HEPL
Haute Ecole de la Province de Liège

Année Académique 2024 – 2025

IOT :
Gestion de flottes de vélos connectés

Andrea SPELGATTI
M. Ing. Ind. - Informatique

Martin VAN DEN BOSSCHE
M. Ing. Ind. - Informatique

Table des matières

1 Introduction

1.1 Problématique

Lors de nos diverses balades en ville, nous pouvons désormais constater l'apparition de plus en plus fréquente de vélos en libre-service. C'est ainsi que l'idée de gérer une flotte de vélos universitaires, disponibles pour les étudiants et le personnel sur un campus, nous est venue à l'esprit. Des problématiques apparaissent directement avec cette idée, comme l'exemple du campus de Google à Mountain View, où une succession de vols des vélos mis à disposition sur le site a eu lieu, avec près de 250 disparitions par semaine [1].

On ne sait également pas qui utilise les vélos, si les règles d'utilisation sont respectées ou si les vélos sont en bon état.

Il semble donc nécessaire de mettre en place un système de gestion de ces vélos pour éviter ce genre de désagrément. Cet ajout est aussi l'occasion de proposer une série de fonctionnalités qui pourraient être utiles pour les utilisateurs.

1.2 Objectif

L'objectif de ce projet est de réaliser un système de gestion permettant de suivre l'activité des vélos en libre-service à l'intérieur d'un campus.

Ce système devra permettre de gérer les vélos, les stations, les utilisateurs et les trajets. Nous ajouterons par ailleurs quelques fonctionnalités intelligentes pour améliorer le confort de l'utilisateur.

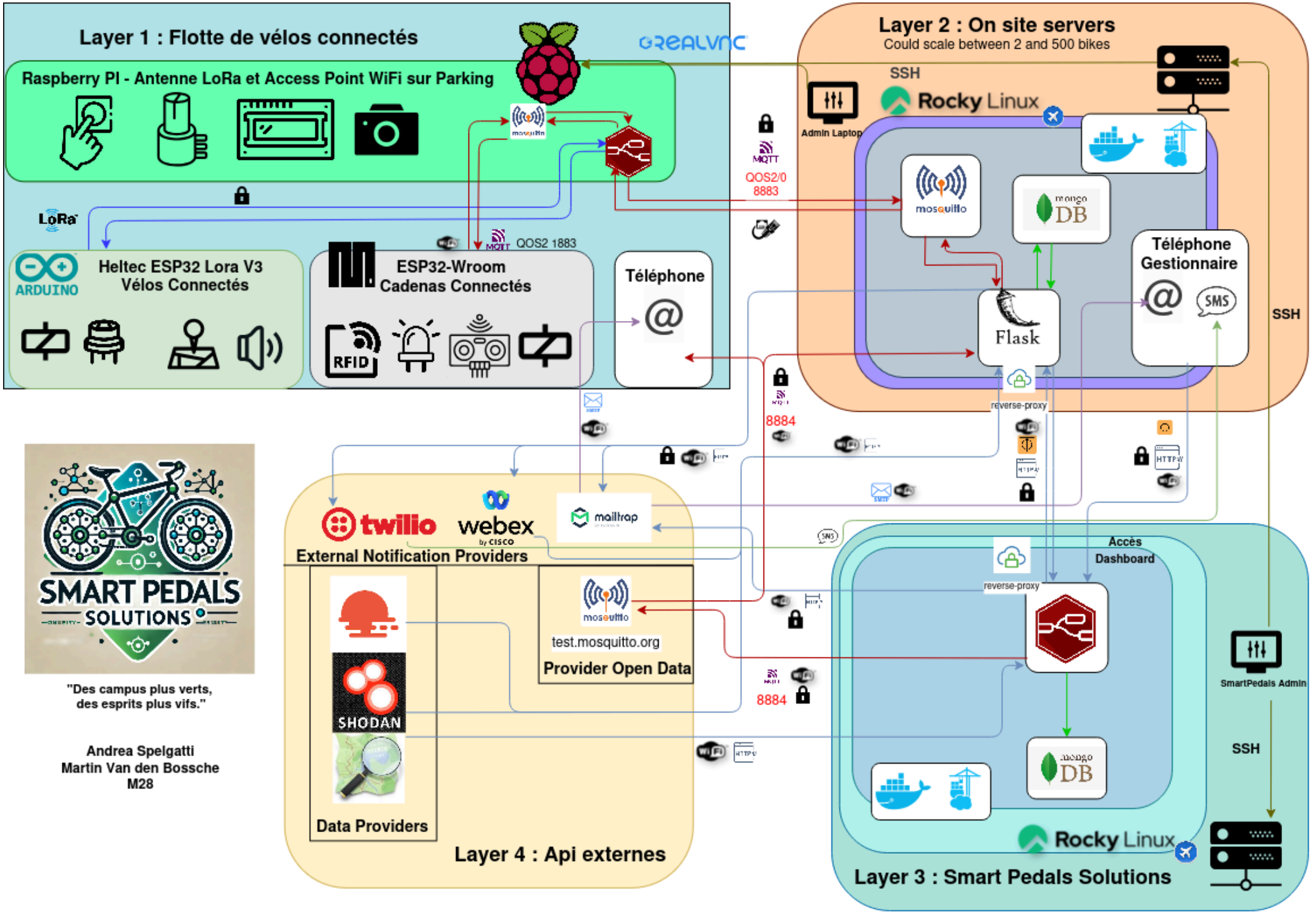
1.3 Proposition

Le projet consiste à réaliser trois objets connectés avec des capteurs différents. Ces objets sont les suivants :

- un vélo connecté avec un ESP32 compatible LoRa, intégrant une gestion des lumières en fonction de la luminosité, un buzzer, un GPS pour prévenir les vols ;
- une station connectée de sécurité/recharge avec un ESP32 compatible Wi-Fi, comprenant la détection de la présence d'un vélo et un système de déverrouillage par badge ;
- Une antenne connectée avec un edge processing basé sur une Raspberry Pi, se connectant aux deux autres objets et faisant office de point relais (hub).

Le serveur de gestion sera développé en Python avec une base de données MongoDB. Un serveur sera déployé chez le client, et un second sera géré par nos soins pour la distribution des mises à jour, le dépannage et la télémétrie.

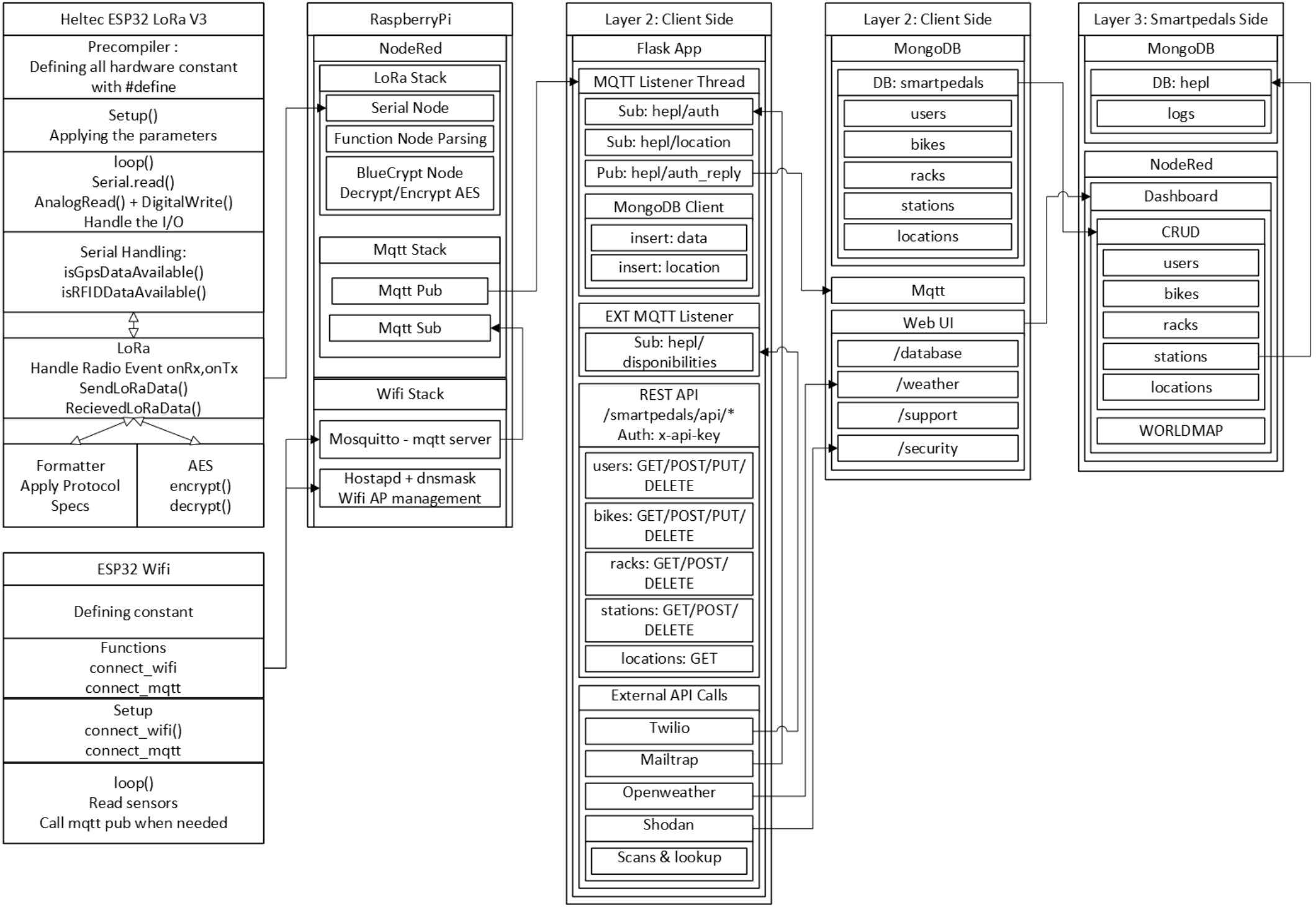
Les deux serveurs seraient basés sur Rocky Linux en utilisant une architecture containerisée avec Docker.



SMART PEDALS SOLUTIONS

"Des campus plus verts, des esprits plus vifs."

Andrea Spelgatti
Martin Van den Bossche
M28


Heltec ESP32 LoRa V3
Precompiler :
Defining all hardware constant
with #define
Setup()
Applying the parameters
loop()
Serial.read()
AnalogRead() + DigitalWrite()
Handle the I/O
Serial Handling:
isGpsDataAvailable()
isRFIDDataAvailable()
LoRa
Handle Radio Event onRx,onTx
SendLoRaData()
RecievedLoRaData()
Formatter
Apply Protocol
Specs
AES
encrypt()
decrypt()
ESP32 Wifi
Defining constant
Functions
connect_wifi
connect_mqtt
Setup
connect_wifi()
connect_mqtt
loop()
Read sensors
Call mqtt pub when needed
RaspberryPi
NodeRed
LoRa Stack
Serial Node
Function Node Parsing
BlueCrypt Node
Decrypt/Encrypt AES
Mqtt Stack
Mqtt Pub
Mqtt Sub
Wifi Stack
Mosquitto - mqtt server
Hostapd + dnsmask
Wifi AP management
Layer 2: Client Side
Flask App
MQTT Listener Thread
Sub: hepl/auth
Sub: hepl/location
Pub: hepl/auth_reply
MongoDB Client
insert: data
insert: location
EXT MQTT Listener
Sub: hepl/
disponibilities
REST API
/smartpedals/api/*
Auth: x-api-key
users: GET/POST/PUT/
DELETE
bikes: GET/POST/PUT/
DELETE
racks: GET/POST/
DELETE
stations: GET/POST/
DELETE
locations: GET
External API Calls
Twilio
Mailtrap
Openweather
Shodan
Scans & lookup
Layer 2: Client Side
MongoDB
DB: smartpedals
users
bikes
racks
stations
locations
Mqtt
Web UI
/database
/weather
/support
/security
Layer 3: Smartpedals Side
MongoDB
DB: hepl
logs
NodeRed
Dashboard
CRUD
users
bikes
racks
stations
locations
WORLDMAP

2 Prix

2.1 Prix de fabrication

Pour une station : D'après le site Dexter Industries (dexterindustries.com), le kit GrovePi est affiché à 150 $.

- Raspberry Pi Model 4B (4 Go) - 75 $
- Dragino LoRa Adapter - 30 $

Par vélo :

- Carte Heltec LoRa - 20 $
- ESP32 - 5 $
- Adafruit GPS v3 **(nous avons la v2)** - 30 $

Total (station avec une borne et un vélo) : 310 $ (hors prix du vélo)

2.2 Estimation d'un coût plus raisonnable

En disignant un custom PCB et en privilégiant des composants "non dev", on devrait pouvoir baisser les coûts :

Capteurs et affichage — En regardant sur aliexpress :

- PCB sur mesure + ESP32 nus (hors cartes de développement) : 2 $ + 0.50 $ par unité
- Antennes LoRa (AliExpress) : 5 $
- Capteurs simples (LED, LDR, buzzer, etc.) : 1 $ l'unité

Donc en se fournissant en chine on voit des prix inférieurs.

Vélo (ESP32 + module LoRa externe) — En optant pour des ESP32 nus et un module LoRa externe, le coût estimé descend à 45 $ par vélo.

Station — Avec la même logique d'optimisation, on peut viser 20 $ pour l'électronique seule, en tenant compte du boîtier, de la fixation et des à-côtés, l'estimation réaliste se situe plutôt entre 75 et 100 $.

Antenne LoRa — Une antenne LoRa de base ne nécessite probablement pas une Raspberry Pi ; une alternative plus économique comme une **Orange Pi Zero** peut convenir (15 $).

Conclusion : — En choisissant des composants moins "grand public" (moins plug-and-play), on réduit nettement les **coûts**. Toutefois, le **coût par vélo** demeure significatif.

3 Clientèle cible

Dans notre exemple initial, nous avons parlé de vélos en libre-service (type Google), fréquemment victimes de vols. Notre solution vise en priorité des organisations capables de déployer et gérer des flottes :

- Etablissements universitaires et hautes écoles ;
- Institutions publiques (villes / communes) ;
- Grandes entreprises disposant.

Avec un exemple chiffré : La HEPL compte près de 10 500 étudiants et 1 200 membres du personnel. Un déploiement initial pourrait couvrir 5 % de cette population, soit 585 vélos répartis sur ses 3 campus :

- Campus 2000 : 195 vélos ;
- Campus Barbou : 195 vélos ;
- Campus Gloesner : 195 vélos.

Avec une augmentation annuelle de 2 % du nombre de vélos déployés, la couverture atteindrait 10 % de la population du campus en environ 2 ans (calcul basé sur une progression simple). Autres clients potentiels D'autres institutions pourraient adopter SmartPedals, comme :

- L'Université de Liège ;
- Gramme ;
- Des entreprises locales telles qu'Eurogentec et EVS.

4 Consommation électrique

4.1 Général

Hypothèses : alimentation 5 V ; courants moyens en fonctionnement (hors pics TX/RX et démarrage). L'énergie/jour est estimée par $\mathbf{E = P \times 24\ h\ /\ 1000\ [kWh]}$ (avec $P = U \times I$).

Appareil	Courant moyen	Puissance (pour 5 V)	Energie / jour	Remarques
Heltec ESP32 LoRa v3	140–150 mA	0,70–0,75 W	0,0168–0,0180 kWh	Le GPS et l'ESP32 dominent la consommation
ESP32-WROOM	150 mA	0,75 W	0,0180 kWh	Le Wi-Fi domine la consommation
Passerelle Raspberry Pi	740–960 mA	3,7–4,8 W	0,0889–0,1152 kWh	Pi 4B ≈ 4,5–4,8 W ; ventilateur jusqu'à 1 W

4.2 Noeud vélo (ESP32)

Hypothèses : alimentation 5 V ; émission LoRa 0,1 s toutes 30 s (0,33 % de duty cycle). La puissance se déduit via $\mathbf{P = U \times I}$ ($U = 5$ V).

Sous-ensemble	Courant moyen
Carte ESP32 (base)	110 mA
LoRa en écoute	+ 5 mA
LoRa en émission **(0,1 s / 30 s, moyenne)**	+ 0,4 mA
GPS Adafruit	+ 22 mA
Capteur de lumière	+ 1 mA
OLED embarqué	+ 2 mA
LED **(occasionnelle)**	+ 0,2 mA
Relais **(10 % du temps)**	+ 7 mA
Total (arrondi)	**140–150 mA**

Puissance estimée à 5 V : 0,70–0,75 W.

4.3 ESP32-WROOM (borne)

Hypothèses : Wi-Fi actif en continu ; alimentation 5 V. La puissance se déduit via $\mathbf{P = U \times I}$ ($U = 5$ V).

Sous-ensemble	Courant moyen
ESP32 (Wi-Fi continu)	120 mA
Lecteur RFID (MFRC522)	+ 20 mA
LED (1 allumée + 1 occasionnelle)	+ 2,2 mA
Relais (10 % du temps)	+ 7 mA
Capteur ultrason **(rare)**	+ 0,8 mA
Total (arrondi)	**150 mA**

Puissance estimée à 5 V : 0,75 W.

4.4 Antenne (station)

Variante / ajout	Puissance
Pi 3B+ (repos typique)	1,9 W
Pi 4B (repos typique)	2,7 W
Point d'accès Wi-Fi (charge légère)	+0,3–0,7 W
LoRa HAT (réception)	+0,06 W
Grove LCD (rétroéclairage)	+0,3 W
Ventilateur 5 V	+1,0 W
Autres (GrovePi, boutons, encodeur)	+ faible
Total Pi 4B (ex. avec Wi-Fi léger + ventilateur)	**4,5–4,8 W**

4.5 Notes

Ces données sont une aggrégation des consommation **moyennes** relevées principalement sur des sites d'e-commerce (p. ex. sparkfun.com). Pour une estimation fiable, il est recommandé d'effectuer des mesures réelles :

- **Wattmètre** : mesure de la consommation moyenne sous charge représentative ;
- **Oscilloscope** : capture des transitoires et pics (TX/RX, démarrages, etc.).

5 Coûts mensuels

- Electricité : négligeable à l'échelle d'un noeud.
- Postes difficiles à estimer a priori :
  - Usure et maintenance du matériel (capteurs, batteries, boîtiers, cadenas) ;
  - Coûts d'API et d'abonnements (SMS/e-mail, etc.) ;
  - Connectivité réseau (éventuellement 4G) ;
  - Montée en charge de l'architecture serveur (compute, stockage, logs, sauvegardes).

Structure des coûts

- **Fixes** : passerelles, infrastructure de base, supervision, se diluent avec la taille de la flotte (une station pour plusieurs vélos).

- **Variables** : matériel par vélo, trafic, stockage, API, croissent **quasi linéairement** avec le nombre d'utilisateurs / vélos.

A priori, on peut s'attendre à une progression **approximativement linéaire** des coûts variables avec la taille de la flotte, l'estimation globale reste fortement dépendante du dimensionnement final.

6 Sécurité

6.1 Mots de passe hardcodés et faibles

Nos constats et contre-mesures actuelles :

6.1.1 Docker

- Utilisation de **variables d'environnement** pour éviter d'écrire les mots de passe en clair dans les fichiers.
- Emploi de **mots de passe uniques et non par défaut**

6.1.2 Raspberry Pi

- Le mot de passe par défaut **raspberrypi** n'est pas utilisé ; un **mot de passe distinct et robuste** est défini.

6.1.3 Embarqué (ESP32 / Arduino)

- Sur les ESP32, les **secrets sont hardcodés**.
- Le code Arduino étant compilé, l'accès direct est **plus difficile** : cela **ne constitue pas** une mesure de sécurité.
- **Améliorations recommandées**
  - Mettre en place **TLS** entre noeuds et serveur, idéalement en **authentification mutuelle (mTLS)** avec **certificats par appareil**.
- **Etat du projet**
  - **Nous aurions dû privilégier TLS plutôt qu'un chiffrement 100 % symétrique partagé.**
  - **Les ESP32 disposent d'un chiffrement du stockage intégré ; nous n'avons malheureusement pas encore exploré ni mis en place cette piste.**

6.2 Services réseau non sécurisés

6.2.1 LAN

Le réseau LAN reliant les différents équipements n'est pas sécurisé et est considéré comme étant **physiquement inaccessible**. Néanmoins, cette hypothèse reste fragile. L'usage de **802.1X** aurait permis de **sécuriser** davantage ce réseau.

6.2.2 Wi-Fi

La Raspberry Pi, utilisée comme **access point**, est en **WPA2**. Une **gestion centralisée des identités** via **802.1X / RADIUS (WPA2-Enterprise)** - idéalement avec **EAP-TLS** et certificats par appareil serait bien plus sûr.

6.2.3 Lora

Le chiffrement **AES** est actuellement utilisé, mais il est malheureusement appliqué via un protocole **obsolète**. Une solution plus sécurisée consisterait à implémenter TLS sur LoRa, afin de renforcer la protection des échanges.

6.2.4 Internet

Toutes nos communications inter-sites transitent via des **tunnels VPN** chiffrés et authentifiés, ce qui **simplifie** et **sécurise** les échanges entre nos serveurs distants.

6.3 Interfaces d'Ecosystème non sécurisées

6.3.1 Flask API

Double barrière en place :

- Clé **API** requise ;
- **Reverse proxy** en frontal (terminaison TLS).

6.3.2 MQTT

Identification par client_id — Filtrage et vérification des **client_id** (avec stratégie de nommage par appareil). **Remarque : à compléter par des contrôles d'authentification plus forts. DEPRECATED**

Authentification / Autorisation —

- **utilisateur / mot de passe** pour **publish** / **subscribe** ;
- Droits **partiels** appliqués via des **ACL** (topics autorisés / interdits).

TLS — Utilisation de **certificats** pour le broker et les clients avec le port 8883.

6.4 Utilisation de Composants Non Sécurisés ou Obsolètes

6.4.1 Logiciels à jour

Nous utilisons une version de **Rocky Linux** toujours **supportée**, recevant des correctifs de sécurité réguliers. C'est d'autant plus important que certaines machines sont **partiellement exposées à Internet**.

6.4.2 NodeRed

Dashboard **DEPRECATED**, besoin d'être passé à dashboard 2.0 par exemple.

6.4.3 Docker

Nous employons les **dernières versions stables** de nos images et pouvons les **mettre à jour facilement** avec **Docker Compose** (bump de version dans `docker-compose.yml`).

6.4.4 Raspberry Pi

Les logiciels **ne sont pas à jour** :

- **Node-RED** présente des failles de sécurité connues ;
- **Debian Buster** n'est plus supportée.

La décision de ne pas mettre à jour vient du fait que **Dexter Industries** ne tient plus à jour les **bibliothèques GrovePi** pour les versions modernes de **Raspbian**.

Atténuation actuelle — La **Raspberry Pi** n'a **aucun accès direct à Internet** et ne peut communiquer vers l'extérieur que via le **réseau filaire** ou son **Wi-Fi sécurisé**.

6.5 Protection Insuffisante de la Vie Privée

Notre principe de base était de **conserver toutes les données personnelles** des utilisateurs sur les **serveurs du client**, avec des **purges automatiques** des bases (p. ex. **7 jours** pour les données de localisation). (**RGPD**)

Cependant, nous avons ajouté une **sauvegarde** sur un **serveur L3** afin de produire des **métriques** qui font l'objet d'un **nettoyage tous les 30 jours**.

6.6 Transfert et Stockage de Données Non Sécurisés

Côté **transit**, notre architecture **n'implique pas de transferts non sécurisés** : les communications sont **chiffrées** (VPN/TLS).

En **stockage**, les **bases de données ne sont pas chiffrées sur disque** pour l'instant. Des mesures d'**accès système** sont en place : un **seul utilisateur Linux** dispose des droits nécessaires pour les accès distants, et l'**accès root est désactivé** par défaut.

6.7 Absence de Gestion des Appareils

6.7.1 Raspberry Pi

Accès **SSH** via la **Layer 2** : un **bastion (jump host)** est requis, une fois connecté à la L2, des accès **SSH** et **VNC** sont disponibles.

6.7.2 Layer 2

Un accès **SSH depuis Internet** est possible. Si le client **bloque le port 22** sur le firewall nous **conservons l'administration** via **ZeroTier**.

6.7.3 Layer 3

Accès disponibles via **Internet** et via le **réseau SmartPedals** (tunnels chiffrés).

6.7.4 ESP32

Pas de fonctionnalité similaire : les ESP32 **ne disposent pas** actuellement d'un mécanisme d'**OTA** déployé. Nous savons que des solutions d'OTA existent, mais nous **n'avons pas approfondi** ni mis en place cette piste.

6.8 Paramètres par défaut non sécurisés

Les paramètres par défaut de notre **architecture** sont plutôt **sécurisés**. Les utilisateurs doivent toutefois **modifier immédiatement** tous les mots de passe initiaux.

Bonnes pratiques complémentaires :

- Désactiver les comptes et identifiants par défaut ;
- Régénérer les secrets/clefs au déploiement ;
- Restreindre les ports/services exposés.

6.9 Durcissement physique insuffisant

6.9.1 Raspberry Pi

En production, la passerelle serait idéalement **placée en hauteur** (mât/piquet) pour **accroître la portée** et **réduire les manipulations non autorisées**. Notre **prototype** actuel **n'est pas sécurisé physiquement**.

Pistes de renforcement : boîtier IP65+, vis/écrous anti-effraction, scellement, fixation antivol, alimentation protégée.

6.9.2 ESP32 sur vélo

L'usage d'un **devkit** avec **broches de debug accessibles** et un **boîtier non intégré** au cadre n'offre pas une sécurité suffisante. Un **PCB personnalisé** permettrait un **boîtier adapté**, intégré au cadre.

Pistes de renforcement : scellement (résine/potting), visserie inviolable, détection d'ouverture/choc.

6.9.3 ESP32 cadenas

Le **prototype 3D** illustre le concept. En production, il faudrait utiliser un **acier épais** (anse blindée) et **sceller** le cadenas au niveau de la station (béton/platine métallique).

6.9.4 Serveurs

Ils peuvent être placés dans des **data centers** classiques, où l'accès est déjà restrein et géré avec une copie sur le **cloud** des données sensibles.

7 Conclusion

Pour maximiser notre apprentissage et sortir de notre zone de confort, nous avons inversé nos rôles habituels. Ainsi, chacun d'entre nous a pris en charge la partie du projet qu'il maîtrise le moins. Cette approche nous a permis, d'une part d'élargir nos compétences respectives, et d'autre part d'aborder des défis avec un regard nouveau, favorisant la collaboration et la compréhension du système dans son ensemble (ce qui ne nous a tout de même pas empêché d'avoir des soucis de communication en interne).

7.1 Martin

Ce projet m'a permis de passer de la théorie à la pratique, surtout côté **DevOps** et **administration système**.

- **Docker / Compose.** Je suis passé d'une compréhension superficielle (une fois il y a longtemps pour OS avec guacamole) à une utilisation concrète : images, réseaux, volumes. La containerisation de **ZeroTier** (initialement prévue en service natif sur Rocky Linux 10) m'a montré qu'on peut contourner l'absence de paquets tout en gardant l'accès aux interfaces réseau.
- **Caddy (reverse proxy).** Objectif : donner des **URL claires** au lieu d'IP:port. La bascule **HTTP vers HTTPS** a révélé pas mal de pièges (redirections, certificats). J'ai touché **OpenSSL** et je pense avoir mieux compris les histoires de **Common Name (CN)** en environnement conteneurisé. Parfois, le problème vient de la **configuration**, pas du code.
- **MongoDB.** Utilisation d'**init scripts** et d'**index TTL** pour maîtriser la rétention (utile pour le **RGPD**). Ça évite d'alourdir la base inutilement.
- **Flask + front.** Première vraie intégration web (HTML, routes). J'ai corrigé des erreurs de logique (ex. mauvais usage de `render_template` au lieu de **redirections** HTTP). Les **SSE** ont permis un rafraîchissement en direct (mais montrent leurs limites quand les messages arrivent en rafale : à optimiser).
- **Traitement de données. JSONPath** m'a aidé à formater proprement les messages **MQTT**, particulièrement utile avec l'application mobile : Iot MQTT Panel.

En somme, j'ai renforcé mes bases et gagné en confiance d'utilisation : je sais déployer une pile conteneurisée, diagnostiquer des soucis de réseau/certificats, structurer la donnée (TTL, init scripts).

7.2 Andrea

Je me suis concentré sur la partie hardware et j'ai principalement renforcé mes compétences sur :

- **Développement embarqué** : ESP32, Arduino et MicroPython.

- **Débogage de capteurs** ; compréhension des protocoles UART et I²C, ainsi que des capteurs analogiques et numériques.
- **Compréhension plus approfondie des pilotes Linux**, utilisation de dmesg et gestion des permissions via les groupes Unix.
- Premier contact avec **JavaScript et Node.js** ; gestion des dépendances NPM dans Node-RED.
- Administration de base de **systemd** (unités, services).
- Utilisation plus fluide de la **hiérarchie de configuration** sous /etc/.
- **Configuration et routage réseau sous Linux** ; mise en place d'un point d'accès Wi-Fi avec hostapd et dnsmasq (plus simple à prendre en main que dhcpd).

Frustrations rencontrées :

- Certains aspects du projet ont représenté des défis stimulants, mais parfois exigeants. D'une part, la volonté de donner du sens à chaque étape, combinée à une liste de tâches ambitieuse, a parfois rendu le développement plus complexe que prévu. Cela a nécessité un équilibre délicat entre l'autonomie créative et le respect des contraintes techniques et organisationnelles.
- L'absence de support de GrovePi sur les OS plus récents complique l'utilisation de Node-RED sous Raspberry Pi OS (ex-Raspbian).

Bibliographie

[1] A. Ruggiero, “Security Flaw: Google 'Free Bikes' Stolen by the Hundreds Each Week.” Accessed: Oct. 09, 2024. [Online]. Available: https://gearjunkie.com/biking/google-bikes-stolen-theft