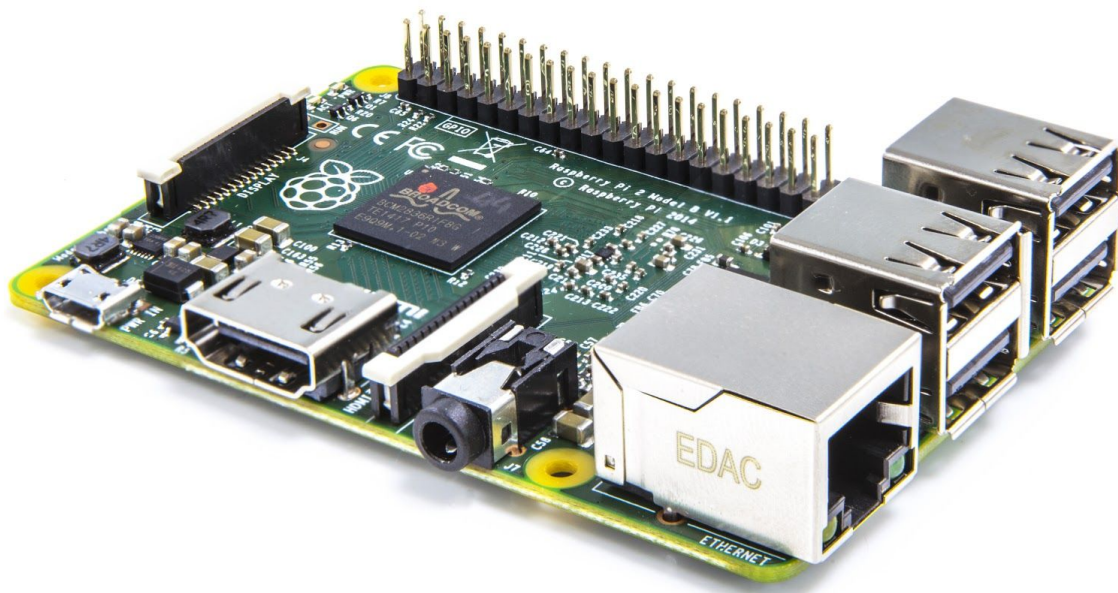


# Rapport de projet

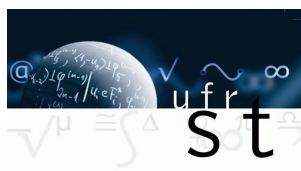
## Système informatisé de contrôle d'accès



**Étudiants créateurs du projet :**

- Amiot Nicolas
- Marc Petot
- Rousselet Corentin

**Tuteur de projet :** - Mr Éric Merlet, enseignant chercheur à l'université de Franche-Comté.



## Remerciements

Nous tenions d'abord à remercier M. Guillaume Paquette, responsable de la licence professionnelle Conception et développement orientés objet d'applications multi-tiers, pour nous avoir accordé sa confiance lors de notre inscription.

Nous souhaitons également remercier M. Eric Merlet, notre tuteur de projet, pour sa disponibilité et ses conseils autant sur la partie conception que sur la partie rédaction du rapport qui nous ont permis de mener à bien ce projet.

Nous n'oublions pas non plus le reste de l'équipe enseignante de cette licence professionnelle pour l'ensemble des connaissances qu'ils nous ont apportées tout au long de cette année.

## Sommaire

- I. Introduction**
- II. Étude du système**
  - 1. Expression du besoin
  - 2. Architecture du système
  - 3. Répartition des tâches
- III. Mise en oeuvre**
  - 1. Matériels
  - 2. Méthode de travail
- IV. Modélisation conceptuelle**
  - 1. Diagramme de cas d'utilisation
  - 2. Diagramme de classe
  - 3. Diagramme de déploiement
- V. Base de données**
  - 1. Modèle logique de données
  - 2. Administration de la base de données
- VI. Client lourd du personnel**
  - 1. Diagramme de séquence de l'ajout d'un badge
  - 2. Structure de l'application
  - 3. L'interface Homme-Machine
- VII. Contrôle des leds**
  - 1. Schéma de câblage des leds
  - 2. Utilisation des leds
- VIII. Caméras de surveillance**
  - 1. Mise en place
  - 2. Le Client/Serveur
- IX. Autorisation d'accès par mot de passe**
  - 1. Diagramme de séquence d'une tentative d'accès
  - 2. L'interface Homme-Machine
- X. Gestion du lecteur RFID**
  - 1. Mise en place et analyse des trames
  - 2. Gestion du lecteur en JAVA
  - 3. Diagramme de séquence de lecture d'un badge dans une zone non sensible
- XI. Conclusion**

## I. Introduction

Une entreprise peut avoir besoin de contrôler les passages qui s'effectuent au sein des bâtiments afin d'éviter toute intrusion qui pourrait provoquer de grave conséquence.

Pour cela, chaque employé de l'entreprise ou visiteur devra posséder un badge avec des accès spécifiques à certaines zones. Ce badge devra être présenté à chaque nouvelle zone devant un lecteur RFID autorisant ou non l'accès à cette personne. L'historique des accès est sauvegardé en cas de problème.

De plus, certaines zones sont sous haute protection. Ces zones là sont sous vidéo-surveillance et leurs accès demandent en plus de posséder un badge valide, d'indiquer le mot de passe correspondant au badge. Cela permet d'éviter tout risque d'intrusion à ces zones importantes suite à la perte ou vol d'un badge. L'employé devra néanmoins signaler la perte du badge au plus vite pour qu'on lui en réattribue un. Toutes les zones sous vidéo-surveillance sont contrôlées par un vigile.

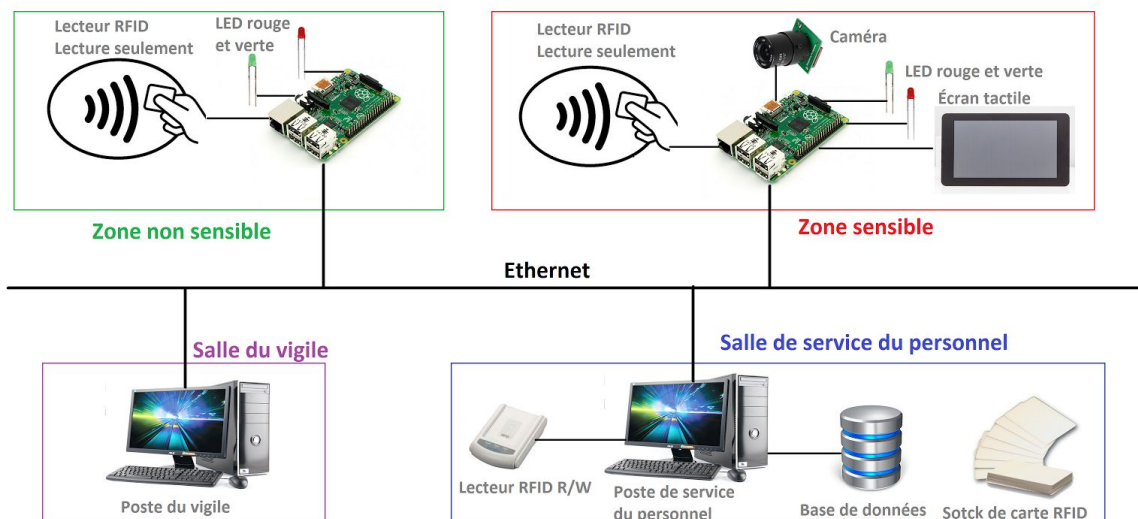
## II. Étude du système

### 1. Expression du besoin

Le système doit permettre de réaliser les fonctions suivantes :

- Créer des badges avec des accès spécifiques à chacun d'eux.
- Modifier les badges en cas de changement de poste.
- Supprimer un badge en cas de perte ou licenciement.
- Gérer l'agencement d'une structure à travers d'une application.
- Détecter un badge passé devant un lecteur RFID pour y lire son numéro et en conserver l'historique.
- Autoriser ou non l'accès à la zone selon la liste des accès propre à cette personne et si la demande correspond à l'horaire d'ouverture.
- Allumer la bonne led selon si l'utilisateur est autorisé ou non.
- En cas de zone sensible, demander en plus le bon mot de passe.
- Pour les zones sensibles, capturer le flux vidéo de chaque caméra et les afficher.

## 2. Architecture du projet



A chaque entrée, on a un lecteur RFID relié par une liaison série RS232 à une carte raspberry PI qui est elle-même reliée en liaison Ethernet au serveur. Le lecteur et la raspberry fonctionnent comme un client/serveur, la raspberry qui joue le rôle de serveur attend une nouvelle requête du client qui s'effectue lors du passage d'une carte RFID devant le lecteur. La raspberry va ensuite demander à la base de données si la personne est autorisée à passer.

Dans le cas d'une zone sensible, on retrouve bel et bien la caméra pour la vidéo surveillance et l'écran tactile pour y indiquer le mot de passe correct.

Dans la salle de service du personnel, zone qui sera sensible, on a un lecteur RFID relié au poste qui permet de lire ou d'écrire des badges. Ce poste est aussi relié à la base de données afin d'enregistrer les badges dans la base de données et y récupérer les informations relatives à ce badge.

Dans la salle du vigile, zone qui sera sensible, on peut visionner l'ensemble des caméras en temps réels.

## 3. Répartition des tâches

### ➤ **Amiot Nicolas**

- Administration de la base de données nécessaire au bon fonctionnement du système. Développement des requêtes SQL pour le projet entier.
- Création d'un client lourd pour le personnel du service. Tout d'abord, cette application permet d'ajouter, de modifier ou de supprimer des badges. Ensuite, elle permet de visualiser l'historique des passages d'une zone ou d'un badge. Pour finir, elle permet aussi l'agencement d'une structure. Une structure étant composé de bâtiment, de zone et de lecteur.

### ➤ **Petot Marc**

- Gestion du lecteur RFID :
- Analyse des trames échangées avec le lecteur.
- Création d'une fonction permettant de mettre en relation le lecteur à l'ordinateur et au raspberry pi.
- Création d'une fonction permettant d'écrire sur un badge et une autre permettant d'en lire un.

### ➤ **Rousselet Corentin**

- Création d'un système d'allumage et d'extinction d'une led verte et d'une led rouge grâce au contrôleur GPIO du Raspberry pi. Ces leds seront utilisées pour informer l'utilisateur du résultat du passage de son badge.
- Création d'une interface graphique pour permettre à l'utilisateur d'entrer son mot de passe dans les zones sous haute surveillance. L'interface graphique interroge la base de données du système où est stocké le mot de passe correspondant au badge.
- Création d'un client/serveur Python-Java affichant l'image transmise par la pi-camera depuis le Raspberry pi vers le poste du vigile.

## III. Mise en oeuvre

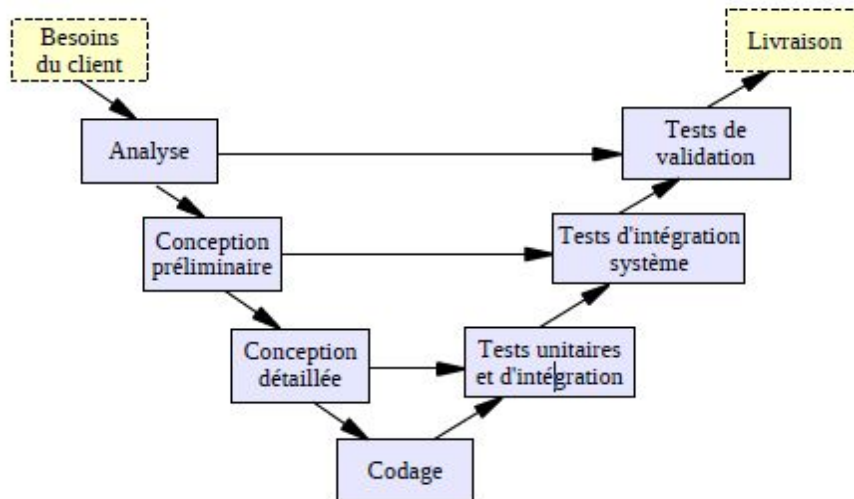
### 1. Matériels

- 2 cartes Raspberry PI 2 avec bloc d'alimentation;
- 2 cartes micro SD avec comme système d'exploitation Raspbian;
- 1 caméra et son boîtier pour carte Raspberry PI 2;
- 1 convertisseur HDMI - VGA (module PiView);
- 1 convertisseur USB - RS232;
- 1 LED verte et 1 LED rouge;
- 1 Breakout Board;
- 1 écran (en attendant l'écran tactile);
- 1 modules d'extension prototypage pour carte Raspberry PI 2;
- 1 lecteurs/encodeurs RFID;
- 1 caméra camera PI pour Raspberry;
- 5 tags RFID;
- 3 PC avec environnement de développement Java.



## 2. Méthode de travail

Nous avons travaillé selon le modèle du cycle en V. Le cycle en V est une méthode de gestion de projet qui permet de définir les étapes pour arriver le plus rapidement possible du besoin du client à la livraison du produit.



Le cycle en V est composé de plusieurs étapes:

### L'analyse :

En partant du cahier des charges on détermine les besoins du client, cette analyse permet de déterminer ce qu'il faut faire. Ensuite on passe à la conception pour déterminer comment faire.

### Conception préliminaire :

En partant de l'analyse, on réalise une ébauche de la structure du système, ensuite les choix technologiques devront être faits.

### Conception Détaillée :

Cette phase est dépendante des choix technologiques, on doit prendre en compte la phase de codage en suivant la modélisation UML, et en ayant un point de vue des utilisateurs.

### Codage/Réalisation :

Dans cette partie c'est la rédaction de toutes les méthodes dans les langages définies de chacune des classes décrites dans les phases précédentes.

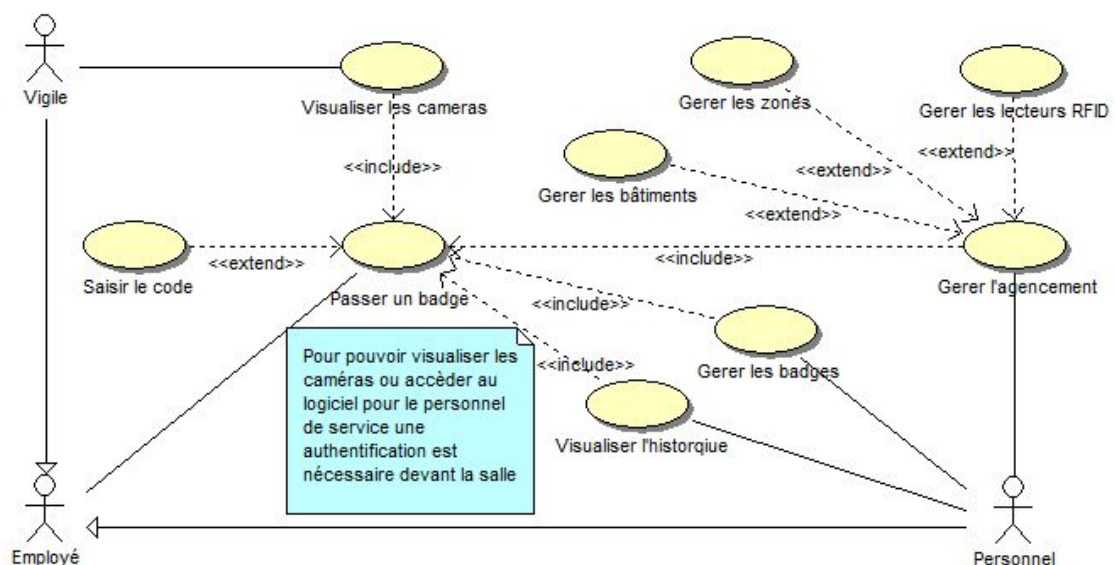
### Les tests :

Le logiciel est testé tout le long du cycle de développement. Ces tests sont spécifiques à leur partie. Ils comportent les tests unitaires, d'intégrations et de validations.

## IV. Modélisation conceptuelle

### 1. Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation sont des diagrammes UML (Unified Modeling Language) utilisés pour donner une vision globale du comportement fonctionnel d'un système. Un cas d'utilisation représente une interaction entre un utilisateur (humain ou machine) et un système.



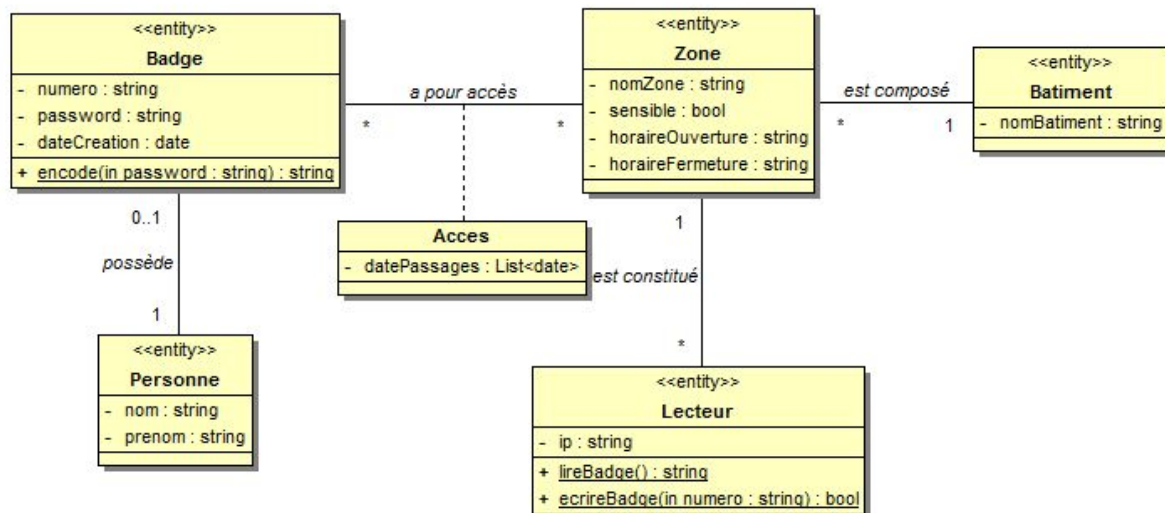
Un employé peut passer un badge afin d'accéder à une zone.

Un vigile quant à lui peut visualiser les caméras. Pour accéder à la zone de vidéo-surveillance, il doit auparavant s'authentifier en passant un badge voir en saisissant un code en plus du badge.

Un personnel de service peut quant à lui effectuer 3 actions. Tout comme le vigile, il doit passer une badge voir saisir un code pour accéder à la zone dédiée. Une fois authentifié, il peut soit gérer les badges, soit visualiser l'historique des passages soit gérer l'agencement. Gérer l'agencement correspond à gérer les bâtiments, les zones et/ou les lecteurs.

## 2. Diagramme de classe

Le diagramme de classe nous permet de vous représenter nos classes entités. On y retrouve obligatoirement les attributs de la classes dont on a besoin qui devront tous être privé afin de respecter le principe fondamentale d'encapsulation (idée de protéger l'information contenue dans un objet et de ne proposer que des méthodes de manipulation de cet objet).



Un lecteur est identifié par son adresse ip fixe. Un lecteur peut lire le numéro d'un tag RFID ou en écrire un.

Une zone est défini par un nom, si elle est sensible ou non, d'un horaire d'ouverture et d'un horaire de fermeture pour préciser à quel moment les accès à cette zone sont autorisés. Une zone possédera autant de lecteur qu'il y a de porte.

Un bâtiment possède un nom et est composé d'une ou de plusieurs zones.

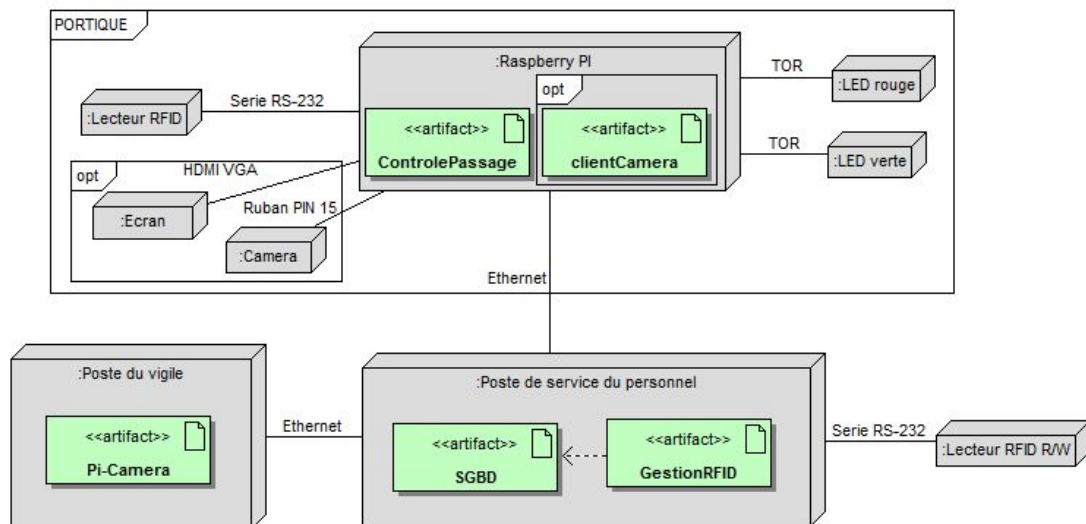
Un badge est composé d'un numéro qui correspond à celui qui est stocké sur le tag RFID, d'une date de création et d'un mot de passe, crypté en MD5 grâce à la fonction encode, qui peut être demandé pour l'accès d'une zone sensible. Un badge permet l'accès à une ou plusieurs zones.

Pour chaque accès dans une zone, on a comme donnée porteuse la liste des dates de passage permettant de générer un historique pour un badge ou une zone précise.

Une personne est caractérisé par un nom et un prénom et possédera son propre badge si elle a besoin d'un accès.

## 3. Diagramme de déploiement

Un diagramme de déploiement est 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 leurs relations entre eux.

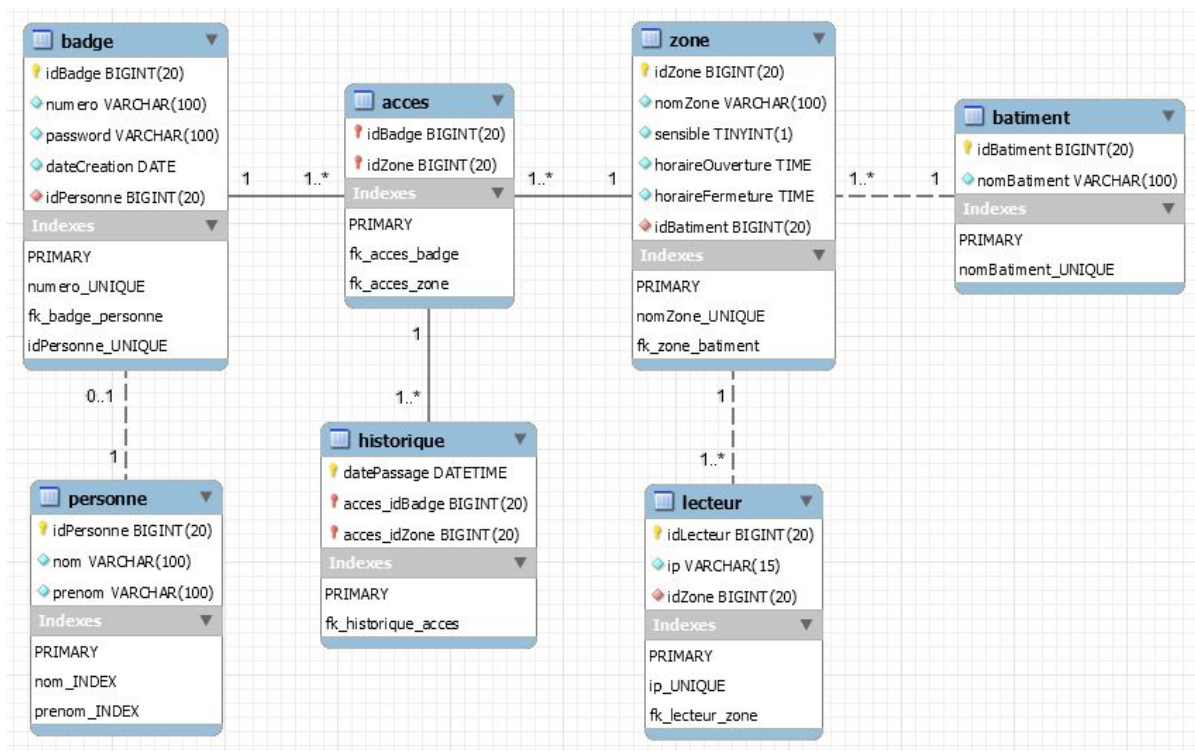


Dans ce diagramme de déploiement plusieurs composants ont été utilisés tels que :

- Les noeuds, représentés par des cubes en trois dimensions, qui sont des composants mécaniques de l'infrastructure.
- Les associations, représentées par de simples lignes, qui sont des liens de communication et s'établissent entre les différents composants du système.
- Les dépendances qui permettent de spécifier qu'un élément dépend d'un autre.
- Les artefacts qui sont une manière de définir un fichier, un programme, une bibliothèque ou une base de données construite ou modifiée dans un projet.
- Les fragments qui ici permettent de définir un ensemble ou de préciser les éléments optionnels.

## V. Base de données

### 1. Modèle logique de données



Le MLD permet de représenter la structure de la base de données. On peut y retrouver, en plus des informations présentes sur le diagramme entités, un identifiant comme clé primaire pour chaque table ne possédant pas d'attribut pouvant jouer ce rôle. On y retrouve aussi des tables d'associations pour les relations plusieurs à plusieurs.

Certaines tables sont reliées par une relation non identifiée, c'est à dire que l'identifiant faisant référence à une autre table est une clé étrangère. D'autres sont reliées par une relation identifiée, c'est à dire que le champ qui fait référence à une autre table est la clé primaire de la table référençant. Les champs faisant références à une autre table sont représentés en rouge.

La clé primaire (ensemble de clé jaune et/ou rouge d'une table) permet d'identifier de manière unique chaque ligne de la table et les valeurs doivent, par convention, ne jamais être modifiées. Il ne peut y avoir qu'une clé primaire par table. Elle permet à un champ d'être unique, indexé et sa valeur ne peut pas être nulle. Chaque clé primaire rajoutée s'auto-incrémente lors de l'insertion d'une nouvelle ligne.

Les clés étrangères (losanges rouges) permet d'établir des liens entre plusieurs tables et garantissent que les valeurs de chaque ligne de la table référençant existent dans la table référencée. Les colonnes de la table référencée doivent faire partie d'une contrainte de clé primaire ou d'une contrainte d'unicité.

Une champ unique ne peut pas avoir deux fois la même valeur et il est également indexé.

Un champ indexé permet d'organiser ses valeurs lorsqu'ils sont stockées dans la base de données (donc plus gourmand en écriture et en mémoire). En contrepartie, cela permet à la base de données de retrouver plus rapidement une information lors de la lecture comme par exemple la recherche d'une personne par son nom et/ou prénom.

## 2. Administration de la base de données

Pour ce projet, le système de gestion de base de données était laissé libre au choix. La décision a été de s'orienter vers MySQL avec l'interface web phpMyAdmin pour sa gestion. Ce choix a été fait car il s'agit du SGBD open source le plus populaire.

En plus de la création de la base de données, il a fallut créer un utilisateur pouvant se connecter à distance et ayant des droits limités sur la base de données. Il a dans ce projet comme nom d'utilisateur et mot de passe "user". Ses droits se limitent sur la manipulation des données, c'est à dire au méthode CRUD (Create, Read, Update, Delete) qui en langage SQL correspond à Insert, Select, Update et Remove.

Créer un utilisateur pouvant se connecter à distance ne suffit pas pour que l'accès à la base de données soit accessible depuis le réseau puisque phpmyadmin est configuré par défaut en local. Pour résoudre cela, il faut faire une modification dans le fichier de configuration de phpmyadmin.

## VI. Client lourd du personnel

### 1. Diagramme de séquence de l'ajout d'un badge

Ce diagramme de séquence me permet de vous montrer chronologiquement les interactions entre le personnel de service, une personne, le lecteur et un badge dans le cas de la création d'un nouveau badge.

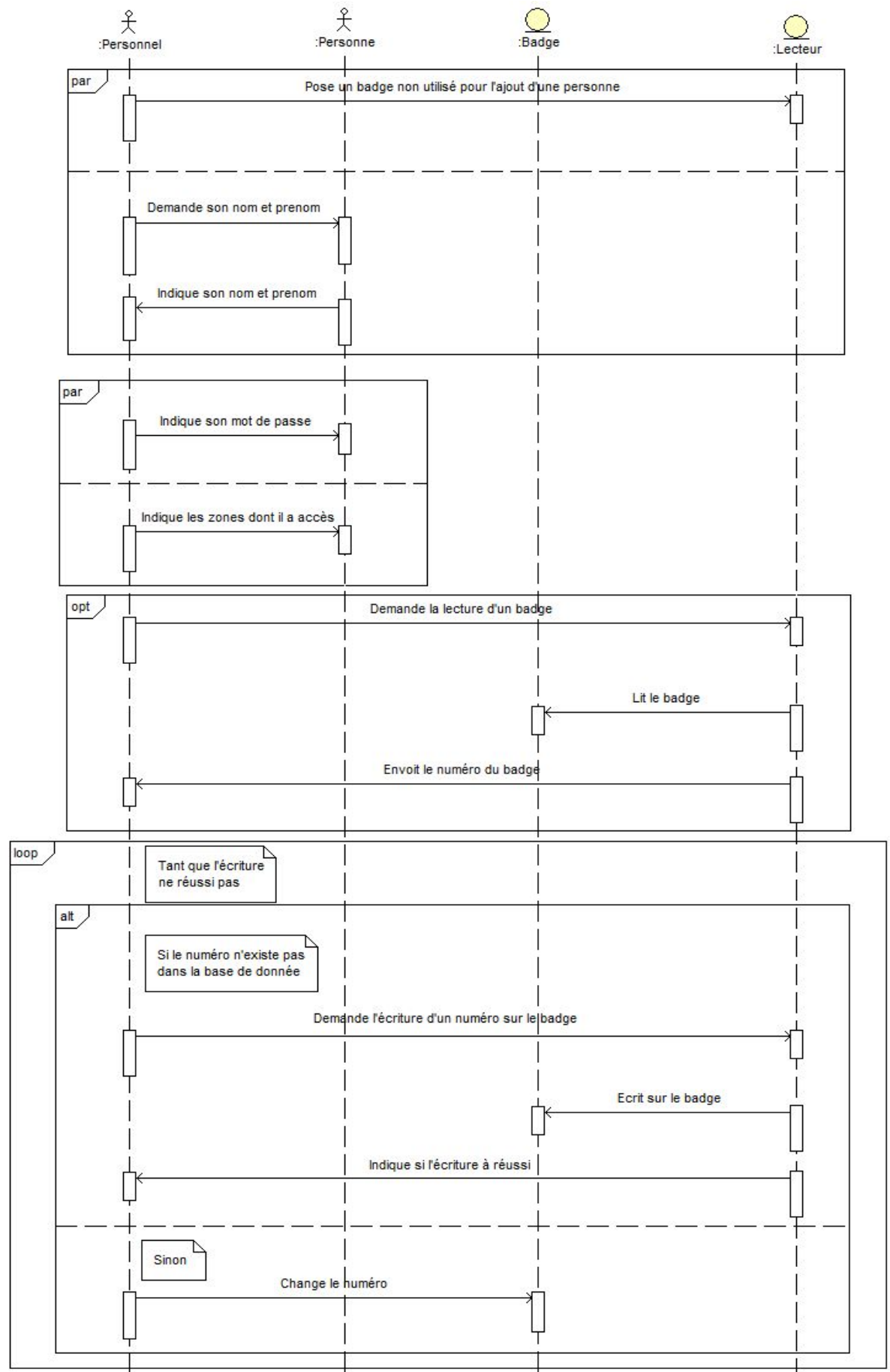
Le but lors de la création d'un nouveau badge est d'y attribuer un numéro unique qui permettra d'identifier une personne. Pour cela, on a besoin d'un lecteur RFID qui puisse écrire ce numéro sur une tag et d'un accès à la base de données pour y ajouter cette information.

Lors de la saisie des informations pour la création d'un badge, on va demander le nom et le prénom de la personne qui va posséder ce badge et on va lui indiquer les zones qu'il a accès ainsi que son mot de passe dans le cas d'un accès à une zone sensible.

Le numéro peut être choisi par le personnel de service mais il peut aussi reprendre le numéro qui est déjà inscrit sur le badge s'il y en a un.

Lorsque le personnel de service va valider la saisie des informations, deux vérifications devront être faites par rapport au badge. On va tout d'abord vérifier que la base de données ne possède pas ce numéro dans ses enregistrements. Si le numéro existe déjà, le personnel devra proposer un autre numéro. Sinon on va demander l'écriture sur le badge et l'on va vérifier si l'écriture a bien été effectuée. Si l'écriture échoue, cela signifie qu'il y a eu un problème matériel. Une des causes qui va revenir le plus souvent et certainement la moins grave est simplement l'oubli de mettre un badge sur le lecteur.







## 2. Structure de l'application

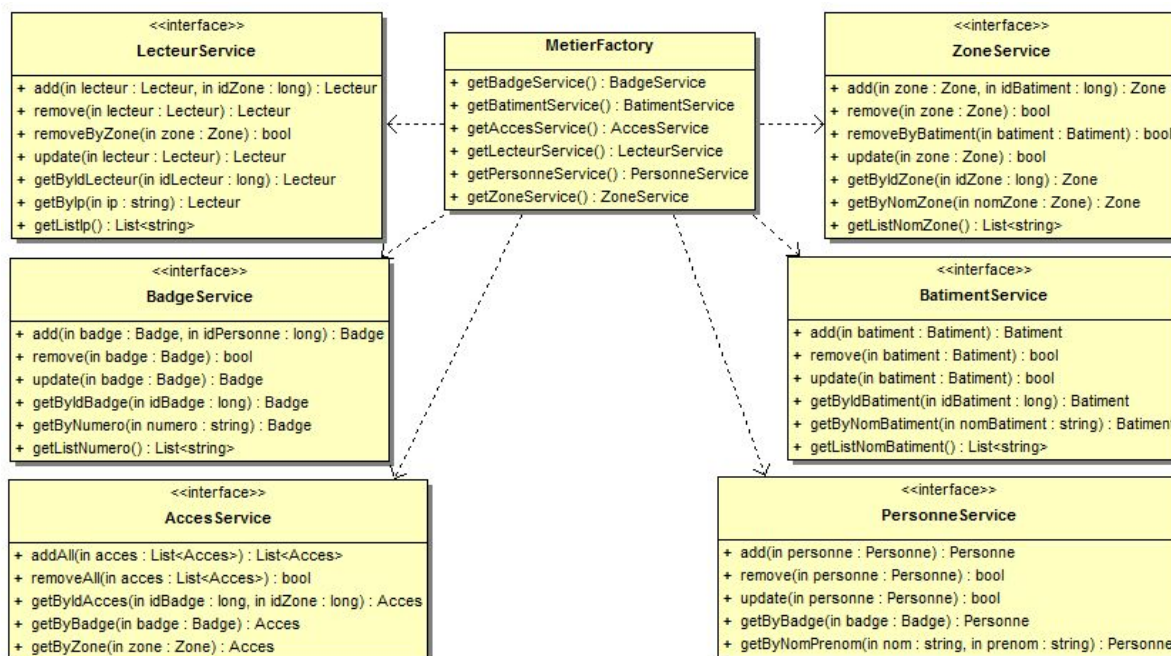
L'application pour le personnel de service à été faite en Java selon l'architecture trois tiers afin de rendre l'application la plus flexible possible en cas de besoin évolutif de la part de l'entreprise. Cette architecture permet de séparer l'application en trois couches:

- La couche présentation s'occupe de l'affichage de l'IHM et des interactions avec l'utilisateur.
- La couche métier correspond au traitement des données, c'est à dire à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative.
- La couche des données permet quant à elle l'accès au données qui sont dans notre cas stocké de façon définitive dans une base de donnée.

La couche métier contient les entités et les interfaces associées. Une interface ne possède aucun attribut mais que des méthode abstraites qui devront être implémentées. Une interface ne peut pas être instanciée, c'est à dire que l'on ne peut pas créer d'objet issus de cette classe, mais on peut créer des classes qui implémenteront le concept qu'on instanciera grâce à une Factory.

La Factory est un patron de conception utilisé en programmation orientée objet. Elle permet d'instancier des objets dont le type est dérivé d'un type abstrait (classe abstraite, interface). La classe exacte de l'objet n'est donc pas connue par l'appelant. De plus la Factory génère seulement des singleton, dont le but est de restreindre l'instanciation d'une classe à un seul objet et ainsi éviter les erreurs.

Les classes d'implémentation définissent les règles métiers, c'est à dire qu'elles contrôlent les informations qui y circulent.



Pour la couche des données, la structure utilisée est la même que pour la couche métier à l'exception des classes entités qui ne sont pas présentes et qu'ici les classes d'implémentation permettent d'exécuter des requêtes sur la base de données. Pour exécuter ces requêtes, chacune des implémentations est associée à la classe permettant l'accès à la base de données.

ServiceSql
- cnx : Connection
- dbHost : string
- dbPort : int
- dbName : string
- dbUser : string
- dbPasswd : string
- dbDriver : string
- dbUrl : string
+ connect() : Connection
+ disconnect() : void

On peut y retrouver plusieurs informations telles que le numéro ip sur laquelle se trouve la base de données, son numéro de port (par défaut 3306), le nom de la base de données, le nom d'utilisateur, son mot de passe et le driver utilisé pour la connexion. On y retrouve aussi deux méthodes qui permettront d'ouvrir la connexion à la base de données ou de la fermer.

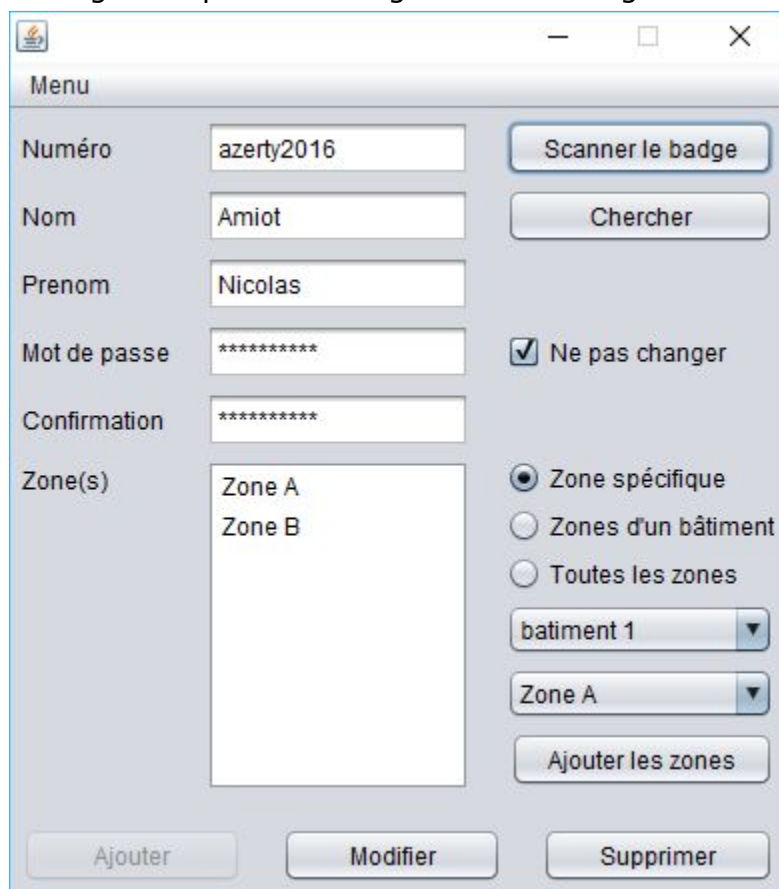
Les requêtes utilisées pour récupérer les informations au niveau de la base de données sont des requêtes préparées. Leurs avantages contrairement aux requêtes classiques sont, en premier, d'éviter les injections SQL puisqu'elles effectuent des vérifications portant sur les variables transmises avant d'exécuter la requête. Le deuxième avantage est qu'elles sont plus rapides pour les requêtes exécutées plusieurs fois dans le script puisqu'elles sont pré-compilées par le SGBD.

```
@Override
public List<Acces> addAll(List<Acces> acces) throws Exception {
    try {
        cnx = sql.connect();
        String query = "INSERT INTO acces (idBadge, idZone) VALUES (?, ?)";
        statement = cnx.prepareStatement(query);
        for (Acces a : acces) {
            statement.setLong(1, a.getIdBadge().getIdBadge());
            statement.setLong(2, a.getIdZone().getIdZone());
            statement.executeUpdate();
        }
        return acces;
    } finally {
        close();
    }
}
```

Sur cet exemple, on voit bien que la requête n'est préparée qu'une seule fois et que pour chaque accès autorisé pour une personne, on ne fait que changer la valeur des paramètres avant chaque exécution.

### 3. L'interface Homme-Machine

L'IHM dédié au personnel du service faite à partir de la bibliothèque graphique Swing comporte 3 grandes parties que l'on peut choisir à travers la barre de menu. La première grande partie est la gestion des badges RFID.



Menu

Numéro: azerty2016 [Scanner le badge]

Nom: Amiot [Chercher]

Prenom: Nicolas

Mot de passe: \*\*\*\*\* ☒ Ne pas changer

Confirmation: \*\*\*\*\*

Zone(s):  
Zone A  
Zone B

☒ Zone spécifique  
☐ Zones d'un bâtiment  
☐ Toutes les zones

batiment 1 [v]  
Zone A [v]  
[Ajouter les zones]

[Ajouter] [Modifier] [Supprimer]

On retrouve sur cette partie les principales informations pour permettre l'accès à une zone à partir d'un badge. C'est à dire qu'on retrouve le numéro unique du badge, le nom et prénom de la personne possédant ce badge, son mot de passe encodé ensuite en MD5 en cas d'une demande d'accès à une zone sensible et la liste des zones dont la personne à accès.

Le bouton scanner le badge permet de lire le badge et permettre selon si le badge est déjà enregistré dans la base de données ou non, d'ajouter ou de modifier/supprimer un badge. Si le badge existe, les champs de texte sont pré-remplis avec les informations correspondantes.

Le bouton chercher permet quant à lui de rechercher une personne dans la base de donnée selon son nom est prénom. Si cette personne existe, les champs seront aussi pré-remplis et le bouton modifier et supprimer seront disponibles, sinon seul le bouton ajouter le sera.

Dans le cas d'une recherche qui a été fructueuse, une checkbox "ne pas changer" apparaîtra à côté du mot de passe. Elle permet de demander si le mot

de passe doit être changé ou non lors de la modification. Cette option est présente pour des raisons de sécurité puisque le champ n'est pas directement lisible par le client mais surtout pour éviter d'encoder à nouveau le mot de passe récupéré depuis la base de donnée qui lui est déjà encodé.

Pour l'ajout des zones dont la personne aura accès, on a le choix entre trois options. La première est l'ajout d'une zone spécifique d'un bâtiment. La deuxième est l'ajout de toutes les zone se trouvant dans le bâtiment choisi. La dernière est l'ajout de toutes les zones de chaque bâtiment. Une fois les zones ajoutées, l'utilisateur peut supprimé une par une les zones présentent dans la liste si jamais il s'est trompé ou simplement s'il veut par exemple donné l'accès à toutes les zones d'un bâtiment excepté une. Pour faire cela il lui suffit de faire un double clic, dans la liste, sur la zone qu'il veut supprimée.

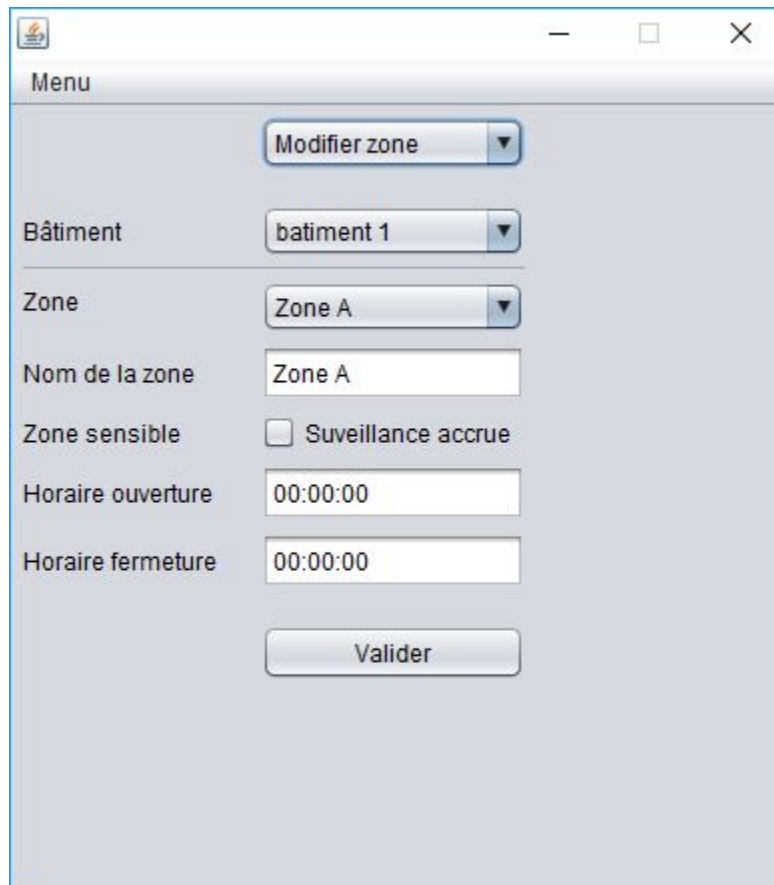
La deuxième grande partie de l'IHM est la visualisation de l'historique.

Personne	Zone	Date
Amiot Nicolas	Zone A	01/01/2016 10:15:38
Amiot Nicolas	Zone A	01/01/2016 15:01:53
Amiot Nicolas	Zone B	01/01/2016 17:48:13

On peut choisir si l'on veut visualiser l'historique selon une zone ou selon un badge. Une liste déroulante présentera alors toutes les zones ou badges présents dans la base de données. Dans le cas d'une visualisation selon un badge, un bouton scanner le badge apparaîtra pour permettre de lire le numéro du badge que l'on souhaite sans devoir le rechercher à la main ce qui est plutôt fastidieux dans le cas d'un numéro.

Une fois le choix fait, la liste présentera pour chaque accès lié à la zone ou au badge le nom et prénom de la personne, le nom de la zone et la date de passage.

La dernière grande partie de l'IHM est la gestion de l'agencement, c'est à dire pouvoir ajouter, supprimer ou modifier des bâtiments, zones et lecteurs.



Comme pour le badge, si l'on souhaite modifier un élément, les champs seront pré-remplis. Chaque élément est classé selon une nomenclature.

Les champs horaire ouverture et fermeture possèdent un masque obligeant la personne à entrer seulement des chiffres. Une fonction ensuite vérifiera si l'horaire donné correspond bien à un moment d'une journée. Si jamais l'horaire d'ouverture est la même que celle de fermeture, cela signifie que la zone est disponible à n'importe quel moment de la journée.

Un autre masque a été utilisé pour gérer l'agencement, il est associé à l'adresse ip pour les lecteurs afin toujours d'y simplifier la saisie.

```
MaskFormatter maskIp = new MaskFormatter("###.###.###.###");  
maskIp.setPlaceholderCharacter('0');  
maskIp.install(jFormattedTextFieldPanelAgencementIpLecteur);
```

Comme vous pouvez le voir ci-dessus, la création d'un masque pour un champ de texte se fait grâce à l'objet MaskFormatter. Dans son constructeur, on va indiquer le format souhaité. Ici le croisillon '#' appelé souvent dièse '#' par abus



de langage permet de signifier que seulement les chiffres sont valides et dans le cas contraire, le caractère ne sera pas ajouté au champ. Les points n'ont aucune signification particulière, ils apparaîtront là où ils ont été placés et seront ignorés lors de la saisie d'un caractère dans le champ de texte.

On pourra donc se retrouver avec des chaînes de caractère tel que "127.000.000.001". Cette chaîne de caractères ne correspond néanmoins pas à une véritable adresse ip. Il faut pour cela supprimer les zéros en trop.

```
private String formattedTextToIp(String ft) {
    String[] tft = ft.split("\\.");
    int ip1 = Integer.parseInt(tft[0]);
    int ip2 = Integer.parseInt(tft[1]);
    int ip3 = Integer.parseInt(tft[2]);
    int ip4 = Integer.parseInt(tft[3]);
    if (ip1 < 256 && ip2 < 256 && ip3 < 256 && ip4 < 256) {
        return ip1 + "." + ip2 + "." + ip3 + "." + ip4;
    } else {
        return "";
    }
}
```

On va récupérer dans un tableau chacune des chaînes découpées par le séparateur point. Le point ayant une signification pour la méthode split, il faut échapper ce caractère à l'aide de deux slash. On va ensuite mettre en entier nos chaînes de caractères ce qui aura pour effet de supprimer les zéros en trop. Ensuite on vérifie que l'adresse ip est valide et si c'est le cas, on retourne sa valeur.

Cette fonction permet de retourner une véritable adresse ip mais il est aussi essentielle pour pouvoir afficher une adresse ip enregistré dans la base de données dans ce champ possédant un masque.

```
private String ipToFormattedText(String ip) {
    String[] tip = ip.split("\\.");
    for (int i = 0; i < 4; i++) {
        while (tip[i].length() < 3) {
            tip[i] = "0" + tip[i];
        }
    }
    return tip[0] + "." + tip[1] + "." + tip[2] + "." + tip[3];
}
```

Cette fonction comme pour la précédente récupère un tableau de chaîne de caractère selon le séparateur point et pour chaque chaîne de caractère, on ajoute des zéros supplémentaires devant la chaîne initiale tant qu'elle ne se sera pas composée de 3 caractères.



au GPIO numéro 2 (pin 13). Les deux leds sont protégées par des résistances de 330 ohms.

En réalité le câblage des leds est effectué sur une Breakout Board pour simplifier la réalisation des soudures entre les résistances et les anodes des leds.

## 2. Utilisation des leds

La carte Raspberry Pi donne accès à des entrées et sorties numériques appelées GPIO (en anglais "general purpose input & output") contrôlées par le processeur ARM. L'utilisation des GPIO est faite en Java, pour cela nous avons utilisé le projet PI4J. PI4J est un projet développé par la société Raspberry pi Foundation qui a pour but de donner l'accès au développeur à une API Java orientée objet permettant de gérer les GPIO.

Dans le programme Java chaque GPIO utilisé sera instancié et grâce à des méthodes incluses dans l'api, il est possible de faire passer les GPIO d'un état à un autre.

```
51     final GpioController gpio = GpioFactory.getInstance();
52
53     // provision gpio pin #01 as an output pin and turn on
54     final GpioPinDigitalOutput pin = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "MyLED", PinState.HIGH);
55
56     // set shutdown state for this pin
57     pin.setShutdownOptions(true, PinState.LOW);
58
59     System.out.println("--> GPIO state should be: ON");
60
61     Thread.sleep(5000);
62
63     pin.low();
64     System.out.println("--> GPIO state should be: OFF");
```

Sur l'image ci dessus, le GPIO numéro 1 est instancié à la ligne numéro 54 et mis à l'état "High" et à la ligne 63 ce même GPIO est mis à l'état "low".

Pour un souci de simplicité d'utilisation du programme java, j'ai décidé de créer deux méthodes, une méthode "AllumerLedVerte" et une autre "AllumerLedRouge". Elles serviront chacune à allumer la led pendant 2 secondes et donc de passer le GPIO concerné à l'état "High" pendant 2 secondes puis à l'état "low".



## VIII. Caméras de surveillance

### 1. Mise en place

Notre projet contiendra des salles sous haute surveillance, ces salles seront équipées d'une ou plusieurs caméras de surveillance, ces caméras seront visionnables par le vigile.

Le vigile pourra les visionner depuis n'importe quel poste situé dans le même réseau que la caméra. Cette dernière est installée sur le Raspberry pi, il s'agit d'une pi-camera qui est un module caméra pour raspberry-pi qui utilise le connecteur CSI (Camera Serial Interface) pour fonctionner.



### 2. Le Client/Serveur

Malheureusement la caméra commandé pour notre projet ne fonctionnait pas, lors du lancement de la commande `"raspistill -o cam.jpg"` permettant d'enregistrer une photo l'erreur suivante apparaît : `"mmal: Received unexpected camera control callback event, 0x4f525245"`, nous avons donc contacté le service après vente pour un remplacement de la caméra.

Cette caméra est utilisable via deux méthodes, des commandes bash ou par programmation python grâce à l'API picamera. Pour remplir notre objectif de pouvoir visionner l'application depuis n'importe quel poste dans le même réseau que la caméra nous avons décidé de créer un Client/Serveur avec un client python qui envoie le flux d'image de la caméra vers un serveur Java qui ouvre le flux d'image reçu dans une interface graphique réalisée à partir de la bibliothèque graphique Swing. Ce serveur java sera Multi-Threadé afin de permettre le visionnage de plusieurs caméras simultanément. Pour parer à l'absence d'un caméra fonctionnelle, le flux d'image était constitué pendant la

phase de développement de l'application d'une image choisi au hasard sur le site "Finda.photo".

L'API swing comporte deux fonctionnalités, la visualisation du flux d'image et la possibilité de prendre une capture d'écran qui par la suite est enregistrée sur le poste ou l'API est exécutée. Ci-dessous l'API utilisée par un vigile.



Côté client, c'est-à-dire au niveau du Raspberry où est câblé la caméra, le programme est composé d'un fichier python qui crée une connexion avec le serveur Java grâce à une socket qui prend en paramètre un port (le port 8080) et l'adresse IP du serveur. Le fichier est ensuite ouvert et envoyé au serveur grâce à la socket.

Côté serveur, c'est-à-dire au niveau du poste du vigile, à l'exécution le programme attend une connexion sur le port 8080 en créant une socket serveur. Une fois la connexion établie, le serveur crée un nouveau thread qui appelle la classe "Transfert", cette classe crée un buffer et le remplit tant que le thread reçoit quelque chose de la part du client. Ensuite le thread crée un nouveau fichier et appelle la classe "IG" qui prend en paramètre le chemin du fichier créé. La classe "IG" utilise la bibliothèque graphique Swing pour créer une JFrame composée de l'image et d'un bouton permettant de prendre une capture d'écran.

Lors de l'appui sur le bouton "Prendre un ScreenShot", la classe "IG" crée un nouveau fichier .jpg contenant une capture d'écran de l'api. Ce fichier est sauvegardé sur le poste du vigile dans le dossier où est situé l'API et un message d'information apparaît. Ci-dessous l'API après l'appui sur le bouton.

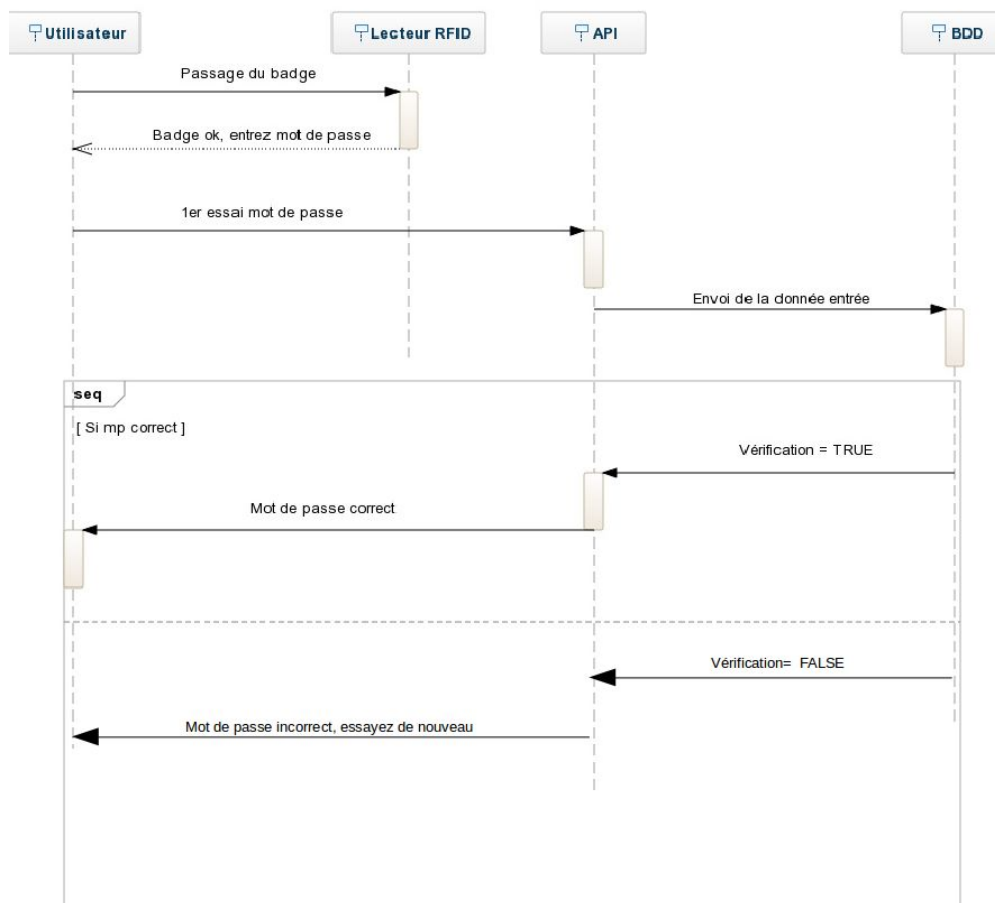


Ci-dessous le fichier contenant la capture d'écran.



## IX. Autorisation d'accès par mot de passe

### 1. Diagramme de séquence d'une tentative d'accès



Ci-dessus, un diagramme de séquence représentant une tentative d'accès à une zone sous haute surveillance. Dans ce diagramme nous partons du principe que le badge est reconnu par le lecteur RFID. Une fois le badge reconnu l'utilisateur essaye une première fois d'entrer son mot de passe grâce à l'interface graphique de l'application prévue à cette effet. L'application contacte ensuite la base de données du système pour tester si le mot de passe entré correspond au mot de passe du badge présent dans la base de données.

Si le mot de passe est bon, l'application affiche le message suivant "Mot de passe correct" et l'utilisateur a accès à la salle. Si non l'application affiche "Mot de passe incorrect, essayez de nouveau" et tant que l'utilisateur n'entre pas le bon mot de passe, l'utilisateur n'a pas accès à la salle.

## 2. L'interface Homme-Machine

Dans les zones sous haute surveillance, l'utilisateur en plus du passage de son badge devant le capteur RFID doit entrer un mot de passe qui correspond à ce badge. Ce mot de passe est constitué uniquement de chiffres. Pour qu'il puisse entrer ce mot de passe nous avons décidé de créer une application à partir d'une interface homme machine utilisant la bibliothèque graphique Swing. Cette interface est constituée de onze boutons (chiffres de 0 à 9 et un bouton valider), d'un JTextfield permettant l'affichage du mot de passe entré par l'utilisateur et d'un JLabel permettant d'informer l'utilisateur du résultat de son appui sur le bouton "Valider". Après l'appui sur le bouton valider, l'application va consulter la base de données du système afin de comparer le mot de passe entré par l'utilisateur et celui enregistré sur la base de données correspondant au badge précédemment passé devant le lecteur RFID.

Coté programmation, le code de l'interface se trouve dans la classe "SwingPasswd" qui hérite de la classe "JFrame" et implémente la classe "ActionListener". Cette classe est constituée d'un "setter" et d'un "getter" qui permettent d'accéder à la variable "passwdValider", variable qui contient le mot de passe entré par l'utilisateur une fois l'appui sur le bouton valider effectué. Il y a également une méthode "Verif" prenant un booléen en paramètre qui affiche le message "Mot de passe correct" si le booléen est égale à "true" et "Mot de passe incorrect" si le booléen est égale à "false" et une méthode "ActionPerform" prenant en paramètre un "ActionEvent" qui permet d'attribuer des actions au bouton après un appui. La JFrame a une taille de 210 pixels par 200 pixels ce qui correspond à la taille de l'écran tactile. La disposition des différents éléments dans cette fenêtre est réalisée grâce à trois JPanel respectivement positionnés en Nord, Centre et Sud. Le premier JPanel contient le JLabel et le JTextfield, le second contient les neuf chiffres et le dernier contient le bouton "Valider".

La connections à la base de données est réalisée dans la classe "ServiceSql" qui utilise le driver "com.mysql.jdbc.driver". Cette classe permet la connexion et la déconnexion à la base de données grâce à l'appelle des méthodes "connect()" et "disconnect()".

Ci-dessous une image représentant l'interface graphique :





## X. Gestion du lecteur RFID

### 1. Mise en place et analyse

Afin de dialoguer entre le lecteur et la Raspberry Pi ou l'ordinateur, nous devons utiliser des trames UART.

Les trames qui sont envoyées au lecteur doivent avoir ce format :

Module address	Frame length	Command	Parameters 1..n	CRCH	CRCL
1 byte	1 byte	1 byte	n bytes	1 byte	1 byte

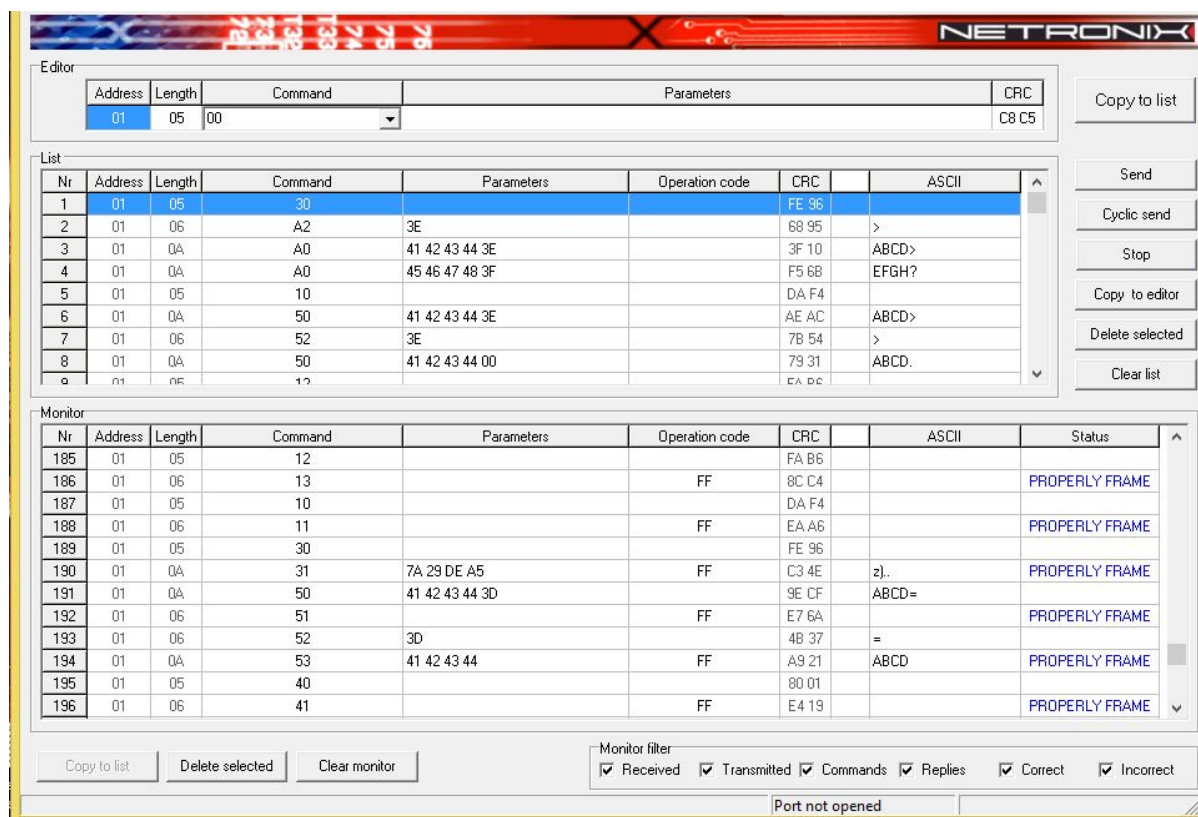
L'adresse du module correspond à l'adresse unique du lecteur. Ensuite, « Frame length » représente le nombre d'octets de la trame, puis il y a un numéro pour chaque commande existante. C'est possible selon certaines commandes comme l'écriture et la lecture d'ajouter certains paramètres (par exemple ce qui doit être écrit sur un badge). A la fin de la trame, CRCH et CRCL correspondent au CRC : Contrôle de redondance cyclique qui est un outil logiciel permettant de détecter les erreurs de transmission ou de transfert par ajout, combinaison et comparaison de données redondantes.

Les trames que renvoie le lecteur sont de cette forme :

Module address	Frame length	Response	Parameters 1..n	Operation code	CRC H	CRC L
1 byte	1 byte	1 byte	n bytes	1 byte	1 byte	1 byte

Elles ont quasiment les mêmes propriétés que celles ci-dessus mise à part que « Response » remplace « Command ». C'est un numéro qui identifie la trame qui est égale au numéro de commande incrémenté de un. Il y a également un code de l'opération qui s'ajoute permettant de connaître si l'envoi de la trame a réussi ou non.

Le logiciel FRAMER de Netronix a permis de simuler l'envoi et la réception de trame.



Les commandes présentes dans la documentation du module H1M05 sont insérées dans l'éditeur afin d'être envoyés au lecteur RFID. Le logiciel affiche dans le moniteur la trame envoyée et la trame réponse du lecteur comme ci-dessus.

Avec les tests réalisés grâce au logiciel FRAMER, une procédure a été établie. Pour interagir avec un badge RFID, l'antenne électromagnétique doit d'abord être allumée (commande 10) puis la carte en contact avec le lecteur doit être sélectionnée (commande 30). Ensuite, il y a possibilité d'écrire sur le badge (commande 50) et/ou lire le badge (commande 52). Si un autre badge RFID doit être utilisé, la commande 40 qui permet de mettre en veille le badge utilisé jusqu'ici doit être envoyée. Une fois que l'utilisation des badges est terminée, l'antenne doit être éteinte (commande 12).

Chaque trame réponse a un code d'opération qui prend comme valeur FF si l'opération réussit, dans le cas contraire l'opération a échoué.

## 2. Gestion du lecteur en JAVA

Pour interagir avec le lecteur RFID via le câble RS32, nous utilisons l'API de communication JAVA RXTX.

### 1) Mise en lien avec le lecteur

Une fonction Connect() prenant comment paramètre le nom du port utilisé pour faire le lien avec le lecteur a été créée. Celle-ci vérifie si le port est bien disponible :

```
CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(portName);
```

Si c'est le cas, alors elle se connecte à ce port et l'initialise.

```
CommPort commPort = portIdentifier.open(this.getClass().getName(), 2000);
```

```
if (commPort instanceof SerialPort)
{
    //si le port est présent mais pas connecté
    SerialPort serialPort = (SerialPort) commPort;
    serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);

    is = serialPort.getInputStream();
    os = serialPort.getOutputStream();
}
```

La méthode setSerialPortParams() permet de paramétrer le port comme indiqué dans la documentation.

Ensuite, on initialise les InputStream et OutputStream qui permettront d'envoyer et lire les trames.



## 2) Calcul du CRC

La documentation fournit une fonction permettant de calculer le CRC en C. La voici retranscrit en java :

```
public static String CalculCRC(String t)
{
    byte [] trame = hexStringToByteArray(t);
    int Many = trame.length;
    String CRC;
    int i,NrBajtu;
    short C, crc = 0;
    for (NrBajtu=0;NrBajtu<Many;NrBajtu++)
    {
        C=(short) (((crc>>8)^trame[NrBajtu])<<8);
        for (i=0;i<8;i++)
        {
            if (C < 0) C=(short) ((C<<1)^0x1021);
            //if ((C & 0x8000) == 0x8000) C=(short) ((C<<1)^0x1021);
            else C=(short) (C<<1);
            // la condition (C < 0) est l'équivalent de ((C & 0x8000) == 0x8000)
            // <=> C est de type short
        }
        crc=(short) (C^(crc<<8));
        //System.out.format("%d:%x\n", NrBajtu, crc); //pour debug
    }
    CRC=String.format("%04x", crc);
    return CRC;
}
```

Elle prend pour paramètre une chaîne de caractère correspondant au début de la trame à partir de laquelle on doit calculer le crc.

Premièrement, cette chaîne est convertit en tableau de d'octet sur lequel on va garder en mémoire sa taille.

Puis, une première boucle allant de 0 à la taille de ce tableau afin de prendre octet par octet est mise en place. Dans celle-ci, on stocke dans une variable C le résultat d'un ou exclusif entre l'octet de la trame à l'indice NrBajtu et ce qui sera le résultat crc décalé de 8 vers la droite. Le tout, décalé de 8 vers la gauche.

Ensuite, une deuxième boucle est mise en place où C est décalé à chaque fois de 1 vers la gauche. Seulement quand C est égale à 0x8000, alors un ou exclusif avec 0x1021 est fait sur C.

Puis le crc prend la valeur du résultat de lui-même décalé de 8 et du ou exclusif avec C.

Pour faire ces calculs, C et crc devaient être de type short. Par choix, la fonction renvoie le crc en chaîne de caractère.

## 3) Écriture et lecture d'un badge RFID

A partir du logiciel de framer, nous avons établi une procédure permettant d'écrire et lire des badges. Dans ce sens, les fonctions AllumerAntenne(), EteindreAntenne(), SelectionnerBadge(), MiseEnVeilleBadge(), EcritureBadge() et LectureBadge().

Nous allons seulement détailler LectureBadge() et EcrireBadge() car la méthode de coder est la même pour toutes les fonctions. Une chaîne de caractère correspondant au début de la trame est créée à laquelle on ajoute le CRC calculé préalablement avant d'être converti en tableau d'octet grâce à la fonction `hexStringToByteArray(String trame)`. Ensuite, écrit la trame au lecteur grâce à la méthode `Write(byte[] trameConvertie)` de `OutputStream` et on lit la trame réponse avec la méthode `Read(byte[] trameRecue)` de `InputStream` afin de tester si l'opération s'est déroulée correctement.

`EcritureBadge()` et `LectureBadge()` sont plus complexes car des paramètres entrent en jeu.

Avant de pouvoir utiliser ces fonctions, les méthodes `Connect()` et `AllumerAntenne()` doivent être appelées.

`EcritureBadge()` prend deux paramètres : `aEcrire` et `zoneLecteur` qui sont des chaînes de caractère correspondant à ce qui doit être écrit et la zone du badge où l'on doit écrire. `aEcrire` ne doit pas dépasser 4 octets, `zoneLecteur` 1 octet. Dans cette fonction on commence par appeler la fonction `SelectionnerBadge()`. Si l'opération réussit alors on forme une chaîne de caractères composée du début de la trame permettant d'écrire, les chaînes `aEcrire` et `zoneLecteur` et le CRC comme ceci :

```
String t = "010A50"+aEcrire+zoneLecteur; //Trame permettant d'écrire sur le badge
//ajout du CRC à la trame
String crc = CalculCRC(t);
t+=crc;
```

Ensuite, cette chaîne est convertie en tableau d'octets et on l'envoie au lecteur.

```
byte[] trame = hexStringToByteArray(t);

//Ecriture de la trame
try {
    os.write(trame);
} catch (IOException ex) {
    Logger.getLogger(LecteurRFID.class.getName()).log(Level.SEVERE, null, ex);
}
```

Puis, on lit la trame réponse :

```
//Lecture de la trame réponse
byte[] trameRecue = new byte[15];
int nbByteT = 0, nbByteAtt = 6, nbByte;
while (nbByteT < nbByteAtt)
{
    try {
        nbByte = is.read(trameRecue, nbByteT, nbByteAtt - nbByteT);
        //System.out.println("Taille:" + nbByte);
        nbByteT += nbByte;
    } catch (IOException ex) {
        Logger.getLogger(LecteurRFID.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Et enfin on vérifie que tout s'est déroulé correctement. On regarde si le code de l'opération est égal à - 1 soit FF en hexadécimal. La fonction renvoi vrai si elle réussit, faux sinon.

```
if(trameRecue[3]==-1)
{
    reussi = true;
}
```

LectureBadge() ne prend qu'un paramètre : zoneLecteur

Même processus que pour EcritureBadge(), on crée une chaîne de caractère avec le début de la trame, la zone du lecteur et le CRC qu'on convertit en tableau d'octets qu'on envoie au lecteur.

On lit la trame réponse mais cette fois on stocke le contenu de la carte dans une variable de type String nommé idBadge comme ceci :

```
//Si le code de l'opération est égale à FF soit -1 en byte : l'opération a réussi
if(trameRecue[7]==-1)
{
    byte[] id = new byte[]{trameRecue[3],trameRecue[4],trameRecue[5],trameRecue[6]};
    idBadge = getHexString(id);
}
```

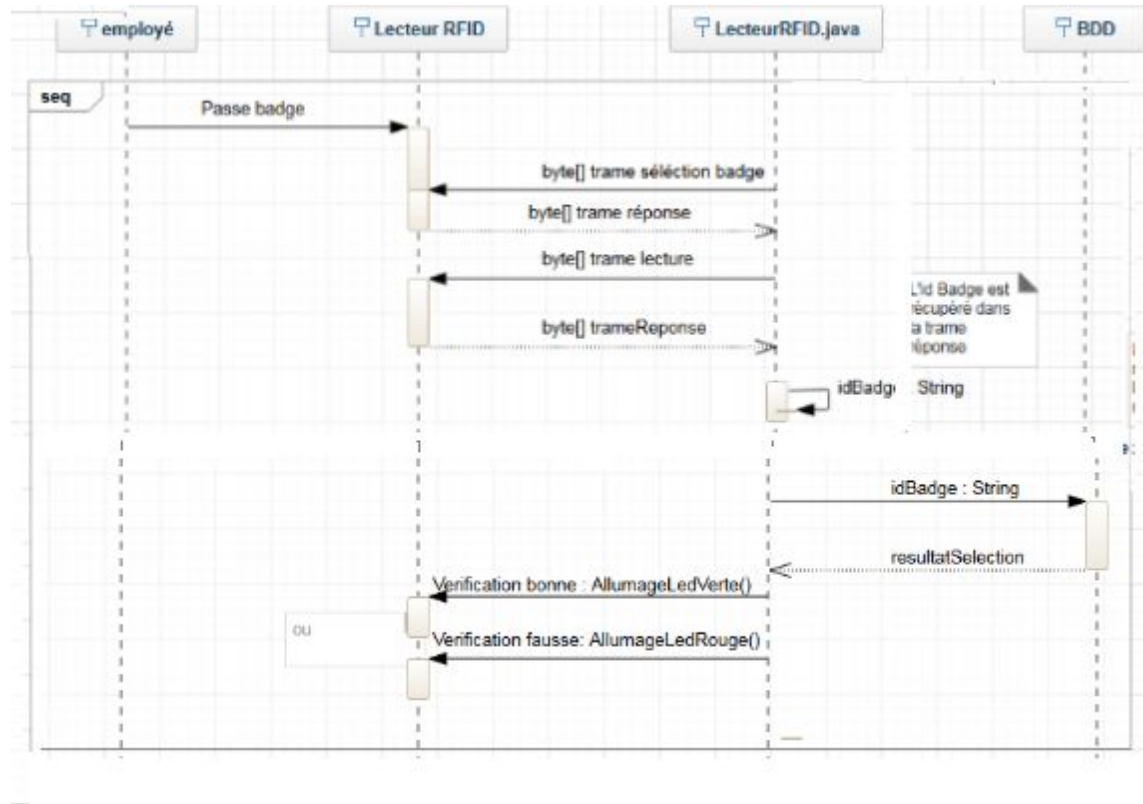
La fonction getHexString(byte[] b) permet de convertir un tableau de byte en chaîne hexadécimal.

La fonction LectureBadge() renvoie l'id du badge ou une chaîne de caractère vide si l'opération a échoué.

Enfin, à la fin de ces 2 fonctions, on appelle la fonctions MiseEnVeilleBadge() pour pouvoir écrire ou lire sur d'autres badges.

L'antenne devra être éteinte si aucune manipulation avec les bagdes n'est pas prévu : appel de la méthode EteindreAntenne().

### 3. Diagramme de séquence lecture badge zone non sensible



Le diagramme de séquence ci-dessus représente l'événement où un employé essaye d'accéder dans une zone sensible.

On suppose ici que l'antenne électromagnétique est allumée et que le lecteur RFID est bien initialisé.

La méthode LireBadge() est appelé jusqu'à qu'un employé passe un badge, une fois que l'employé passe son badge, la trame de sélection de badge est envoyé au lecteur qui lui renvoie la trame réponse, une fois fait le programme envoie la trame permettant de lire ce badge et récupère la trame réponse. Celle-ci contient l'identifiant du badge récupéré et convertit en chaîne de caractère. Cet identifiant est comparé avec ceux de la base de données. Si celui-ci est juste, une led verte s'allume et l'employé est autorisé à rentrer, sinon la led s'allume en rouge et l'employé ne peut pas passer.

## X. Conclusion

Ce projet fût intéressant car il manié non seulement le côté logiciel, mais aussi le coté matériel. On s'est vite aperçu que travailler avec du matériel devenait tout de suite plus complexe et nécessitait de passer un bon moment à lire les différentes documentations pour comprendre comment il fonctionnait et donc comment dialoguer avec lui.

Malgré les difficultés rencontrées notamment sur la mise en commun des différentes applications et la PI-caméra défectueuse, nous avons vraiment apprécié développer un système de contrôle d'accès car comme nous le savons, la sécurité est un élément important de nos jours pour les entreprises.

Certaines améliorations ont été apportées au projet comme la sécurisation du mot de passe lié au badge pour les zones sensibles à l'aide d'un cryptage en MD5.

Nous garderons donc un très bon souvenir de ce projet que nous avons principalement pu développer en langage Java qui est pour nous le langage informatique que nous apprécions le plus.