# 4IRC – Projet Scientifique

# Sujet – 2023-2024

*Yannick Joly, Oscar Carrillo – 4 décembre 2023*

## Table des matières

# Préambule

## Compétences visées

À l'issue de ce module vous serez capable de :

- Analyser et synthétiser des spécifications / un cahier des charges.
- Développer des solutions logicielles et matérielles en prenant en compte des contraintes de déploiement en réseaux et en respectant les bonnes pratiques de conception et de programmation orientée objet.
- Concevoir un protocole de transmission des données issues des objets communicants.

## Moyens

Il s'agit de mettre en œuvre et d'intégrer de multiples connaissances/compétences à travers un projet réalisé en équipe de 4 élèves, munis de compétences en développement logiciel et en réseau :

- Analyse d'un cahier des charges, des interactions entre les différents éléments et découpage du projet.
- Conception et réalisation d'une architecture réseau.
- Conception d'une chaîne IoT et définition du protocole d'échange des informations.
- Conception de l'architecture logicielle (Design Pattern utilisés (si utilisés...), Diagrammes de classes, Schéma de la BD) et développement de la partie applicative en Java.
- Réalisation d'une infrastructure web qui permettra de récupérer, transférer et présenter de l'information.
- Analyse, traitement et visualisation de grands volumes de données (Big Data).

# Projet

## Objectifs

L'objectif de ce projet est de réaliser d'une part un simulateur d'incendies permettant la création, le suivi et la propagation de feux de différents types (localisés sur une carte), et d'autre part de créer un dispositif de gestion de services d'urgences permettant, à partir d'informations collectées par des capteurs, de déployer et gérer les dispositifs adaptés pour éteindre les incendies.

La figure 1 ci-dessous représente le fonctionnement global de l'application.

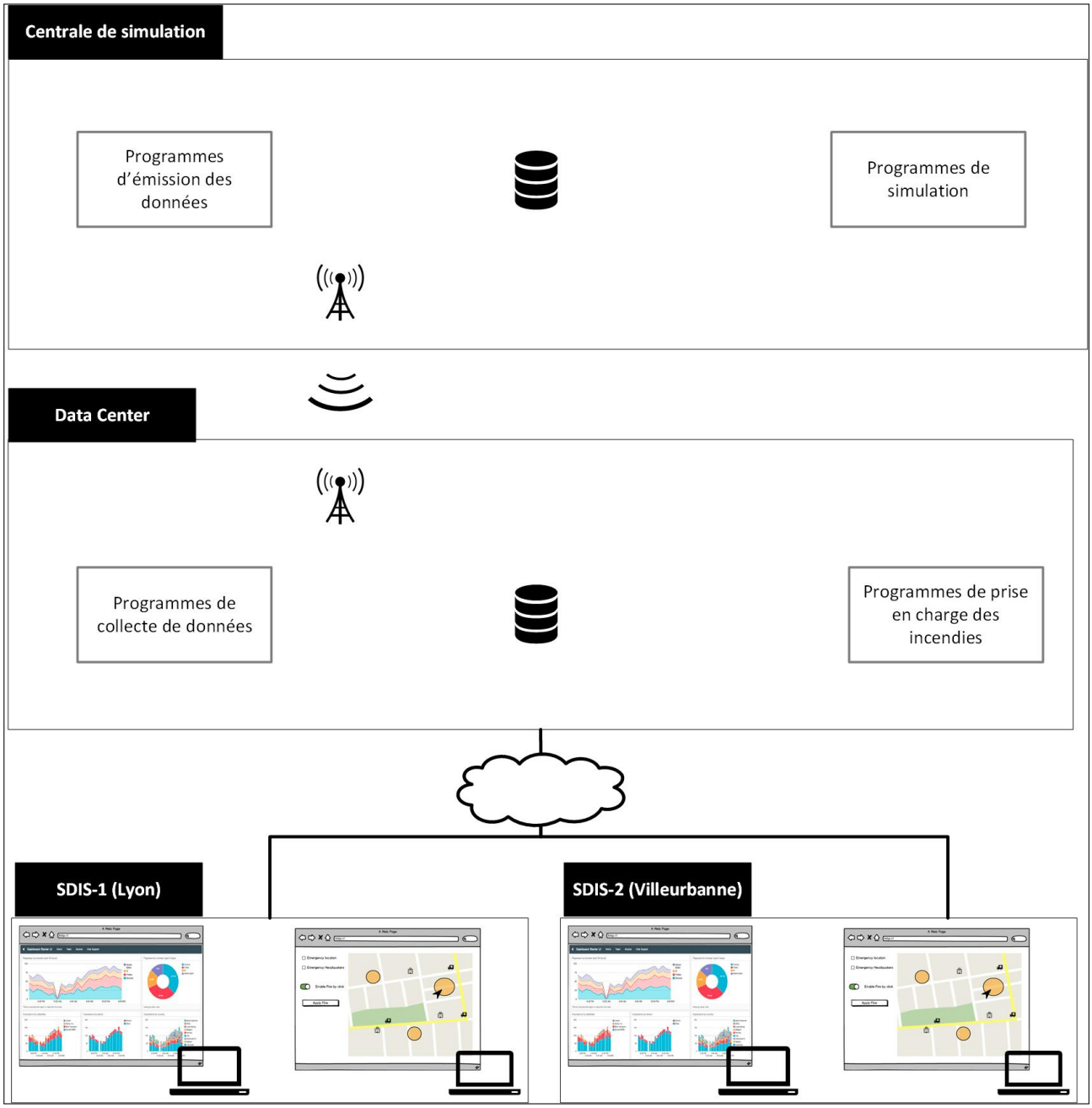


Figure 1 – Fonctionnement global de l'application

### La centrale de simulation (au début du cycle)

Son rôle est de générer des feux dont les coordonnées, l'intensité et la fréquence sont à définir dans le programme.

Ces feux sont stockés dans une BD de simulation.

Les informations détectées par les capteurs simulés (e.g. intensité, coordonnées) de ces feux sont envoyées par ondes radio au Data Center, à l'image de détecteurs de feux qui transmettent une alerte à une centrale de supervision.

### Le Data Center

Le Data Center est la partie « métier » des services de gestion d'incendie. Il est constitué d'un ensemble de programmes allant de la réception des alertes incendies à leur prise en charge. La première action de ce groupe de programmes est donc l'acquisition des données des feux simulés et envoyées par radio (étape précédente), pour alimenter et mettre à jour une BD de feux « réels ».

Ensuite, la logique métier : un programme appelé *Emergency Manager* gère une flotte de services d'urgence (casernes, camions, pompiers), détecte les feux (dans la BD) et déploie les ressources nécessaires pour les éteindre (quel(s) camion(s) de quelle(s) caserne(s) est (sont) affectés à quel(s) feu(x)). Il pourra libérer les ressources affectées à un feu lorsqu'il est éteint.

Enfin, un certain nombre de données doivent être envoyées dans un serveur de stockage sur le Cloud à des fins de statistiques et visualisations.

### Les écrans de supervision des casernes

Grâce à des écrans de supervision, les casernes peuvent visualiser dans un navigateur web les feux en cours dans la ville (ou toute autre zone géographique choisie), ainsi que la position des différents véhicules de secours. Différents contrôles permettent d'afficher / masquer les feux, les véhicules de secours, etc.

Un Dashboard affiche des statistiques en fonction de l'historique des données collectées.

### La centrale de simulation

Le simulateur détecte la présence ou non des services d'urgence près des feux (dans BD de feux réels) et réduit ou augmente leur intensité (dans la BD de feux simulés).

Les coordonnées et intensité de ces feux sont envoyées par ondes radio au Data Center. . ., et ainsi de suite jusqu'à. . .

## Architecture fonctionnelle de la solution

Une architecture fonctionnelle globale vous est proposée figure 2. Elle méritera d'être affinée : il existe en effet plusieurs manières de gérer les interactions entre le (les) Serveur Web, la (les) DataBase, le Simulator et l'Emergency Manager. Vous êtes également très libre de concevoir chaque partie comme vous l'entendez. Gardez cependant à l'esprit que la partie Simulator et la partie Emergency doivent être le plus faiblement couplées possible. En effet, la partie Emergency doit rester totalement fonctionnelle si l'on remplace le Simulator par un autre simulateur, ou même si on la remplace par un système réel de détection d'incendies. Par ailleurs, tous les composants de votre solution doivent être suffisamment génériques pour s'adapter facilement à toute extension (par exemple, on peut décider de gérer également des urgences médicales, des accidents de circulation, etc.). Identifiez donc bien pour chaque responsabilité si elle relève plutôt du Simulateur, du Manager ou si elle est partagée (et sous quelle forme).

Quelques contraintes de langage vous sont imposées (Java pour les parties « métier »), d'autres suggérées.

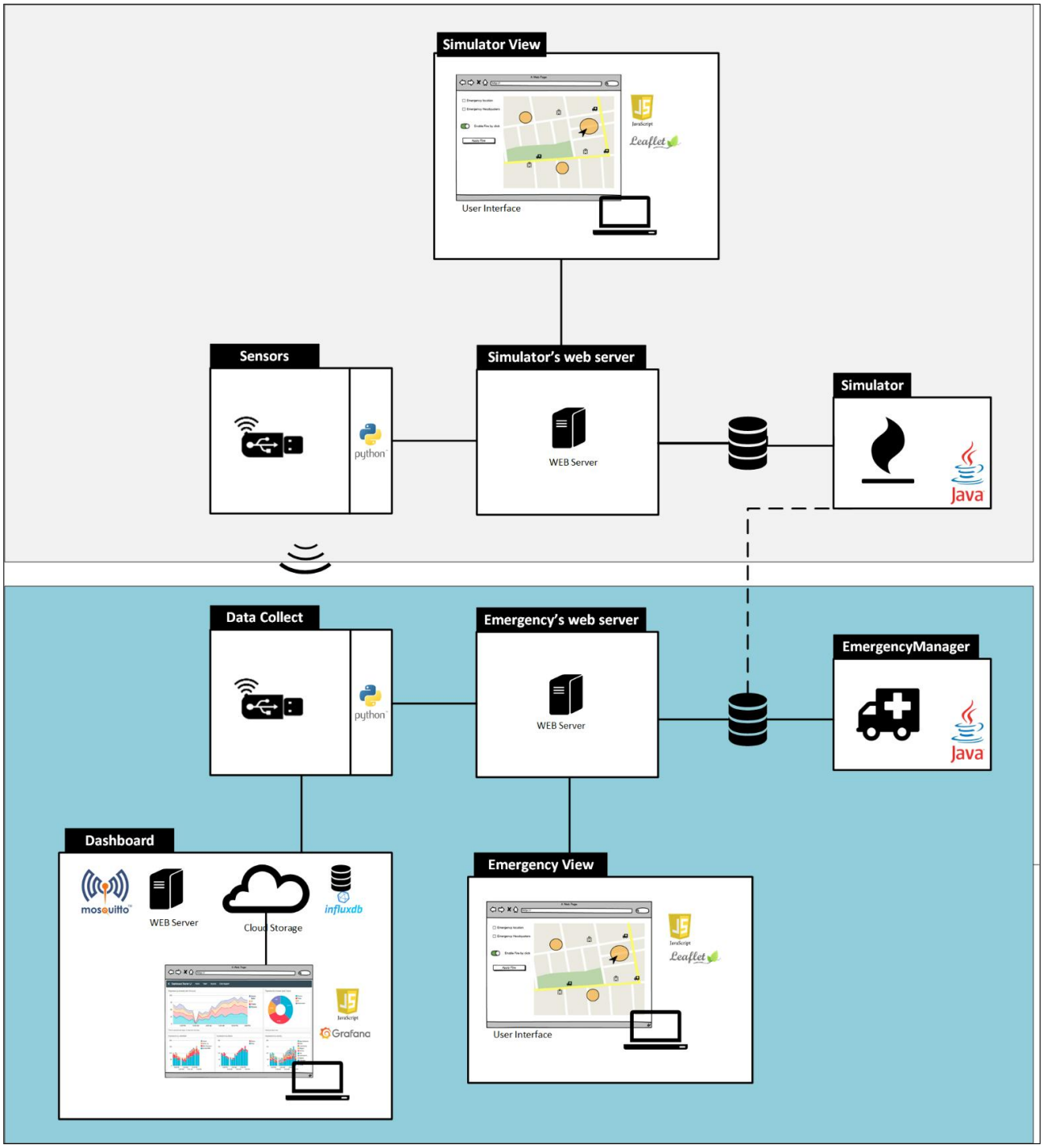


Figure 2 – Architecture globale de l'application

Quelques bonnes pratiques de développement à mettre en œuvre :

- Chaque fonctionnalité doit pouvoir être testée séparément. Prévoyez des jeux de tests en conséquence.
- Les responsabilités doivent être bien séparées : accès BD, objets métier, rendu visuel, etc.
- Concevez et développez de manière incrémentale :
  - Gérez dans un premier temps un seul feu, une seule caserne dotée d'un seul camion, camion qui se « téléporte » instantanément vers le feu en 1 tour de jeu. Les ressources (caserne, camion, éventuellement pompier) et le feu simulé sont « codées en dur » dans la BD.
  - Ensuite, ajoutez des feux (par programme), des camions, des casernes avec affectation des ressources au feux dans l'ordre d'arrivée.
  - En fonction du temps disponible (au choix) :
    - Gérez le déplacement visuel des camions, d'abord en ligne droite entre leur position initiale et le feu puis utilisez des API de calcul d'itinéraires disponibles sur le web pour déplacer vos camions le long des routes réelles.
    - Ajoutez d'autres règles « métier » à la phase d'affectation des ressources :
      * Différents types de camions (capacité, type de produit),
      * Choix du ou des camion(s) à envoyer sur un sinistre (le plus près, le premier disponible, le mieux adapté, etc.),
      * Optimisation des itinéraires des camions pour refaire le plein ou se ravitailler en produits,
      * Fatigue des pompiers durant l'intervention,
      * etc.
    - Prévoyez une interface de saisie des ressources, les affectations des ressources restant gérés par programme (sans intervention humaine).

Pour cela, respectez les principes « Responsabilité unique » et « Ouvert aux extensions - Fermé aux modifications » en pensant abstraction, encapsulation, polymorphisme et délégation.

## Architecture réseau de la solution

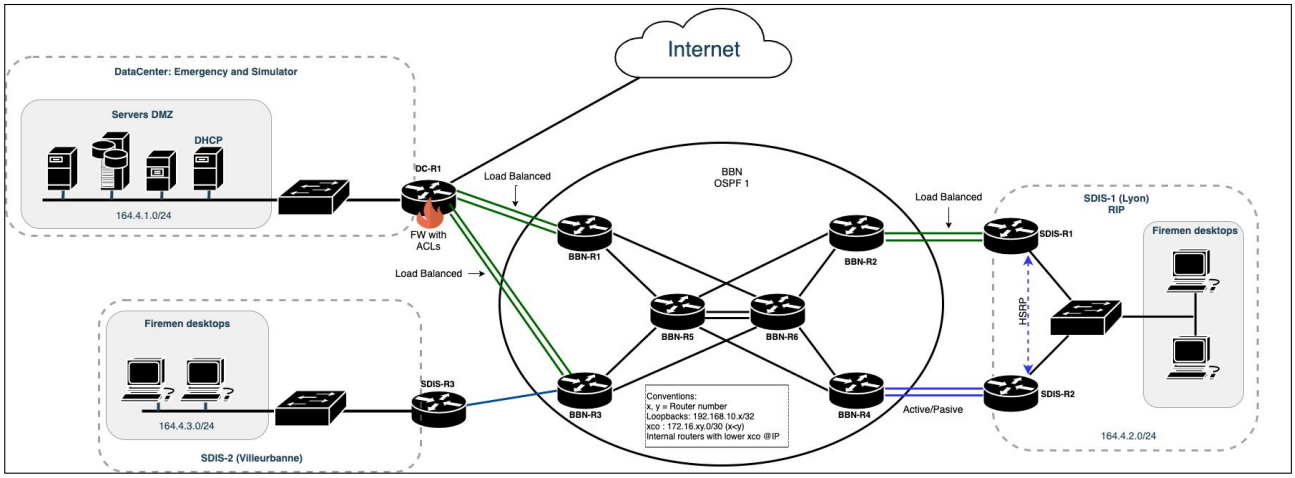


Figure 3 – Architecture Réseau

L'architecture réseau de la solution est présentée dans la Figure 3 ; dans cette architecture il y aura 3 sites principaux :

- Un Data Center
- SDIS de Lyon qui compte un central de supervision
- SDIS de Villeurbanne

Tous les sites seront interconnectés grâce à un BackBone OSPF.

Cette configuration sera la base pour pouvoir généraliser le déploiement à d'autres casernes, n'hésitez pas à fournir des scripts de génération des configurations pour les routeurs des nouvelles casernes en utilisant des moteurs de templates Jinja.

Pour la modélisation de votre architecture, vous allez utiliser le logiciel GNS3 et les ordinateurs pour les validations d'interconnections devront être virtualisés par des machines virtuelles ou conteneurs docker créées par vous. A minima, il faut créer les VMs suivantes :

- Serveur simulation
- Serveur Bases de Données PostgreSQL
- Serveur Emergency Management
- Serveurs Web
- Client SDIS Lyon
- Client SDIS Villeurbanne

*Remarque : les différents serveurs du DataCenter peuvent être dans la même VM.*

## Architecture IoT de la solution

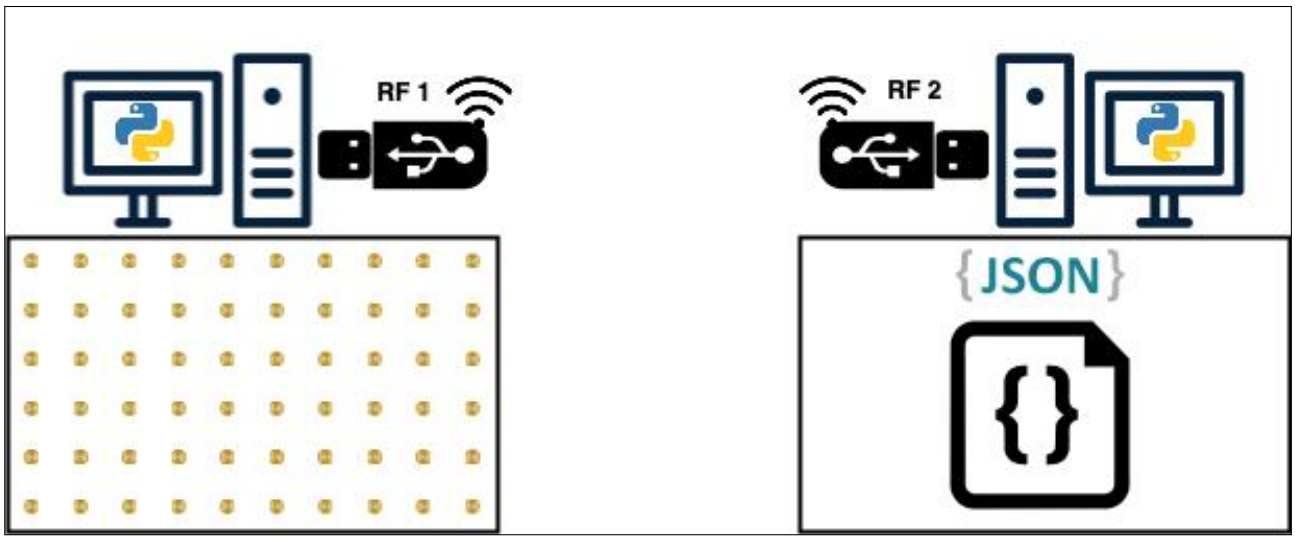


Figure 4 – Schéma IoT

L'architecture à implémenter pour votre solution IoT sera à faire à l'aide des puces `Micro:bit` comme dans la figure 4 et en respectant les chemins de génération et acquisition des données suivantes :

- les données à traiter seront issues d'une matrice de 10 points horizontaux par 6 points verticaux représentant le déploiement des capteurs d'intensité de feu à différents endroits dans la ville. L'intensité des feux sera une valeur entière entre [0,9].
- des données de tests de simulation pourront être fabriquées à partir de l'outil de simulation graphique disponible dans le git du projet.

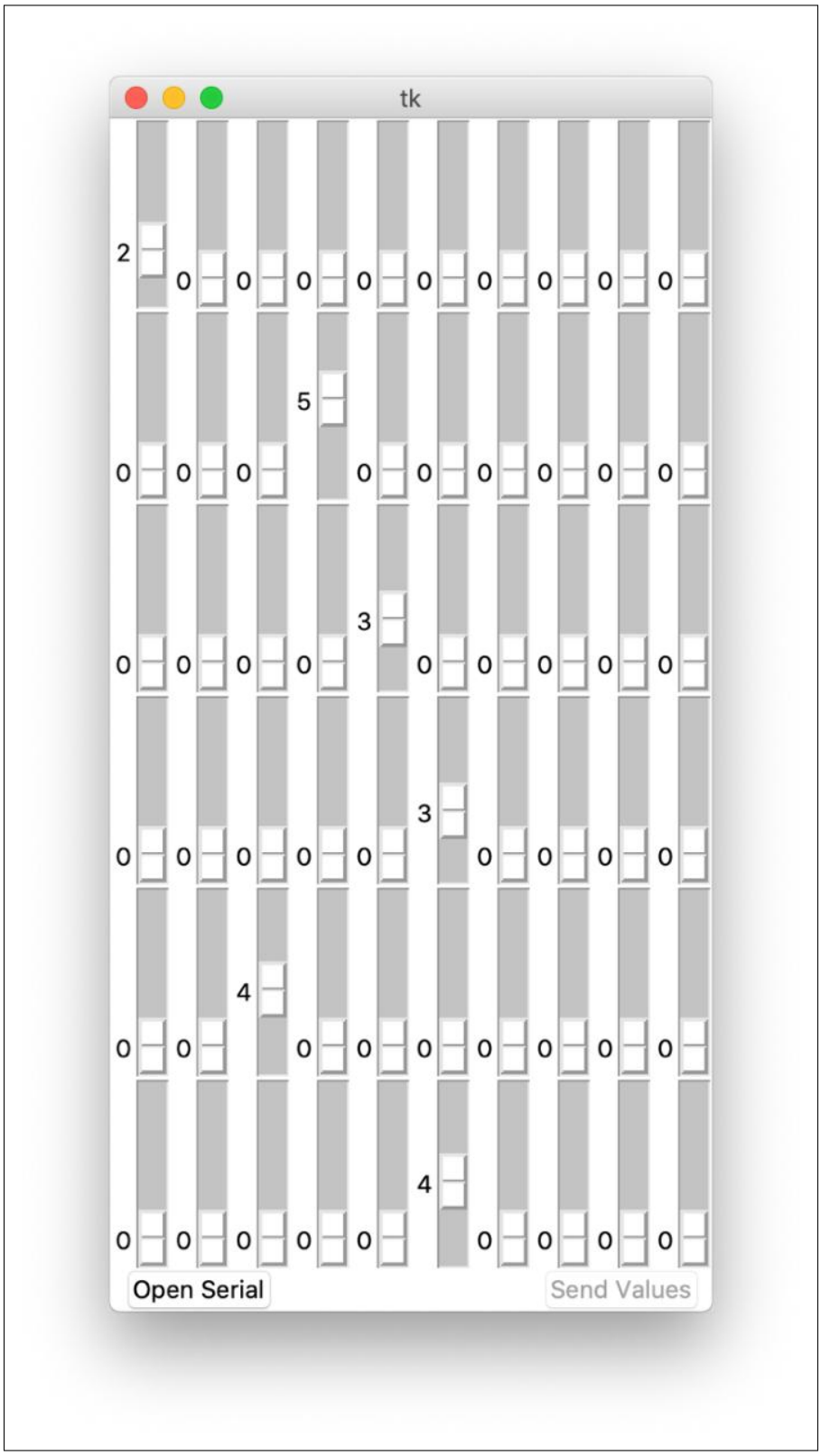


Figure 5 – Application génération données capteurs

- L’outil de "génération de données capteur" va écrire dans le port USB les valeurs des différents points en suivant le format (colonne, ligne, intensité) pour chaque point de la grille. Par exemple, la sortie USB série des données générées dans la Figure 5 sera la suivante :
- Un microcontrôleur `micro:bit` (RF1) sera connecté au port USB du simulateur pour lire les sorties dans le format spécifié au point précédent, un protocole sécurisé devra être créé pour transmettre ces valeurs à un autre microcontrôleur par voie sans fils.
- Un microcontrôleur `micro:bit` (RF2) sera connecté à un autre port USB et devra collecter par un protocole sans fils sécurisé les informations des intensités des feux. Ces valeurs devront être écrites par la sortie UART dans un format de votre choix.
- Les valeurs écrites par RF2 dans UART, seront collectées par une application qui va faire une double fonction :
  - Faire appel à un API REST DAO (format JSON) pour enregistrer dans la base de données Emergency Management les valeurs des intensités des feux reçus.
  - Envoyer par messages MQTT les valeurs des intensités des feux à un dashboard dans le cloud.

Nous mettons donc à votre disposition un petit programme de "génération de données capteur" (sorties potentielles de la carte) pour tester votre chaine IoT indépendamment du programme réel de simulation. Il est disponible sur le git :

https://github.com/CPELyon/4irc-19-20-proj-transversal

Pour intégrer l’ensemble des couches, il suffira alors de modifier ce programme pour qu’il interroge la BD et écrive les données sur le RF1.

# Annexes

## Déroulement du projet

| | | |
|---|---|---|
| 1 | 04/12 PM | Définition de l'architecture globale et découpage du projet en tâches et affectation aux membres de l'équipe (Trello ou autre). Prise en main des différentes technos à utiliser. Conception et mise en œuvre de l'architecture réseau. |
| 2 | 05/12 AM | |
| 3 | 05/12 PM | |
| 4 | 06/12 AM | |
| 5 | 06/12 PM | |
| 6 | 07/12 AM | Présentation architecture globale et gestion du projet. Présentation/Démo architecture réseau. |
| 7 | 08/12 AM | Conception logicielle de l'application. Mise en œuvre de la chaîne IoT. Début développement. |
| 8 | 18/12 PM | |
| 9 | 19/12 AM | |
| 10 | 19/12 PM | |
| 11 | 20/12 AM | |
| 12 | 20/12 PM | Présentation Conception logicielle : Diag. de classes/séquences et Schéma BD. |
| 13 | 21/12 PM | Développement logiciel (métier, rendu Web, interactions avec BD).Développement partie analyse/visualisation Big data. Intégration de l'ensemble des composants. |
| 14 | 22/12 AM | Présentation/Démo Chaîne IOT. |
| 15 | 22/12 PM | Développement logiciel (métier, rendu Web, interactions avec BD).Développement partie analyse/visualisation Big data. Intégration de l'ensemble des composants. |
| 16 | 11/01 PM | |
| 17 | 12/01 AM | |
| 18 | 12/01 PM | |
| 19 | 15/12 AM | Démo applicatif métier avec intégration de tous les composants. |
| 20 | 15/01 PM | |
| 21 | 16/12 AM | Soutenance + Démo finales. |
| 22 | 16/01 PM | |

## Évaluation

- Infrastructure Réseau : Démo maquette GNS3 connectivité sites de casernes et data center en respectant des contraints d'accès depuis des machines virtuelles.
- Infrastructure IoT : Démo collecte, envoi et réception des données des feux dans la ville ainsi que génération d'appel REST.
- Analyse fonctionnelle de l'application, diagrammes de classes et schéma de la BD.
- Qualité des programmes (respect des principes SOLID).
- Soutenance et démonstration finale.

| Réseau | IoT | Dev et intégration | Qualité Présentation |
|---|---|---|---|
| 25% | 25% | 40% | 10% |

## Documentation à consulter

| | |
|---|---|
| Java | Javadoc<br>• https://docs.oracle.com/javase/8/docs/api/ |
| | Lire/écrire dans BD en Java (Java DataBase Connectivity)<br>• https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/26258-jdbc-decouvrez-la-porte-dacces-aux-bases-de-donnees<br>• https://www.tutorialspoint.com/postgresql/postgresql_java.htm |
| | Invoquer une requête http en Java<br>• https://www.baeldung.com/java-http-request |
| | Librairie et tuto pour convertir objet <-> JSON et réciproquement<br>• https://github.com/FasterXML/jackson-databind/ |
| | Planification de tâches<br>• https://docs.oracle.com/javase/7/docs/api/java/util/Timer.html<br>• https://www.jmdoudoux.fr/java/dej/chap-planification_taches.htm |

| | |
|---|---|
| Partie Web | Python Flask<br>• https://openclassrooms.com/fr/courses/1654786-creez-vos-applications-web-avec-flask<br>• https://www.tutorialspoint.com/flask/index.htm<br>• https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world |
| | Système d'information géographique<br>• https://leafletjs.com/<br>• https://leafletjs.com/plugins.html<br>• https://docs.mapbox.com/api/ |
| | Test d'API<br>• Postman : https://www.getpostman.com/ |
| Partie réseau | Configuration routeurs<br>• https://cisco.netacad.com<br>• https://blogs.cisco.com/developer/network-configuration-template |
| Partie IoT | Accès REST API<br>• https://www.geeksforgeeks.org/get-post-requests-using-python/ |
| Partie Cloud | MQTT<br>• https://thingsmatic.com/2016/06/24/a-self-hosted-mqtt-environment-for-internet-of-things-part-2/<br>• https://hub.docker.com/_/eclipse-mosquitto |
| | Grafana<br>• https://thingsmatic.com/2017/03/02/influxdb-and-grafana-for-sensor-time-series/<br>• https://hub.docker.com/r/grafana/grafana/ |
| | InfluxDB<br>• https://hub.docker.com/_/influxdb |