**Compte rendu**

**Projet Embarquai2**

****

|  |
| --- |
| BENAITIER Loris, DORIATH Arthus  ICSSA – BTS SNIR 2021/2022 |

[INTRODUCTION 7](#_Toc104823231)

[REPARTITION DES TACHES 8](#_Toc104823232)

[1. Etudiant 1 DORIATH Arthus 8](#_Toc104823233)

[2. Etudiant 2 BENAITIER Loris 8](#_Toc104823234)

[3. Etudiant 3 BOSSUET Baptiste 8](#_Toc104823235)

[PLANNING PREVISIONNEL 9](#_Toc104823236)

[SPRINT 10](#_Toc104823237)

[BASE DE DONNÉES 13](#_Toc104823238)

[ANALYSE UML 14](#_Toc104823239)

[1. Diagramme des exigences 14](#_Toc104823240)

[2. Diagramme de déploiement 15](#_Toc104823241)

[3. Diagramme de cas d’utilisation 16](#_Toc104823242)

[4. Diagramme d’activité 17](#_Toc104823243)

[5. Diagramme de classe 18](#_Toc104823244)

[6. Diagramme de séquence 19](#_Toc104823245)

[7. Diagramme de définition de bloc 20](#_Toc104823246)

[MATERIELS UTLISES 21](#_Toc104823247)

[1. Présentation du M5Stack Core2 23](#_Toc104823248)

[LOGICIELS ET LIBRAIRIES UTILISES 26](#_Toc104823249)

[1. Logiciels 26](#_Toc104823250)

[2. Librairies 27](#_Toc104823251)

[SYSTEME D’EXPLOITATION UTILISE 28](#_Toc104823252)

[LANGAGES DE PROGRAMMATIONS UTILISES 29](#_Toc104823253)

[RECUPERATIONS DONNEES CAPTEURS VIA LE M5STACK CORE2 30](#_Toc104823254)

[ENVOIENT DES DONNEES DEPUIS LE M5STACK CORE2 31](#_Toc104823255)

[1. Auteur 32](#_Toc104823256)

[2. Objectif 32](#_Toc104823257)

[3. Cahier des charges 32](#_Toc104823258)

[4. Matériels nécessaires 32](#_Toc104823259)

[4.1. Module UNIT LoRaWan868 33](#_Toc104823260)

[4.2. Module Lorawan : Xbee Waveshare + Carte ATIM LoRaWan 34](#_Toc104823261)

[5. Problème rencontré 35](#_Toc104823262)

[6. Environnement de développement 35](#_Toc104823263)

[7. Présentation de Lora et Lorawan 36](#_Toc104823264)

[7.1. Introduction 36](#_Toc104823265)

[7.2. Qu’est-ce que LoRa ? 36](#_Toc104823266)

[7.3. Qu’est-ce que LoRaWAN ? 36](#_Toc104823267)

[7.4. Quels sont les avantages des protocoles LoRa et LoRaWAN ? 37](#_Toc104823268)

[8. Liaison physique : module Lorawan/M5Stack Core2 37](#_Toc104823269)

[9. Choix du format de la trame contenant les données 39](#_Toc104823270)

[9.1. Caractères au début et à la fin de la trame de données 39](#_Toc104823271)

[9.2. Caractères de séparations entre chaque champ de la trame 39](#_Toc104823272)

[9.3. FLAG 39](#_Toc104823273)

[9.4. Données 41](#_Toc104823274)

[9.5. CRC 42](#_Toc104823275)

[10. Envoi de la trame contenant les données 47](#_Toc104823276)

[11. Test unitaire pour l’envoi de la trame contenant les données 48](#_Toc104823277)

[12. Test unitaire pour la génération de fausse valeur 49](#_Toc104823278)

[13. Intégration avec l’étudiant 1 51](#_Toc104823279)

[14. Conclusion 51](#_Toc104823280)

[RECUPERATIONS DES DONNEES ENVOYEES PAR LE M5STACK CORE2 52](#_Toc104823281)

[1. Auteur 53](#_Toc104823282)

[2. Objectif 53](#_Toc104823283)

[3. Cahier des charges 53](#_Toc104823284)

[4. Matériels et logiciels nécessaires 53](#_Toc104823285)

[4.1. Routeur multitech MTCAP 868 54](#_Toc104823286)

[4.2. Raspberry pi 55](#_Toc104823287)

[5. Configuration routeur multitech Lorawan (MTCAP 868) 56](#_Toc104823288)

[5.1. Introduction 56](#_Toc104823289)

[5.2. Configuration des paramètres réseau du routeur 56](#_Toc104823290)

[5.3. Démarrage du NETWORK SERVEUR 57](#_Toc104823291)

[5.4. Ajouts des identifiants d’un module Lorawan (END-DEVICE) 57](#_Toc104823292)

[5.5. Ajout d’un appareil pour le lié avec un module Lorawan 59](#_Toc104823293)

[5.6. Transferts des données en MQTT 60](#_Toc104823294)

[5.7. Paramètre en fonction de notre localisation 61](#_Toc104823295)

[6. Configuration du serveur Mosquitto sur le PC administrateur 61](#_Toc104823296)

[7. Test du serveur Mosquitto 62](#_Toc104823297)

[8. Test serveur Mosquitto + routeur Lorawan + module Lorawan 63](#_Toc104823298)

[9. Conclusion 64](#_Toc104823299)

[ENREGISTREMENT DES DONNEES DANS LA BDD 65](#_Toc104823300)

[1. Auteur 66](#_Toc104823301)

[2. Objectif 66](#_Toc104823302)

[3. Cahier des charges 66](#_Toc104823303)

[4. Matériel et logiciels Nécessaires 66](#_Toc104823304)

[5. Décodage de la trame 66](#_Toc104823305)

[5.1. Récupérer le paquet qui contient la donnée 66](#_Toc104823306)

[5.2. Extraire la trame donnée du paquet Lorawan 67](#_Toc104823307)

[5.3. Extraire les différentes informations contenues dans la trame 67](#_Toc104823308)

[5.4. Traitement du CRC 67](#_Toc104823309)

[5.5. Traitement du flag 68](#_Toc104823310)

[6. Enregistrement dans la base de données 69](#_Toc104823311)

[7. Gestion performance du serveur 70](#_Toc104823312)

[8. Gestion erreurs du serveur 70](#_Toc104823313)

[9. Test unitaire 71](#_Toc104823314)

[10. Intégration avec l’étudiant 3 73](#_Toc104823315)

[11. Conclusion 73](#_Toc104823316)

[TRAITEMENT DES DONNEES VIA UNE IHM 74](#_Toc104823317)

[1. Auteur 75](#_Toc104823318)

[2. Situation 75](#_Toc104823319)

[IHM DU M5STACK CORE2 77](#_Toc104823320)

[DOSSIER MAINTENANCE ENVOI/RECEPTION/ENREGISTREMENT DONNEE 78](#_Toc104823321)

[1. Auteur 79](#_Toc104823322)

[2. Objectif 79](#_Toc104823323)

[3. Debug 79](#_Toc104823324)

[INSTALLATIONS PARTIES ENVOI/RECEPTION/ENREGISTREMENT DONNEE 81](#_Toc104823325)

[1. Auteur 82](#_Toc104823326)

[2. Objectif 82](#_Toc104823327)

[NOTICE UTILISATION ROUTEUR/SERVEUR MOSQUITTO/SERVEUR PYTHON 83](#_Toc104823328)

[1. Auteur 84](#_Toc104823329)

[2. Objectif 84](#_Toc104823330)

[TEST FINAL DE L’ENSSEMBLE DU PROJET 85](#_Toc104823331)

[1. Test qui rassemble les parties de chaque étudiant 86](#_Toc104823332)

[CONCLUSION 88](#_Toc104823333)

[ANNEXES 89](#_Toc104823334)

[1. Annexes techniques (documents constructeurs) 90](#_Toc104823335)

[1.1. Routeur Multitech Lorawan (LORIS BENAITIER) 90](#_Toc104823336)

[1.2. Module Lorawan : Carte ATIM LoRaWan (LORIS BENAITIER) 90](#_Toc104823337)

[1.3. M5Stack Core2 90](#_Toc104823338)

[2. Codes sources 90](#_Toc104823339)

[2.1. Envoient des données (LORIS BENAITIER) 90](#_Toc104823340)

[2.2. Réceptions et enregistrements des données (LORIS BENAITIER) 90](#_Toc104823341)

[2.3. Script Bash (LORIS BENAITIER) 90](#_Toc104823342)

[3. Codes sources Doxygens 90](#_Toc104823343)

[3.1. Récupérations et envoient des données 90](#_Toc104823344)

[3.2. Réceptions et enregistrements des données (LORIS BENAITIER) 90](#_Toc104823345)

# INTRODUCTION

Dans le cadre de notre projet informatique de BTS SNIR, on nous a confié la réalisation du projet nommé « embarquai2 ».

Le but du projet consiste à développer un système de suivi de courses nautiques. Le public pourrait visualiser en temps réel l’état de la course, des bateaux sur le principe de l’interface graphique “Virtual Regatta”.

La création et la suppression des participants et des courses seront accessible par un administrateur via l’IHM. Toutes les courses ainsi que les participants seront enregistrées dans la base de données et rejouable par l’administrateur et visible par tout le monde.

Ce système ne concernerait que les courses proches des côtes de manière à pouvoir utiliser le système LoRaWan comme support de transmission des données.

Le système “mobile” embarqué sur le bateau doit récolter les informations issues des différents capteurs (vitesse, position, vent, direction du vent) et les envoyer à intervalles réguliers au système “fixe”. Ce dernier stocke dans une base de données toutes les informations reçues et les affiche sur l’écran de télévision afin que le public puisse suivre la course en temps réel.

Afin de mener à bien ce projet, nous serons 3 étudiants à travailler dessus avec des tâches spécifiques pour chacun. Pour le versionning du projet, nous avons utilisé Github qui rassemble le travail de chaque étudiant. Le projet Github est accessible à l’aide du lien suivant : <https://github.com/Sorann753/embarquai>

Le but, à terme, est de rassembler le travail de toute l'équipe pour avoir un projet fonctionnel qui répondent aux attentes et aux exigences du cahier des charges. Les moyens qui seront alloués pour la mise en place de ce projet seront de 100€.

# REPARTITION DES TACHES

## Etudiant 1 DORIATH Arthus

* Installer et câbler la carte M5Stack ou RPI, ainsi que les modules SIMRAD
* Configurer la BusCAN et les modules SIMRAD
* Installer l’environnement de développement
* Développer le module CAN de récupération des données issues du bus CAN (NMEA 2000)
* Tester le module CAN de récupération des données issues du bus CAN
* Intégrer le module CAN au module LoRaWan de l’étudiant 2
* Documenter le module CAN

## Etudiant 2 BENAITIER Loris

* Installer et câbler la carte M5Stack ou RPI
* Configurer la carte additionnelle LoRaWan et le routeur Multitech
* Installer l’environnement de développement
* Développer le module LoRaWan d’envoi/réception/enregistrement des données
* Tester le module LoRaWan
* Intégrer le module LoRaWan avec le module CAN de l’étudiant 1 et la base de données de l’étudiant 3
* Documenter le module LoRaWan

## Etudiant 3 BOSSUET Baptiste

* Installer et câbler, le mini-ordinateur RPi et la télévision
* Configurer le mini-ordinateur RPi et la télévision
* Installer l’environnement de développement
* Développer le module WEB de configuration, le module IHM d’affichage et le module STAT de visualisation a posteriori
* Tester les modules WEB, IHM et STAT
* Intégrer les modules WEB, IHM, STAT entre eux ainsi qu’avec l’étudiant 2
* Documenter les modules

# PLANNING PREVISIONNEL

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | SPRINT 1 | SPRINT 2 | SPRINT 3 | SPRINT 4 | SPRINT 5 |
| Arthus DORIATH | ~ Installer l’environnement de développement « Visual studio code »  ~ Commander matérielles | ~ Faire schéma UML  ~ Câblé + configurer M5stack | Non défini | Non défini | Non défini |
| Loris BENAITIER | ~ Rechercher de librairie  ~ Installer l’environnement de développement « Visual studio code »  ~ Commander matérielle | Non défini | Non défini | Non défini | Non défini |
| Baptiste BOSSUET | ~ Installer le Raspberry PI  ~ Installer PhP  ~ Installer SQLite  ~ Installer Bootstrap  ~ Commander matérielle | Non défini | Non défini | Non défini | Non défini |

# SPRINT

|  |  |
| --- | --- |
|  | Sprint 1 |
| Arthus DORIATH | * Installer l’environnement de développement « Visual studio Community » * Etude du projet |
| Loris BENAITIER | * Recherche de librairie décodage Lorawan + M5Stack * Installer l’environnement de développement « Visual studio Community » * Etude du projet |
| Baptiste BOSSUET | * Etude du projet |

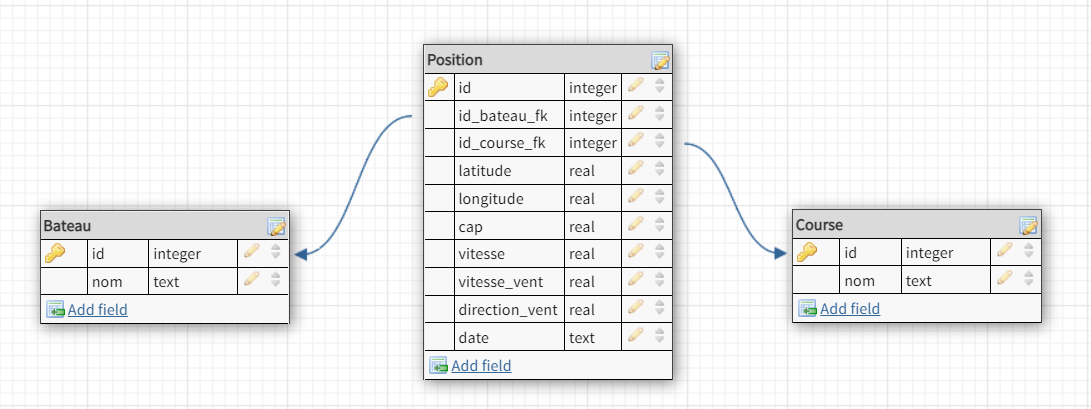
|  |  |
| --- | --- |
|  | Sprint 2 |
| Arthus DORIATH | * Tester matériel M5stack * Câblé Bus NMEA2000 et Capteur |
| Loris BENAITIER | * Programme de test / simuler fausses valeurs des capteurs * Faire programme génération fausse valeur |
| Baptiste BOSSUET | * Installer PHP (v7) et Installer SQLite * Installer Bootstrap et Installer Raspberry PI |

|  |  |
| --- | --- |
|  | Sprint 3 |
| Arthus DORIATH | * Câblé + configure M5Stack avec le NMEA2000 (CAN) * Recherche et test de librairie décodage CAN + M5Stack * Faire module NAVI * Tester module NAVI * Faire programme Principal MStack * Décodage CAN NMEA |
| Loris BENAITIER | * Faire programme pour prouver l’envoie * Faire un client de test pour le serveur * Programme de test / fausse base de données * Faire module Lorawan (envoie) * Câblé + configurer MStack & Lorawan * Faire module Lorawan Réception/ Enregistrement * Configurer le routeur Lorawan |
| Baptiste  BOSSUET | * Faire IHM configuration * Faire IHM viewer * Tester IHM |

|  |  |
| --- | --- |
|  | Sprint 4 |
| Arthus DORIATH | * Faire IHM M5Stack * Faire le boitier + carte électronique pour l’intégration * Intégrer le module NAVI au Lorawan (software) |
| Loris BENAITIER | * Intégrer le Lorawan Réception/Enregistrement avec BDD * Intégrer le module Lorawan envoi avec le module NAVI * Faire script Bash pour l’installations des modules python nécessaire * Faire script Bash pour l’installation et l’utilisation de mosquitto * Test pour savoir si mes productions répondent au cahier des charges |
| Baptiste BOSSUET | * Faire IHM STAT (à posteriori) * Intégré IHM avec BDD |

|  |  |
| --- | --- |
|  | Sprint 5 |
| Arthus DORIATH | * Documentation(s) technique(s) |
| Loris BENAITIER | * Documentation(s) technique(s) |
| Baptiste BOSSUET | * Documentation(s) technique(s) |

# BASE DE DONNÉES



La base de données nous permet de stocker toutes les informations nécessaires à la réussite du projet. Nous avons par exemple l'identifiant du bateau ou encore l’identifiant de la course qui permettent de savoir à qui appartiennent les données que nous recevons et à quelle course elles correspondent. SQLite nous a été imposé comme outil pour créer notre base de données.

La base de données comporte 3 tables :

* Bateau : La liste des noms de chaque bateau.
* Course : La liste des noms de chaque course
* Position : Stocke l'ensemble des informations récoltées par les capteurs et les paramètres entrées par l'utilisateur.

# ANALYSE UML

## Diagramme des exigences

Le diagramme d'exigence est un diagramme spécialement utilisé en SYSML. Il précise les fonctions à réaliser ainsi que le niveau de performance à atteindre. Une exigence comporte un intituler, un identifiant et une description.

Le diagramme permet de mettre en évidence toutes les exigences attendues sur le projet. De façon générale, notre système doit pouvoir récolter les données des capteurs en utilisant la technologie Lorawan. Et enfin, toutes ces données doivent être exploitées afin de créer une IHM.

* IHM : IHM signifie interface homme machine et fait référence à un tableau de bord qui permet à un utilisateur de communiquer avec une machine, un programme informatique ou un système. En l'occurrence, notre IHM affichera les différentes données récupérer depuis une base de données afin de répondre aux exigences du client.

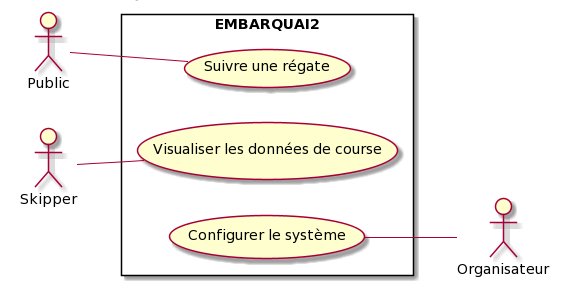
## Diagramme de déploiement

SCREEN DIAGRAMME DEPLOIMENT

Dans le contexte du langage de modélisation unifié (UML), un diagramme de déploiement fait partie de la catégorie des diagrammes structurels, car il décrit un aspect du système même.

Le but de notre diagramme de déploiement est d'avoir une vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leur relation entre eux.

## Diagramme de cas d’utilisation



Le diagramme de cas d'utilisation est un diagramme UML utilisé pour une représentation du comportement fonctionnelle d'un système logiciel. On va pouvoir y modéliser toutes les interactions entre le système et ses utilisateurs.

Sur ce diagramme, nous pouvons observer que le public doit pouvoir suivre le parcours d’une régate (bateau) en temps réel. Le skipper pourra visualiser les données de la course en direct depuis son bateau. Et enfin, l'organisateur pourra effectuer des réglages afin de configurer la course.

## Diagramme d’activité

SCREEN DIAGRAMME D’ACTIVITE

Le diagramme d'activité est un diagramme comportemental d'UML, permettant de représenter le déclenchement d'événements en fonction des états du système.

Ici, le diagramme d'activité représente les différents états du système embarquai2 dans son ensemble.

## Diagramme de classe

SCREEN DIAGRAMME CLASSE

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que leurs relations.

Le diagramme de classe ci-dessus permet de modéliser et de décrire la classe NAVI qui est utilisée pour récupérer les données des capteurs depuis le M5stack.

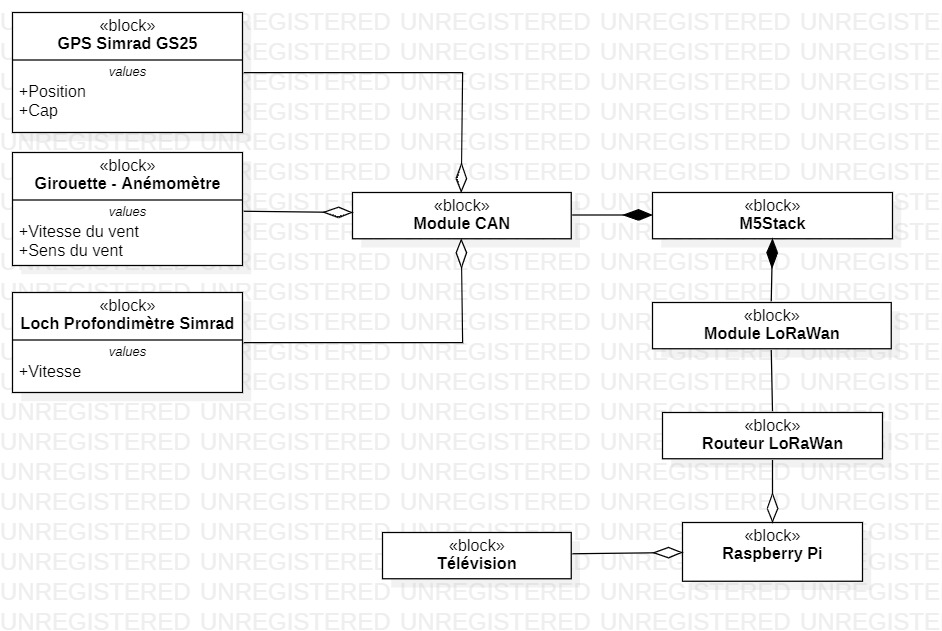
## Diagramme de séquence

SCREEN DIAGRAMME SEQUENCE

Le diagramme de séquence est la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique.

Ici, on peut observer toutes les interactions entre le système embarquai2 et les acteurs comme le skipper, le public ou encore l’administrateur dans l’ordre chronologique.

## Diagramme de définition de bloc



Le diagramme de bloc permet d’exprimer la structure d'un système, d'un sous-système ou d'un composant. Les blocs peuvent représenter des entités physiques où logiques. Ils sont décomposables, ils peuvent posséder des propriétés et un comportement.

Notre diagramme de bloc permet de représenter la structure des composants de notre système embarquai2. Il modélise les composants électroniques présent pour récupérer les données et les envoyer jusqu'au M5stack. Il y a également les composants nécessaires au transfert des données en MQTT du M5stack jusqu’au Raspberry Pi pour être affiché sur une télévision via une IHM.

# MATERIELS UTLISES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Disponibilité | Prix en euro (s) | Description | Quantité | Respect du cahier des charges |
| M5Stack Core2 :  Réf : ESP32 | Acheté | 40 € | M5Stack est un système de modules composé d'ordinateurs à cartes uniques à destination de l'Internet des objets | 2 | Oui |
| Module Lorawan :  Réf : UNIT LoRaWan868, ASR6501 868MHz, UART interface pour M5Stack | Acheté | 8,50 € | Module permettant de communiquer et de transférers des données avec la norme Lorawan | 1 | Oui |
| Module CAN :  Ref : ….. | Acheté | 9.50 € | Le module Grove CANbus est un transceiver isolé pour bus CAN (Controller Area Network) pouvant être utilisé pour réaliser de complexe réseaux de communication tel qu'on les trouve dans les voitures. | 1 | Oui |
| Routeur Multitech Lorawan :  Réf : MTCAP 868 | Existant | 0 € | Routeur capable de récupérer les données émises par les appareils équipés de modules lorawan | 1 | Oui |
| Raspberry Pi | Existant | 0 € | Le Raspberry Pi est un nano-ordinateur mono-carte à processeur ARM | 1 | Oui |
| Ecran compatible HDMI :  Réf : Acer KA2 ecran | KA222QA | Noir | Existant | 0 € | Dalle pour afficher du contenu... | 1 | Oui |
| Simrad (Modules et cables NMEA2000) :  Ref :….. | Existant | 0 € | Matériels nautique utilisé sur les bateaux pour la récupération des données environnante. | 1 | Oui |
| Module Lorawan :  Réf : Xbee Waveshare + Carte ATIM LoRaWan | Existant | 0 € | Module permettant de communiquer et de transférer des données avec la norme Lorawan | 1 | Oui |

Le cahier des charges et ses contraintes ont été respectés au niveau du matériel acheté et utilisé. Le budget final est inférieur à 100€ donc le cahier des charges a été respecté concernant le budget maximal alloué pour notre projet.

* Budget final -> 98 euros
* Budget maximal -> 100 euros
* 98 < 100 -> Cahier des charges respecté

## Présentation du M5Stack Core2



Une image contenant texte, équipement électronique

Description générée automatiquement



M5Stack Core2, un module de développement basé sur ESP32 riche en fonctionnalités qui permet de prototyper des idées IoT dès la sortie de la boîte. Ce kit de développement comprend une batterie intégrée et de nombreux autres capteurs et modules intéressants. Et en plus de tout cela, il peut être facilement programmé avec Arduino IDE, Platformio, ou micro-python.

L'unité principale est équipée d'un écran tactile capacitif de 2,0 pouces qui fournit une interface homme-machine fluide et réactive. Le moteur de vibration intégré peut être utilisé pour fournir un retour haptique ou des alertes. Le module RTC intégré fournit une heure précise de la journée. L'alimentation est fournie via une puce de gestion de l'alimentation AXP192, pour surveiller et contrôler les attributs d'alimentation de l'appareil. L'emplacement pour carte TF inclus prend en charge les cartes microSD jusqu'à 16 Go. Le haut-parleur intégré est associé à une puce d'amplificateur de puissance d'interface audio numérique I2S pour réduire la distorsion du signal et fournir une sortie audio plus claire. Il y a des boutons physiques indépendants d'alimentation et de réinitialisation (RST) sur les côtés de Core2 pour AWS, avec 3 boutons tactiles programmables à l'avant de l'écran.

Une image contenant table

Description générée automatiquement

# LOGICIELS ET LIBRAIRIES UTILISES

## Logiciels

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Disponibilité | Prix en euro (s) | Description | Respect du cahier des charges |
| Visual Studio Code | Existant | 0 € | Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégré. | Oui |
| Plateformio | Existant | 0 € | PlatformIO est un écosystème open source dédié au développement IoT. PlatformIO IDE est l'environnement de développement C/C++ pour les systèmes embarqués supportés. Il est multi-plateformes (Windows, Mac et GNU/Linux) et il fournit une extension à un éditeur de texte existant : soit Atom soit Visual Studio Code. | Oui |
| Pycharm | Existant | 0 € | PyCharm est un environnement de développement intégré utilisé pour programmer en Python | Oui |

## Librairies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Disponibilité | Prix en euro (s) | Description | Respect du cahier des charges |
| Librairie M5Stack | Existant | 0 € | Librairie réutilisant les librairies déjà existantes du c/c+ adapté pour l’embarquée | Oui |
| Librairies Python :  from socket import timeout, gaierror | Existant | 0 € | Cette librairie est utilisée pour détecter et gérer les timeouts et les problèmes de chaine de caractère incorrecte sur un fichier | Oui |
| Librairie Python :  from threading import Thread | Existant | 0 € | Cette librairie est utilisée pour gérer les Threads et notamment le multithreading | Oui |
| Librairie Python :  import paho.mqtt.client as mqtt | Existant | 0 € | Cette librairie permet aux applications de se connecter à un courtier MQTT pour publier des messages, s’abonner à des rubriques et recevoir des messages publiés. | Oui |
| Librairie Python :  import re | Existant | 0 € | Cette librairie a été utilisé pour récupérer des informations précises dans des chaînes de caractères. | Oui |
| Librairie Python :  import base64 | Existant | 0 € | Ce module fournit des fonctions pour encoder des données binaires en caractères ASCII imprimables et décoder ces encodages en données binaires. | Oui |
| Librairie Python :  import sqlite3 | Existant | 0 € | Cette librairie a été utilisé pour manipuler les bases de données SQLite | Oui |
| Librairie Python :  import configparser | Existant | 0 € | Cette librairie a été utilisée pour la création de fichiers | Oui |
| Librairie Python :  import sys | Existant | 0 € | Ce module fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier. | Oui |
| Librairie Python :  import logging | Existant | 0 € | Cette librairie a été utilisée pour écrire des messages d'erreur dans un fichier | Oui |
| Librairie Python :  import datetime | Existant | 0 € | Cette librairie a été utilisée pour récupérer l'heure et la date à l’instant t. | Oui |
| Librairie Python :  Import time | Existant | 0 € | Ce module fournit différentes fonctions liées au temps. | Oui |

# SYSTEME D’EXPLOITATION UTILISE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Disponibilité | Prix en euro (s) | Description | Respect du cahier des charges |
| Raspbian | Existant | 0 € | Système d’exploitation pour Raspberry PI | Oui |

# LANGAGES DE PROGRAMMATIONS UTILISES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Disponibilité | Prix en euro (s) | Description | Respect du cahier des charges |
| C++ | Existant | 0 € | C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes, dont la programmation générique et d’objet. | Oui |
| Python | Existant | 0 € | Python est un langage de programmation interprété, multi-paradigme et multiplateformes. | Oui |
| HTML | Existant | 0 € | Le HyperText Markup Language, généralement abrégé HTML est le langage de balisage conçu pour représenter les pages web. | Oui |
| Bash | Existant | 0 € | Langage de programmation utilisé sur Linux qui est beaucoup utilisé pour faire des scripts. | Oui |
| Shell | Existant | 0 € | Langage de programmation utilisé notamment pour faire des scripts sur des systèmes UNIX | Oui |
| SQL | Existant | 0 € | Langage de programmation utilisé pour gérer les bases de données en ligne de commande. | Oui |

# RECUPERATIONS DONNEES CAPTEURS VIA LE M5STACK CORE2

# ENVOIENT DES DONNEES DEPUIS LE M5STACK CORE2

## Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

## Objectif

L'objectif est de faire transiter les données des capteurs entre le M5Stack (Partie mobile) jusqu'au PC administrateur par le biais du réseau Lorawan.

## Cahier des charges

* La communication des données doit utiliser le réseau LoRaWan
* L'envoi des données doit se faire soit par le biais d'un M5Stack ou d'un Raspberry Pi
* Envoyer les informations récoltées au poste administrateur pour exploitation à intervalle fixe et configurable
* Permettre une installation/configuration aisée pour les enseignants ainsi qu'une utilisation facile et documentée pour les élèves.

## Matériels nécessaires

Pour réaliser cette partie, nous avons choisi d'utiliser le M5Stack Core 2 avec son module Lorawan (UNIT LoRaWan868).

Pour des raisons pratiques et financières, nous avons acheté 2 M5Stack Core 2 afin que l’étudiant 1 (Arthus Doriath) et moi puissions travailler chacun de notre côté en effectuant nos tests.

### Module UNIT LoRaWan868

Une image contenant texte

Description générée automatiquement



Une image contenant table

Description générée automatiquement

L'unité LoRaWAN868 est un module de communication LoRaWAN adapté à la fréquence 868 MHz lancé par M5Stack. Le module adopte le schéma ASR6501, qui prend en charge la communication longue distance et a à la fois une consommation d'énergie ultra-faible et une sensibilité élevée.

 Le module intègre la pile de protocoles LoRaWAN et adopte une interface de communication série (utilisant le jeu de commandes AT pour le contrôle). Lorsqu'il est utilisé, il peut être utilisé comme nœud de collecte pour accéder à un grand nombre de passerelles pour la collecte et la gestion des données.

Ce module convient aux applications de communication IoT longue distance à faible consommation, telles que le déploiement de nœuds de surveillance environnementale.

### Module Lorawan : Xbee Waveshare + Carte ATIM LoRaWan

Une image contenant texte, équipement électronique, circuit

Description générée automatiquement

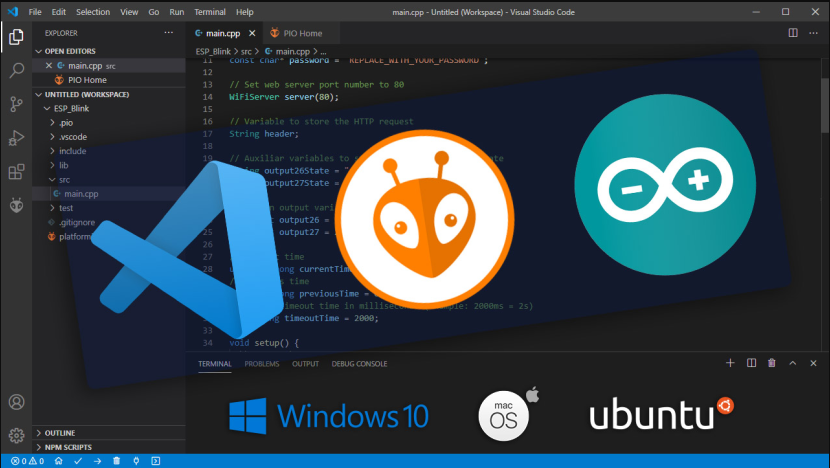
La carte ATIM Lorawan (module Lorawan) est un module permettant de communiquer en Lorawan. La carte XBE Waveshare est un support utilisé pour transférer les données à envoyer sur le module Lorawan (Carte ATIM LoRaWan).

## Problème rencontré

Après beaucoup de tests, je n'ai pas réussi à faire fonctionner le module Lorawan (UNIT LoRaWan868). Il faut savoir que ce module est extrêmement peu documenté, même après avoir suivi le peu d'instructions que j'avais sur la documentation technique, le module ne fonctionnait toujours pas. J'ai donc envoyé un message au support technique des créateurs de ce module (M5Stack), mais sans succès puisqu’eux-mêmes mon certifié qu'il ne l'avait jamais testé.

Par conséquent, pour ne pas prendre de retard sur le projet, j'ai utilisé la Carte ATIM LoRaWan avec son support (Xbee Waveshare) qui était disponible dans mon établissement sans frais supplémentaires.

## Environnement de développement



Pour pouvoir envoyer un programme sur le M5Sstack Core2, il fallait un IDE capable d’interagir avec lui. Par conséquent, j’ai choisi Plateformio qui est un IDE qui peut s’installer sur Visual studio code en tapent son nom dans la barre de recherche des extensions installables. Après l’avoir installé, un logo apparaitra sur le côté gauche de Visual studio code ou on pourra crée son propre projet selon notre matériel.

L’avantage est qu’une fois le projet crée, toutes les bibliothèques du matériel choisi seront directement importées et disponibles dans l’IDE.

Etant donné qu’on a utilisé un M5Stack Core2, toutes les bibliothèques liées à cette carte ont été importé. Les librairies de cette dernière étant codé en c++, tout mon code sur le M5Stack Core2 sera en c++.

## Présentation de Lora et Lorawan

### Introduction

Développés par la start-up grenobloise Cycleo, LoRa et LoRaWAN sont des protocoles majeurs de l’internet des objets. Véritable révolution dans le domaine, cette technologie facilite la communication radio entre les appareils connectés.

### Qu’est-ce que LoRa ?

LoRa (*Long Range*) est une technologie de communication radio bas débit et très longue portée. C’est la couche physique d’un réseau étendu à faible puissance et longue distance (LPWAN). Améliorant le système de [modulation par déplacement de fréquence](https://fr.wikipedia.org/wiki/Modulation_par_d%C3%A9placement_de_fr%C3%A9quence) (FSK), LoRa se base sur la modulation à spectre étalé (*chirp*).

Ce protocole propriétaire offre ainsi une meilleure portée de la communication, avec une puissance tout aussi faible que les autres systèmes et une bonne résistance aux interférences.

### Qu’est-ce que LoRaWAN ?

LoRaWAN (Long Range Wide-area network) est l'architecture du système, le réseau étendu à faible puissance et longue distance. Ce protocole permet la communication à longue portée et faible débit entre des objets connectés ayant une faible consommation électrique. Plus précisément, LoRaWAN aide ces appareils à communiquer avec les serveurs d’applications.

En amont, tous les appareils connectés au réseau possèdent une puce LoRa qui établit la communication avec des passerelles (gateways) en utilisant la modulation par étalement de spectre. Ces passerelles sont des stations de base pourvues d’antennes et connectées à internet. Elles servent alors d’intermédiaires pour contacter le serveur.

Tandis que la couche physique de modulation LoRa établit la communication à longue portée avec les passerelles, le protocole IP prend le relai entre les gateways et le serveur applicatif.

Ainsi, LoRaWAN ne crée pas directement la communication entre des objets intelligents. Il fournit une architecture leur permettant de dialoguer avec le serveur applicatif – via les passerelles et la technologie LoRa.

### Quels sont les avantages des protocoles LoRa et LoRaWAN ?

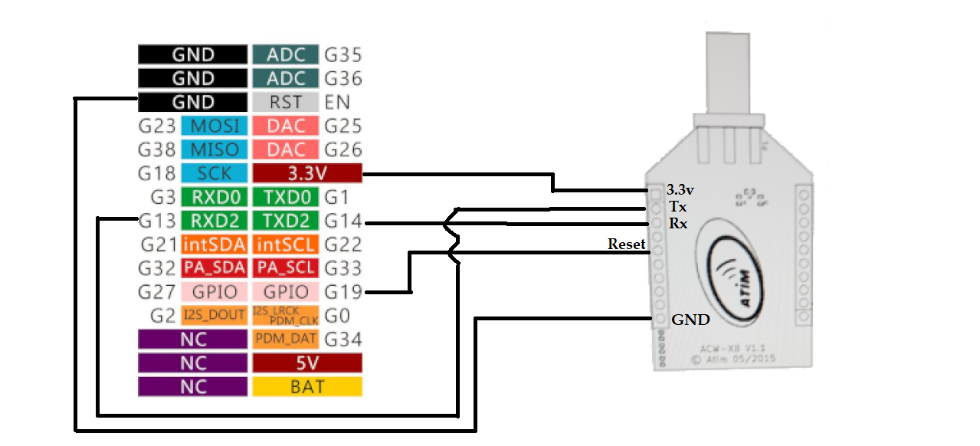
L’avantage principale de LoRa et LoRaWAN est leur grande portée, associée à une puissance aussi faible que les protocoles concurrents. De ce fait, une seule passerelle ou station de base peut couvrir des villes entières ou des centaines de kilomètres carrés. Avec un minimum d'infrastructure, il est alors possible de couvrir facilement des pays entiers.

## Liaison physique : module Lorawan/M5Stack Core2

Afin de relier le module Lorawan avec le M5Stack, il faut brancher la carte ATIM LoRaWan sur les pins de la partie supérieure de la Xbee Waveshare (Du coté où se trouve la prise pour une liaison sur un ordinateur). Puis, relier les pins qui se trouve en dessous de la carte Xbee Waveshare sur les pins disponibles dans le M5Stack Core2 comme sur le schéma :

Montage provisoire pour les tests :

PINS M5Stack Core 2 : Carte ATIM LoRaWan + Xbee Waveshare :



Les pins dans le M5Stack sont accessibles en séparent les deux parties qui le compose qui sont maintenu par des vis.

Une image contenant texte, orange, équipement électronique

Description générée automatiquement

Montage Finale :

Une image contenant texte

Description générée automatiquement

## Choix du format de la trame contenant les données

Le format de la trame de données est extrêmement important puisque ça va déterminer la taille finale du message et la façon dont on pourra récupérer les données par la suite. On verra également que tout sa aura une incidence sur le taux d’erreur des messages récupéré par le serveur python.

Format trame : b'@Flag;id\_course;id\_bateau;Donnee\_1;Donnee\_2;!CRC~'

### Caractères au début et à la fin de la trame de données

Afin que le message soit envoyé, il est impératif de mettre au début de la trame le caractère b et le caractère ' à la suite. Puis à la fin de la trame, il faut rajouter le caractère '. Ces caractères sont obligatoires car cela va avoir pour effet d’encoder la trame en UTF8 qui est le format d’envoi standard des messages utilisant le protocole Lorawan. Si on n’utilise pas ce format d’envoi, les messages envoyés seront complètement corrompus et inexploitables.

### Caractères de séparations entre chaque champ de la trame

Le choix des caractères de séparations entre les différents champs ne sont pas obligatoires pour l’envoi d’un message. Par conséquent, nous avons choisi les caractères les plus propices afin d’avoir un message clair et facile à décoder pour mon serveur python.

* @ et ! sont les deux caractères qui séparent le flag et les données avec le CRC
* Le caractère ; sera ajouté juste après le champ flag et chaque champ de donnée
* Le caractère ~ sera ajouté juste après le champ CRC

### FLAG

Le flag a été mis en place car étant donné que les capteurs ne donnent pas toujours de valeurs, on peut avoir seulement deux valeurs sur les 6 qu’on souhaite envoyer. La deuxième raison est que si jamais par exemple, la donnée\_1 est la même que l’ancienne qui a été envoyé, alors ça ne sert à rien de renvoyé cette valeur. Et enfin, le fait de n’envoyer que les donnée disponible et utile réduit considérablement la taille de la trame de données. Par conséquent, le taux d’erreur et de corruption diminue lui aussi drastiquement.

Flag : 0b00000000;

* Le Flag est composé de bit à 0 ou à 1 avec 8 champs donc 2 champs inutile à la fin. Si le champ est à 1, alors la donnée à été mise à jour, sinon, la donnée n’est pas disponible donc pas la peine d’envoyé cette donnée.
* Le ‘0b’ dans le flag veut dire que le flag est en binaire (0x-> en hexadécimale…)
* Le flag est contenu dans un byte qu’on va récupérer dans la structure data\_navi du namespace navi qui est mis à jour par mon coéquipier Arthus Doriath en fonction de ce qu’il reçoit.
* L’ordre de lecture se lit de droite à gauche et pour savoir quelle donnée correspond à quel bit dans le flag, on se réfère aux champs de l’énumération data\_navi\_content qui est dans le namespace navi.
* Les champs donnés sont récupérés dans la structure data\_navi qui est dans le namespace navi. Les champs donnés sont mis dans la trame dans l’ordre de lecture du flag, comme ça, on peut, en réutilisant la méthode inverse, retrouver à quoi correspondent chaque valeur sur le serveur python qui traitera la trame de donnée.
* Afin que le serveur python puisse savoir à son tour quel donnée correspond à quel champs (altitude, vitesse…), on doit lui passer le flag dans la trame message. Par conséquent, pour réduire la taille de la trame, on va passer la valeur décimale du flag au lieu de passer le flag brut.
* Pour la vérification des bits dans le flag, on effectue un traitement bit à bit.

Extrait code qui gère le flag « fonction\_send\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

Extrait du code qui gère la conversion base 2 à Base 10 « fonction\_send\_lora.cpp » dans la fonction « Traitement\_flag\_data » :



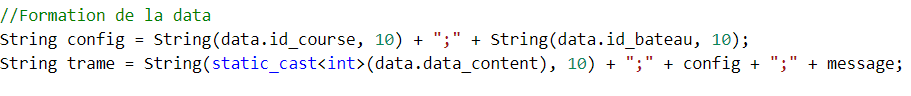
### Données

La trame peut contenir au maximum 6 champs de données récupérées grâce aux capteurs. Il y a id\_course et id\_bateau qui seront configurables et placés juste avant les données capteurs pour savoir à qui appartiennent ces données.

Les données capteurs sont : Heading, Speed, Altitude, Longitude, WindSpeed, WindAngle.

Toutes ces données seront récupérées depuis la structure data\_navi du namespace navi en fonction du flag.

Extrait code qui gère la concaténation des données « fonction\_send\_lora.cpp » dans la fonction « Traitement\_flag\_data » :



* Message -> Données bateau récupérées en fonction du flag
* Data\_content -> Nom de la variable de type ‘byte’ qui contient le flag (Transformation base 2 -> base 10)

### CRC

Pour que le serveur python qui sera charger de traiter les données qui ont été envoyé puisse savoir si elles sont complètes ou non, j’ai mis en place un algorithme de CRC qui permet en autre de vérifier si le message reçu est conforme par rapport à l’originale (avant son envoi).

Le principe du CRC est très simple, on prend les données sur lesquelles on veut appliquer le CRC, on les convertit en binaire, on passe la valeur binaire + une clé binaire qu’on a défini au préalable dans la fonction generation\_crc qui va générer comme son nom l’indique un CRC en binaire. Et enfin, on met ce CRC dans la trame donnée. Une fois que le serveur python qui traite les données à reçu cette trame, il va convertir les données ou le CRC y est appliqué en binaire et les passés dans la même fonction ‘generation\_crc’ avec la même clé binaire que celle utilisée pour l’envoi. Pour finir, on va comparer le CRC trouver et le CRC contenu dans la trame et voir s’ils sont égaux. S’ils ne sont pas égaux, alors les données sont corrompues, sinon, elles sont conformes et peuvent être utilisé.

Extrait code qui génère un CRC « fonction\_send\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

* Donnée en binaire + clé binaire définis au préalable en rajoutant des bits à 0 selon la taille de la clé choisit à l’aide de la fonction add\_bit.
* On fait appel à la fonction mod2div qui va prendre en paramètre le résultat précédent et la clé en binaire pour générer le remainder (reste == CRC)
* Les données passées dans cette fonction seront les suivantes : flag;id\_course;id\_bateau;donnee\_1;donnee\_2…; le tout convertit en binaire.

Extrait code -> Fonction qui va diviser les données binaires par la clé et stocker le reste de la division « fonction\_send\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

Extrait code -> Fonction qui va faire un ou exclusif entre deux types string « fonction\_send\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

Extrait code -> Fonction qui va s’occuper de faire la conversion binaire d’une chaine de caractère « fonction\_send\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

Extrait code qui s’occupe d’appeler les fonctions nécessaires pour générer le CRC et le placer dans la trame « fonction\_send\_lora.cpp » dans la fonction « Send\_Message\_Lorawan » :

Une image contenant texte, intérieur, capture d’écran

Description générée automatiquement

* On place crc\_bin juste après les données (flag + id\_course + id\_bateau) suivis du caractère ~.

## Envoi de la trame contenant les données

On envoi la trame à l’aide de la commande Serial2.println qui correspond à l’envoi de donnée pour le port C (G13 = RXD2 et G14 = TXD2), le tout dans un type string.

Extrait code qui envoi message « fonction\_send\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

* Le delay choisi entre chaque envoi est de 2 seconde car en dessous de cette valeur, il y a beaucoup trop de messages qui sont corrompus et donc inexploitable.
* Après avoir rencontré le client, il m’a dit que ce n’était pas nécessaire que le temps entre l’envoient de chaque message soit configurable donc cette fonctionnalité n’a pas été implémenté.

## Test unitaire pour l’envoi de la trame contenant les données

Grace au dossier test qui est fourni avec Plateformio, en exécutant une commande spécifique dans le terminal, on peut lancer ces propres test sans utiliser la fonction Setup() et la fonction loop() principale.

Dans ce cas précis, on va pouvoir tester l’envoie des données grâce à cet outil.

Commande de base : pio test -e m5stack-core2 -f nom\_dossier\_a\_tester --verbose

Commande : pio test -e m5stack-core2 -f test\_LoRaWan --verbose

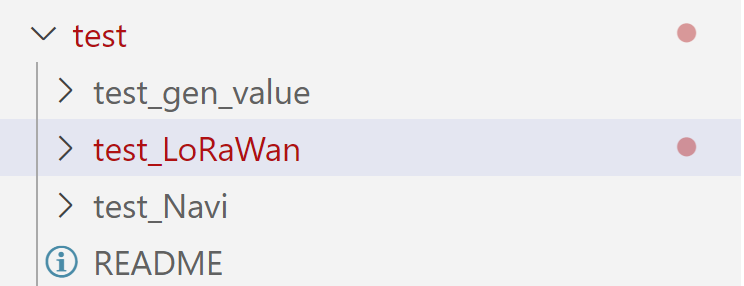
Extrait code source pour tester l’envoie des données « test\_lora.cpp » :

Une image contenant texte

Description générée automatiquement

* On appelle 1000 fois la fonction Send\_Message\_Lorawan

Extrait de l’arborescence du projet avec le dossier test\_LoRaWan de Plateformio :



Résultat :

Après le lancement du test, aucune erreur de compilation n’est apparue et les tests n’ont révélé aucune erreur. Le message ok1 envoyé sur le Serial pour s’assurer que la boucle fonctionnait était présent.

De plus, l’affichage du message complet dans la fonction Send\_Message\_Lora à l’aide de la commande Serial.println(born) s’affichait correctement et était conforme à la structure de la trame voulu.

* Test validé

## Test unitaire pour la génération de fausse valeur

J’ai mis en place ce test pour deux raisons :

* Si l’étudiant 1 (Arthus Doriath) ne parvenait pas à récupérer les valeurs des capteurs
* Faire des test d’envoient avec des valeurs qui varient pour être en condition réelle sans dépendre de l’étudiant 1 (Arthus Doraith) pour avancer.

Code pour la génération de fausse valeur « fonction\_gener\_value.h » :

Une image contenant texte

Description générée automatiquement

Code pour tester si ma fonction me donne bien les résultats attendus « test\_gen\_value.h » :

Une image contenant texte

Description générée automatiquement

Extrait code pour lancer le test de ma fonction de génération de nombre « main\_test\_genValu.cpp » dans la fonction loop :

Une image contenant texte

Description générée automatiquement

Extrait résultat compilation test fausse valeur :



* On obtient bien des valeurs aléatoires sans aucune erreur de compilation, tous les tests sont passés.
* test\_gen\_value = PASSED
* Si jamais il y avait un test qui ne passait comme celui du type par exemple, on obtiendrait une erreur « test\_gen\_value = Error »
* Commande utilisé : pio test -e m5stack-core2 -f test\_gen\_value --verbose

Grace à ce test, je peux maintenant utiliser ma fonction de génération de fausses valeurs en condition réelle pour simuler les valeurs des capteurs en étant sur quelle génèrent des résultats conformes.

* Test validé

## Intégration avec l’étudiant 1

Pour fusionner mon module « Envoi des données depuis le M5Stack Core 2 « et le module « récupération données capteurs depuis le M5Stack Core2 » de l’étudiant 1, il suffira que l’étudiant 1 appelle ma fonction Send\_Message\_Lorawan en lui passant la structure data\_navi qu’il aura mise à jour préalablement. Etant donné que le premier module Lorawan (UNIT LoRaWAN 868) ne fonctionnait pas, j’ai utilisé le module ATIM Carte Lorawan + Xbee Waveshare. Cependant, la liaison se fait seulement avec les pins qui sont situés à l’intérieure du M5Stack Core2. Par conséquent, l’étudiant 1 ne pouvait plus accéder à son port pour connecter son module Bus CAN. De ce faite, il a fallu concevoir une carte électronique pour que nos deux modules puissent s’intégrer ensemble au niveau hardware.

## Conclusion

Le cahier des charges sur cette partie est validé entièrement car le module « Envoient des données depuis le M5Stack Core2 » est totalement fonctionnelle et répond aux attentes du client et du cahier des charges.

# RECUPERATIONS DES DONNEES ENVOYEES PAR LE M5STACK CORE2

## Auteur

Cette partie a été rédiger et réalisée par l'étudiant Loris Benaitier.

## Objectif

L'objectif de cette partie est de réussir à récupérer les trames contenant les données envoyées depuis le M5Stack Core2 sur le pc administrateur (Poste fixe).

## Cahier des charges

* Envoyer les informations récoltées au poste “fixe” donc le pc administrateur

## Matériels et logiciels nécessaires

Afin de pouvoir récupérer les trames contenant les données, nous avons besoin d'un routeur qui est capable d’intercepter et de récupérer les trames Lorawan émise par des appareils qui sont équipé d’un module Lorawan. Nous avons également besoin d’un Raspberry pi pour respecter le cahier des charges avec l’OS Raspbian. Cependant, n’importe quel autre ordinateur avec un os diffèrent marcherait aussi puisque le système est multiplateforme.

* Raspberry pi avec OS Raspbian
* Routeur multitech Lorawan (MTCAP 868)
* Pour réaliser le code en python, j’ai utilisé l’IDE Pycharm qui est spécialisé pour le langage de programmation python.

### Routeur multitech MTCAP 868

Une image contenant table

Description générée automatiquement

Le routeur MTCAP 868 est un routeur qui a été conçu pour pouvoir réceptionner les données venant des différents modules lorawan dans sa zone (10 km de portée).

Il possède beaucoup de configurations différentes, on peut notamment comme n'importe quel routeur le mettre sur notre réseau, il possède un port Ethernet et des petites LED vertes qui indiquent si oui ou non le service Lorawan est actif.

### Raspberry pi



Raspberry Pi est fondamentalement un ordinateur. Malgré sa petite carte, son faible coût et son système d'exploitation open source Raspbian, il s'agit toujours d'un ordinateur adapté aux besoins de base en matière de programmation.

Carte MicroSD = 16 Go

OS = Raspbian (image iso à récupérer sur internet)

## Configuration routeur multitech Lorawan (MTCAP 868)

### Introduction

Pour que le routeur puisse intercepter et récupérer les trames de données déchiffrer (besoins des identifiants du module Lorawan) envoyer depuis le M5Stack Core 2, il faut le configurer pour faire en sorte qu’il reconnaisse notre module Lorawan. Puis par la suite, redirigé les trames contenant les données en MQTT sur l’ordinateur fixe (administrateur).

Le principe de fonctionnement est le suivant : le routeur une fois allumée va être en écoute constante sur un périmètre de 10 km. Il aura la capacité de capter les trames de données émises par n'importe quel appareil équipé d'un module Lorawan mais ne pourra pas les déchiffrer puisqu’il n’aura pas leurs identifiants.

### Configuration des paramètres réseau du routeur

On part du principe que le pc fixe (administrateur) sera sur un réseau local, par conséquent, pour que le routeur puisse transférer les paquets en MQTT, on doit le mettre sur le même réseau que le pc fixe (administrateur).

Cette démarche est nécessaire car pour le transfert en MQTT, on verra après qu’on doit rentrer une adresse IP qui correspond à l’appareil qui recevra les données du routeur. Par conséquent, afin que le routeur puisse rediriger les données sur cette adresse IP, il doit la connaitre et donc être sur le même réseau.

Configuration routeur :

1. SETUP-> DHCP Configuration
2. Cocher : enable
3. SETUP -> Network Interface
4. Cliquer sur la ligne de br0 avec le crayon
5. IPV4 Setting -> Mode = static && IP\_disponible sur le réseau local voulu
6. Le routeur sera accessible depuis le réseau local en question en tapant

son IP : IP\_disponible et pourra donc transférer les données en MQTT.

### Démarrage du NETWORK SERVEUR

Afin que le routeur puisse gérer la réception des paquets, il faut activer le service Lora en mode NETWORK SERVER.

Configuration routeur :

1. LoRaWAN -> Network interface
2. Dans le menu LoRa Mode, il y a une case qui nous permet de choisir NETWORK SERVER

Image de ma configuration :

Une image contenant texte

Description générée automatiquement

### Ajouts des identifiants d’un module Lorawan (END-DEVICE)

* Chemin paramètres routeur = LoRaWAN -> Key Management :
* Location = Local Keys

Afin qu'il récupère seulement les trames de notre appareil et puisse les déchiffrer, il suffit juste dans les paramètres du routeur qu’on appelle également filtre, de rentrer les identifiants de notre module Lorawan en allant dans : Local End-Device Credentials -> Add new et pareil pour la partie Local Network Settings. Ces identifiants sont donnés par le constructeur lors de l’achat de l’appareil (notice d’utilisation). Si jamais ces identifiants ne sont pas disponibles, on peut pour certains les générer ou les modifiés.

On doit également choisir le type de jonction (Mode de communication entre routeur et module Lorawan) qu’on souhaite utiliser (OTA ou ABP). La différence entre ces deux types de jonction est que OTA est une jonction sécurisé cryptée en AES 128 (cryptage sécurisé) car il fait ce qu’on appelle une jointure alors que ABP n’est pas crypté et est relativement plus lent. De plus, ce type de connexion n’est quasiment plus utiliser de nos jours.

* ABP : Activation By Personnalization
* OTAA : Over The Air

Il faut également choisir le type de connexion en fonction de notre localisation puisqu’en Europe, la fréquence est de 868 Mhz donc on utilise LW102-OTA-EU868

Pour le choix de la classe, il faut se référer aux spécifications de l’appareil en question mais la classe A est la classe par default de tous les modules Lorawan.

Signification des identifiants et paramètres pour OTA :

* Dev EUI : c’est un identifiant qui rend unique chaque objet, programmé en usine.

Ce paramètre n’est théoriquement pas modifiable.

* App EUI : C’est un identifiant unique d’application qui permet de regrouper les objets. Cette adresse, sur 64 bits, permet de classer les périphériques par application. Ce paramètre est modifiable.
* App Key : Il s’agit d’un secret partagé entre le périphérique et le réseau, utilisé pour dériver les clefs de session. Ce paramètre peut être modifié.
* Device profile = Profil de l’appareil qu’on configure (LW102-OTA-EU868)
* Network Profile = Classe par default (Classe A)

Image de ma configuration :

Une image contenant texte

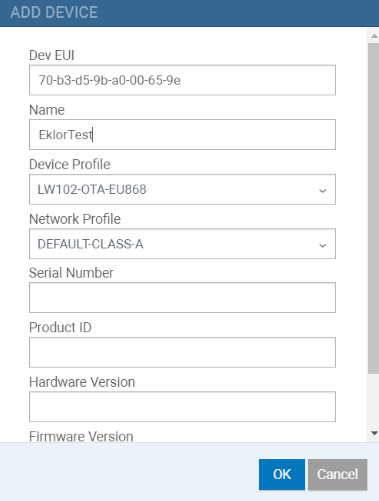
Description générée automatiquement

### Ajout d’un appareil pour le lié avec un module Lorawan

* Chemin paramètre routeur = LoRaWAN -> DEVICES

Pour finir, il est nécessaire d’ajouter un appareil dans les paramètres du routeur en allant dans End Devices -> Add New. Il faut rentrer le DEV EUI qu’on a rentré précédemment, le nom de notre appareil pour le reconnaitre, le Device Profil qui est identique à celui rentrer juste avant et enfin le Network Profile en Classe A. Cette configuration servira notamment à faire que si on reçoit des données de telle module Lorawan, on sait que ca correspond à un thermomètre par exemple.

Image de ma configuration :



### Transferts des données en MQTT

Etant donnée que le routeur à la possibilité de transférer les paquets qu’il a reçu en MQTT, je vais utiliser cette méthode afin de rediriger les trames de données sur l’ordinateur administrateur.

MQTT : MQTT est un protocole standardisé reposant sur TCP/IP. "Il est particulièrement utilisé pour transporter des données des objets connectés sur le cloud. Pour communiquer avec MQTT, les objets connectés utilisent un broker, c'est-à-dire un programme en charge de la réception des informations publiées afin de les transmettre aux clients abonnés. Le broker joue un rôle de relais. Il existe plusieurs types de brokers : ActiveMQ, JoramMQ, Mosquitto ou encore, RabbitMQ.

Paramètres routeur à modifier :

1. Aller dans les parèmtres du routeur
2. Aller dans LoRaWAN puis Network Setting
3. Aller dans Payload Broker
4. Modifier ces champs :
   * Username = ne rien mettre (sauf si on veut un identifiant)
   * Password = ne rien mettre (sauf si on veut un mot de passe)
   * Port = 1883 (Port par default)
   * Hostname = IP de l’appareil ou l’on veut rediriger les paquets du routeur

Image de ma configuration :

Une image contenant table

Description générée automatiquement

### Paramètre en fonction de notre localisation

Etant donné qu’on se trouve en Europe, il est nécessaire de modifier le paramètre suivant :

Paramètre routeur à modifier :

1. Aller dans les parèmtres du routeur
2. Aller dans LoRaWAN puis Network Setting
3. Aller voir dans LoRaWAN Network Serveur Configuration et Channel Plan
   * + - Channel Plan = EU868

Image de ma configuration :

Une image contenant texte

Description générée automatiquement

## Configuration du serveur Mosquitto sur le PC administrateur

Afin de pouvoir réceptionner les paquets que le routeur redirige sur le pc administrateur, il est nécessaire d’avoir un serveur sur le pc administrateur qui puisse les recevoir. Pour cela, j’ai choisi d’utiliser un serveur Mosquitto car c’est simple d’utilisation et multiplateforme.

Ouvrir terminale commande sur Raspbian (Raspberry pi) :

1. Installer mosquitto : sudo apt-get install mosquitto
2. Création fichier conf afin de pouvoir écouter tous les appareils (cond.d est le dossier ou l’on doit mettre les fichiers de configurations pour un mosquitto) : sudo nano emplacement\_mosquitto/conf.d/nom\_new\_fichier\_conf
3. Dans le fichier de conf on met : allow\_anonymous true (on accepte les appareils anonyme…)

Listener 1883 (Port par default)

1. Lancer serveur : sudo service mosquitto start
2. Arrêter le serveur : sudo service mosquitto stop

## Test du serveur Mosquitto

Grace à un client de test en python, j’ai pu tester et savoir si mon serveur mosquitto était fonctionnelle avant d’envoyer de vraies données avec le routeur et le module Lorawan.

Extrait code python du client de test pour mosquitto « Client\_Python\_test.py » :

Une image contenant texte

Description générée automatiquement

Après avoir testé, j’ai pu confirmer que le serveur mosquitto était fonctionnelle car j’ai bien reçu les paquets publiés visible sur le serveur mosquitto. De plus, mon client était sur un autre ordinateur donc cela veut dire que le serveur mosquitto accepte bien les connexions extérieures. Par conséquent, il ne devrait pas y avoir de problème en condition réelle.

Exemple paquet visible sur serveur mosquitto : Received ….’lora/…’ (44 bytes)

* Test validé

## Test serveur Mosquitto + routeur Lorawan + module Lorawan

Pour faire le test en condition réelle, j’ai créé un serveur qui va récupérer et afficher tous les paquets de type lora qui seront publié sur le serveur mosquitto.

Extrait code serveur python test lora « Serveur\_Python\_test.py » :

Une image contenant texte

Description générée automatiquement

Après avoir lancé le serveur mosquitto et le serveur python sur le pc administrateur + lancé le programme de test de fausses données avec le module Lorawan depuis le M5Stack Core2 et configurer le routeur, on obtient le résultat attendu qui est qu’on reçoit bien des paquets lora sur le serveur mosquitto.

Exemple : serveur mosquitto en activité :

Une image contenant texte

Description générée automatiquement

* On peut observer ici qu’un envoi est constitué de toutes ces étapes, la donnée se trouve dans le /up.
* Format paquet contenant la donnée : lora/DEV EUI/up
* Toutes ces étapes constituent la procédure nécessaire à l’envoi d’un message de type Lorawan.

Exemple : serveur mosquitto en activité mais sans paquet :



* Si aucune donnée n’est envoyée, alors le routeur envoi ce type de paquet.

Exemple : serveur python qui récupère les messages lora publié sur le serveur mosquitto :



* On voit bien ici que j’ai reçu un paquet de mon module Lorawan car il possède bien l’identifiant contenu entre les 2 « / » en tant que DEV EUI.
* Tests validés

## Conclusion

Pour conclure, toutes les fonctionnalités attendues sont présentes et répondent au cahier des charges car je reçois bien les paquets lora envoyés du M5Stack Core2 jusqu’au serveur mosquitto sur le pc administrateur.

# ENREGISTREMENT DES DONNEES DANS LA BDD

## Auteur

Cette partie a été rédiger et réalisée par l'étudiant Loris Benaitier.

## Objectif

L’objectif de cette partie est de réussir à enregistrer les données reçues sur le pc administrateur dans une base donnée envoyées depuis le M5Stack Core2.

## Cahier des charges

* Enregistrer les informations des bateaux dans une base de données
* Faire en sorte que le système soit simple d’utilisation et robuste

## Matériel et logiciels Nécessaires

* Aucun matériel n’est nécessaire pour cette partie.
* Pour réaliser le code en python, j’ai utilisé l’IDE Pycharm qui est spécialisé pour le langage de programmation python.
* DB Browser qui est un logiciel pour visualiser et construire ses bases de données.

## Décodage de la trame

### Récupérer le paquet qui contient la donnée

Pour récupérer le paquet qui contient nos données, il faut que le serveur récupère tous les messages avec ce format : lora/DEV EUI/up

Extrait code qui récupère ce type de paquet « Serveur\_Python.py » dans la fonction « on\_connect » :

Une image contenant texte

Description générée automatiquement

### Extraire la trame donnée du paquet Lorawan

Afin d’extraire la trame du paquet, on va récupérer en premier ce qui se trouve entre « "data":" » et « " » puis le décoder car il est en base64.

Extrait code qui décode et extrait la donnée du paquet « Serveur\_Python.py » dans la fonction « on\_message » :

Une image contenant texte

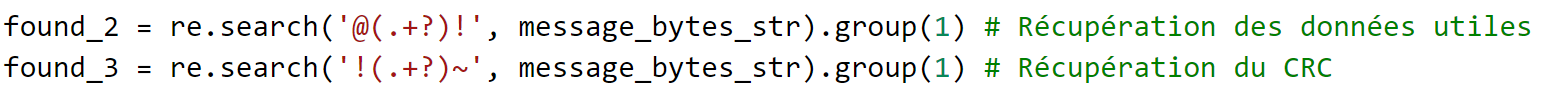
Description générée automatiquement

Une fois que la donnée est extraite on obtient notre trame : b'@9;1;0;9.0;48.25868;!01000101~'

### Extraire les différentes informations contenues dans la trame

Extraction des 2 id et des données capteurs, puis nous allons extraire le CRC et le flag.

Extrait code qui extrait les données et le CRC « Serveur\_Python.py » dans la fonction « on\_message » :



### Traitement du CRC

Afin de vérifier que le message n’est pas corrompu, on va comparer le CRC dans la trame et le nouveau CRC générer en passant les données suivante convertit en binaire : flag;id\_course;id\_bateau;donnee\_1;donnee\_2…;

Code qui génère le nouveau CRC « Serveur\_Python.py » :

Une image contenant texte

Description générée automatiquement

Code qui convertit la donnée en binaire « Serveur\_Python.py » :

Une image contenant texte

Description générée automatiquement

### Traitement du flag

Pour traiter le flag, même opération que le module « envoie données depuis le M5Stack Core 2 », il faut parcourir le flag envoyé et en fonction du modèle de correspondance des bits qui est connu du serveur python et du M5Stack Core2, on peut déterminer la correspondance des données reçu (exemple : si donnee\_1 == au cap ou bien à la vitesse…).

Extrait code qui gère le flag « Serveur\_Python.py » dans la fonction « traitement\_donnee » :

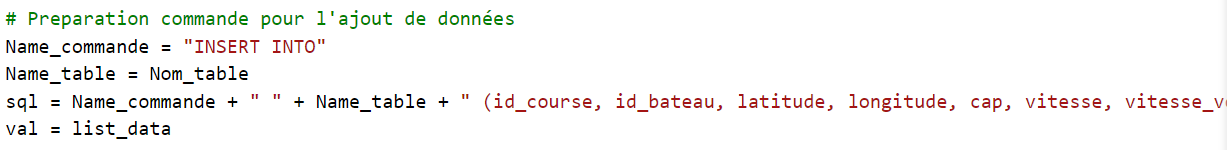
Une image contenant texte

Description générée automatiquement

## Enregistrement dans la base de données

Pour l’enregistrement des données dans une base de données, je vais faire appel à la bibliothèque sqlite3 de python. La fonction « Ajout\_Base\_Donnee » dans le fichier « Serveur\_Python.py » va enregistrer chaque donnée dans le champ correspondant de la table.

Extrait code enregistrement base de données « Serveur\_Python.py » dans la fonction « Ajout\_Base\_Donnee » :

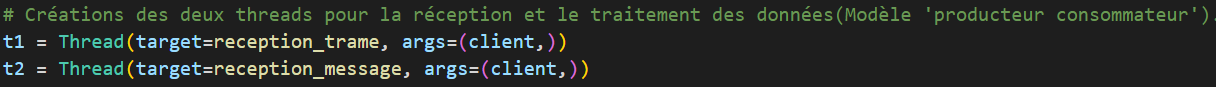


## Gestion performance du serveur

Pour faire en sorte que le serveur récupère et traite les données rapidement et efficacement, j’ai imaginé un système ‘Producteur consommateur’ à l’aide de la bibliothèque ‘thread’.

Le principe est le suivant : Un thread va gérer la récupération des données et l’autre thread va s’occuper de traiter la donnée (extraction, décodage, enregistrement BDD).

Extrait code qui gère les threads « Serveur\_Python.py » dans la fonction « main » :

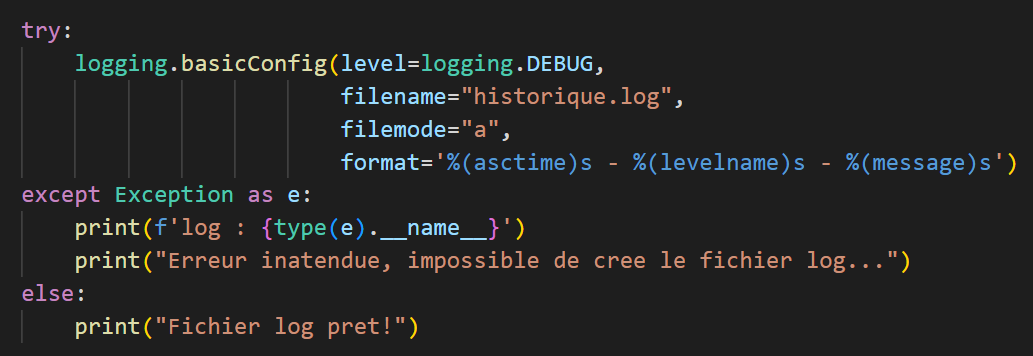


## Gestion erreurs du serveur

Pour faire en sorte que le serveur soit robuste, qu’en cas d’erreur ou de situation imprévu, le serveur continu de fonctionner ou affiche un message d’erreur, j’ai mis en place une gestion intelligente des erreurs en utilisant notamment le ‘try except’ pour toutes les situations à risques.

Je me suis également servi de la bibliothèque ‘logging’ pour la création d’un fichier log qui enregistrera en temps réelle tout ce qui se passe sur le serveur en précisant l’heure, la date, et la ligne exacte.

Extrait code pour la création et la gestion d’un fichier log « Serveur\_Python.py » dans la fonction « log\_init » :



Extrait code pour la gestion d’erreur du serveur « Serveur\_Python.py » dans la fonction « main » :

Une image contenant texte

Description générée automatiquement

## Test unitaire

Afin de tester le système d’enregistrement dans une BDD, j’ai envoyé de fausses données au niveau du M5 Stack Core2 puis j’ai lancé le serveur mosquitto et le serveur python sur le pc administrateur.

Protocole de mise en service du serveur python (Serveur\_Python.py) :

* Mettre le serveur python (Serveur\_Python.py), le créateur de fichier conf (creation\_fichier\_conf.py), et le script bash (Installation\_module\_mqtt.bash) dans le même dossier
* Le fichier .sqlite contenant la base de données peut être place n’importe où car il suffit de rentrer le chemin du fichier dans le fichier de configuration (param.ini crée grâce au programme « creation\_fichier\_conf.py). Si jamais le fichier de configuration est mauvais, le serveur python prendra les configurations par defaults : nom\_bdd = bdd\_embarquai.sqlite. Par conséquent, il faudra crée une base de données avec ce nommage située dans le même dossier que le serveur python avec la même structure que notre diagramme de base de données.

1/ Installer module paho-mqtt :

1. Lancer le script bash nommé : Installation\_module\_mqtt.bash
2. Exécution ligne commande : ./nom\_fichier (si cela ne marche pas à cause de droit d’exécution : chmod 755 nom\_fichier)

2/ Lancer le créateur de fichier conf (creation\_fichier\_conf.py) pour rentrer l’adresse ip, le port et le nom de la base de données que le serveur python doit exploiter afin qu’il puisse se connecter et enregistrer ses données :

1. Python3 nom\_fichier.py
2. Génération d’un fichier :param.ini
3. Rentrer les configurations voulu dans les champs disponibles et enregistrer le fichier
4. Si les champs sont invalides, alors le serveur utilisera ses propres configurations

3/ Lancer le serveur python :

1. Python3 nom\_fichier.py
2. Rentrer chemin du fichier de conf (param.ini)
3. Si les paramètre était valides dans le fichier conf et que ces paramètres ne permettaient pas au serveur python de se connecter, alors le serveur redemandera de rentrer une configuration valide

Extrait des messages en sortie du serveur python pour l’enregistrement des données dans une base de données « Serveur\_Python.py » :

Une image contenant texte

Description générée automatiquement

Extrait du fichier de conf (param.ini) « creation\_fichier\_conf.py » :

Une image contenant texte

Description générée automatiquement

Extrait du fichier log (historique.log) « Serveur\_Python.py » :

Une image contenant texte

Description générée automatiquement

Extrait des messages en sortie du serveur python lors de la connexion du serveur python « Serveur\_Python.py » :

Une image contenant texte

Description générée automatiquement

Extrait des données récupéré dans un fichier sqlite « DB Browser » :

Une image contenant table

Description générée automatiquement

Après avoir lancé tous les programmes nécessaires pour le test, toutes les fonctionnalités voulu et attendu sont fonctionnelles et répondent au cahier des charges car on peut voir qu’on enregistre bien des données dans une base de données.

* Test Validé

## Intégration avec l’étudiant 3

Afin d’assurer l’intégration avec l’étudiant 3, il suffira juste que je rentre le nom de son fichier ou le chemin qui contient la base de données dans le fichier conf (param.ini générer par le programme creation\_fichier\_conf.py) en ayant le même nom de table que lui dans mon serveur python (Serveur\_Python.py).

## Conclusion

Toutes les exigences et les attendus du client sont présentes et fonctionnelle. La partie enregistrement dans une base de données répond au cahier des charges.

# TRAITEMENT DES DONNEES VIA UNE IHM

## Auteur

Cette partie aura dû être rédigé par Baptiste Bossuet.

## Situation

Baptiste Bossuet ayant quitté le BTS pendant la période du projet, cette partie ne sera pas complétée.

## 

# IHM DU M5STACK CORE2

# DOSSIER MAINTENANCE ENVOI/RECEPTION/ENREGISTREMENT DONNEE

## Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

## Objectif

* Résolutions de potentiels problèmes sur le système d’envoi, réception et enregistrement

## Debug

Par default, les messages de debugs renvoyé par python sont suffisants pour comprendre d’où vient le problème. Si on souhaite revoir en détail ce qu’il s’est passé, il suffit d’aller voir dans le fichier de log (historique.log qui contient toutes les activités du serveur)

* Si le serveur python (Serveur\_Python.py) ne se connecte pas et redemande en permanence le chemin complet du fichier de conf, et que les sorties de debugs précédentes indiquent -> Récupération configuration personnalisé puis fichier de configuration dans les normes et connexion impossible :
  + - Vérifier les champs (ip et port) dans le fichier de configuration personnalisé
* Si le serveur python (Serveur\_Python.py) ne se connecte pas et redemande en permanence le chemin complet du fichier de conf, et que les sorties de debugs précédentes indiquent -> Récupération configuration personnalisé puis fichier de configuration personnalisé invalide et connexion impossible :
  + - Le port et l’IP choisi sont donc ceux par defaults donc IP = 127.0.0.1 port = 1883 donc cela veut dire que l’IP et le port par default sont mauvais
* S’il y a le moindre problème pour lire ou récupérer les données du fichier de conf (param.ini « Instalation\_fichier\_conf.py »), le serveur python ira chercher sa configuration par defaults : IP = 127.0.0.1 et port = 1883. Et nom\_bdd = bdd\_embarquai.sqlite
  + - Exemple : Le chemin du fichier de conf est mauvais
    - Exemple : Modification d’un champ qui ne doit pas être modifié dans le fichier conf (param.ini)
* Si le serveur python c’est connecté avec succès (connexion Broker réussi) et qu’il ne se passe rien, alors cela veut dire que soit le serveur mosquitto ne reçoit rien ou bien il s’est déconnecté entre temps.
* Si on réussit à se connecter au Serveur (Connexion Broker réussi) et qu’il n’y a pas de message de debugs qui dit : Opération effectué correctement sur la base de données après le message suivant (Donnée lisible en cours d’enregistrement crc valide), alors cela veut dire qu’il y a un problème au niveau de la base de données
  + - Vérifier nom\_bdd rentrer dans fichier configuration ou bien ça peut être la structure de la base de données qui ne correspond pas à la norme établie (voir diagramme base de données)

# INSTALLATIONS PARTIES ENVOI/RECEPTION/ENREGISTREMENT DONNEE

## Auteur

Cette partie a été rédiger et réalisée par l'étudiant Loris Benaitier.

## Objectif

# NOTICE UTILISATION ROUTEUR/SERVEUR MOSQUITTO/SERVEUR PYTHON

## Auteur

Cette partie a été rédiger et réalisée par l'étudiant Loris Benaitier.

## Objectif

# TEST FINAL DE L’ENSSEMBLE DU PROJET

## Test qui rassemble les parties de chaque étudiant

Nous avons installé tout le matériel nécessaire pour récupérer les données capteurs (Système NMEA 2000). Puis nous avons alimenter le système embarqué (Carte ATIM LoraWan + Xbee Waveshare + M5Stack Core2 le tous relié grâce à la carte électronique) via un câble USB depuis un PC.

Système finale pour le test :

IMAGE SYSTEME FINALE COMPLET

On peut constater que le système complet qui rassemble le travail de toute l’équipe interagie bien ensemble puisque le système final est totalement fonctionnel et répond au cahier des charges. On arrive à envoyer des données depuis le M5Stack Core2 qui viennent des capteurs du système NMEA 2000. Les données sont bien reçues par le PC Fixe (administrateur) et enregistrer dans la base de données.

# CONCLUSION

# ANNEXES

## Annexes techniques (documents constructeurs)

### Routeur Multitech Lorawan (LORIS BENAITIER)

### Module Lorawan : Carte ATIM LoRaWan (LORIS BENAITIER)

### M5Stack Core2

## Codes sources

### Envoient des données (LORIS BENAITIER)

### Réceptions et enregistrements des données (LORIS BENAITIER)

### Script Bash (LORIS BENAITIER)

## Codes sources Doxygens

### Récupérations et envoient des données

### Réceptions et enregistrements des données (LORIS BENAITIER)