la POO (abstraction, héritage, composition, encapsulation, polymorphisme) tout en intégrant des capteurs, des actionneurs et une interface WiFi HTTP.

2 Objectifs et périmètre

2.1 Objectifs

- Concevoir une architecture logicielle orientée objet pour un système embarqué.
- Intégrer plusieurs capteurs (présence, lumière, tactile) et actionneurs (servo, buzzer, LED, LCD).
- Fournir une interface de contrôle et de supervision via **WiFi** et **HTTP**.
- Documenter l'architecture, les choix techniques et le fonctionnement global.

2.2 Périmètre fonctionnel

Le système :

- détecte une présence via un capteur ultrason (seuil : distance ≤ 10 cm) ;
- détecte le mode nuit via un capteur de luminosité ;
- déclenche une animation (danse) et des effets (LED / musique / LCD / etc.) selon les conditions ;
- permet de changer de chanson via un bouton tactile et/ou WiFi ;
- expose des points d'accès HTTP pour consulter l'état et piloter certaines fonctions.

3 Contexte matériel et logiciel

3.1 Matériel

- Carte : ESP8266
- Capteurs : ultrason, luminosité, bouton tactile
- Actionneurs : servo-moteur, buzzer, LED, écran LCD RGB

3.2 Environnement de développement

- Arduino IDE
- Bibliothèques typiques : ESP8266WiFi, serveur HTTP, Servo, LCD
- Langage : C++ (Arduino)

3.3 Organisation du dépôt

La structure du dépôt est organisée de façon modulaire afin de respecter la séparation des responsabilités :

- `core_Sensor.h / core_Actuator.h` : interfaces abstraites
- `sensors_*` : implémentations concrètes des capteurs
- `actuators_*` : implémentations concrètes des actionneurs
- `music_*` : représentation et lecture de chansons
- `wifi_*` : service WiFi et API HTTP
- `app_*` : logique applicative (orchestration)
- `*.ino` : point d'entrée Arduino (`setup/loop`)

4 Analyse fonctionnelle

4.1 Cas d'utilisation

La figure 1 présente les cas d'utilisation principaux du système.

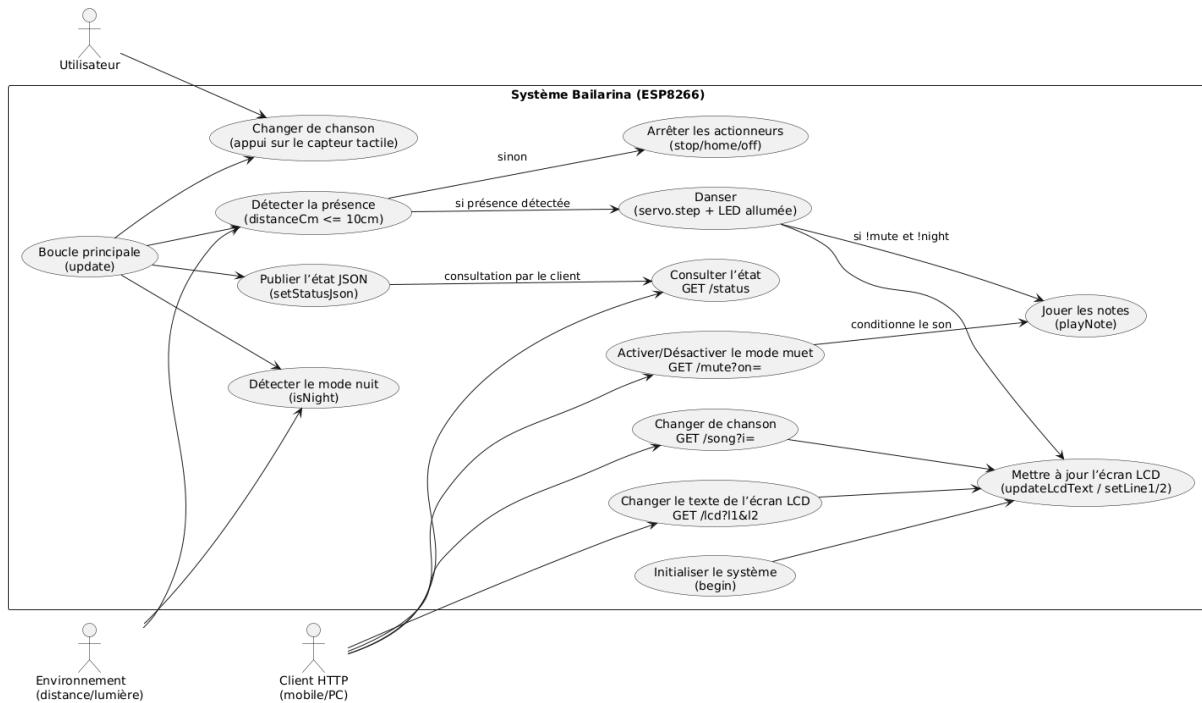


FIGURE 1 – Diagramme de cas d'utilisation (UML) – Projet Danseuse Capybara

4.2 Scénario nominal

Un scénario typique est le suivant :

1. Initialisation du système : capteurs, actionneurs, WiFi, LCD.
2. Boucle principale : acquisition des mesures (distance, luminosité), lecture du bouton tactile.
3. Si présence détectée : animation (servo), LED active, musique si autorisée (non muet, non nuit).
4. Publication / mise à jour de l'état (LCD et/ou JSON via HTTP).
5. Si absence : arrêt des actionneurs et retour à un état stable.

5 Architecture logicielle (POO)

5.1 Principes de conception

Les principes suivants ont guidé l'implémentation :

- **Abstraction** : interfaces communes pour capteurs et actionneurs.
- **Encapsulation** : chaque classe gère son état et son comportement.
- **Composition** : l'application agrège les modules et orchestre le système.
- **Extensibilité** : ajout de capteurs/actionneurs sans modifier la logique globale.

5.2 Diagramme de classes

La figure 5 illustre l'architecture objet et les relations entre classes.

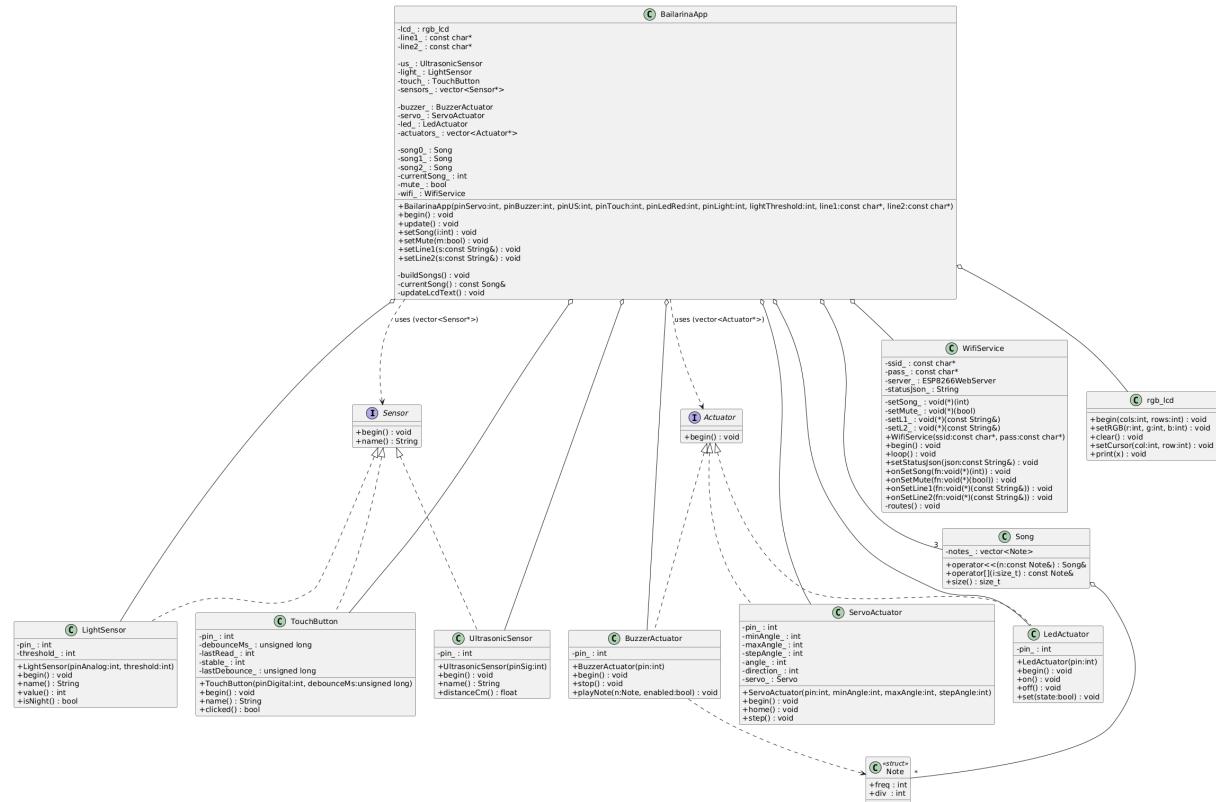


FIGURE 2 – Diagramme de classes (UML) – Projet Danseuse Capybara

5.3 Description des modules

5.3.1 Interfaces Sensor et Actuator

- **Sensor** définit le contrat minimal d'un capteur : initialisation et lecture/actualisation.
- **Actuator** définit le contrat minimal d'un actionneur : initialisation, activation/désactivation et actions spécifiques.

5.3.2 Capteurs

- **Capteur ultrason** : calcule une distance et permet une logique de présence.
- **Capteur de luminosité** : fournit un indicateur `isNight` pour contraindre le comportement (ex. silence la nuit).
- **Bouton tactile** : fournit un événement de changement de chanson.

5.3.3 Actionneurs

- **Servo** : animation mécanique (pas de danse / positionnement).
- **Buzzer** : génération de notes et lecture de mélodies.
- **LED** : retour visuel d'activité.
- **LCD** : affichage d'état (lignes de texte, mode, etc.).

5.3.4 Musique

Le module musique encapsule la représentation d'une chanson et la logique d'itération sur les notes (durée, fréquence, etc.). L'exécution peut être conditionnée par un mode *mute* ou par le mode nuit.

5.3.5 Service WiFi / HTTP

Le module WiFi expose une interface HTTP permettant :

- GET `/status` : consultation de l'état du système (JSON).
- GET `/song?i=` : changement de chanson par index.
- GET `/mute?on=` : activation/désactivation du mode muet.
- GET `/lcd?l1&l2` : mise à jour des deux lignes affichées.

6 Description détaillée du fonctionnement

6.1 Initialisation (`begin/setup`)

Lors du démarrage :

- configuration des broches et initialisation des capteurs/actionneurs ;
- connexion WiFi et configuration du serveur HTTP ;
- initialisation LCD et affichage d'un état prêt.

6.2 Boucle principale (`update/loop`)

À chaque itération :

1. acquisition des mesures (distance, luminosité) ;
2. prise en compte d'événements utilisateur (tactile) ;
3. décision : danser / arrêter selon la présence ;
4. décision audio : jouer si non muet et non nuit ;
5. mise à jour de l'affichage et publication de l'état.

6.3 Gestion des états et conditions

- **Présence** : déclenche l'animation.
- **Absence** : arrêt servo, extinction LED, arrêt buzzer.
- **Mode nuit** : désactive la musique (ou limite les effets) selon la politique retenue.
- **Mode muet** : priorité sur la lecture musicale.

7 API HTTP

7.1 Résumé des endpoints

Route	Méthode	Description
/status	GET	Retourne l'état courant du système (JSON)
/song?i=	GET	Sélectionne une chanson via un index
/mute?on=	GET	Active/désactive le mode muet
/lcd?l1&l2	GET	Met à jour les lignes LCD (texte)

TABLE 1 – API HTTP – Projet Danseuse Capybara

7.2 Exemples de requêtes

Listing 1 – Exemples de requêtes HTTP

```

1 # Etat du système
2 GET http://<ip_esp8266>/status
3
4 # Changer de chanson (ex: 2)
5 GET http://<ip_esp8266>/song?i=2
6
7 # Activer le mode muet
8 GET http://<ip_esp8266>/mute?on=1
9
10 # Modifier LCD
11 GET http://<ip_esp8266>/lcd?l1=Bonjour&l2=Danseuse

```

8 Procédure de compilation et déploiement

8.1 Téléversement sur ESP8266

1. Ouvrir le fichier .ino du projet dans l'IDE Arduino.
2. Sélectionner la carte ESP8266 et le port série.
3. Vérifier la présence des bibliothèques requises.
4. Compiler puis téléverser.

8.2 Vérifications

- Vérifier la connexion WiFi et récupérer l'adresse IP affichée (LCD ou moniteur série).
- Tester /status depuis un navigateur ou un outil HTTP.
- Vérifier la réaction à la présence et la cohérence des modes (nuit/muet).

9 Gestion de projet et travail collaboratif (Git)

9.1 Organisation

Le projet est versionné avec Git et hébergé sur GitHub. Un flux de travail basé sur des branches a été utilisé :

- branche **main** : version initial du professeur
- branches **feature/*** : développements et intégrations progressives
- intégration finale via *pull request* (*non réalisé*)

9.2 Justification

Ce flux permet :

- une traçabilité des modifications,
- une meilleure gestion des conflits,
- une intégration contrôlée et revue.

10 Conclusion

Le projet Danseuse Capybara met en œuvre une application embarquée complète combinant capteurs, actionneurs et communication réseau, tout en structurant le code selon des principes POO. L'architecture modulaire facilite l'extension (ajout de nouveaux capteurs/actionneurs) et la maintenance.

Améliorations possibles :

- ajout d'un diagramme de séquence UML (scénario nominal),
- enrichissement de l'état JSON (diagnostic, timestamps),
- gestion plus avancée des états (automate) et des priorités.

A Annexe A – Captures du projet (PNG)

Les figures suivantes illustrent le projet dans son état final. Les fichiers PNG sont fournis dans le dépôt dans un dossier `demo/images/`.

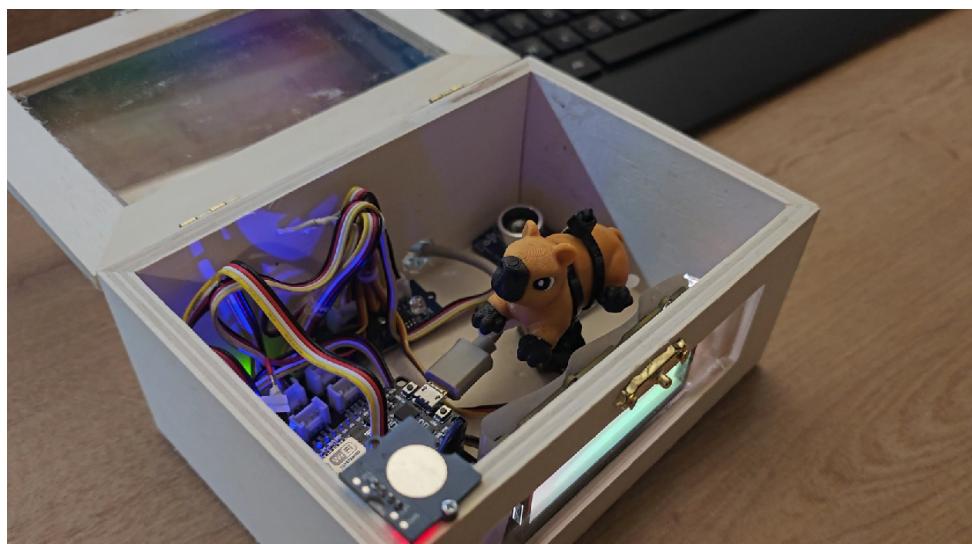


FIGURE 3 – Vue 1 : montage matériel



FIGURE 4 – Vue 2 : montage matériel

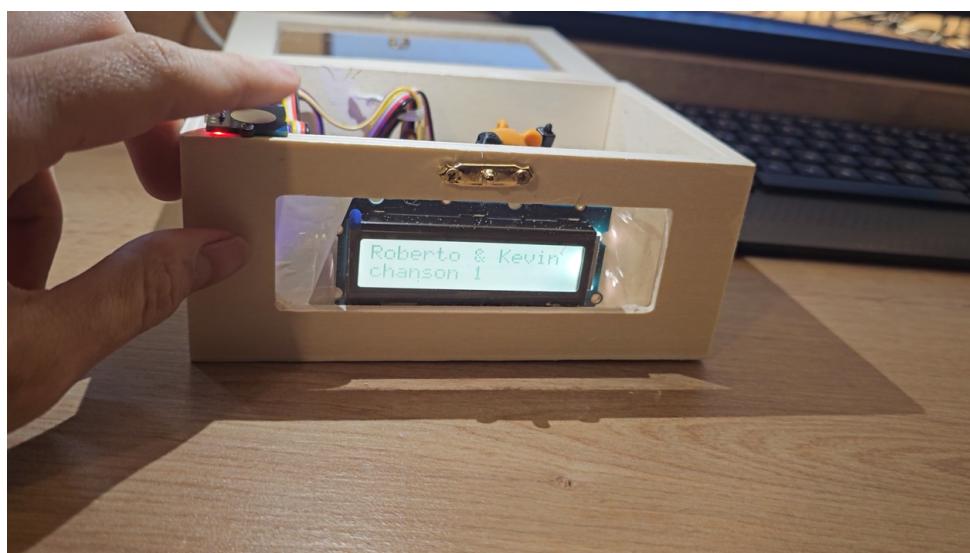


FIGURE 5 – Vue 3 : montage matériel