Province de Liège
Enseignement

Haute Ecole de la Province de Liège

Projet

IOT
Pierre De Fooz

Projet IOT : Smart Bike

SPELGATTI Andrea
Alexandre Gallez
Master Ingénieur Industriel | M18
Année 2022-2023

30 août 2023

Table des matières

Introduction

Le Prototype Révolutionnaire de Vélo Intelligent

Dans le vaste panorama de l'innovation technologique, émerge une étoile qui brille pour tous. Notre projet, porteur de changement, aspire à réaliser un rêve partagé : rendre les avantages des vélos intelligents modernes accessibles à chacun, quel que soit le portefeuille. Le prototype que nous présentons aujourd'hui s'épanouit pour concrétiser cette vision inclusive.

À travers cette initiative, nous aspirons à donner à tous les passionnés de la mobilité à deux roues les opportunités et les prouesses de l'univers des vélos intelligents, sans égard aux contraintes financières. Ce projet englobe une gamme d'innovations qui repoussent les frontières de la sécurité, de la praticité et de la connectivité pour tous les cyclistes.

Examinons de plus près ce prototype qui brille par ses caractéristiques exceptionnelles :

- **Détection d'Accident avec Caméra de Bord :** Une caractéristique qui va au-delà de la simple sécurité en offrant une couche supplémentaire de protection. Non seulement ce vélo peut détecter les accidents potentiels, mais il peut également enregistrer ces moments cruciaux grâce à une caméra embarquée .
- **Déverrouillage du Vélo via RFID :** La commodité est à l'honneur avec cette fonctionnalité. Grâce à la technologie RFID, il vous suffit d'approcher votre vélo pour qu'il se déverrouille automatiquement, éliminant ainsi le besoin de clés physiques pour garentir une sécurité maximal.
- **Détection de Vol :** Pour dissuader les voleurs et protéger votre investissement, ce vélo intelligent est équipé d'un système de détection de vol. En cas d'une tentative d'ouverture forcée, le vélo saura réagir.
- **Traceur GPS :** La perte de votre vélo ne sera plus source d'inquiétude grâce au traceur GPS intégré. Même lorsque vous n'êtes pas à proximité, vous pourrez localiser votre vélo sur une carte en temps réel.
- **Éclairage Automatique :** Adaptabilité est le maître-mot avec cet éclairage automatique. Le capteur de lumière ambiante ajuste automatiquement l'éclairage du vélo en fonction des conditions extérieures, garantissant une visibilité optimale à tout moment.
- **Sauvegarde sur un Cloud Dédié :** Toutes les informations pertinentes liées à votre vélo seront sauvegardées de manière sécurisée sur un cloud spécialement dédié, garantissant que vous ne perdrez jamais de données cruciales.
- **Tableau de Bord Android :** Pour une expérience utilisateur enrichissante, ce vélo intelligent est livré avec un tableau de bord fournissant une mine d'informations utiles à portée de main.

En résumé, ce prototype de vélo intelligent incarne une démarche qui met la technologie au service de tous. Il ouvre la voie à une ère où les avantages des vélos intelligents ne sont plus réservés à quelques privilégiés, mais deviennent une réalité tangible pour tous les amateurs de deux-roues. Restez à l'écoute pour découvrir comment ce projet ambitieux pourrait redéfinir notre façon de

percevoir la mobilité urbaine.

0.0.1 l architecture dossier

Git s'est imposé comme l'outil clé favorisant le travail d'équipe, l'organisation du code et la traçabilité des modifications dans notre projet de vélo intelligent. Toutefois, pour des raisons structurelles du système, certains éléments nécessaires ne sont pas inclus, car ils doivent résider à des emplacements spécifiques. Cette approche responsable vise à garantir la sécurité des données tout en tirant parti des avantages de Git pour l'innovation collaborative.

en dossier principal, on retrouve :

le dossier esp32 qui regroupe tous les codes de test Arduino et un dossier final qui mélange le code créé dans les autres dossiers (finalesp32).

dans le dossier esp8266, on retrouve le code python de test et le code final en Arduino.

dans le dossier sensor exemple, on retrouve tous les codes de tests des composants qui ont été la base du développement des codes finaux.

le dossier rapport contient l'architecture Latex de ce rapport.

le dossier RaspberryPi contient le code de node red , le code de chaque sous script exécuté par node red.

dans le dossier rocky-linux on y retrouve tous les différents dockers et leur installation.

voici le résultat de la commande tree -d -a de notre archive :

```
        |-- .git
        |    |-- branches
        |    |-- hooks
        |    |-- info
        |    |-- logs
        |    |-- objects
        |    |    |-- info
        |    |    +-- pack
        |    +-- refs
        |         |-- heads
        |         |-- remotes
        |         |    +-- origin
        |         +-- tags
        |-- esp32
        |    |-- accelero
        |    |-- accelerometre
        |    |    +-- main
        |    |-- final
        |    |    |-- finalesp32
        |    |    +-- main
        |    |-- gps
```

```
|   |   +-- m
|   |-- lora
|   |   +-- main
|   +-- temp and humidity detector
|       +-- main
|           +-- main
|-- esp8266
|   +-- arduino
|       +-- esp8266
|-- Rapport
|   |-- bibliographie
|   |-- media
|   |   +-- images
|   |-- textes
|   +-- titres
|-- RaspberryPi
|   +-- usages
|       |-- bluetooth
|       |   +-- node
|       |-- nodered
|       +-- scripts
|           |-- 6axis
|           |   +-- __pycache__
|           |-- lora
|           +-- __pycache__
|-- rocky-linux
|   +-- docker
|       |-- eclipse-mosquitto
|       +-- node-red
+-- sensors-examples
|-- grove_6_axis
|   |-- LSM6DS3-for-Raspberry-Pi
|   +-- __pycache__
+-- __pycache__
```

Chapitre 1

Hardware

1.1 Raspberry Pi

1.2 ESP32

1.3 ESP8266

Chapitre 2

Interface Utilisateur

2.1 Gestion des données

2.1.1 Hyper-X

Nous avons choisi hyperX pour host la machine virtuelle de Rocky Linux, car disponible par défaut sur les machines windows et a de très bonne performance. Ceci est de plus que une architecture de test donc ce choix n'est pas réelement important.

2.1.2 Rocky Linux

Le choix de Rocky Linux Minimal en tant que base pour le déploiement de conteneurs Docker s'avère judicieux pour plusieurs raisons. En optant pour cette distribution Linux, qui est issue d'une fourche de CentOS et conçue dans un esprit de stabilité et de fiabilité, les utilisateurs peuvent bénéficier d'un environnement solide et sécurisé pour exécuter leurs applications conteneurisées. La version "Minimal" de Rocky Linux offre une empreinte réduite, ce qui permet d'allouer davantage de ressources aux conteneurs eux-mêmes plutôt qu'au système d'exploitation sous-jacent.

2.1.3 Docker

Dans la quête d'une maintenance et d'un déploiement plus simples et efficaces des serveurs,nous avons choisi Docker qui se distingue comme la solution idéale. Grâce à sa capacité à encapsuler des applications et leurs dépendances dans des conteneurs isolés, Docker nous garantit une gestion harmonieuse des versions et des configurations.

2.1.4 Mosquitto

C'est là que Mosquitto entre en jeu en tant que courtier MQTT (Message Queuing Telemetry Transport) open source, offrant une infrastructure pour la transmission de données entre les appareils.

2.1.5 MangoDB

Nous utiliserons ensuite MangoDB, en tant que système de gestion de base car il offre une flexibilité sans pareille pour modéliser des données variées.

2.1.6 Node-Red

Nous avons choisi node-red pour être le subscriber au serveur Mosquitto, et pour gérer les données reçus et les stocker dans la base de données MangoDB. Ce choix est peut être pas le plus efficace, mais il est le plus simple à mettre en place et à maintenir.

2.1.7 Docker Compose

On a choisi d'écrire nos configuration de serveur sous la forme d'un docker compose pour simplifié le déploiement.

```
version: '3'
services:
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    ports:
      - "1883:1883"  # MQTT port
    networks:
      - mqtt_network
    restart: always

  mongodb:
    image: mongo
    container_name: mongodb
    volumes:
      - mongodb_data:/data/db  # Persistence volume for MongoDB data
    ports:
      - "27017:27017"  # MongoDB port
    networks:
      - mqtt_network
    restart: always

  nodered:
    image: nodered/node-red
    container_name: nodered
    ports:
      - "1880:1880"  # Node-RED UI port
    networks:
      - mqtt_network
    volumes:
      - nodered_data:/data  # Persistence volume for Node-RED data and
          configuration
    restart: always
```

```
33
34 networks:
35   mqtt_network:
36
37 volumes:
38   mongodb_data:
39   nodered_data:
```

Listing 2.1 : Orchestration des containeurs

Chapitre 3

Maintenance et Mise à jour

Chapitre 4

Interaction avec des services en ligne