

HE-ARC

CONCEPTION OS

---

# Arduinozore

---

*Auteurs :*

PEDRETTI Maël  
NETO DA SILVA André  
GANDER Laurent

*Responsables :*

CORTINOVIS Claudio

2 mai 2018

**haute école**  
neuchâtel berne jura



**ingénierie**  
saint-imier le locle delémont

## Résumé

Le projet Arduinozore a été réalisé dans le cadre du cours de Conception OS, un module de 3ème année de Bachelor au sein de la Haute-École Arc — Ingénierie, section Développement Logiciel et Multimédia. Le présent document décrit le déroulement du projet et présente les fonctionnalités implémentées. Arduinozore est un programme en Python qui permet de visualiser les valeurs de capteurs branchés sur un Arduino et connectés à un ordinateur ou un RaspberryPi. Au travers d’une interface web, il facilite la configuration et la récupération des données.

## Abstract

The Arduinozore project was realized within the context of the OS conception course, a 3rd year Bachelor module within the Arc-Engineering High School, section Software and Multimedia Development. This document describes the progress of the project and presents the implemented features. Arduinozore is a program in Python that allows you to visualize datas from sensors wired to an Arduino and connected to a computer or a RaspberryPi. Through a web interface it facilitates the configuration and fetch of datas.

## Table des matières

<b>1</b>	<b>Lexique</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>1</b>
<b>3</b>	<b>Rappel des objectifs</b>	<b>3</b>
3.1	Récupération des données . . . . .	3
3.2	Activation de relais . . . . .	3
3.3	Global . . . . .	3
3.4	Secondaires . . . . .	3
<b>4</b>	<b>Fonctionnalités implémentées</b>	<b>3</b>
4.1	Arduino . . . . .	3
4.2	Communication avec les Arduinos . . . . .	4
4.3	Interface web . . . . .	4
<b>5</b>	<b>Architecture</b>	<b>4</b>
5.1	Arborescence du projet . . . . .	4
5.2	Diagramme UML . . . . .	6
5.3	Description des classes . . . . .	8
5.4	Multiprocessing . . . . .	9
5.5	Format de données . . . . .	9
<b>6</b>	<b>Méthodologie</b>	<b>9</b>
6.1	Integration continue . . . . .	10
6.2	Déploiement continu . . . . .	10
<b>7</b>	<b>Difficultés rencontrés</b>	<b>10</b>
<b>8</b>	<b>Résultats</b>	<b>10</b>
8.1	Communication . . . . .	10
8.2	Interface . . . . .	10
<b>9</b>	<b>Évolutions possibles</b>	<b>13</b>
<b>10</b>	<b>Conclusion</b>	<b>13</b>
<b>11</b>	<b>Bibliographie</b>	<b>13</b>

## Table des figures

1	Diagramme de communication entre les composants . . . . .	2
2	Diagramme du protocole de communication . . . . .	4
3	Arborescence du projet . . . . .	5
4	Diagramme uml . . . . .	7
5	Page d'accueil . . . . .	11
6	Paramètres . . . . .	11
7	Paramètres des cartes . . . . .	11
8	Formulaire d'enregistrement de carte . . . . .	12
9	Configuration pour la carte Uno . . . . .	12
10	Liste des cartes configurées . . . . .	12

# 1 Lexique

Afin de simplifier la lecture, un lexique est présenté ci-après.

TAB. 1 : Lexique

Mot clé	Définition
Python	Le langage de programmation utilisé pour le développement de l'application. Un programme python nécessite d'être exécuté par un interpréteur Python. Un programme Python dépend souvent d'un certain nombre de packages Python.
Package	Un ensemble de fonctionnalités implémentées et offertes par des membres de la communauté Python, afin d'étendre les possibilités du langage. Un package utilisé par un programme Python doit être installé sur l'ordinateur du client. Docker Hub Taxonomy utilisant de multiples packages, il est nécessaire de les installer avant de pouvoir utiliser l'application.
Arduino	Un Arduino est une petite carte électronique facilitant l'apprentissage de l'électronique et la programmation ainsi que le prototypage [?].
RaspberryPi	Un RaspberryPi est un mini-ordinateur équipé d'un microprocesseur ARM et 256 à 512 mo de RAM. L'intérêt du produit se trouve dans sa très faible consommation en énergie et son coût très bas. [2].
WebSocket	Le protocole WebSocket vise à développer un canal de communication full-duplex sur un socket TCP pour les navigateurs et les serveurs web [5]. En une phrase simple : les WebSockets permettent de créer des applications temps-réel sur le web [6].
YAML	YAML Ain't Markup Language. YAML est un format de représentation de données par sérialisation Unicode. L'idée de YAML est que presque toute donnée peut être représentée par combinaison de listes, tableaux associatifs et données scalaires. YAML décrit ces formes de données (les représentations YAML), ainsi qu'une syntaxe pour présenter ces données sous la forme d'un flux de caractères (le flux YAML) [7]. Ce langage de stockage de données permet d'éviter l'utilisation d'une base de donnée qui serait lourde pour ce projet. Il stocke les données sérialisée directement dans un fichier sur le disque. De plus, il permet de charger les fichiers de manière sécurisée afin d'éviter les injections de code.

# 2 Introduction

La domotique prenant de plus en plus de place dans les foyers modernes et les prix de ces systèmes étant élevés, il a été nécessaire d'essayer de trouver une alternative moins coûteuse.

Les cartes Arduinos ainsi que RaspberryPi restant relativement bon marché malgré leurs capacités toujours plus grandes, le choix s'est porté sur ces modèles. Ces deux équipements n'étant pas prévu pour communiquer ensemble de manière native, il est nécessaire de développer une solution leur permettant de se comprendre. Cependant, le projet étant destiné à un usage open source, il est préférable de garder un système dans lequel l'Arduino peut communiquer avec un autre matériel que le PI.

La figure 1 représente la communication entre les différents composants de notre système

Le but de ce projet est de pouvoir visualiser via un interface web les valeurs de capteurs connectés à un RaspberryPi.

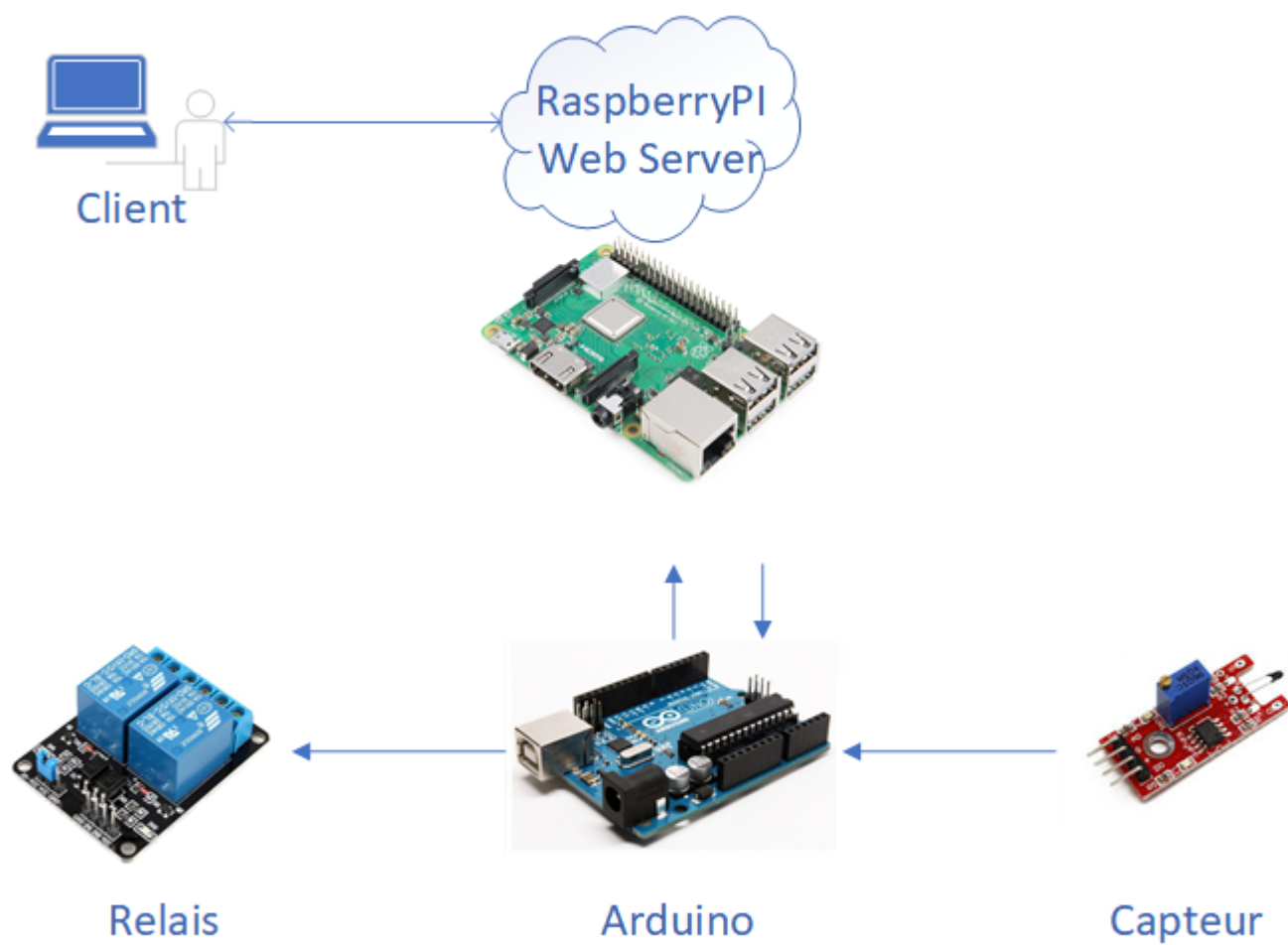


FIG. 1 : Diagramme de communication entre les composants

## 3 Rappel des objectifs

### 3.1 Récupération des données

1. Trouver un moyen de pouvoir lire n'importe quel capteur depuis l'arduino et envoyer la valeur sur le port série. Ne pas faire de traitement, juste lire les valeurs et les transmettre. Produire un code "générique".
2. Développer un logiciel (python ou autre) qui lise ce paramètre sur le port série et qui l'interprète en fonction de configurations mises en place auparavant.
3. Développer une appli web ou desktop qui permette de se connecter au pi et d'afficher les données de manière lisible.

### 3.2 Activation de relais

1. Produire un code générique pour pouvoir activer ou désactiver une gpio de manière générique.
2. Ajouter au logiciel la possibilité de trigger ce relai.
3. Ajouter à l'appli web la même possibilité.

### 3.3 Global

1. Trouver un moyen de donner un nom à l'arduino pour l'identifier sur le port com. "Salon" donne plus d'infos que "Arduino Uno sur COM1"

### 3.4 Secondaires

1. Modifier l'appli web pour ajouter la possibilité de paramétrer les nouveaux capteurs ou relais depuis là de manière simple. (Des clics sur des boutons sont toujours plus agréables que la ligne de commande pour les utilisateurs lambda)
2. Ajouter la possibilité de rajouter des capteurs sur les arduinos. Ils disposent de 6 entrées analogiques, autant toutes les utiliser.

## 4 Fonctionnalités implémentées

En l'état, une fois lancé, le programme est capable de récupérer les données des capteurs branchés à un Arduino ainsi que de changer l'état de ses sorties afin, par exemple, de faire commuter un relais.

### 4.1 Arduino

Pour que l'Arduino soit capable de communiquer via le port série, il est nécessaire de déterminer un protocole de communication.

La figure 2 illustre ce protocole.

Tout d'abord, lorsque l'Arduino est allumé, il s'annonce en continu tant que l'appareil en face ne lui envoie pas la commande ok. Suite à cela, il est possible de consulter la valeur d'une entrée ou de changer l'état d'une sortie.

Pour ce faire, les commandes sont les suivantes :

```
1 usage :
  - lire les données disponibles sur le port série
  - envoyer "ok" pour commencer
  - lire les données disponibles sur le port série
  - envoyer :
6   - "w" + [NUMERO_SORTIE] pour changer l'état d'une sortie
  - "r" + [NUMERO_ENTREE] pour lire la valeur d'une entrée
```

Lors du changement d'état d'une sortie, l'Arduino ne renvoie rien.

Par contre, lors de la lecture d'une entrée, l'Arduino renvoie la valeur lue sans traitement. La valeur est comprise entre 0 (0 volt) et 1023 (5 volts). La précision est donc de 0.005 volts.

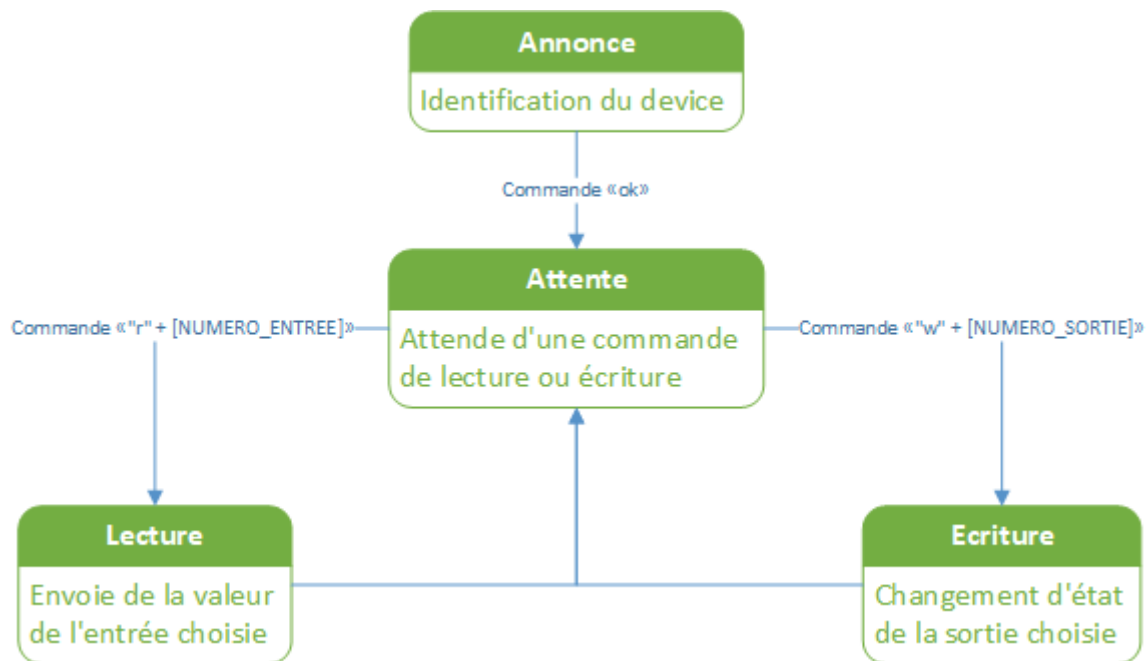


FIG. 2 : Diagramme du protocole de communication

## 4.2 Communication avec les Arduinos

Afin de communiquer avec les arduinos, le package Python PySerial est utilisé. Il permet de faciliter l'usage d'un port série en python. Dès lors, il est simple de respecter le protocole de communication décrit ci-dessus.

## 4.3 Interface web

Afin d'afficher les mesures et de pouvoir changer l'état des sorties, un interface web est présent. Il permet depuis la page d'accueil d'afficher les Arduinos connectés et de les configurer. Il permet la création de capteurs afin d'enregistrer les configurations. Il en est de même pour les différents types de cartes Arduino.

# 5 Architecture

Ce chapitre décrit l'architecture du projet. En premier lieu, une brève explication sur les fichiers est donnée puis l'organisation du projet est représentée.

## 5.1 Arborescence du projet

Comme l'illustre la figure 3, le dossier de projet arduinozore est sous la forme d'un projet github [3].

Il est possible d'y trouver un fichier .gitignore, un script de test concernant l'assurance qualité du code et son fichier de configuration, un fichier README expliquant brièvement le projet, les configurations pour rendre le paquet installable, les dossiers doc et arduino et le package arduinozore. Ce sont ces derniers qui sont expliqués.

Le premier dossier, arduino, contient le code arduino à flasher sur les devices.

Le deuxième dossier, appelé lui arduinozore, contient lui les codes sources permettant de faire fonctionner le projet.

Le fichier `__main__.py` est le point d'entrée du projet. C'est ce package qui lance le serveur et instancie les différents process qui seront utilisés pour récupérer les données.

Le fichier `__init__.py` est un fichier utilisé pour que python traite le dossier comme un package afin de repérer les sous packages.

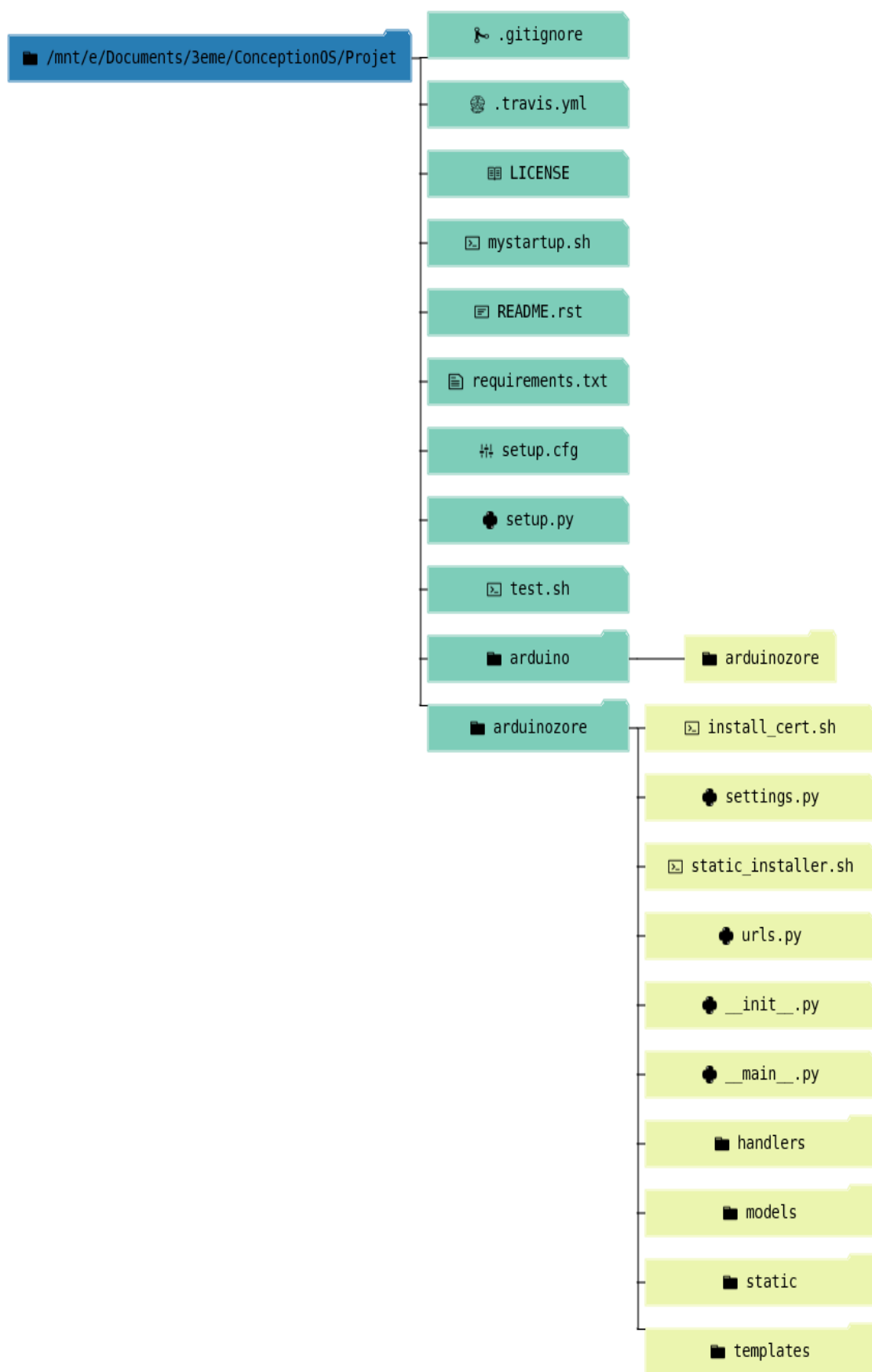


FIG. 3 : Arborescence du projet



Le fichier `install_cert.sh` sert à générer les certificats ssl pour communiquer via https.

Le fichier `static_installer.sh` sert à télécharger les fichiers statiques utilisés pour le rendu graphique de l'application.

Le fichier `settings.py` contient les réglages pour le serveur web.

Le fichier `urls.py` contient la liste des urls atteignables et leurs actions respectives.

Le dossier `handlers` contient les gestionnaires qui exécutent les actions relatives aux urls.

Le dossier `static` contient les fichiers statiques qui seront servis par le serveur (feuilles de style en cascade, fichiers javascript, etc.).

Le dossier `templates` contient les templates de pages web utilisées pour le rendu.

Comme on peut le constater, ce package est sous la forme d'un package web.

Le dernier dossier, `Doc`, contient le rapport et le manuel utilisateur. Ces 2 fichiers étant réalisés en `RestStructuredText`, ils sont ensuite convertis en pdf en utilisant le projet `Technical Report`[4]. Ceci libère d'une tâche de mise en page étant donné qu'elle est générée automatiquement.

## 5.2 Diagramme UML

La figure 4 représente le diagramme UML du projet. Il est expliqué ci-après.

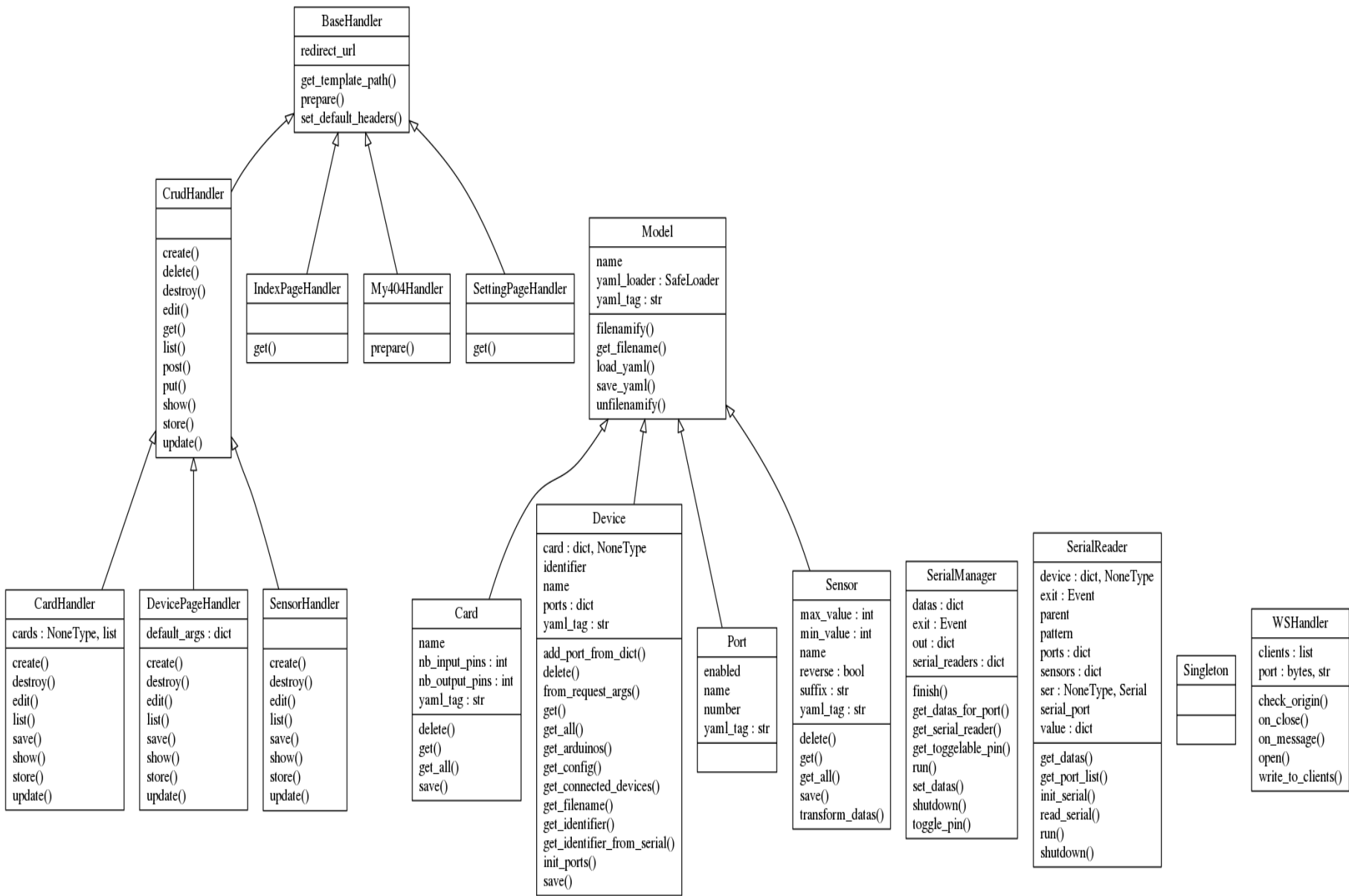


FIG. 4 : Diagramme uml

## 5.3 Description des classes

La majorité des classes du package handlers dérivent de "BaseHandler". Cette base contient des configurations qui sont identiques à tous les autres gestionnaires.

Les classes SerialManager et SerialReader dérivent de "multiprocessing.Process" afin de pouvoir travailler simultanément. De plus amples explications peuvent être trouvées dans la section suivante.

La classe WSHandler dérive elle de "tornado.websocket.WebSocketHandler". Il s'agit du gestionnaire pour toutes les connexions WebSocket.

Les classes du package models dérivent quant à elles de la classe de base Model.

### 5.3.1 BaseHandler

Cette classe est la classe de base de laquelle héritent tous les gestionnaires.

Elle fixe les entêtes de communication et redirige http vers https. De plus, elle localise le dossier contenant les templates.

### 5.3.2 CrudHandler

Cette classe est la classe de base de laquelle héritent tous les gestionnaires qui permettent le CRUD [8]. Elle hérite de la classe BaseHandler.

Cette classe permet de tromper l'utilisateur sur les méthodes HTTP utilisées. Comme les navigateurs ne peuvent pas, à l'heure actuelle, utiliser les méthodes PUT, PATCH, DELETE, cette classe permet de faire comme si ces méthodes étaient utilisées au travers de requêtes POST.

### 5.3.3 CardHandler

Cette classe est le gestionnaire de tout ce qui touche aux cartes. Elle est capable de lire les configurations de cartes déjà enregistrées, de sauvegarder les modifications s'il y en a et d'éventuellement supprimer ces configurations. En fonction des liens atteints, elle affiche la configuration, son formulaire de création, son formulaire de modification ou la liste des configurations à disposition.

### 5.3.4 DevicePageHandler

Cette classe est idem à la précédente si ce n'est qu'elle agit pour tout ce qui touche aux devices.

### 5.3.5 SensorHandler

Cette classe est idem à la précédente si ce n'est qu'elle agit pour tout ce qui touche aux capteurs.

### 5.3.6 IndexPageHandler

Cette classe est le gestionnaire de la page d'accueil. Elle est capable de récupérer les Devices connectées et de les afficher.

### 5.3.7 SettingPageHandler

Cette classe est le gestionnaire de la page des configurations. Elle permet d'afficher les types de configuration disponibles.

### 5.3.8 SerialManager

Cette classe est un singleton responsable d'attribuer un SerialReader pour chaque Arduino connecté. De ce fait, il est sûr que les nouveaux processus sont lancés et arrêtés proprement et qu'un seul processus est lancé par Device.

### 5.3.9 SerialReader

Cette classe est responsable de la communication avec les devices. Afin de rendre les échanges asynchrones et de ne pas bloquer le serveur pour lire une valeur, ces classes sont des processus lancés à côté du processus parent. Ils sont gérés par la classe SerialManager.

Les processus ont été préférés aux threads car leur manipulation est plus simple en python.

### 5.3.10 WSHandler

Cette classe est le gestionnaire des connections aux websockets. Lors de l'ouverture d'une connection (Donc lorsque l'utilisateur souhaite manipuler un arduino), elle s'assure qu'un processus de communication est lancée et fait le pont entre l'utilisateur et le SerialManager qui communique avec les SerialReaders.

### 5.3.11 Model

Cette classe est la classe de base de laquelle héritent tous les modèles. Elle fixe les fonctions de base disponibles dans tous les modèles et configure le chargement et l'écriture des configurations sur le disque.

### 5.3.12 Card

Cette classe est le modèle de données pour les cartes. Elle hérite de la classe Model et permet de lire et écrire les configurations sur le disque.

### 5.3.13 Device

Cette classe est idem à la précédente si ce n'est qu'il s'agit du modèle pour les devices.

### 5.3.14 Sensor

Cette classe est idem à la précédente si ce n'est qu'il s'agit du modèle pour les capteurs.

## 5.4 Multiprocessing

Le package Multiprocessing permet d'exécuter des tâches de manière concurrente. Les classes SerialManager et SerialReader sont lancées comme un ou plusieurs process et permettent donc d'exécuter des tâches en parallèle. De ce fait, les process ne peuvent pas communiquer de manière normale entre eux avec des listes, ils doivent utiliser des queues ou des variables spéciales qui empêchent les conflits d'écriture ou de lecture.

## 5.5 Format de données

Afin de simplifier le stockage des configurations, le format de données YAML a été choisi. Il est plus léger d'utilisation qu'une base de données pour un projet de petite envergure comme le notre.

# 6 Méthodologie

Le projet s'est principalement déroulé selon une méthodologie de recherches puis d'implémentation. Il a fallu se renseigner sur beaucoup d'aspect tel que la meilleure technologie à utiliser, les limitations et possibilités des Arduinos, comment communiquer entre le RaspberryPi et le(s) Arduino(s), etc.

Plus de la moitié du temps consacré au projet s'est déroulé dans le cadre de recherches. De ce fait, aucun planning n'a été défini à l'avance car il était impossible d'évaluer le temps nécessaire pour effectuer ces recherches. Le projet s'est donc déroulé de manière itérative. C'est à dire que chaque fois qu'une fonctionnalité était implémentée, elle était testée à la main et de manière automatisée puis une nouvelle recherche et une nouvelle implémentation suivait.

Au fur et à mesure il a été possible de développer des solutions suivant les objectifs. Tout d'abord, un simple script permettait la communication entre RaspberryPi et Arduino, ensuite un interface web permettait de visualiser les données de ce Device. Suite à cela, le multiprocessing a été implémenté afin de gagner en performances et de ce fait, la communication entre les process a du être implémentée de manière concurrente. Ne connaissant pas le multiprocessing en python, il a fallu à nouveau effectuer des recherches. Finalement, l'interface web avec les configurations disponibles a pu être implémenté.

## 6.1 Intégration continue

Durant tout le processus de développement, des tests d'assurance qualité concernant la qualité du code ont été réalisés afin de garder le code lisible et compréhensible pour un autre développeur. Ces tests ont été automatisés grâce à l'utilisation de l'intégration continue. Il s'agit de réaliser des tests sur le code qui est mis en ligne sur un dépôt Git. Dans ce projet, le choix s'est porté sur Travis CI [9]. Ce système de test d'intégration continue permet de configurer plusieurs choses dont le système d'exploitation sur lequel les tests sont menés et également quelles versions de python sont testées. Pour ce projet, le code est testé avec les versions 3.4, 3.5 et 3.6 de Python sur un os Linux.

## 6.2 Déploiement continu

Suite à cela, nous avons choisi de créer une application installable au moyen du gestionnaire de dépendance intégré à Python, "Pip".

Afin de simplifier les déploiements, nous avons rajouté une étape à notre système d'intégration continue. Si tous les tests sont validés et que le commit est tagué avec une version qui est sous la forme 0.1.0, l'application est construite et déployée sur l'hébergeur de paquets Python Pypi.

De ce fait, beaucoup de temps a été gagné lors des tests et des déploiements étant donné qu'ils étaient réalisés automatiquement.

## 7 Difficultés rencontrés

Plusieurs difficultés ont été rencontrées durant la réalisation de ce projet. En effet, n'ayant pas un cahier des charges basé sur des actions à effectuer mais sur des recherches à approfondir, il a fallu trouver des technologies compatibles et apprendre des nouvelles pratiques avant de pouvoir développer le projet.

Tout d'abord, il a fallu trouver quels utilitaires utiliser pour communiquer entre les différents composants de l'application.

Suite à cela, pour pouvoir améliorer les performances, il a été nécessaire d'implémenter du parallélisme. La communication inter-process n'étant pas toujours facile à cerner et réaliser de manière correcte, un certain temps a été utilisé à ces fins.

Il a fallu ensuite implémenter l'interface web complet et d'autres difficultés sont apparues. Il a été difficile de lier WebSockets, Processus et Arduinos. De plus, le stockage des configurations est resté longtemps un problème.

Après avoir obtenu des résultats concluants concernant le fonctionnement de l'application sur nos systèmes respectifs, il a fallu réaliser l'installer pour le paquet. L'emplacement des fichiers de configuration posé des problèmes car il a fallu trouver dans quels dossiers il était possible de créer un dossier sans privilèges. Nous avons décidé de créer un dossier caché situé dans le dossier personnel de l'utilisateur.

## 8 Résultats

Grace à l'utilisation des technologies citées dans la section précédente, il a été possible de créer une interface web permettant la configuration et l'affichage des communications avec les Arduinos.

### 8.1 Communication

Comme expliqué au chapitre 4.1, la communication est possible avec l'Arduino de manière simple.

Tout d'abord, l'Arduino s'annonce sur le port série jusqu'à ce que l'utilisateur lui indique qu'il est prêt.

Dès lors, il est possible de communiquer avec une console série ou via l'interface web de manière simple.

### 8.2 Interface

La figure 5 illustre la page d'accueil sans devices connectées.

La figure 6 illustre la page des paramètres disponibles. Il est possible de configurer des cartes (Uno, Mega, etc.), des capteurs et les devices possédant déjà une configuration.

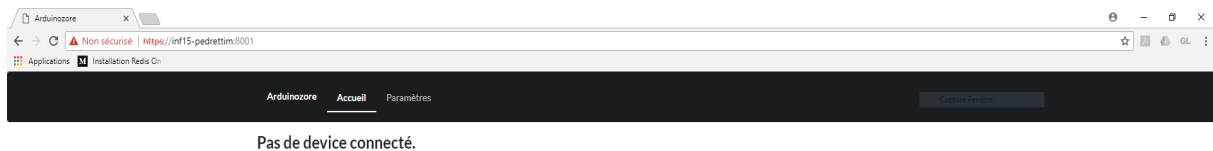


FIG. 5 : Page d'accueil

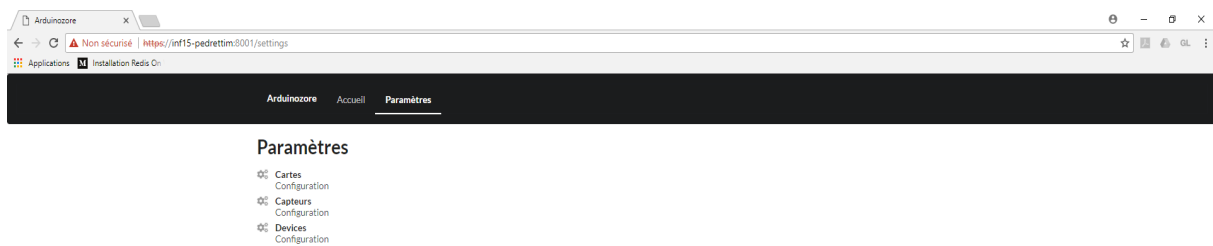


FIG. 6 : Paramètres

La figure 7 illustre la page des paramètres des cartes. Comme aucune carte n'a encore été configurée, il est possible de créer une configuration.

La figure 8 illustre le formulaire de création de cartes.

La figure 9 illustre l'affichage d'une configuration créée.

La figure 10 illustre la page des paramètres des cartes. Étant donné que des cartes ont été créées, les configurations sont listées.

Les pages concernant les capteurs et les devices sont identiques. Il est à noter que lors du branchement d'une nouvelle carte, si l'utilisateur souhaite visualiser ses données, le formulaire de création de configuration de device lui sera affiché. Une fois le device configuré, il sera possible d'interagir avec.

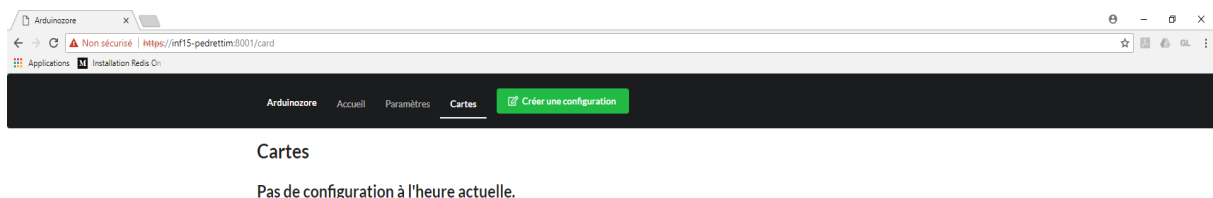


FIG. 7 : Paramètres des cartes

Arduinozore Accueil Paramètres Cartes [Créer une configuration](#)

Nom de la carte  
Uno

Nombre d'entrées analogiques  
6

Nombre de sorties digitales  
14

Envoyer

FIG. 8 : Formulaire d'enregistrement de carte

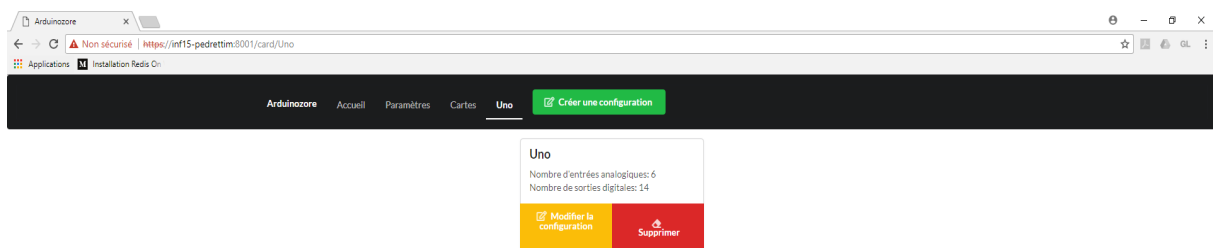


FIG. 9 : Configuration pour la carte Uno

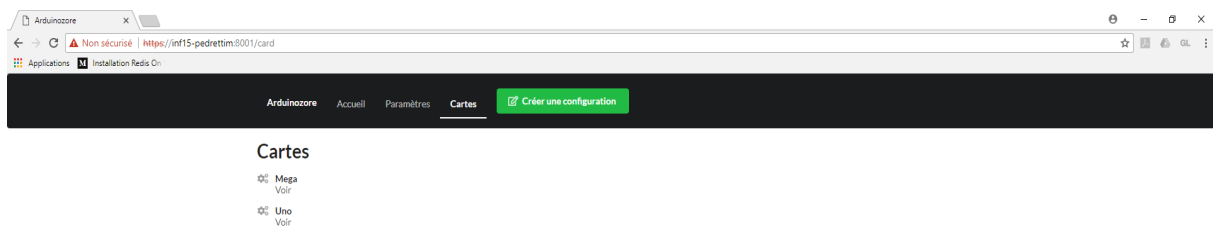


FIG. 10 : Liste des cartes configurées

## 9 Évolutions possibles

Pour le moment, aucune protection n'a été implémentée. Tout utilisateur disposant de l'adresse du serveur peut y accéder et modifier les configurations. Ce projet étant destiné à un usage domestique dans un réseau isolé, nous n'avons pas mis un point d'honneur à développer cet aspect.

Il n'est également pas possible d'utiliser des capteurs qui ont besoin d'être activé au moyen d'une impulsion sur une entrée. Cet aspect reste relativement simple à développer étant donnée qu'il est possible d'activer des pins de sortie sur les Arduinos.

S'ajoute à cela l'impossibilité de lire les données de capteurs codées sur plus de 10 bits. Il n'est pour le moment pas possible d'aller lire des registres supplémentaires de l'Arduino pour récupérer l'entièreté d'une mesure. Si le temps l'avait permis, une solution à ce problème aurait été cherchée.

Si un capteur ne peut communiquer que via une connection série avec l'Arduino, il n'est également pas possible de l'utiliser. Ceci est également du au manque de temps.

## 10 Conclusion

Au premier abord, le problème paraît simple à résoudre. Récupérer des données sur un Arduino, les afficher sur une page web, pouvoir configurer les devices. Cependant, plus le projet progresse et plus la difficulté augmente. Après avoir résolu des solutions pour les paliers précédents, il s'est parfois avéré que des problèmes avaient été créés pour lesquels il a fallu également trouver des solutions.

Durant tout le déroulement du projet, des avancées ont été réalisées mais presque toutes ont été ralenties par l'apparition de nouvelles difficultés.

À l'heure actuelle, l'application Arduinozore fonctionne et les objectifs principaux ont presque entièrement été respectés. Il est possible de lire les données de capteurs de manière générique, l'interface web permet l'affichage de ces données et le pilotage des sorties. L'Arduino est identifiable via un nom depuis l'application web grâce à son Hardware ID unique. De plus, les objectifs secondaires l'ont également été. Il est aussi possible de paramétrer les capteurs et cartes de manière simple. Les configurations sont également stockées. Cependant, le produit étant fonctionnel, il n'est pas terminé. Plusieurs d'améliorations, citées dans le chapitre précédent, sont encore possibles.

De plus, l'efficacité exacte d'une telle application est difficile à tester. En effet, tester que tous les messages envoyés de l'arduino sont affichés sur la page est compliqué. Il faudrait mettre en place des simulateurs et d'autres batteries de tests beaucoup plus compliquées.

## Références

### 11 Bibliographie

- [1] Site web Arduino 25.04.2018. <https://www.arduino.cc>
- [2] Site web RaspberryPi 25.04.2018. <https://www.raspberrypi.org>
- [3] Arduinozore sur github 26.01.2018. <https://github.com/S-Amiral/arduinozore>
- [4] TechnicalReport sur github 26.01.2018. <https://github.com/73VW/TechnicalReport>
- [5] WebSocket sur Wikipédia 26.01.2018. <https://fr.wikipedia.org/wiki/WebSocket>
- [6] WebSocket sur Binio.io 26.01.2018. <https://blog.bini.io/intro-websocket/>
- [7] YAML sur Binio.io 26.01.2018. <https://fr.wikipedia.org/wiki/YAML>
- [8] CRUD sur Wikipédia 26.01.2018. <https://fr.wikipedia.org/wiki/CRUD>
- [9] Travis CI 26.01.2018. <https://about.travis-ci.com>