

Projet

E-STOCK

Dossier technique
version 0.1



Trachat Joffrey
Legger Pierre-Antoine

BTS SN-IR - LaSalle Avignon

Sommaire

Présentation Général du projet	4
Expression du besoin	4
Présentation du projet	5
Diagramme de déploiement de l'armoire	6
Diagramme de classes du projet	8
Choix du type de liaison série	9
Répartition des tâches	11
Objectifs attendus	12
Partie personnelle : Legger Pierre-Antoine	13
Rappel du besoin initial	13
Le lecteur OMNIKEY	13
Organisation	14
Objectifs	14
Gantt	15
Outils de développement	16
Diagramme de cas d'utilisation pour l'acteur utilisateur	17
Le protocole de communication	18
Maquette IHM	19
Scénarios	22
Scénario Authentification	22
Scénario Authentification par badge	24
Scénario Authentification par identifiant	29
Scénario Recherche article	33
Scénario Consulter le stock	37
Tests de validation	43
Rechercher un article	43
Consulter le stock	43
Authentification par identifiant	44
Authentification par badge RFID	44
Partie Personnelle : Tranchat Joffrey	45
Objectifs	45
Ressource logicielles et matérielles	45
Les logiciels:	45
Planification	46
Répartition des tâches	46
Gantt	47
Diagramme de cas d'utilisation	48
Protocole de communication	49

Format des trames	49
Trame de requête Raspberry Pi → SE	49
Trame de réponse SE → Raspberry Pi :	49
Liaison Série	50
Base de données	53
Structure de la base de données e-stock	53
Exemple de création et de remplissage d'une table	55
Utiliser la base de données dans le programme	56
Mettre à jour le stock a partir d'une trame de poids	59
Scénario Mettre à jour le stock	59
Exemple de requête SQL	61
Récupérer les données d'un article dans une armoire	61
Mettre à jour le stock dans une armoire	62
Le comptage automatique	63
Test de mise à jour du stock	64
Mise en place du lecteur code-barres	65
Scénario lecteur code-barres	65
Interface	68
Gestion du lecteur code-barres	69
Test scanner objet	72
Annexes	73
Manuel d'installation de Qt pour Raspberry Pi	73

Présentation Général du projet

Expression du besoin

Les enseignants du Lycée technique et professionnel interviennent dans des ateliers dans lesquels de nombreux équipements sont utilisés. Ils souhaitent pouvoir disposer d'armoires communicantes afin :

- De rendre accessible le matériel dans un espace sécurisé
- De faciliter un inventaire des stocks avant de passer une commande
- D'assurer un suivi des activités (Qui a effectué l'activité ? Quand ?)
- De rendre plus autonome et de responsabiliser un groupe d'élève lors D'une activité
- De se libérer de la gestion et du rangement

Les armoires ne seront pas utilisées uniquement pour du stockage de matériel mais aussi comme une ressource pédagogique.

Le développement de l'application doit répondre aux exigences des utilisateurs :

- Simplicité d'utilisation
- Correspondre aux contraintes définies,
- Réalisable dans un délai de 200 heures (IR) et 170 heures (EC).

Contrôler, gérer, assurer la traçabilité, et sécuriser l'accès d'un système de gestion de stocks automatisé.



Présentation du projet

Il s'agit de réaliser un système de gestion de stock automatisé qui permettra :

- De contrôler et gérer l'utilisation de produits stockés dans une armoire sensible
- D'assurer la traçabilité de l'attribution du matériel et des consommables stockés
- De sécuriser l'accès par un contrôle d'accès par badge RFID

Une armoire sera composée de 8 casiers maximum. Chaque casier pourra être équipé :

- D'une gâche électrique afin d'assurer son ouverture/fermeture
- D'une balance pour assurer le comptage automatique des articles

Le comptage automatique de la quantité est déterminé en fonction du poids unitaire et du poids mesuré sur la balance.

Si les casiers ne sont pas munis individuellement :

- De gâche électrique, seule l'armoire en disposera pour accéder à l'ensemble des rangements
- De balance, le comptage des articles se fera manuellement en indiquant la quantité des articles. Un lecteur code-barres pourra être utilisé pour identifier les articles.

Un lecteur de badge RFID est intégré à chaque armoire pour contrôler l'accès. L'exploitation de l'armoire e-stock est possible à partir de l'écran tactile intégré.

Architecture du système

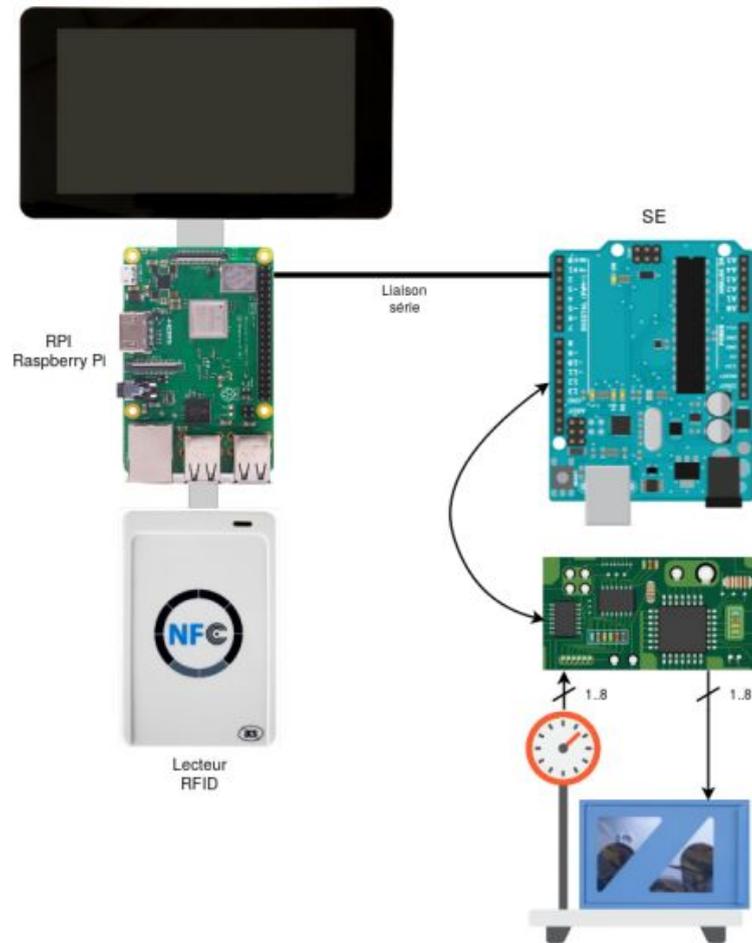
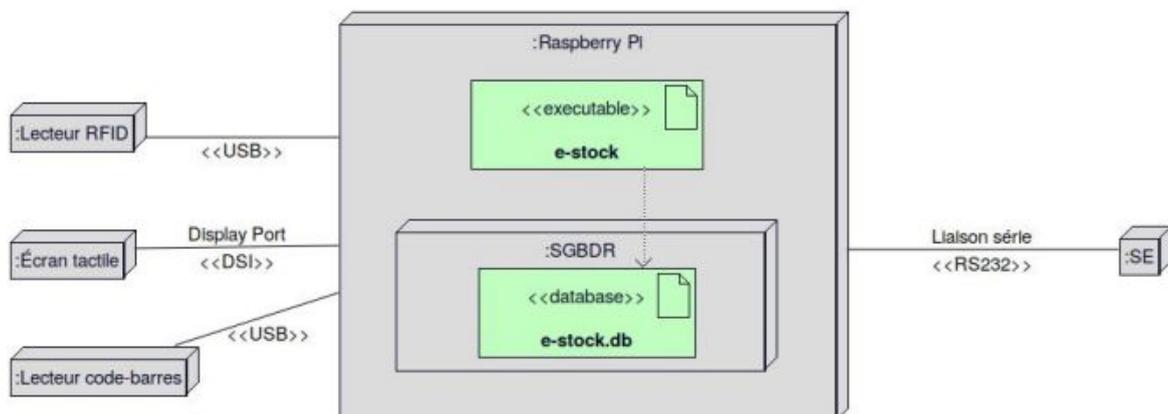


Diagramme de déploiement de l'armoire



Une armoire e-stock est architecturée autour :

- d'un Raspberry Pi pour la gestion du stock
- d'un ESP32 pour le comptage automatique et l'accès au casier

La base de données MySQL sera intégrée au Raspberry Pi.

Un écran tactile sera associé à l'armoire sur lequel on pourra se connecter soit avec un badge grâce à un lecteur RFID (Radio Frequency IDentification) soit avec un compte utilisateur grâce à un clavier virtuel.

Le lecteur RFID (Radio Frequency IDentification) sera relié par USB au Raspberry Pi pour l'authentification.

Un lecteur de code barre sera disponible pour récupérer les articles ou la gestion de stock.

La classe **Ihm** s'occupe de gérer toutes les fonctions d'affichage sur l'interface de l'application.

La classe **Keyboard** permet de créer et gérer le clavier virtuel.

La classe **Supervision** permet la gestion global de l'application, elle permet les liaisons entre les différentes classes.

La classe **Article** permet de modéliser un article et de récupérer les données de l'article à partir de la base de données.

La classe **Utilisateur** modélise l'utilisateur connecté.

La classe **Armoire** permet de récupérer les données associées à l'armoire.

La classe **Bdd** permet la liaison avec la base de données e-stock et d'exécuter des requêtes SQL.

Choix du type de liaison série

Comme défini dans le cahier des charges, une liaison série doit être utilisée entre le SE et la Raspberry Pi. Le choix devait se faire entre la RS232, RS422 et la RS485.

Ci-dessous un tableau les regroupant avec leurs caractéristiques :

Liaison	RS232	RS422	RS485
Distance	15m	1200m	1200m
Débit max.	19200 Bauds	10 MBds	10 MBds
Nombre d'émetteurs	1	1	32
Nombre récepteurs	1	10	32

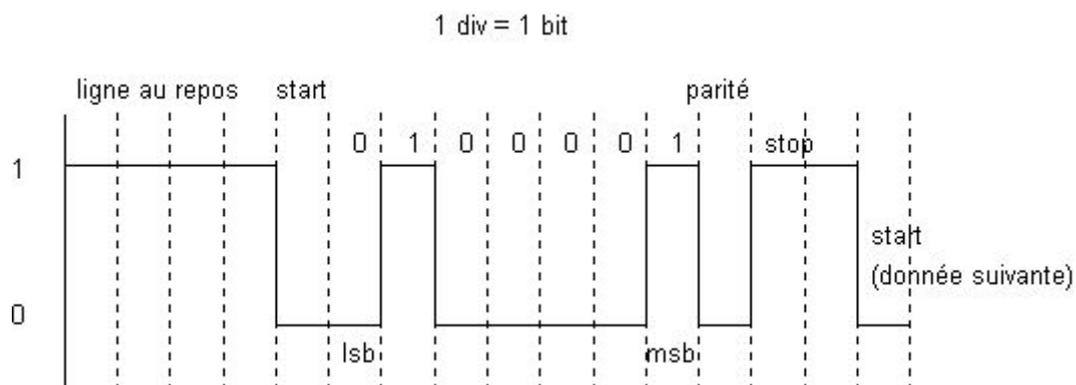
Le choix s'est porté sur la liaison **RS232** car elle correspond aux besoins : la distance de 15 m est largement suffisante (moins d'1 mètre dans une armoire) ainsi que le débit maximum (très peu d'informations sont échangées).

Paramètres d'une trame RS232 :

- Longueur des données : 7 ou 8 bits (par exemple un caractère ASCII)
- La vitesse de transmission : 9600 bauds
- Parité : le mot transmis peut être suivi ou non d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission. Il existe deux types de parité.
 - Parité paire : le bit ajouté à la donnée est positionné de telle façon que le nombre des états 1 soit pair sur l'ensemble donné avec le bit de parité
 - Parité impaire : le bit ajouté à la donnée est positionné de telle façon que le nombre des états 1 soit impair sur l'ensemble donné avec le bit de parité
- Bit de start : la ligne au repos est à l'état logique 1 pour indiquer qu'une trame va être transmise. La ligne passe à l'état bas avant de commencer le transfert. Ce bit permet de synchroniser l'horloge du récepteur.
- Bit de stop : après la transmission, la ligne est positionnée au repos pendant 1, 2 ou 1,5 périodes d'horloge selon le nombre de bits de stop.
- Le bit de start apparaît en premier dans la trame puis les données sont ordonnées en *little-endian*, la parité éventuelle et le ou les bit(s) de stop.

Exemple :

Une trame de parité paire, avec 2 bits de stop, le caractère 'B' dont le codage ASCII est x1000010 (0x42) :



Répartition des tâches

Étudiant EC

- ❖ Melvin Chauvin :
 - Commander l'ouverture/fermeture des casiers,
 - Détecter l'état ouvert/fermé des casiers,
 - Mesurer le poids du conteneur des casiers, Communiquer avec la Raspberry Pi.

Étudiants IR

- ❖ Legger Pierre-Antoine (étudiant 1) :
 - S'authentifier, Rechercher un article,
 - Consulter le stock,
 - Communiquer avec le SE pour commander
 - l'ouverture/fermeture des casiers et
 - Afficher l'état ouvert/fermé des casiers
- ❖ Tranchat Joffrey(étudiant 2):
 - Prendre et rapporter un article,
 - Mettre à jour le stock et les mouvements,
 - consulter les mouvements,
 - Communiquer avec le SE pour
 - Récupérer les pesées des casiers et
 - Assurer le comptage automatique

Objectifs attendus

Chauvin Melvin

- La commande d'une gâche est opérationnelle
- La mesure d'un poids est fonctionnelle
- Le tarage est possible
- La configuration de la liaison série est réalisée
- L'envoi et la réception de trames est opérationnelle
- La communication avec la RPI permet l'ouverture/fermeture d'un casier

Legger Pierre-Antoine

- La lecture d'un badge RFID est réalisée
- L'authentification avec ou sans badge est fonctionnelle
- Une autorisation ou une interdiction d'accès est signalée visuellement
- La consultation du stock est opérationnelle
- La communication avec le SE permet l'ouverture/fermeture d'un casier
- La recherche d'un article est possible

Tranchat Joffrey

- Mise à jour du stock
- Une lecture du code barre d'un article est opérationnelle
- Le comptage automatique est fonctionnel
- La visualisation des mouvements est possible
- La communication avec le SE permet la récupération des pesées
- La gestion des balances est fonctionnelle (pesées, tarage)

Partie personnelle : Legger Pierre-Antoine

Rappel du besoin initial

Le système e-stock est un système de gestion permettant la de contrôler et gérer l'utilisation de produits stockés dans une armoire et d'en assurer là traçabilité de l'attribution du matériel et des consommables stockés. Puis de sécuriser l'accès par un contrôle d'accès par badge RFID.

Le lecteur OMNIKEY

Le lecteur OMNIKEY® 5427CK de HID Global fonctionne dans tous les environnements PC. Indépendant du système d'exploitation et du cas d'usage, les modes CCID ou l'interface Keyboard Wedge fournit la solution idéale sans installer ou assurer des drivers. Cela supprime les problèmes complexes de gestion du cycle de vie du logiciel sur site.

La fonctionnalité d'émulation clavier est configuré en "QWERTY". L'émulation clavier permet de récupérer et de traiter des données du badge pour les entrer directement dans les applications en émulant sur la séquence de touches clavier correspondante.



Le lecteur prend en charge les technologies haute et basse fréquence dans un seul dispositif, il s'intègre donc facilement dans les environnements multi-technologies et simplifie les migrations de technologies de cartes.

Organisation

Pour l'organisation du projet et la communication entre tous les membres du groupe, nous avons utilisé :

- Subversion qui est un logiciel libre de gestion de versions hébergé sur le site RiouxSVN pour l'ensemble du code source du projet.
- L'espace de stockage commun Google Drive pour tous les documents ressources.
- Trello qui permet de savoir l'état de chaque objectif de chaque personne.
- Gantt pour avoir une vue de la planification du projet

Objectifs

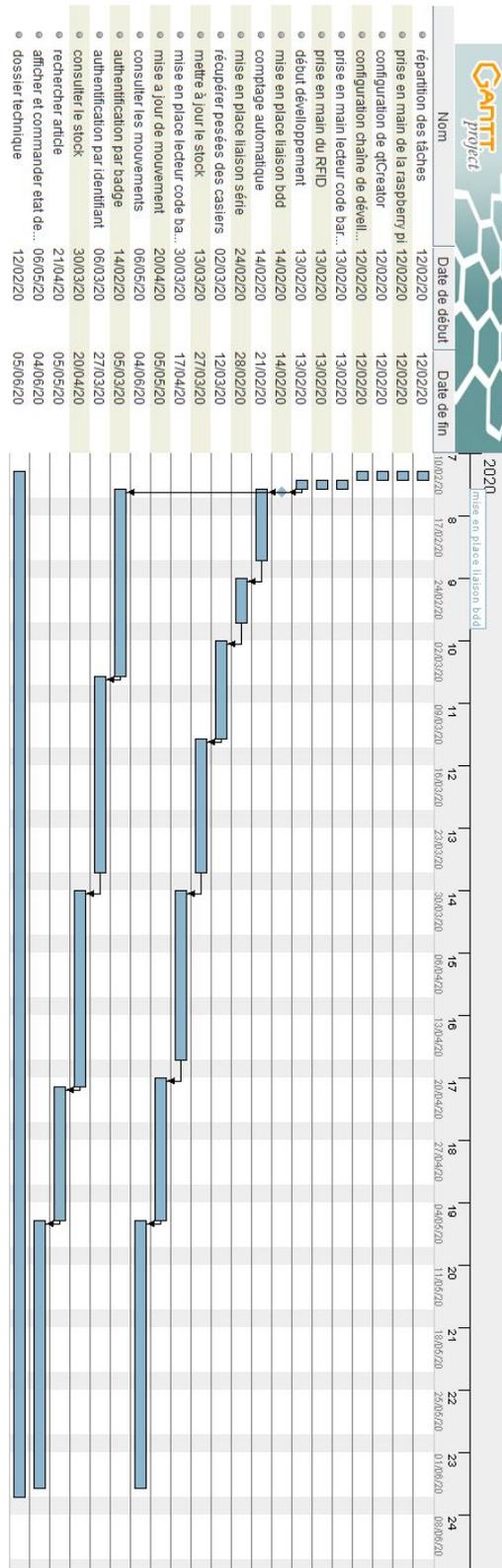
Les utilisateurs peuvent s'authentifier par lecture d'un badge RFID ou par identifiants, et ont la possibilité de rechercher un article, de consulter le stock ainsi que commander l'ouverture/fermeture des casiers.

Répartition des tâches

Tâches à réaliser	Priorité	Itération
Authentification par badge	haute	1
Authentification par identifiant	basse	1
Rechercher un article	haute	2
Consulter les stocks	moyenne	2
Commander l'ouverture des casiers	moyenne	3
Afficher l'état des casiers	moyenne	3

L'Authentification par identifiant a une priorité basse puisqu'il est essentiel pour le client de pouvoir s'authentifier par badge. La tâche sera tout de même réalisée dans l'itération 1 car elle est associée à l'Authentification par badge.

Gantt



Outils de développement

Les outils logiciels :

Désignation	Caractéristiques
Système de gestion de base de données	MariaDB
Atelier de génie logiciel	Bouml V7.9
Logiciel de gestion de version	subversion (RiouxSVN)
Générateurs de documentation	Doxygen version 1.8
Environnement de développement	Qt Creator et Qt Designer
API GUI	Qt 5.11.2

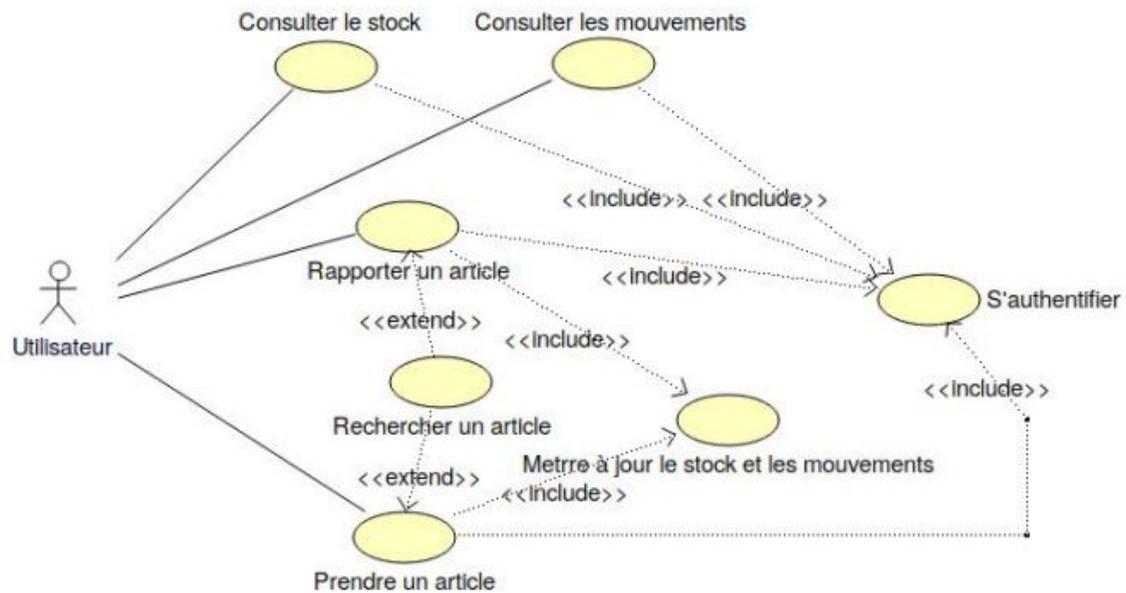
La Raspberry Pi 3 :

Désignation	Caractéristiques
Ordinateur	Raspberry PI 3
Écran	Écran tactile 7" 800x480 relié à la Raspberry PI
Système d'exploitation de la Raspberry PI	GNU/Linux Raspbian

Le lecteur RFID :

Désignation	Caractéristiques
Modèle	OMNIKEY 5427 CK
Clavier virtuel	"QWERTY"
Prise en charge	Haute et basse fréquence

Diagramme de cas d'utilisation pour l'acteur utilisateur



Dans cette partie, les cas d'utilisation sont :

- S'authentifier : se connecter avec un badge RFID ou des identifiants
- Rechercher un article : rechercher un article parmi la liste du matériel présent dans l'armoire
- Consulter le stock : consulter la quantité, la disponibilité et le casier dans lequel se trouve un article

Le protocole de communication

Le protocole **e-stock** définit l'ensemble des trames permettant de communiquer entre le SE (Système Embarqué EC) et la Raspberry Pi (IR). Il est orienté ASCII.

Le protocole **e-stock** est basée sur des trames requête/réponse.

Format des trames

Les trames sont composées d'un en-tête pour identifier leur type.

L'en-tête est "**CASIERS**"

Le délimiteur de fin de champ est le ';'.

Le délimiteur de fin de trame "**\r\n**"

Trame de requête Raspberry Pi → SE :

CASIERS;requete;n_casier;\r\n

n_casier ⇒ 1 à 8 et 0 = *broadcast*

requete ⇒ 1 pour une commande d'ouverture de casier
2 pour une demande d'état

Trame de réponse SE → Raspberry Pi :

CASIERS;reponse;n_casier;donnée;\r\n

n_casier ⇒ 1 à 8

Requête	reponse	donnée
pour une commande d'ouverture du casier	1	0 = erreur, 1 = ok
pour une demande d'état	2	0 = ouvert, 1 = fermé, 2 = inconnu

Remarque : dans le cas d'une requête en *broadcast*, les champs **n_casier;donnée;** sont répétés pour chaque casier.

Maquette IHM

L'utilisateur doit quand il veut accéder au contenu de l'armoire s'authentifier soit par badge RFID soit par ses identifiants.

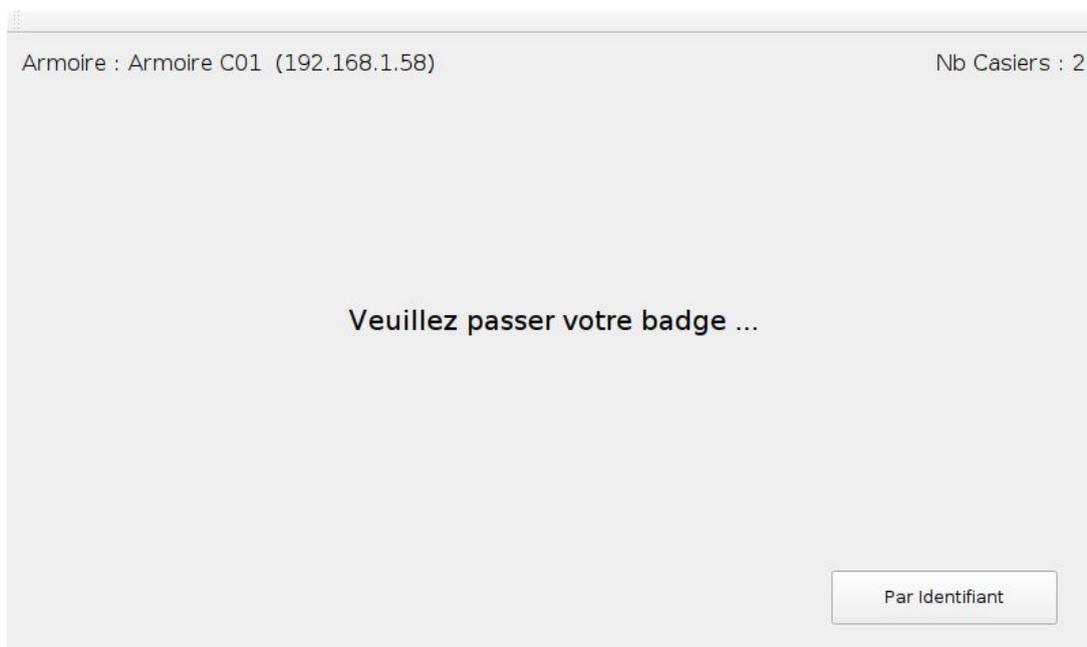
Pour s'authentifier avec un badge RFID (c'est le choix par défaut) :

- Doit présenter son badge

Maquette authentification par badge RFID :



Affichage dans l'application :



Pour s'authentifier par ces identifiants :

- Doit saisir son identifiant et son mot de passe

Maquette authentification par identifiant :

Maquette d'authentification par identifiant. Le formulaire est sur un fond bleu clair et contient :

- Le titre : "Veillez saisir champs ci-dessous"
- Le label "Identifiant :" suivi d'un champ de saisie vide.
- Le label "Mot de passe :" suivi d'un champ de saisie vide.
- Un bouton "Se connecter" centré.
- Un bouton "Par Badge" en bas à gauche.

Affichage dans l'application :

Affichage de la maquette dans l'application. L'interface est grisée et contient :

- En haut à gauche : "Armoire : Armoire C01 (192.168.1.58)"
- En haut à droite : "Nb Casiers : 2"
- Le titre : "Veillez saisir les champs ci-dessous"
- Le label "Identifiant :" suivi d'un champ de saisie contenant "palegger".
- Le label "Mot de passe :" suivi d'un champ de saisie masqué par des points noirs.
- Un bouton "Se Connecter" centré.
- Un bouton "Par Badge" en bas à droite.

L'utilisateur une fois connecté pourra effectuer une recherche de l'article qu'il souhaite et consulter les stocks de l'article concerné.

Maquette recherche et consultation des stocks :

Maquette de l'interface de recherche et consultation des stocks. Le titre est "Stock". Il y a un champ "Rechercher :" suivi de deux champs de saisie. En dessous, un grand rectangle vide est prévu pour l'affichage des résultats. À la base, il y a quatre boutons : "Se déconnecter", "Mouvement", "Casiers" et un bouton non étiqueté.

Affichage dans l'application :

Affichage de l'interface dans l'application. Le titre est "Stock". En haut à gauche, "Armoire : Armoire C01 (192.168.1.58)". En haut à droite, "Nb Casiers : 2".

Section "Rechercher article" : un champ de saisie et un bouton "Rechercher".

Section "Selectionner article" : un menu déroulant affichant "Fluke i30s".

Statistiques : "Quantité : 8", "Disponible : 12", "Casier : 1 et 3".

En bas à gauche, un bouton "Se déconnecter". En bas à droite, un bouton "Mouvements".

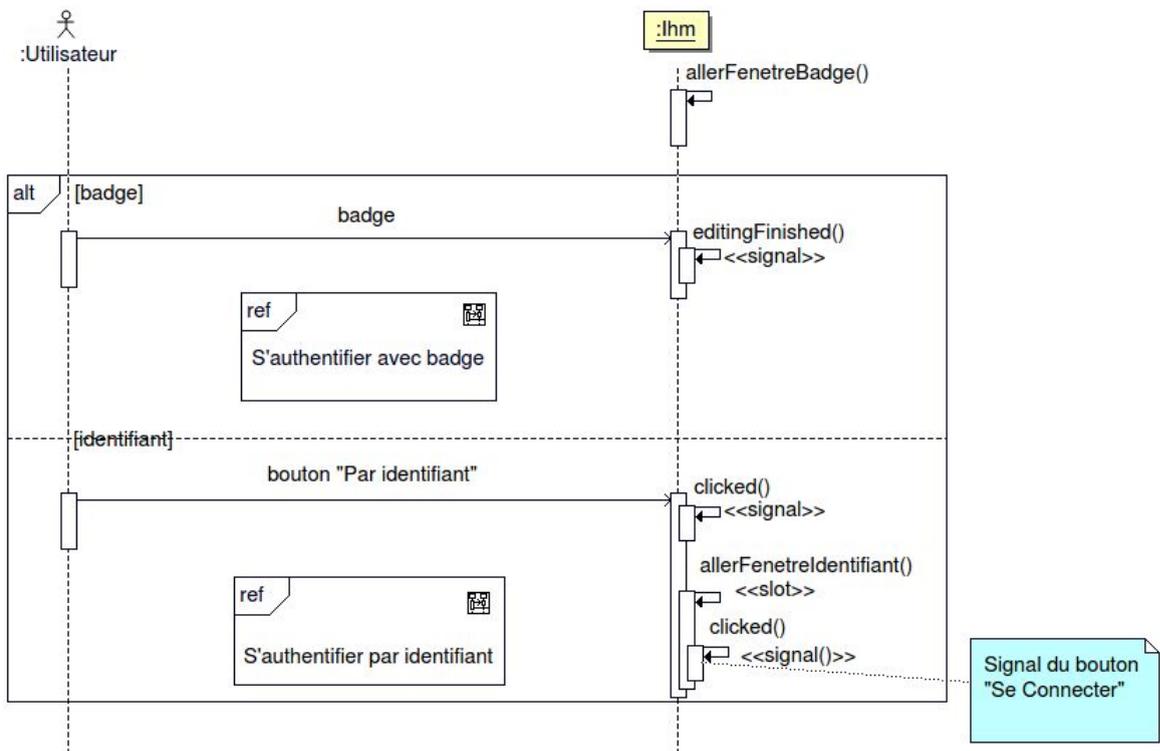
Deux boutons "Casier 1" et "Casier 2" sont visibles en vert.

Scénarios

Scénario Authentification

L'utilisateur a le choix pour s'authentifier, il peut utiliser :

- son badge : comme la trame lue contient un retour à la ligne, cela provoquera l'émission du signal editingFinished() qui va déclencher l'authentification par badge;
- ses identifiants : l'utilisateur doit d'abord cliquer sur le bouton "Par identifiant", puis saisir son identifiant et mot de passe et enfin cliquer sur le bouton "Se connecter" qui provoquera l'émission du signal clicked() et déclenchera l'authentification par identifiant.



Dans de ce scénario, seule la classe Ihm intervient car c'est son rôle d'interagir avec l'utilisateur :

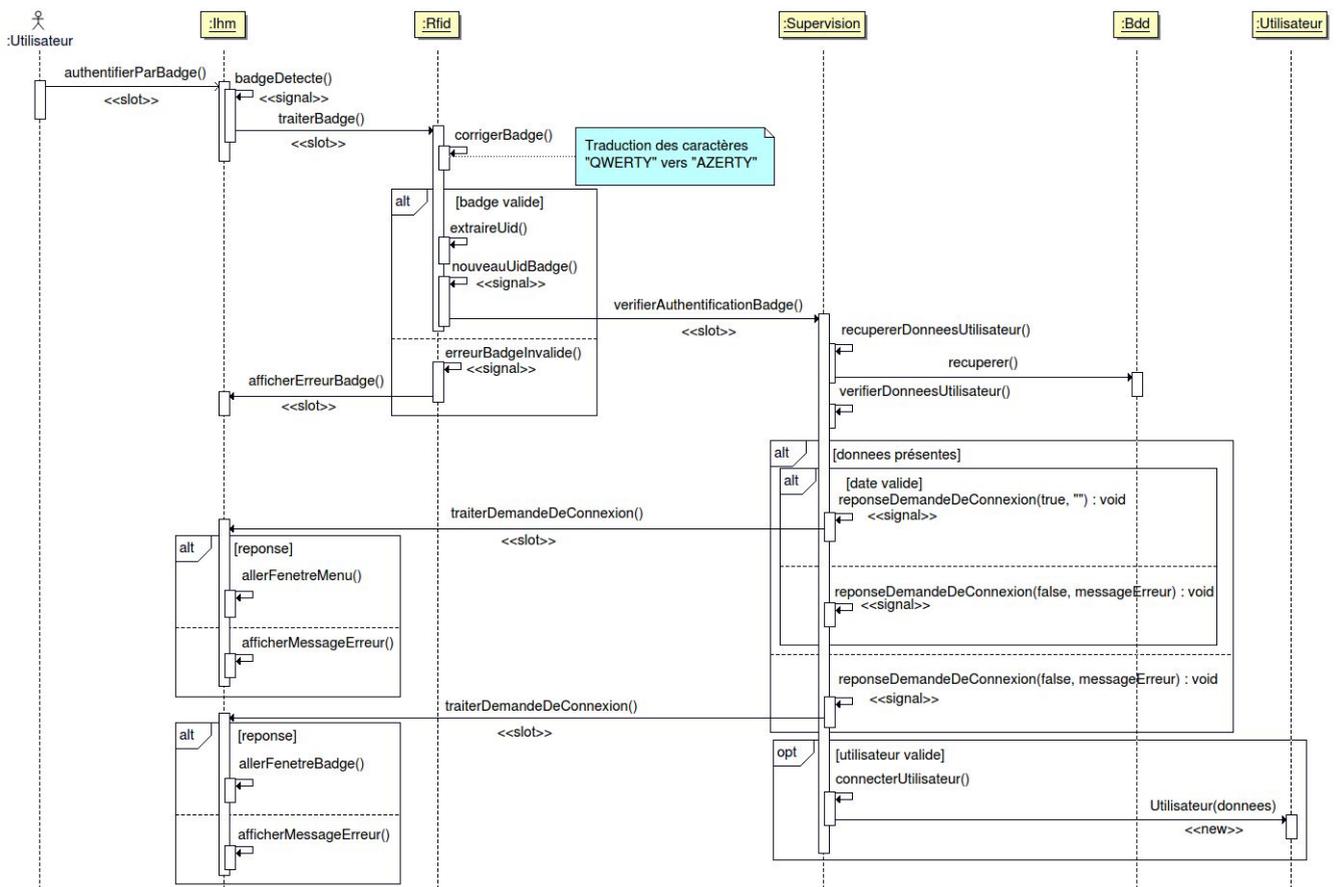
Ihm
- ui : Ui::Ihm
+ Ihm(inout parent : QWidget = nullptr)
+ ~Ihm()
+ changerDeFenetre(in fenetre : int) : void
+ placerCasier(inout casier : Casier) : void
- afficherInformationsArmoire(in informationsArmoire : QStringList) : void
- authentifierParBadge() : void
- authentifierParIdentifiant() : void
- deconnecterUtilisateur() : void
- allerFenetreBadge() : void
- allerFenetreIdentifiant() : void
- allerFenetreMenu() : void
- afficherErreurBadge(in message : QString) : void
- afficherErreurDepassementQuantite() : void
- traiterDemandeDeConnexion(in reponse : bool, in message : QString) : void
- activerRecherche() : void
- rechercherArticle() : void
- effacerRechercheArticle() : void
- mettreAJourListeArticles(in articlesTrouves : QVector<QStringList>) : void
- selectionnerArticle(in index : int) : void
- afficherDonneesArticleSelectionne(in donneesArticle : QStringList) : void
- afficherDonneesArticleSelectionne(in donneesArticle : QVector<QStringList>) : void
- badgeDetecte(in : QString) : void
- identifiantDetecte(in identifiant : QString, in motDePasse : QString) : void
- rechercheArticle(in : QString) : void
- articleSelectionne(in : QString) : void

Scénario Authentification par badge

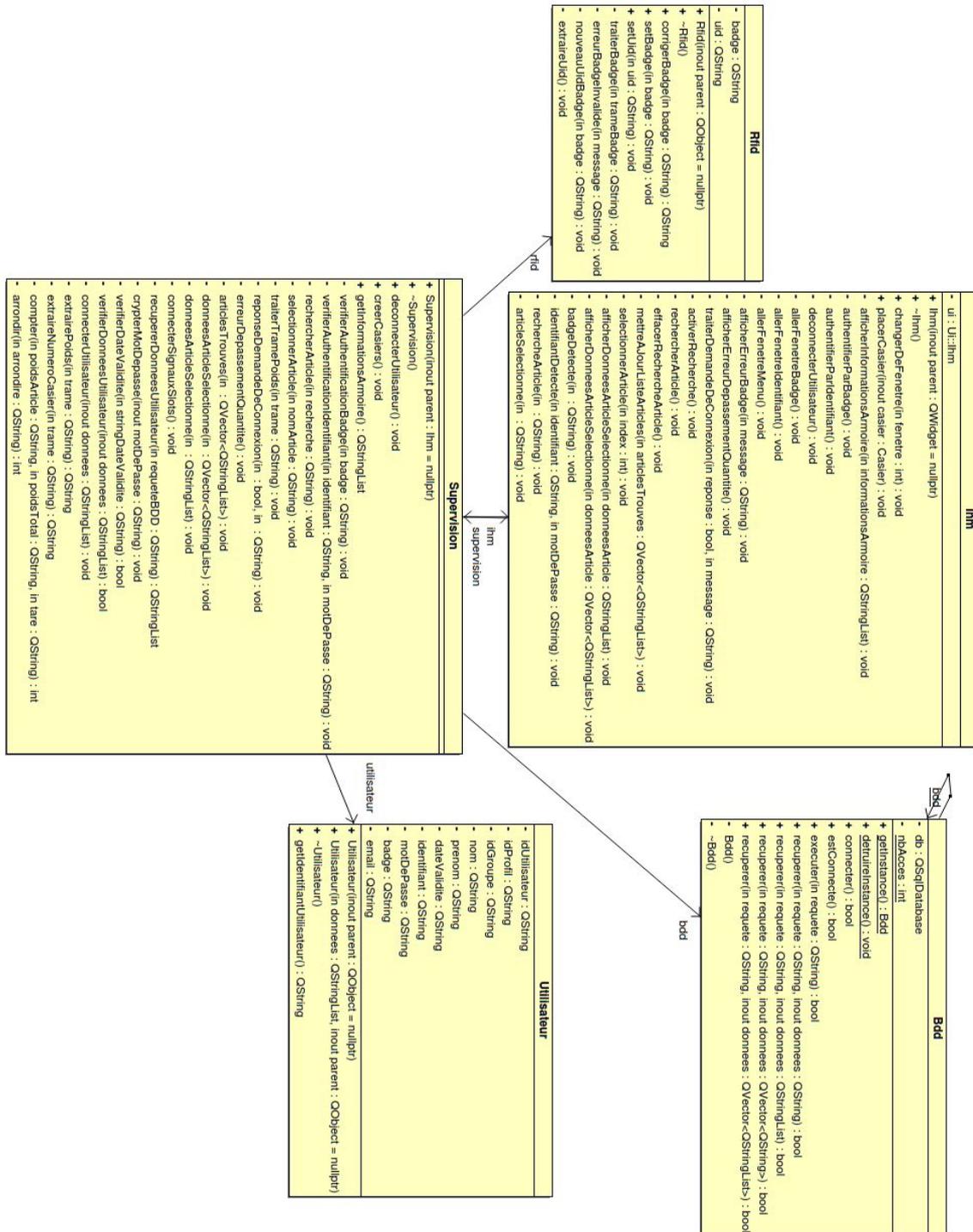
Une fois le “badge” réceptionné par la classe Rfid, il faut corriger le contenu de la trame car le lecteur de badge est configuré en clavier virtuel “QWERTY”. Il faut traduire les caractères en “AZERTY” avec la méthode corrigerBadge().

Puis on vérifie que le début de la trame “badge” contient “RFID:” sinon on envoie une erreur avec erreurBadgeInvalide() et on l’affiche avec afficherErreurBadge(). Si la trame contient un entête valide, extrait l’UID et on le signale. Il faut ensuite le vérifier avec verifierAuthenticationBadge().

Puis la classe Supervision récupère les données liées au badge dans la base de données avec recupererDonneesUtilisateur() qui appelle la méthode recuperer() de la classe Bdd pour exécuter la requête SQL. On vérifie la présence de données utilisateur sinon on envoie l’erreur utilisateur invalide avec reponseDemandeDeConnexion(). Si les données sont présentes, on vérifie ensuite la date de validité du compte pour envoyer une réponse qui sera ensuite traitée par l’Ihm avec traiterDemandeDeConnexion(). En fonction de la réponse, on appellera soit la méthode allerFenetreMenu() soit on affichera une boîte de dialogue d’erreur. Pour finir, Supervision connectera l’utilisateur avec les données récupérées précédemment.



Voici le diagramme de classes pour ce scénario :



- La classe Ihm interagit avec la Supervision avec une relation d'association bidirectionnelle pour permettre l'envoi et la récupération d'information.
- La classe Supervision a une relation unidirectionnelle avec les classes Rfid, Utilisateur et Bdd pour permettre un accès simple aux données et pouvoir les redistribuer à l'Ihm.

La méthode `authentifierParBadge()` permet de signaler la trame d'un badge si celle-ci n'est pas vide :

```
void Ihm::authentifierParBadge()
{
    ui->labelErreurBadge->clear();

    if(ui->lineBadge->text() != "")
    {
        #ifdef DEBUG_IHM
            qDebug() << Q_FUNC_INFO << "Contenu brut badge" <<
ui->lineBadge->text();
        #endif

        QString trameBadge = ui->lineBadge->text();
        ui->lineBadge->clear();
        emit badgeDetecte(trameBadge);
    }
}
```

La méthode `corrigerBadge()` traduit les "caractères QWERTY" de la trame du badge vers l'"AZERTY" :

```
QString Rfid::corrigerBadge(QString badge)
{
    QString badgeCorrige = "";

    if(!badge.isEmpty())
    {
        // effectue les remplacements des touches QWERTY en touches
AZERTY
        badgeCorrige = badge.replace(QString::fromUtf8("Q"), "A");
        badgeCorrige = badge.replace(QString::fromUtf8("W"), "Z");
        badgeCorrige = badge.replace(QString::fromUtf8("q"), "a");
        badgeCorrige = badge.replace(QString::fromUtf8("w"), "z");
        badgeCorrige = badge.replace(QString::fromUtf8("M"), ":");
        badgeCorrige = badge.replace(QString::fromUtf8("à"), "ø");
        badgeCorrige = badge.replace(QString::fromUtf8("&"), "1");
        badgeCorrige = badge.replace(QString::fromUtf8("é"), "2");
        badgeCorrige = badge.replace(QString::fromUtf8("\'"), "3");
        badgeCorrige = badge.replace(QString::fromUtf8("'"), "4");
        badgeCorrige = badge.replace(QString::fromUtf8("("), "5");
        badgeCorrige = badge.replace(QString::fromUtf8("-"), "6");
        badgeCorrige = badge.replace(QString::fromUtf8("è"), "7");
        badgeCorrige = badge.replace(QString::fromUtf8("_"), "8");
    }
}
```

```
        badgeCorrige = badge.replace(QString::fromUtf8("ç"), "9");
    }
    return badgeCorrige;
}
```

La méthode `verifierAuthentificationBadge()` va vérifier les données utilisateur récupérés à partir d'une requête SQL et connecter l'utilisateur si besoin :

```
void Supervision::verifierAuthentificationBadge(QString badge)
{
    QString requeteBDD = "SELECT * from Utilisateur where Badge = '" +
    badge + "'";
    QStringList donnees = recupererDonneesUtilisateur(requeteBDD);
    if(verifierDonneesUtilisateur(donnees))
    {
        connecterUtilisateur(donnees);
    }
}
```

La méthode `recupererDonneesUtilisateur()` effectue une demande à la classe `Bdd` qui effectuera une requête SQL et retournera les données obtenues :

```
QStringList Supervision::recupererDonneesUtilisateur(QString requeteBDD)
{
    QStringList donnees;
    bdd->recuperer(requeteBDD, donnees);

    return donnees;
}
```

La méthode `verifierDonneesUtilisateur()` vérifie que les données ne sont pas vides puis ensuite que la date de validité est correcte :

```
bool Supervision::verifierDonneesUtilisateur(QStringList &donnees)
{
    #ifdef DEBUG_SUPERVISION
        qDebug() << Q_FUNC_INFO << donnees;
    #endif

    if(!donnees.isEmpty())
    {
        if(verifierDateValidite(donnees.at(TABLE_UTILISATEUR_DATE_VALIDITE)))
    }
}
```

```
    {
        emit reponseDemandeDeConnexion(true, "");
        return true;
    }
    else
    {
        emit
reponseDemandeDeConnexion(false,MESSAGE_ERREUR_UTILISATEUR_DATE_NON_VALID
DE);
        return false;
    }
}
else
{
    emit
reponseDemandeDeConnexion(false,MESSAGE_ERREUR_UTILISATEUR_NON_VALID);
    return false;
}
}
```

La méthode connecterUtilisateur() vérifiera si un utilisateur n'est pas déjà connecté sinon créera un objet Utilisateur avec ces données :

```
void Supervision::connecterUtilisateur(QStringList &donnees)
{
    if(utilisateur != nullptr)
    {
        deconnecterUtilisateur();
    }
    utilisateur = new Utilisateur(donnees, this);
#ifdef DEBUG_SUPERVISION
    qDebug() << Q_FUNC_INFO <<
utilisateur->getIdentifiantUtilisateur() << "authentifié";
#endif
}
```

Pour l'authentification par badge, on utilise la requête SQL suivante :

➤ `SELECT * from Utilisateur where Badge = "" + badge + "";`

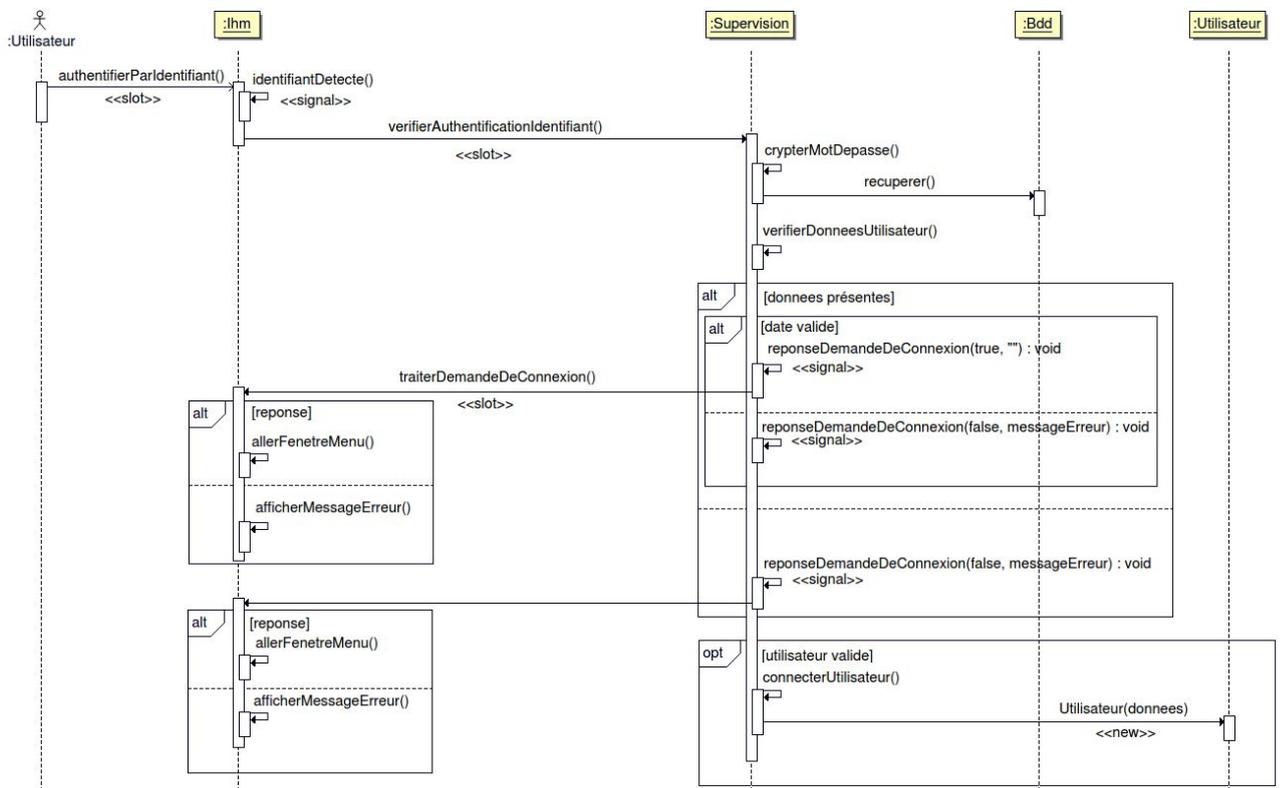
Cette requête permet de récupérer les informations de l'utilisateur en fonction de l'UID du badge.

Scénario Authentification par identifiant

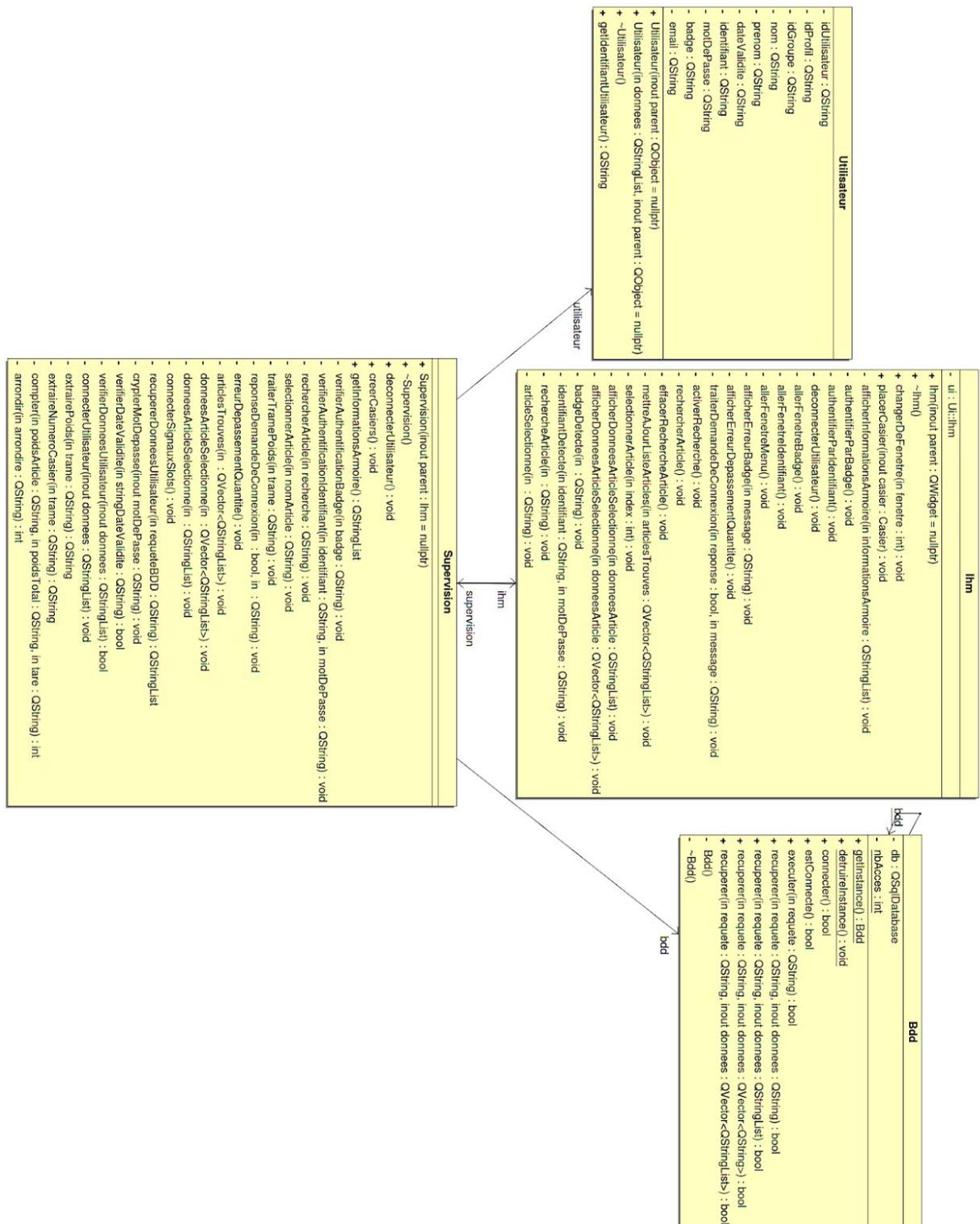
On récupère la saisie des identifiants de l'utilisateur et on le signale avec `identifiantDetecte()` qui déclenche `verifierAuthentificationIdentifiant()` de la classe `Supervision` en lui passant en paramètre l'identifiant et le mot de passe.

Cette méthode va ensuite crypter le mot de passe avec `crypterMotDepasse()`, puis récupérer les données de la base de données en fonction de l'identifiant et du mot de passe crypté avec la méthode `recuperer()` de la classe `Bdd`.

On fera appel à la méthode `verifierDonneesUtilisateur()` qui vérifie la présence de données utilisateur sinon on envoie l'erreur utilisateur invalide avec `reponseDemandeDeConnexion()`. Si les données sont présentes, on vérifie ensuite la date de validité du compte pour envoyer une réponse qui sera ensuite traitée par l'Ihm avec `traiterDemandeDeConnexion()`. En fonction de la réponse, on appellera soit la méthode `allerFenetreMenu()` soit on affichera une boîte de dialogue d'erreur. Pour finir, `Supervision` connectera l'utilisateur avec les données récupérées précédemment.



Voici le diagramme de classe de ce scénario :



- La classe Ihm interagi avec la Supervision avec une relation d'association bidirectionnelle pour permettre l'envoi et la récupération d'information.
- La classe Supervision a une relation unidirectionnelle avec les classes Utilisateur et Bdd pour permettre un accès simple aux données et pouvoir les redistribuer à l'Ihm.

La méthode `authentifierParIdentifiant()` permet d'envoyer les champs identifiant et mot de passe s'ils ne sont pas vides :

```
void Ihm::authentifierParIdentifiant()
{
    if(ui->lineIdentifiant->text() != "")
    {
        #ifdef DEBUG_IHM
            qDebug() << Q_FUNC_INFO << "Identifiant" <<
ui->lineIdentifiant->text() << "MotDePasse" <<
ui->lineMotDePasse->text();
        #endif

        QString identifiant = ui->lineIdentifiant->text();
        QString motDePasse = ui->lineMotDePasse->text();
        ui->lineIdentifiant->clear();
        ui->lineMotDePasse->clear();
        emit identifiantDetecte(identifiant, motDePasse);
    }
}
```

La méthode `verifierAuthentificationIdentifiant()` demandera les informations utilisateur relatives à l'identifiant et au mot de passe crypté, si les données sont correctes on connectera l'utilisateur si nécessaire :

```
void Supervision::verifierAuthentificationIdentifiant(QString
identifiant, QString motDePasse)
{
    this->crypterMotDepasse(motDePasse);

    #ifdef CHANGE_PASSWORD_BEFORE
        QString requete = QString("UPDATE Utilisateur SET MotDePasse='%1'
WHERE Identifiant='%2'").arg(motDePasse).arg(identifiant);
        bdd->executer(requete);
    #endif

    QString requeteBDD = "SELECT * from Utilisateur where Identifiant =
'" + identifiant + "' && MotDePasse = '" + motDePasse + "'";
    QStringList donnees = recupererDonneesUtilisateur(requeteBDD);
    if(verifierDonneesUtilisateur(donnees))
    {
        connecterUtilisateur(donnees);
    }
}
```

La méthode privée crypterMotDepasse() effectue le cryptage du mot de passe seulement si celui-ci n'est pas vide :

```
void Supervision::crypterMotDepasse(QString &motDePasse)
{
    if(!motDePasse.isEmpty())
    {
        motDePasse =
        QString(QCryptographicHash::hash((motDePasse).toLatin1(),
        QCryptographicHash::Md5).toHex());
    }

    #ifdef DEBUG_SUPERVISION
        qDebug() << Q_FUNC_INFO << "Mot de passe crypte" << motDePasse;
    #endif
}
```

Pour l'authentification par identifiant, on utilise la requête SQL suivante :

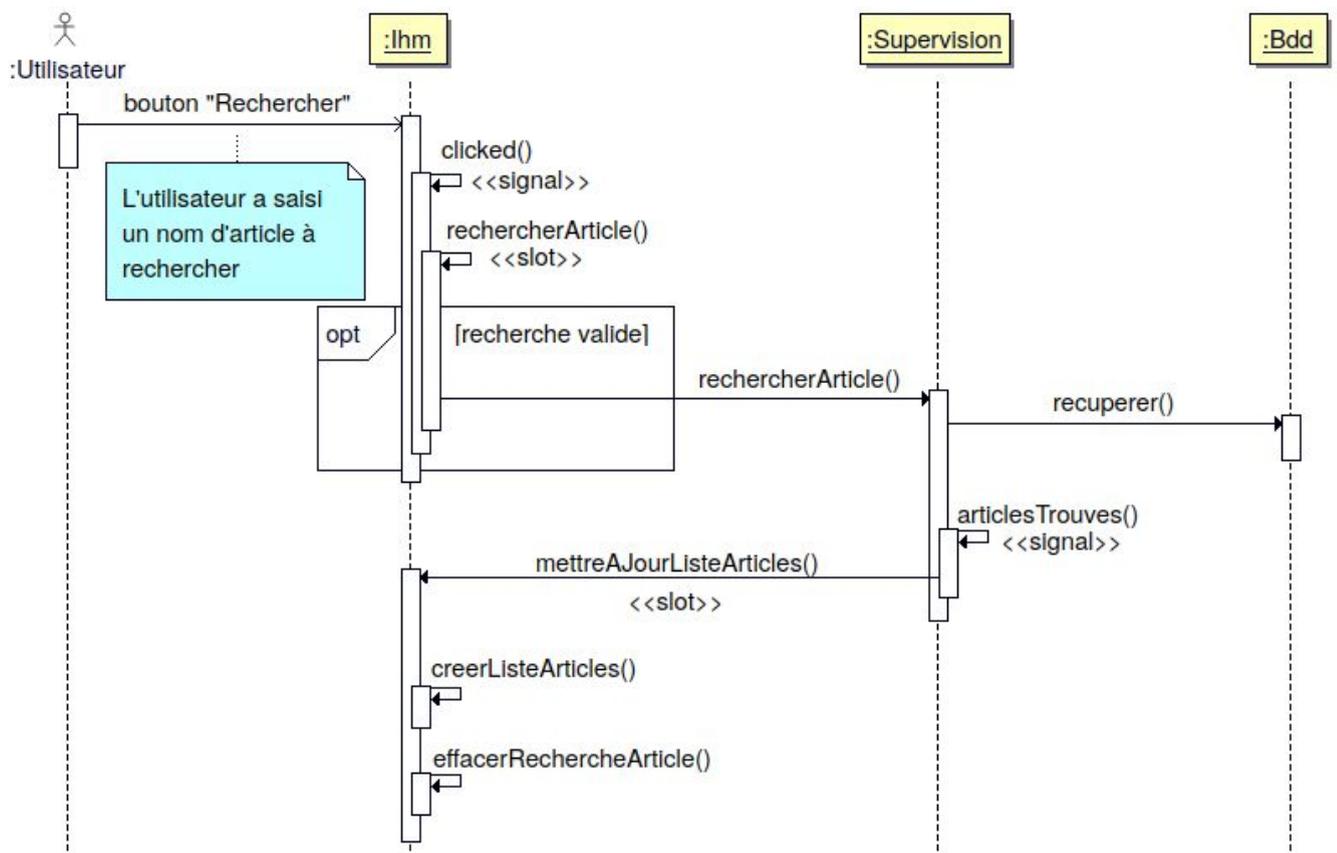
- `SELECT * from Utilisateur where Identifiant = "" + identifiant + "" && MotDePasse = "" + motDePasse + "";`

Cette requête permet de récupérer les informations de l'utilisateur en fonction de son identifiant et mot de passe.

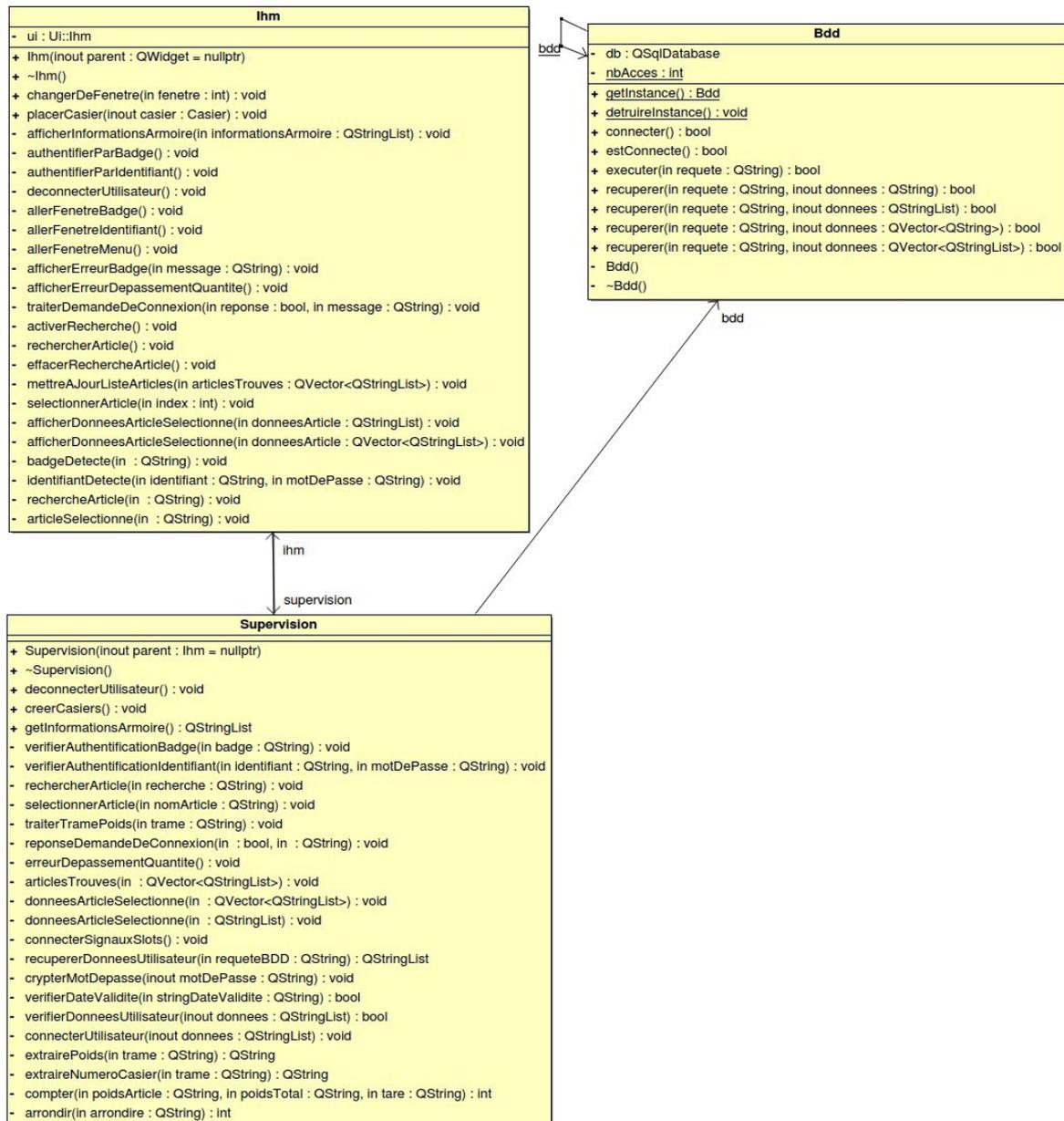
Scénario Recherche article

L'utilisateur saisi un nom de l'article et déclenche l'exécution de la méthode rechercherArticle() en cliquant sur le bouton "Rechercher". La classe Ihm transmet le traitement à la classe Supervision en appelant rechercherArticle() avec en paramètre le nom d'article à rechercher.

Supervision récupérera les articles qui corresponde avec la méthode recuperer() de la classe Bdd. Puis elle renverra la liste des articles à l'Ihm avec la méthode mettreAJourListeArticle(). Cette dernière a pour rôle de mettre à jour la liste déroulante contenant la liste d'articles trouvés suite à la recherche effectuée.



Voici le diagramme de classe de ce scénario :



- La classe Ihm interagit avec la Supervision avec une relation d'association bidirectionnelle pour permettre l'envoi et la récupération d'information.
- La classe Supervision a une relation unidirectionnelle avec la classe Bdd pour permettre un accès simple aux données et pouvoir les redistribuer à l'Ihm.

La méthode `rechercherArticle()` de la classe `Ihm` qui récupère le contenu de la recherche et le signale à la classe `Supervision` :

```
void Ihm::rechercherArticle()
{
    if(!ui->lineRecherche->text().isEmpty())
        emit rechercheArticle(ui->lineRecherche->text());
}
```

La méthode `rechercherArticle()` de la classe `Supervision` va demander à la classe `Bdd` de récupérer la liste des articles qui "ressemble" au contenu recherché :

```
void Supervision::rechercherArticle(QString recherche)
{
    QString requete = "SELECT Stock.NumeroCasier, Article.idType,
Article.Nom, Stock.Quantite, Stock.Disponible, Article.Designation FROM
Stock INNER JOIN Article ON Stock.idArticle = Article.idArticle WHERE
Article.Nom LIKE '%" + recherche + "%' OR Article.Code LIKE '%" +
recherche + "%' OR Article.Designation LIKE '%" + recherche + "%' ORDER
BY Stock.NumeroCasier ASC";

    QVector<QStringList> listeArticlesTrouves;
    bdd->recuperer(requete, listeArticlesTrouves);

    emit articlesTrouves(listeArticlesTrouves);
}
```

La méthode `mettreAJourListeArticles()` met à jour la liste des articles trouvés dans la liste déroulante. Une déconnexion/connexion signal/slot est nécessaire pour éviter un déclenchement pendant la mise à jour :

```
void Ihm::mettreAJourListeArticles(QVector<QStringList> articlesTrouves)
{
    #ifdef DEBUG_IHM
        qDebug() << Q_FUNC_INFO << "articlesTrouves" <<
articlesTrouves.size() << articlesTrouves;
    #endif
    disconnect(ui->comboBoxArticle, SIGNAL(currentIndexChanged(int)),
this, SLOT(selectionnerArticle(int)));
    ui->comboBoxArticle->clear();

    ui->comboBoxArticle->addItem("Sélectionner un article");
    for(int i = 0 ; i < articlesTrouves.size() ; i++)
    {
```

```
if(ui->comboBoxArticle->findText(articlesTrouves[i].at(2)) == -1)
{
    ui->comboBoxArticle->addItem(articlesTrouves[i].at(2));
}
}
connect(ui->comboBoxArticle, SIGNAL(currentIndexChanged(int)),
this, SLOT(selectionnerArticle(int)));

effacerRechercheArticle();
}
```

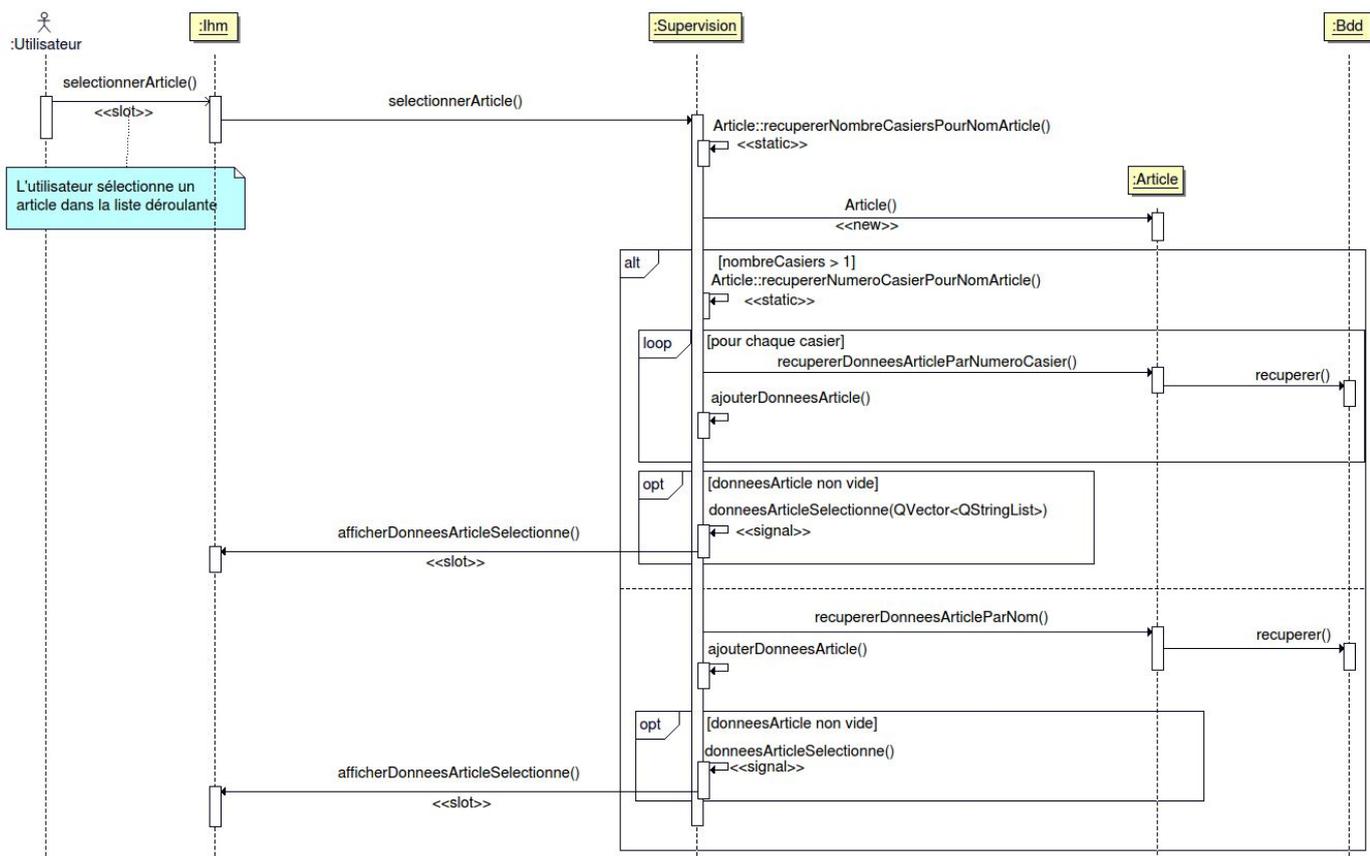
Pour récupérer les informations de l'article en fonction du nom, on effectue une requête SQL qui utilise LIKE. Les informations sur l'article recherché nécessite d'effectuer une jointure entre les tables Stock et Article :

- SELECT Stock.NumeroCasier, Article.idType, Article.Nom, Stock.Quantite, Stock.Disponible, Article.Designation FROM Stock INNER JOIN Article ON Stock.idArticle = Article.idArticle WHERE Article.Nom LIKE '%' + recherche + '%' OR Article.Code LIKE '%' + recherche + '%' OR Article.Designation LIKE '%' + recherche + '%' ORDER BY Stock.NumeroCasier ASC

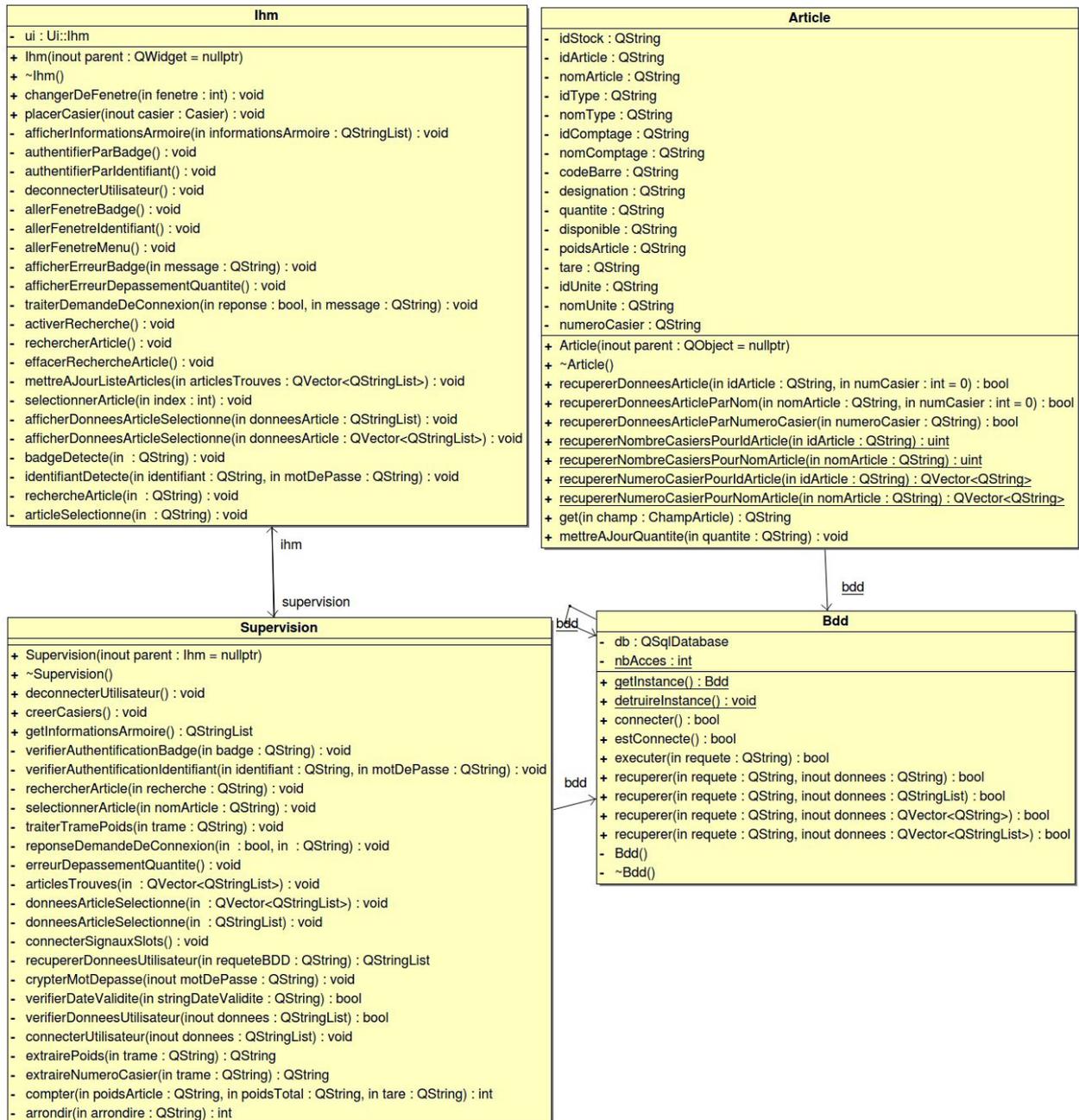
Scénario Consulter le stock

L'utilisateur sélectionne un article dans la liste déroulante. Cela déclenche le slot `selectionnerArticle()` dans la classe `Ihm` qui appelle `selectionnerArticle()` de la classe `Supervision` avec en paramètre le nom de l'article sélectionné. Un objet `Article` est alors instancié. Il faut récupérer le nombre de casiers de l'armoire car il est possible qu'un article soit réparti dans plusieurs casiers. On appelle alors la méthode *static* `recupererNombreCasiersPourNomArticle()` de la classe `Article`.

Pour chaque casier, on récupère les données de l'article avec `recupererDonneesArticleParNumeroCasier()` puis, si des données ont été récupérées, on appelle `afficherDonneesArticleSelectionne()` pour les afficher.



Voici le diagramme de classe de ce scénario :



- La classe Ihm interagit avec la Supervision avec d'association bidirectionnelle pour permettre l'envoi et la récupération d'information.
- La classe Supervision a une relation unidirectionnelle avec la classe Bdd et article pour permettre un accès simple aux données et pouvoir les redistribuer à l'Ihm.

La méthode sélectionnerArticle() de la classe Ihm envoie le nom de l'article sélectionné dans la liste déroulante à partir de son index :

```
void Ihm::selectionnerArticle(int index)
{
    #ifdef DEBUG_IHM
        qDebug() << Q_FUNC_INFO << "index" << index <<
        ui->comboBoxArticle->currentText();
    #endif

    emit articleSelectionne(ui->comboBoxArticle->currentText());
}
```

La méthode sélectionnerArticle() de la classe Supervision récupère les données de l'article différemment si l'article est dans un casier ou plusieurs, puis les envoie à l'ihm pour les afficher :

```
void Supervision::selectionnerArticle(QString nomArticle)
{
    #ifdef DEBUG_SUPERVISION
        qDebug() << "Nom article" << nomArticle;
    #endif

    unsigned int nombreCasiers =
    Article::recupererNombreCasiersPourNomArticle(nomArticle);
    Article *article = new Article(this);

    if(nombreCasiers > 1)
    {
        QVector<QString> numeroDesCasiers;
        QVector<QStringList> donneesArticle;
        QStringList data;

        numeroDesCasiers =
        Article::recupererNumeroCasierPourNomArticle(nomArticle);

        for(int i = 0; i < numeroDesCasiers.size(); i++)
        {
            article->recupererDonneesArticleParNumeroCasier(numeroDesCasiers[i]);
            data << article->get(TABLE_ARTICLE_QUANTITE);
            data << article->get(TABLE_ARTICLE_DISPONIBLE);
            data << article->get(TABLE_ARTICLE_NUMERO_CASIER);

            donneesArticle.push_back(data);
        }
    }
}
```

```
        data.clear();
    }

    if(!donneesArticle.isEmpty())
    {
        emit donneesArticleSelectionne(donneesArticle);
    }
}
else
{
    QStringList donneesArticle;
    article->recupererDonneesArticleParNom(nomArticle);

    donneesArticle << article->get(TABLE_ARTICLE_QUANTITE);
    donneesArticle << article->get(TABLE_ARTICLE_DISPONIBLE);
    donneesArticle << article->get(TABLE_ARTICLE_NUMERO_CASIER);

    if(!donneesArticle.isEmpty())
    {
        emit donneesArticleSelectionne(donneesArticle);
    }
}
}
```

La méthode `afficherDonneesArticleSelectionne()` pour un article dans un seul casier affiche simplement les données dans l'interface graphique :

```
void Ihm::afficherDonneesArticleSelectionne(QStringList donneesArticle)
{

    ui->labelQuantiteNombre->setText(donneesArticle.at(ARTICLE_QUANTITE));

    ui->labelDisponibleNombre->setText(donneesArticle.at(ARTICLE_DISPONIBLE)
);
    ui->labelCasierNombre->setText(donneesArticle.at(NUMERO_CASIER));
}
```

La méthode `afficherDonneesArticleSelectionne()` pour un article dans plusieurs casiers traite l'ensemble des données reçues pour déterminer la quantité et disponibilités totales afin de les afficher :

```
void Ihm::afficherDonneesArticleSelectionne(QVector<QStringList>
donneesArticle)
{
    unsigned int articleDisponible = 0;
    QString casiers;
    int nombreCasiers = donneesArticle.size();

    for(int i = 0; i < nombreCasiers; i++)
    {
        articleDisponible +=
(donneesArticle[i].at(ARTICLE_DISPONIBLE)).toUInt();

        if(i == 0)
        {
            casiers = donneesArticle[i].at(NUMERO_CASIERS);
        }
        else
        {
            casiers += " et " + donneesArticle[i].at(NUMERO_CASIERS);
        }
    }

    ui->labelQuantiteNombre->setText(donneesArticle[0].at(ARTICLE_QUANTITE))
;

    ui->labelDisponibleNombre->setText(QString::number(articleDisponible));
    ui->labelCasierNombre->setText(casiers);
}
```

Dans ce scénario, on utilise plusieurs requêtes SQL :

- Pour récupérer les informations d'un article en fonction de son nom :
 - `SELECT Stock.idStock, Article.idArticle, Article.Nom AS Article, Type.idType, Type.nom AS Type, Comptage.idComptage, Comptage.Nom AS Comptage, Article.Code, Article.Designation, Stock.Quantite, Stock.Disponible, Article.Poids, Stock.Tare, Unite.idUnite, Unite.Nom, Stock.numeroCasier FROM Stock INNER JOIN Article ON Article.idArticle=Stock.idArticle INNER JOIN Type ON Type.idType=Article.idType INNER JOIN Comptage ON Comptage.idComptage=Stock.idComptage INNER JOIN Unite ON Unite.idUnite=Stock.idUnite WHERE Article.Nom = " + nomArticle + ";`

- Pour récupérer les informations d'un article en fonction de son numéro de casier :
 - `SELECT Stock.idStock, Article.idArticle, Article.Nom AS Article, Type.idType, Type.nom AS Type, Comptage.idComptage, Comptage.Nom AS Comptage, Article.Code, Article.Designation, Stock.Quantite, Stock.Disponible, Article.Poids, Stock.Tare, Unite.idUnite, Unite.Nom, Stock.numeroCasier FROM Stock INNER JOIN Article ON Article.idArticle=Stock.idArticle INNER JOIN Type ON Type.idType=Article.idType INNER JOIN Comptage ON Comptage.idComptage=Stock.idComptage INNER JOIN Unite ON Unite.idUnite=Stock.idUnite WHERE Stock.numeroCasier = " + numeroCasier + " ;"`

- Pour récupérer le nombre de casier pour un article en fonction du nom :
 - `SELECT COUNT(Stock.idArticle) FROM Stock INNER JOIN Article ON Stock.idArticle = Article.idArticle WHERE Article.Nom = " + nomArticle + " ;"`

- Pour récupérer les numéros de casiers associer au nom d'un article :
 - `SELECT Stock.numeroCasier FROM Stock INNER JOIN Article ON Article.idArticle=Stock.idArticle INNER JOIN Type ON Type.idType=Article.idType INNER JOIN Comptage ON Comptage.idComptage=Stock.idComptage INNER JOIN Unite ON Unite.idUnite=Stock.idUnite WHERE Article.Nom = " + nomArticle + " ;"`

Tests de validation

Rechercher un article

Test	Article	Résultats attendus	Résultats obtenus	Valide
Recherche mot approché (casse)	fluke	Fluke i30s Fluke 179	Fluke i30s Fluke 179	oui
Recherche mot exact	Fluke i30s	Fluke i30s	Fluke i30s	oui
Recherche d'un article non existant	test	Aucun article qui n'apparaît	Aucun article qui n'apparaît	oui

Consulter le stock

Test	Article	Résultats attendus	Résultats obtenus	Valide
Consultation un article dans un casier	Fluke 179	Quantité : 2 Disponible : 2 Casier : 2	Quantité : 2 Disponible : 2 Casier : 2	oui
Consultation un article dans plusieurs casiers	Fluke i30s	Quantité : 8 Disponible : 6 Casier : 1 et 3	Quantité : 8 Disponible : 6 Casier : 1 et 3	oui

Authentification par identifiant

Test	Identifiant	Mot de passe	Résultats attendus	Résultats obtenus	Valide
Identifiant et mot de passe valide	jbeaumont	""	Affichage fenêtre stock	Affichage fenêtre stock	oui
Identifiant et mot de passe valide date invalide	bounoir.f	""	Affichage "Erreur : le compte n'est plus valide"	Affichage "Erreur : le compte n'est plus valide"	oui
Identifiant valide et mot de passe invalide	jbeaumont	test	Affichage "Erreur : utilisateur non valide"	Affichage "Erreur : utilisateur non valide"	oui
Identifiant non présent dans la base de données	test	test	Affichage "Erreur : utilisateur non valide"	Affichage "Erreur : utilisateur non valide"	oui

Authentification par badge RFID

Test	Badge	Résultats attendus	Résultats obtenus	Valide
Badge non valide	RGFD:30DDA983	Affichage "Erreur badge invalide"	Affichage "Erreur badge invalide"	oui
Badge valide	30DDA983	Affichage fenêtre stock	Affichage fenêtre stock	oui
Badge valide mais date de validité invalide	62A3F560	Affichage "Erreur : le compte n'est plus valide"	Affichage "Erreur : le compte n'est plus valide"	oui
Badge non présent dans la base de données	335C3086	Affichage "Erreur : utilisateur non valide"	Affichage "Erreur : utilisateur non valide"	oui

Partie Personnelle : Tranchat Joffrey

Objectifs

- Mise à jour du stock
- Une lecture du code barre d'un article est opérationnelle
- Le comptage automatique est fonctionnel
- La visualisation des mouvements est possible
- La communication avec le SE permet la récupération des pesées
- La gestion des balances est fonctionnelle (pesées, tarage)

Ressource logicielles et matérielles

Les ressources matérielle:

Désignation	Caractéristiques
Raspberry PI	Raspberry PI 3
Écran	Écran tactile 7" 800x480
lecteur code-barres	Honeywell 1400G2D

Les logiciels:

Désignation	Caractéristiques
Système d'exploitation	GNU/Linux Raspbian
Système de gestion de base de données	MySQL
Atelier de génie logiciel	Bouml v7.9
Logiciel de gestion de version	subversion (RiouxSVN)
Générateurs de documentation	Doxygen version 1.8
Environnement de développement	Qt Creator et Qt Designer
API GUI	Qt 5.11.2

Planification

Répartition des tâches

Tâches à réaliser	Priorité	Itération
Mettre à jour le stock	haute	1
Comptage automatique	haute	1
Communiquer avec le SE	haute	1
Mettre à jour les mouvements	moyenne	2
Lecteur code barre	moyenne	2
consulter les mouvements	moyenne	3

Gantt

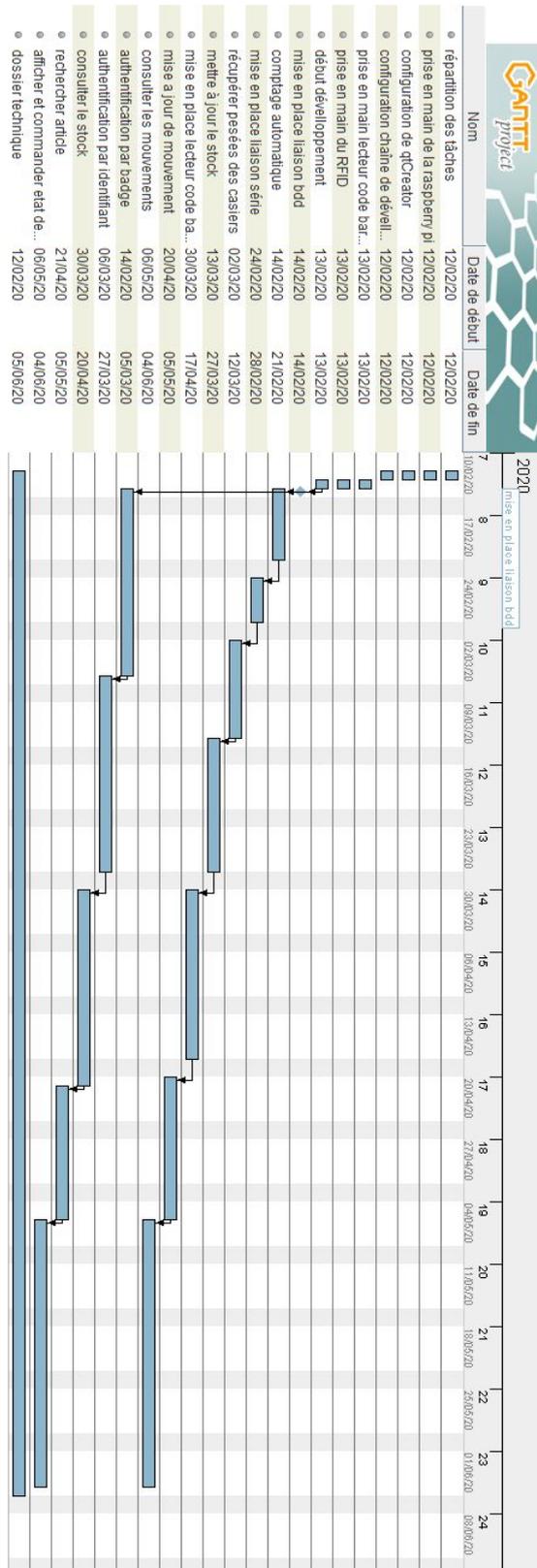
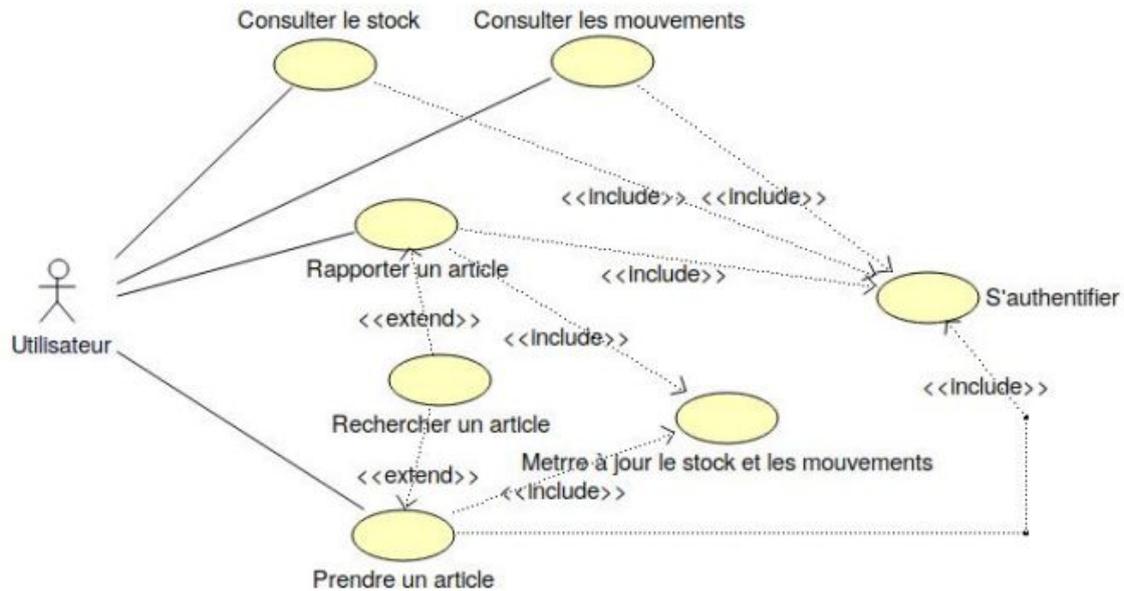


Diagramme de cas d'utilisation



Pour la partie personnelle de l'étudiant, les cas d'utilisation sont :

- **Prendre un article et Rapporter un article** : Mise en place d'un comptage automatique dans les casiers et mise en place d'un lecteur code barre afin d'assurer une identification de l'article et son comptage
- **Mettre à jour le stock et les mouvements** : mettre à jour la base de données du stock et des mouvements lorsque qu'un objet a été pris ou ramené dans un des casiers
- **Consulter les mouvements** : l'utilisateur doit avoir la possibilité de consulter les mouvements (prendre ou rapporter un article) effectués dans l'armoire

Protocole de communication

Le protocole **e-stock** définit l'ensemble des trames permettant de communiquer entre le SE (Système Embarqué EC) et la Raspberry Pi (IR). Il est orienté ASCII.

Le protocole **e-stock** est basée sur des trames requête/réponse.

Format des trames

Les trames sont composées d'un en-tête pour identifier leur type. L'en-tête est "**CASIERS**"

Le délimiteur de fin de champ est le ';'.

Le délimiteur de fin de trame "**\r\n**"

Trame de requête Raspberry Pi → SE

CASIERS;requête;n_casier;\r\n

n_casier ⇒ 1 à 8 et 0 = *broadcast*

requête ⇒ 3 pour une demande de mesure de poids

Exemple, dans le cas d'une demande de mesure de poids pour tous les casiers :

CASIERS;3;0;\r\n

Trame de réponse SE → Raspberry Pi :

CASIERS;réponse;n_casier;donnée;\r\n

n_casier ⇒ 1 à 8

Requête	réponse	donnée	Remarque
pour une demande de mesure de poids	3	poids	en grammes

Remarque : dans le cas d'une requête en *broadcast*, les champs **n_casier;donnée;** sont répétés pour chaque casier.

Exemples dans le cas d'une réponse à une demande de poids:

CASIERS;3;1;1524;\r\n

Liaison Série

Afin de mettre en place la liaison série avec le système embarquée, nous avons créer une classe Communication pour s'occuper de cela.

Communication
<ul style="list-style-type: none"> - port : QSerialPort - trameBrute : QString - nomPort : QString
<ul style="list-style-type: none"> + Communication(inout parent : QObject = nullptr) + ~Communication() + demarrerCommunicationPort() : void + arreterCommunicationPort() : void + configurerPort() : void + ouvrirPort() : void + setNomPort(in nouveauPortSerie : QString) : void + envoyerTrame(in trame : QString) : void + envoyerRequetePoid(in numeroCasier : QString = 0) : void + recevoirTrame() : void - verifierTrame(in trame : QString) : bool - traiterTrame(in trame : QString) : void - envoieTrameOuverture(in trame : QString) : void - envoieTrameEtat(in trame : QString) : void - envoieTramePoids(in trame : QString) : void

Cette classe possède des méthodes qui permettent de gérer efficacement les différentes partis de la communication avec la liaison série. jetons un oeil sur quelques-une de ses fonctions.

```
void Communication::configurerPort()
{
    qDebug() << Q_FUNC_INFO;    port->setPortName(nomPort);
    port->setBaudRate(QSerialPort::Baud9600);
    port->setDataBits(QSerialPort::Data8);
    port->setParity(QSerialPort::NoParity);
    port->setStopBits(QSerialPort::OneStop);
    port->setFlowControl(QSerialPort::NoFlowControl);
}
```

La méthode `configurerPort()` permet de configurer la liaison série. Ainsi, il est possible de configurer les différents paramètres de la liaison comme par exemple, la vitesse de transmission, le nom du port ou encore la présence ou non d'un bit de parité.

Cette méthode nous permet donc de configurer notre liaison série comme nous l'avons définie en collaboration avec l'étudiant de la partie EC du projet.

```
void Communication::envoyerTrame(QString trame)
{
    if (port->isOpen())
    {
        port->write(trame.toLatin1());
    }
}
```

La méthode `envoyerTrames`, nous permet d'envoyer une trame sur la liaison série et ainsi communiquer avec le système embarqués. Nous utilisons cette méthode lorsque nous avons besoin de faire une requête aux système embarqués comme par exemple une requête de poids.

La méthode reçoit en paramètres un `QString` qui correspond au message à envoyer sur la liaison série.

Ensuite, on vérifie si le port est bien ouvert et si cela est le cas, on envoie la trame grâce à la méthode `write` de la classe `QSerialPort` de `qt`.

```
void Communication::recevoirTrame()  
{  
    trameBrute = "\\0";  
  
    while (port->waitForReadyRead(500))  
    {  
        trameBrute.append(port->readAll());  
    }  
  
    if(verifierTrame(trameBrute))  
        traiterTrame(trameBrute);  
}
```

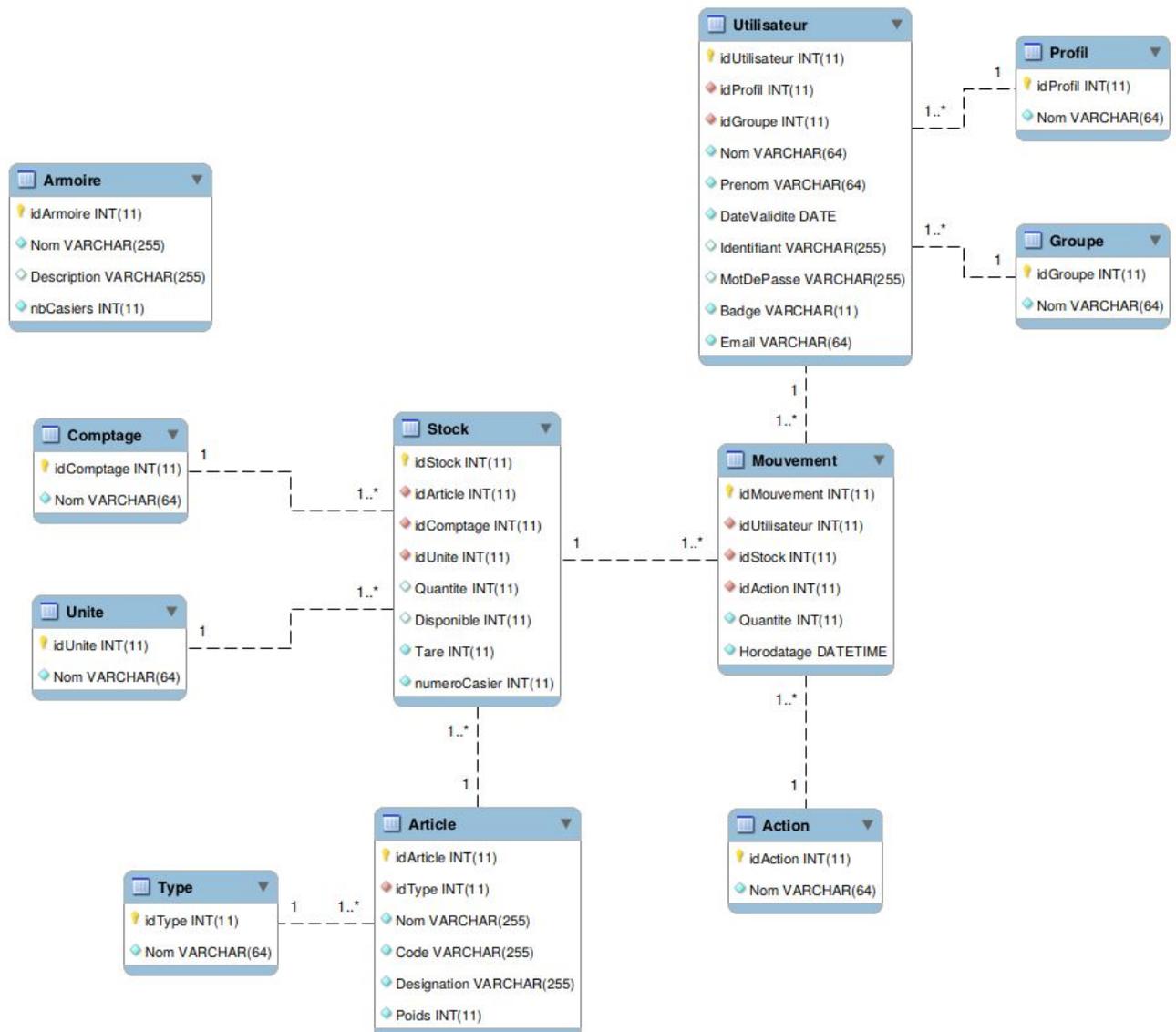
La méthode recevoirTrame est la méthode qui nous permet de récupérer les trames qui nous sont envoyés via la liaison série.

Quand une trame est reçue, la fonction attend 0.5 seconde (500 millisecondes), pendant ce temps, la trame est mis dans un QString.

Une fois cela terminé, on vérifie si la trame est valide grâce à la méthode verifierTrame, et si telle est le cas alors on traite la trame grâce à la méthode traiterTrame.

Base de données

Structure de la base de données e-stock



La table **Armoire** définit les caractéristiques d'une armoire :

- un champ Nom (le nom de l'armoire)
- un champ Description
- un champ nbCasiers (par défaut il est défini à 8)

La table **Comptage** est caractérisée par un champ Nom (Aucun, Automatique, CodeBarre) est définie le type de comptage utilisé.

La table **Profil** est caractérisée par un champ Nom (Administrateur, Gestionnaire, Utilisateur)

La table **Groupe** est caractérisée par un champ Nom (PROFESSEUR, 1-BTS-SN, T-BTS-SN)

La table **Utilisateur** contient toutes les informations des différents utilisateurs:

- une clé étrangère idProfil qui précise son profil
- une clé étrangère idGroupe qui précise son groupe
- un champ Nom
- un champ Prenom
- un champ DateValidite
- un champ Identifiant
- un champ MotDePasse
- un champ Badge
- un champ Email

La table **Unite** est caractérisée par un Nom (Metre, Piece, Pourcentage, Poids g, Poids kg)

La table **Type** est caractérisée par un Nom (Equipement, Consommable)

La table **Action** est caractérisée par un Nom (Entree, Sortie)

La table **Article** contient toutes les informations relative à un article:

- une clé étrangère idType qui précise son type
- un champ Nom
- un champ Code
- un champ Designation
- un champ Poids

La table **Stock** contient les informations de ce qui est contenue dans les casiers:

- une clé étrangère idArticle qui précise l'article
- une clé étrangère idComptage qui précise le comptage
- une clé étrangère idUnite qui précise son unité
- un champ Quantite
- un champ Disponible
- un champ Tare
- un champ numeroCasier

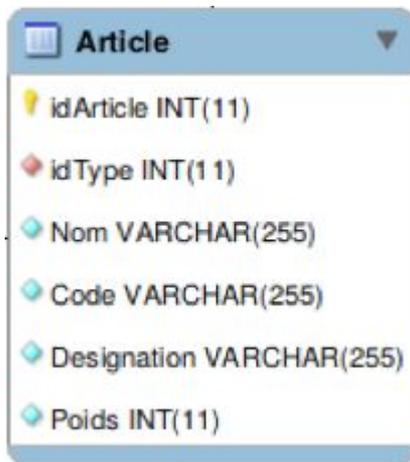
La table **Mouvement** contient les informations des mouvements effectué:

- une clé étrangère idUtilisateur qui précise l'utilisateur
- une clé étrangère idStock qui précise le stock ou à était effectué le mouvement

- une clé étrangère idAction qui précise l'action
- un champ Quantite
- un champ Horodatage

Exemple de création et de remplissage d'une table

Nous prendrons pour exemple la table Article:



Requête SQL de création de la table Article :

```
CREATE TABLE IF NOT EXISTS `Article` (  
  `idArticle` int(11) NOT NULL AUTO_INCREMENT,  
  `idType` int(11) NOT NULL,  
  -- `Type` enum('Equipement','Consommable'),  
  `Nom` varchar(255) NOT NULL,  
  `Code` varchar(255) NOT NULL,  
  `Designation` varchar(255) NOT NULL,  
  `Poids` int(11) NOT NULL,  
  PRIMARY KEY (`idArticle`),  
  CONSTRAINT Article_fk_1 FOREIGN KEY (`idType`) REFERENCES  
  Type(`idType`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Requête SQL de "remplissage" de la table Article :

```
INSERT INTO `Article` (`idType`, `Nom`, `Code`, `Designation`, `Poids`)
VALUES('2','Vis six pans creux M2 8mm','','M2*8', 50);
INSERT INTO `Article` (`idType`, `Nom`, `Code`, `Designation`, `Poids`)
VALUES('2','Vis tête cylindrique M2 8mm','','M2*8', 50);INSERT INTO
`Article` (`idType`, `Nom`, `Code`, `Designation`, `Poids`)
VALUES('2','Vis six pans creux M2 12mm','','M2*12', 50);
INSERT INTO `Article` (`idType`, `Nom`, `Code`, `Designation`, `Poids`)
VALUES('2','Vis tête cylindrique M2 12mm','','M2*12', 50);
INSERT INTO `Article` (`idType`, `Nom`, `Code`, `Designation`, `Poids`)
VALUES('1','Fluke i30s','2584935','Amperemetre AC/DC', 100);
INSERT INTO `Article` (`idType`, `Nom`, `Code`, `Designation`, `Poids`)
VALUES('1','Fluke 179','','Multimetre', 150);
```

Requête SQL de sélection d'un enregistrement de la table Article :

```
SELECT * FROM `Article` FROM `idArticle` = '2';
```

Utiliser la base de données dans le programme

Pour utiliser la base de données e-stock dans notre projet, nous disposons d'une classe **Bdd** :

Bdd
- db : QSqlDatabase
- nbAcces : int
+ <u>getInstance() : Bdd</u>
+ <u>destruireInstance() : void</u>
+ connecter() : bool
+ estConnecte() : bool
+ executer(in requete : QString) : bool
+ recuperer(in requete : QString, inout donnees : QString) : bool
+ recuperer(in requete : QString, inout donnees : QStringList) : bool
+ recuperer(in requete : QString, inout donnees : QVector<QString>) : bool
+ recuperer(in requete : QString, inout donnees : QVector<QStringList>) : bool
- Bdd()
- ~Bdd()

Cette classe permet l'exécution de requête SQL **UPDATE**, **INSERT** et **DELETE** grâce à la méthode **executer()** :

```
bool Bdd::executer(QString requete)
{
    QSqlQuery r;
    bool retour;
    if(db.isOpen())
    {
        if(requete.contains("UPDATE") ||
requete.contains("INSERT") || requete.contains("DELETE"))
        {
            retour = r.exec(requete);
            #ifdef DEBUG_BASEDEDONNEES
            qDebug() << Q_FUNC_INFO << QString::fromUtf8("Retour
%1 pour la requete :
%2").arg(QString::number(retour)).arg(requete);
            #endif
            if(retour)
            {
                return true;
            }
            else
            {
                qDebug() << Q_FUNC_INFO <<
QString::fromUtf8("Erreur : %1 pour la requête
%2").arg(r.lastError().text()).arg(requete);
                return false;
            }
        }
        else
        {
            qDebug() << Q_FUNC_INFO << QString::fromUtf8("Erreur :
requête %1 non autorisée !").arg(requete);
            return false;
        }
    }
    else
        return false;
}
```

Cette classe permet aussi d'exécuter des requêtes de type **SELECT** afin de récupérer des données présente dans la base de données e-stock.

Ceci est possible grâce à quatre méthodes surchargées nommées **récupérer()** :

- Permet de récupérer un seul champ d'un seul enregistrement : `QString`

```
bool recuperer(QString requete, QString &donnees);
```

- Permet de récupérer plusieurs champs d'un seul enregistrement : `QStringList`

```
bool recuperer(QString requete, QStringList &donnees);
```

- Permet de récupérer un seul champ de plusieurs enregistrements : `QVector<QString>`

```
bool recuperer(QString requete, QVector<QString> &donnees);
```

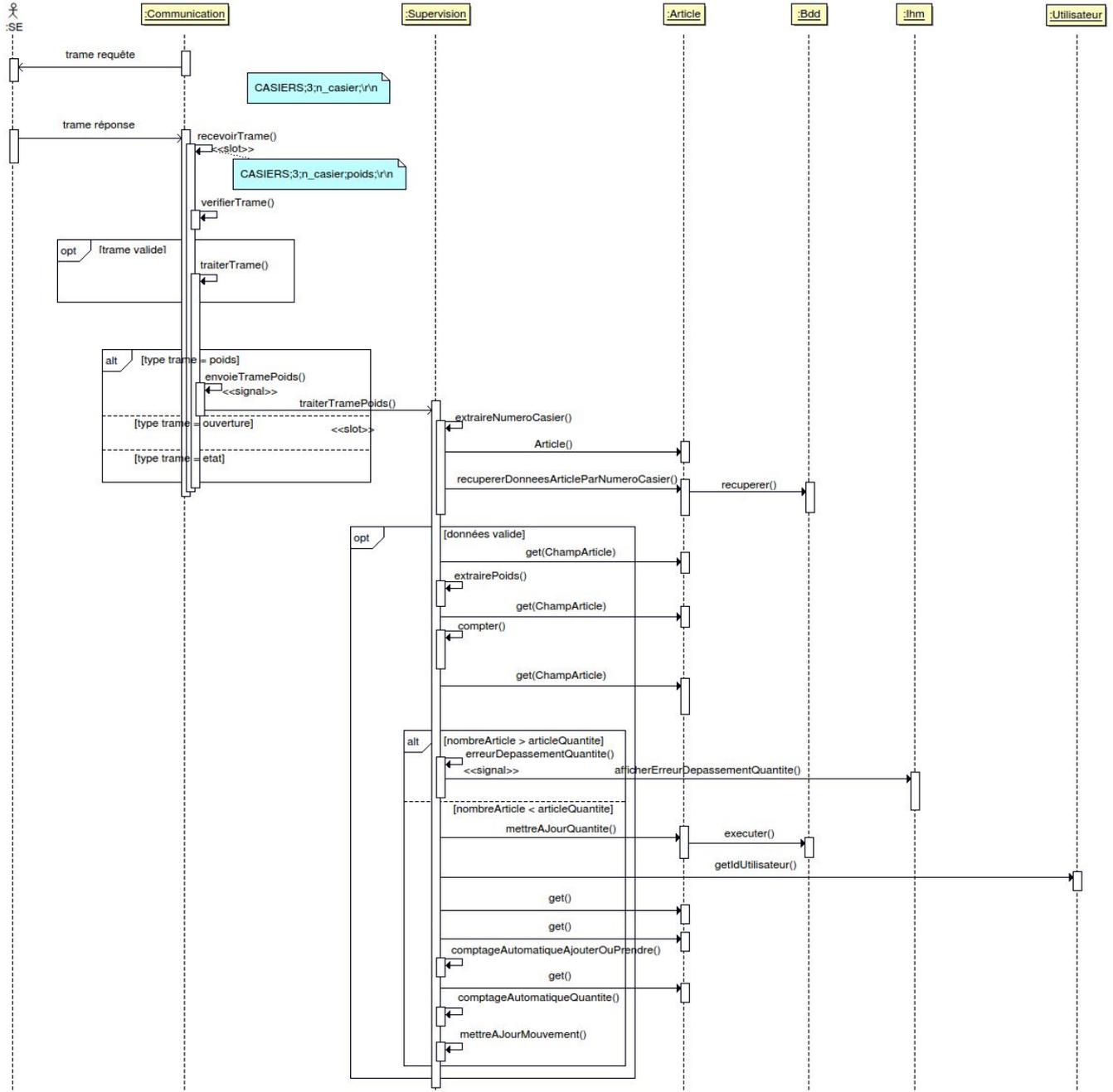
- Permet de récupérer plusieurs champs de plusieurs enregistrements : `QVector<QStringList>`

```
bool recuperer(QString requete, QVector<QStringList> &donnees);
```

La méthodes **executer()** ainsi que les méthodes **recuperer()** retourne tous un booléen qui permet de dire si la requête a été exécutée avec succès ou non.

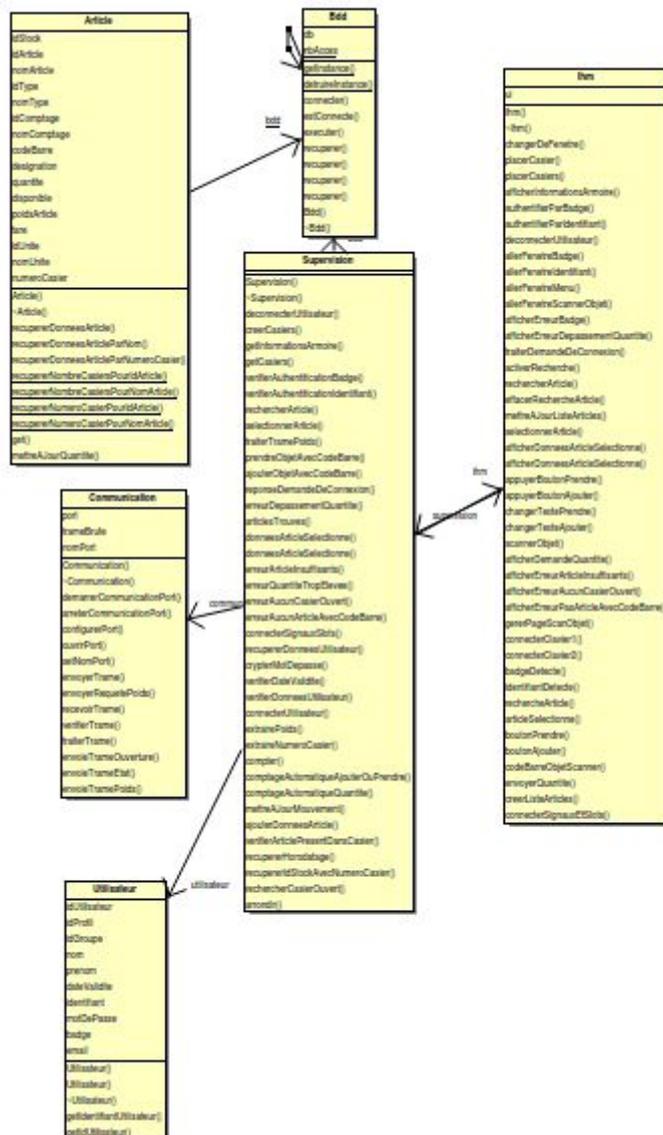
Mettre à jour le stock a partir d'une trame de poids

Scénario Mettre à jour le stock



Ce diagramme de séquence montre comment le stock est mis à jour à jour à partir des trames de poids reçues.

Le diagramme de classe de ce scénario est le suivant :



La méthode **traiterTramePoids()** va permettre de récupérer les données contenues dans la trame puis de les utiliser pour mettre à jour le stock. Pour cela, elle va d'abord extraire le numéro du casier de la trame, pour ensuite récupérer toutes les données de ce casier dans une instance de la classe Article. Elle va ensuite extraire le poids de la trame afin d'effectuer le comptage automatique et pour finir mettre à jour le stock.

```

void Supervision::traiterTramePoids(QString trame)
{
    #ifdef DEBUG_SUPERVISION
        qDebug() << Q_FUNC_INFO << trame;
    #endif
}
  
```

```
QString numCasier = extraireNumeroCasier(trame);

Article *article = new Article(this);
if(article->recupererDonneesArticleParNumeroCasier(numCasier))
{
    #ifdef DEBUG_SUPERVISION
        qDebug() << Q_FUNC_INFO << "Article" <<
article->get(TABLE_ARTICLE_NOM_ARTICLE) << article->get(TABLE_ARTICLE_QUANTITE)
<< article->get(TABLE_ARTICLE_DISPONIBLE);
    #endif

    int nombreArticle = compter(article->get(TABLE_ARTICLE_POIDS),
extrairePoids(trame), article->get(TABLE_ARTICLE_TARE));

    QString strArticleQuantite = article->get(TABLE_ARTICLE_QUANTITE);
    int articleQuantite = strArticleQuantite.toInt();

    if(nombreArticle > articleQuantite)
    {
        emit erreurDepassementQuantite();
    }
    else
    {
        article->mettreAJourQuantite(QString::number(nombreArticle));
    }
}
else
{
    #ifdef DEBUG_SUPERVISION
        qDebug() << Q_FUNC_INFO << "Article introuvable !";
    #endif
}
}
```

Exemple de requête SQL

Pour que la mise à jour du stock soit effectuée, il est nécessaire de faire des requêtes à la base de données **e-stock** pour récupérer ou modifier certaines données.

Récupérer les données d'un article dans une armoire

Lors du traitement d'une trame de poids, il est nécessaire d'instancier un objet Article. Pour cela il est nécessaire de récupérer les informations nécessaire grâce à la requête suivante.

```
SELECT Stock.idStock, Article.idArticle, Article.Nom AS Article,
Type.idType, Type.nom AS Type, Comptage.idComptage, Comptage.Nom
AS Comptage, Article.Code, Article.Designation, Stock.Quantite,
Stock.Disponible, Article.Poids, Stock.Tare, Unite.idUnite,
Unite.Nom, Stock.numeroCasier FROM Stock
INNER JOIN Article ON Article.idArticle=Stock.idArticle
INNER JOIN Type ON Type.idType=Article.idType
INNER JOIN Comptage ON Comptage.idComptage=Stock.idComptage
INNER JOIN Unite ON Unite.idUnite=Stock.idUnite
WHERE Article.idArticle = '5'
```

Pour effectuer cette requête, il est nécessaire de préciser l'id de l'article pour lequel nous voulons récupérer les données. Dans cet exemple l'id de l'article est '5'.

Ce que l'on obtient quand on récupère les données d'un article :

```
QStringList(("1", "5", "Fluke i30s", "1", "Equipement", "3",
"CodeBarre", "2584935", "Amperemetre AC/DC", "8", "6", "100",
"1500", "2", "Piece", "1"))
```

Mettre à jour le stock dans une armoire

A la fin du traitement d'une trame de poids, il est nécessaire de mettre à jour le stock dans la base de données **e-stock**. Pour cela il faut exécuter la requête SQL suivante.

Exemple pour l'article avec l'id '5' est une quantité à changer à 4 :

```
UPDATE Stock SET Disponible = 4 WHERE idArticle = 5;
```

Le comptage automatique

Pour une grande partie des objets stockés dans l'armoire e-stock, la mise à jour du nombre d'objets présents dans l'armoire doit s'effectuer automatiquement.

Pour cela, il a fallu mettre en place un comptage automatique du nombre d'objets présents dans les casiers grâce au poids mesuré par la balance présente à l'intérieur.

La méthode **compter()** permet d'effectuer ce comptage automatique. Pour être utilisé, il est nécessaire de passer en paramètres de cette fonction le poids d'un article, le poids total pesé et la tare du casier en question. Elle va d'abord convertir les paramètres en double. Elle effectuera ensuite le comptage en fonction du poids de l'article, du poids pesé et de la tare du casier.

La méthode **compter()** :

```
int Supervision::compter(QString poidsArticle, QString poidsTotal,
QString tare)
{
    double doublePoidsArticle = poidsArticle.toDouble();
    double doublePoidsTotal = poidsTotal.toDouble();
    double doubleTare = tare.toDouble();

    //comptage du nombre d'articles

    double doubleNombreArticle = qRound((doublePoidsTotal - doubleTare) /
doublePoidsArticle);
    QString strNombreArticle = QString::number(doubleNombreArticle, 'f',
PRECISION);
    int nombreArticle = strNombreArticle.toInt();

    #ifdef DEBUG_SUPERVISION
        qDebug() << Q_FUNC_INFO << "nombreArticle:" << nombreArticle;
    #endif

    return nombreArticle;
}
```

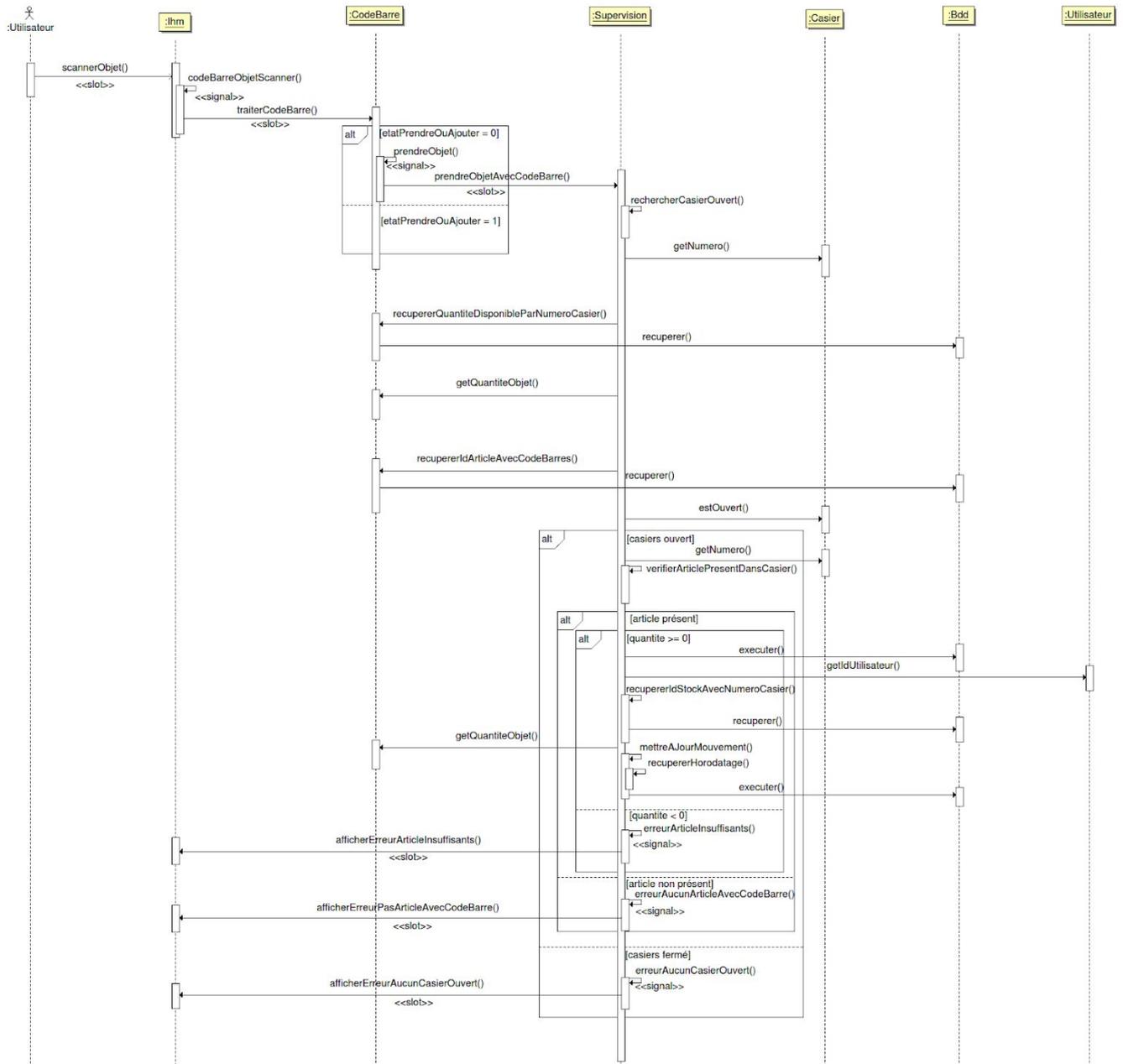
Test de mise à jour du stock

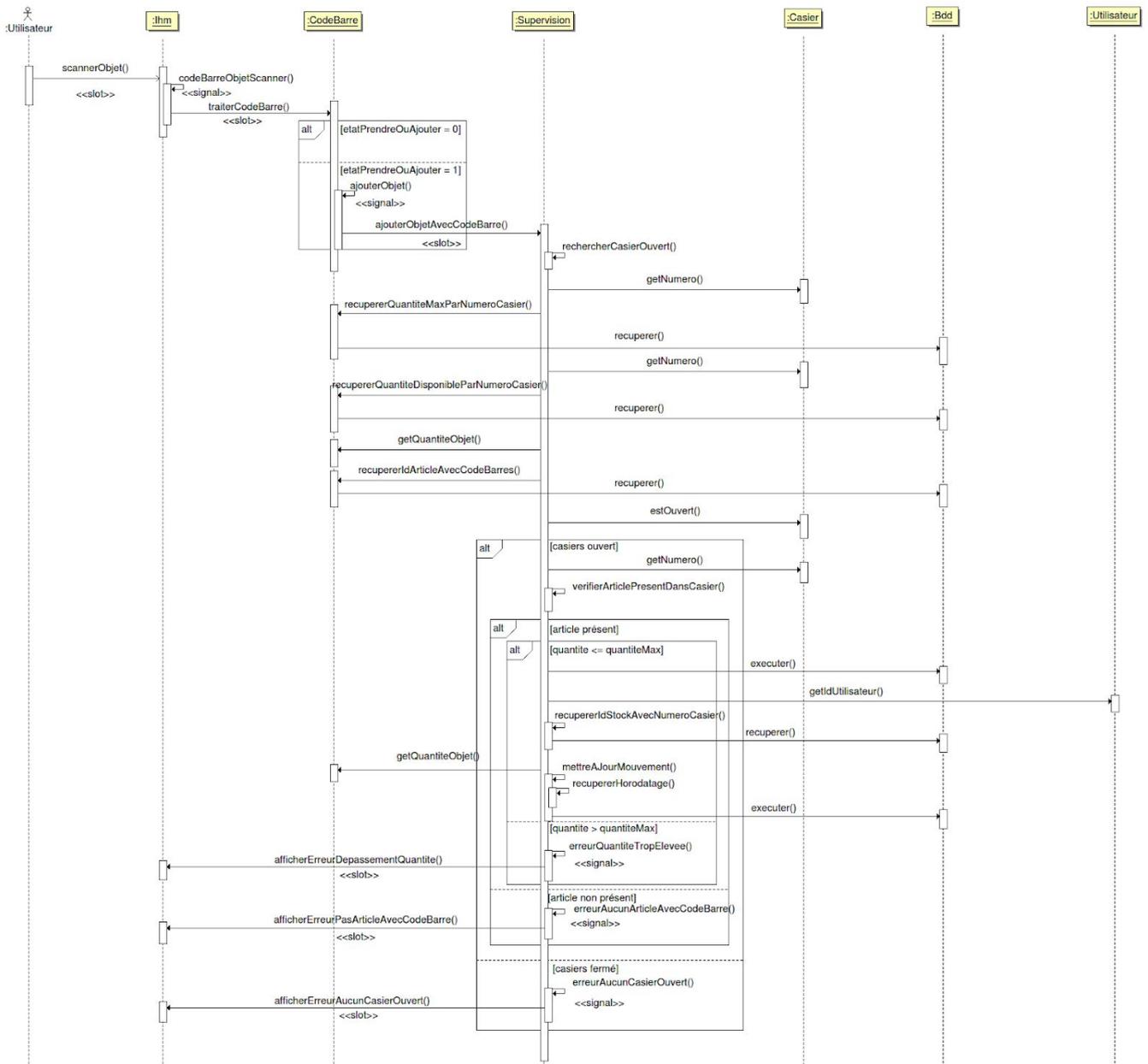
Test	Nombre d'objet	Résultat attendu	Résultat obtenu	Valide
nombre d'objet disponible inférieur au nombre d'objet maximum dans le casier	nombre d'objet disponibles = 5 nombre d'objet maximum =10	Le stock doit bien être mis à jour	Le stock est bien mis à jour	oui
nombre d'objet disponible égal au nombre d'objet maximum dans le casier	nombre d'objet disponibles = 10 nombre d'objet maximum =10	Le stock doit bien être mis à jour	Le stock est bien mis à jour	oui
nombre d'objet disponible supérieur au nombre d'objet maximum dans le casier	nombre d'objet disponibles = 15 nombre d'objet maximum =10	Le stock ne doit pas être mis à jour et un message d'erreur doit être affiché pour signaler qu'il y a trop d'objet dans le casier	Le stock n'est pas mis à jour et un message d'erreur s'affiche pour signaler qu'il y a trop d'objet dans le casier	oui

Mise en place du lecteur code-barres

Scénario lecteur code-barres

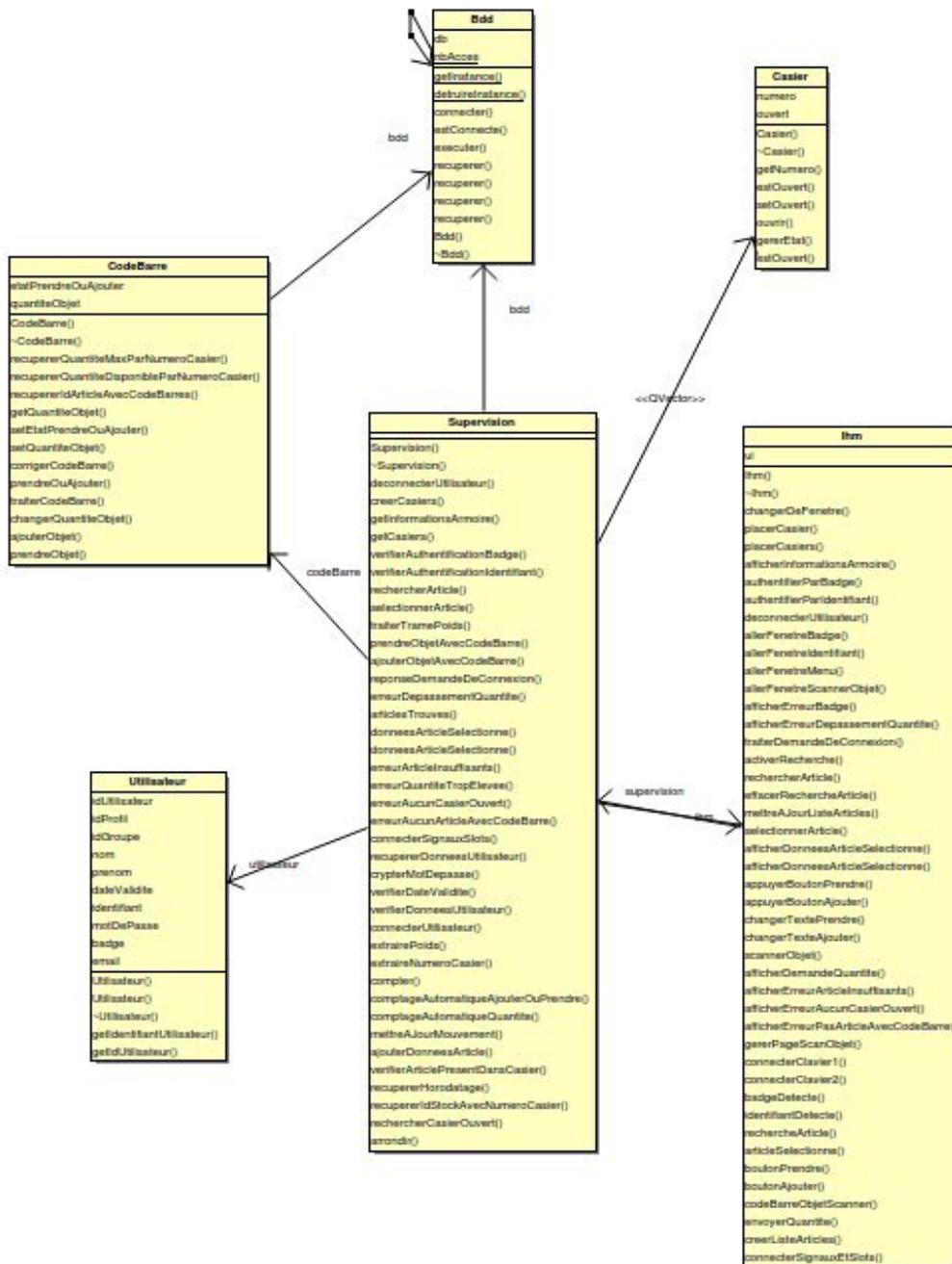
Pour cette partie, nous avons deux scénarios possible, le premier est le scénario dans lequel l'utilisateur se sert du lecteur code-barres pour prendre un objet de l'armoire. Le deuxième est celui où l'utilisateur se sert du lecteur code-barres pour ajouter un objet dans l'armoire.





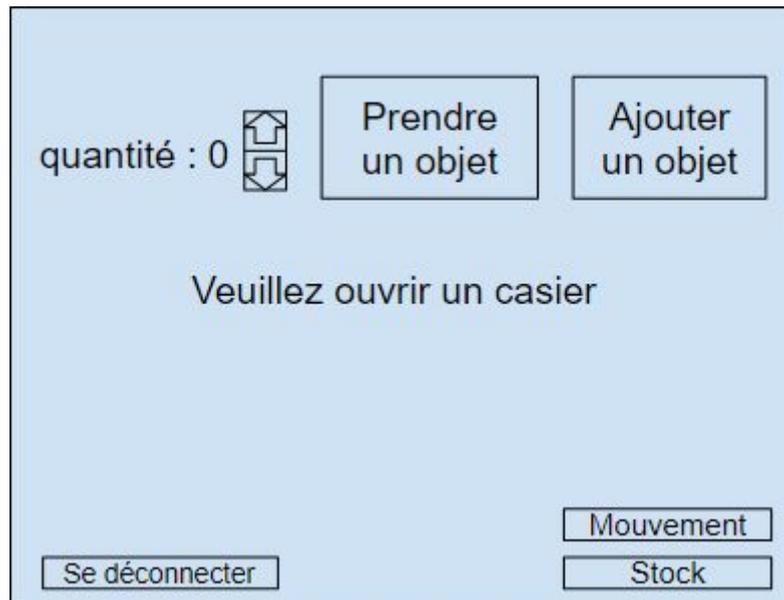
Ces deux diagramme de séquence nous montre le déroulement du processus dans le code pour l'utilisation du lecteur code-barres à des fin de prise d'objet (1er diagramme) ou d'ajout d'objet (2ème diagramme) dans l'armoire

Le diagramme de classe pour pour le scénario d'utilisation du lecteur code-barres est le suivant:

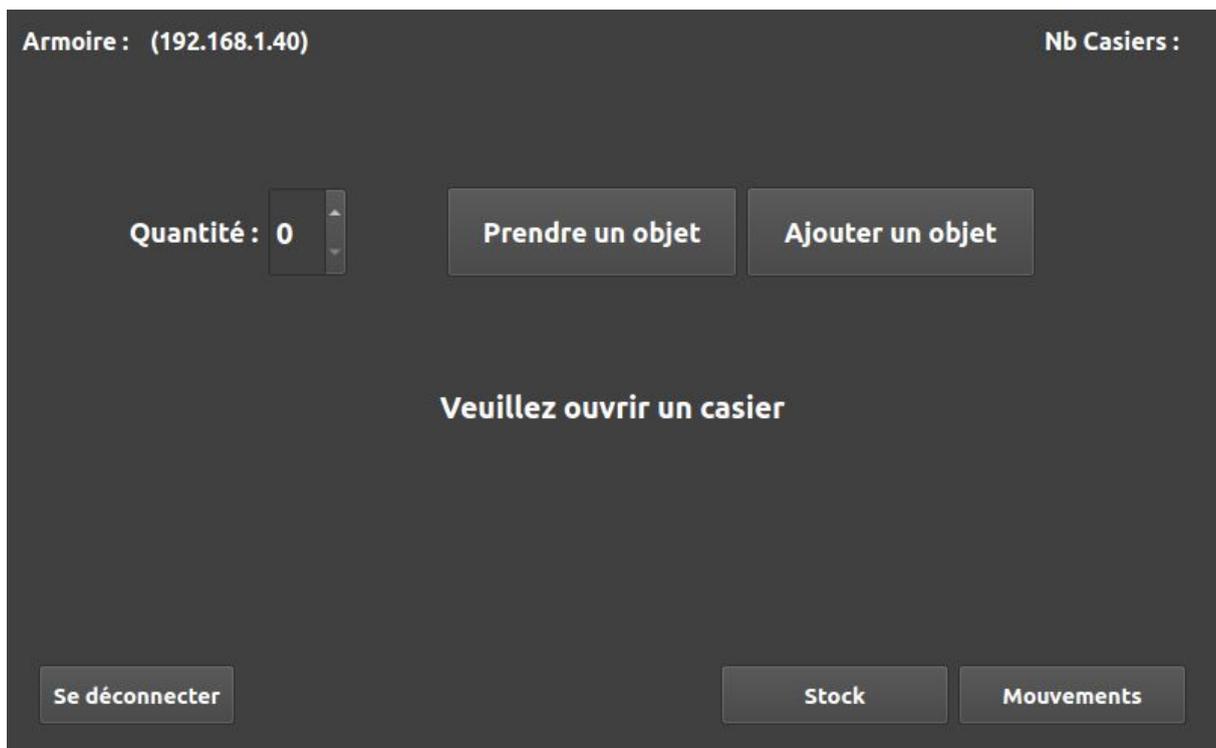


Interface

Avant de commencer la conception de l'interface dans le logiciel, il a fallu faire une maquette afin de savoir comment nous voulions concevoir l'interface. La maquette de l'interface est la suivante:



Une fois la maquette terminée, il a fallu la mettre en place dans le logiciel, le rendu final dans le logiciel est le suivant :



Gestion du lecteur code-barres

Afin de gérer le lecteur code-barres, il y a deux fonctions principal. Une pour s'occuper d'ajouter un objet et une autre pour s'occuper de prendre un objet. La différence si un utilisateur veut ajouter ou prendre un objet se fait grâce au deux boutons sur lequel il peut appuyer sur l'interface selon si il souhaite ajouter ou prendre un objet. Et la gestion de la quantité à ajouter ou à prendre se fait aussi sur l'interface.

Nous allons détailler une seul de ces deux fonctions étant donné que les deux méthodes sont très similaire, ici nous verrons la fonction permettant d'ajouter un objet.

```
void Supervision::prendreObjetAvecCodeBarre(QString codeBarre)
{
    int numeroCasier = rechercherCasierOuvert();

    if(numeroCasier == -1)
        return;

    unsigned int quantiteDisponible =
this->codeBarre->recupererQuantiteDisponibleParNumeroCasier(QString::number(casiers[numeroCasier]-
>getNumero()));
    int quantite = quantiteDisponible - this->codeBarre->getQuantiteObjet();
    QString idArticle =
QString::number(this->codeBarre->recupererIdArticleAvecCodeBarres(codeBarre));

    if(casiers[numeroCasier]->estOuvert())
    {
        if(verifierArticlePresentDansCasier(QString::number(casiers[numeroCasier]->getNumero()),
idArticle))
        {
            if(quantite >= 0)
            {
                QString strQuantite = QString::number(quantite);
                QString requete = "UPDATE Stock SET Disponible = '" + strQuantite + "' WHERE
Stock.idArticle = '" + idArticle + "'";
                bdd->executer(requete);
                QString idUtilisateur = utilisateur->getIdUtilisateur();
                QString idStock = recupererIdStockAvecNumeroCasier(numeroCasier+1 );
                QString idAction = "1";
                QString quantiteMouvement = QString::number(this->codeBarre->getQuantiteObjet());
                mettreAJourMouvement(idUtilisateur, idStock, idAction, quantiteMouvement);
            }
            else
            {
                erreurArticleInsuffisants();
            }
        }
    }
}
```

```
    else
    {
        emit erreurAucunArticleAvecCodeBarre();
    }
}
else
{
    erreurAucunCasierOuvert();
}
}
```

Tout d'abord, nous avons en paramètres le code barres ayant été scanner par l'utilisateur. On recherche d'abord le casier qui est actuellement ouvert grâce à la méthode `rechercherCasierOuvert`.

Nous avons ensuite besoin de récupérer la quantité d'objet actuellement disponible dans le casier, cela se fait grâce à la fonction `recupererQuantiteDisponibleParNumeroCasier` de la classe `codeBarre`

```
unsigned int
CodeBarre::recupererQuantiteDisponibleParNumeroCasier(QString
numeroCasier)
{
    QString requete = "SELECT Stock.Disponible FROM Stock WHERE
Stock.numeroCasier = '" + numeroCasier + "'";

    QString donnees;
    bdd->recuperer(requete, donnees);

    return donnees.toUInt();
}
```

On calcule ensuite la nouvelle quantité après avoir récupérer le nombres d'objet souhaiter. et on récupère ensuite l'id de l'article présent dans le casier grâce à la méthode `recupererIdArticleAvecCodeBarres` de la classe `codeBarre`.

```
unsigned int
CodeBarre::recupererQuantiteDisponibleParNumeroCasier(QString
numeroCasier)
{
    QString requete = "SELECT Stock.Disponible FROM Stock WHERE
Stock.numeroCasier = '" + numeroCasier + "'";

    QString donnees;
    bdd->recuperer(requete, donnees);
```

```
    return donnees.toUInt();  
}
```

Nous possédons donc maintenant toute les données dont nous avons besoin pour effectuer le traitement de la demande.

On regarde donc d'abord si un casiers est ouvert, si cela n'est pas le cas on affiche un message d'erreur sur l'interface pour signaler à l'utilisateur qu'il n'a pas ouvert de casier.

Si un casier est ouvert, nous vérifions ensuite si l'article que l'utilisateur a scannée est bien présent dans le casier actuellement ouvert. Si cela n'est pas le cas on affiche un message d'erreur pour signaler à l'utilisateur que le casier actuellement ouvert ne correspond pas avec l'objet qu'il vient de scanner.

Et ensuite, dernière vérification, on regarde si la quantité est bien supérieur ou égal à zéro car si cela n'est pas le cas cela veut dire que l'utilisateur demande à prendre plus d'objet que ce qui est actuellement disponible dans le casier. Dans ce cas on affiche donc un message d'erreur à l'utilisateur pour le signaler.

Une fois toutes ces vérifications effectuées. nous pouvons donc valider le retrait de l'objet et mettre à jour la base de données en exécutant une requête SQL.

```
    QString requete = "UPDATE Stock SET Disponible = '" +  
    strQuantite + "' WHERE Stock.idArticle = '" + idArticle + "'";  
    bdd->executer(requete);
```

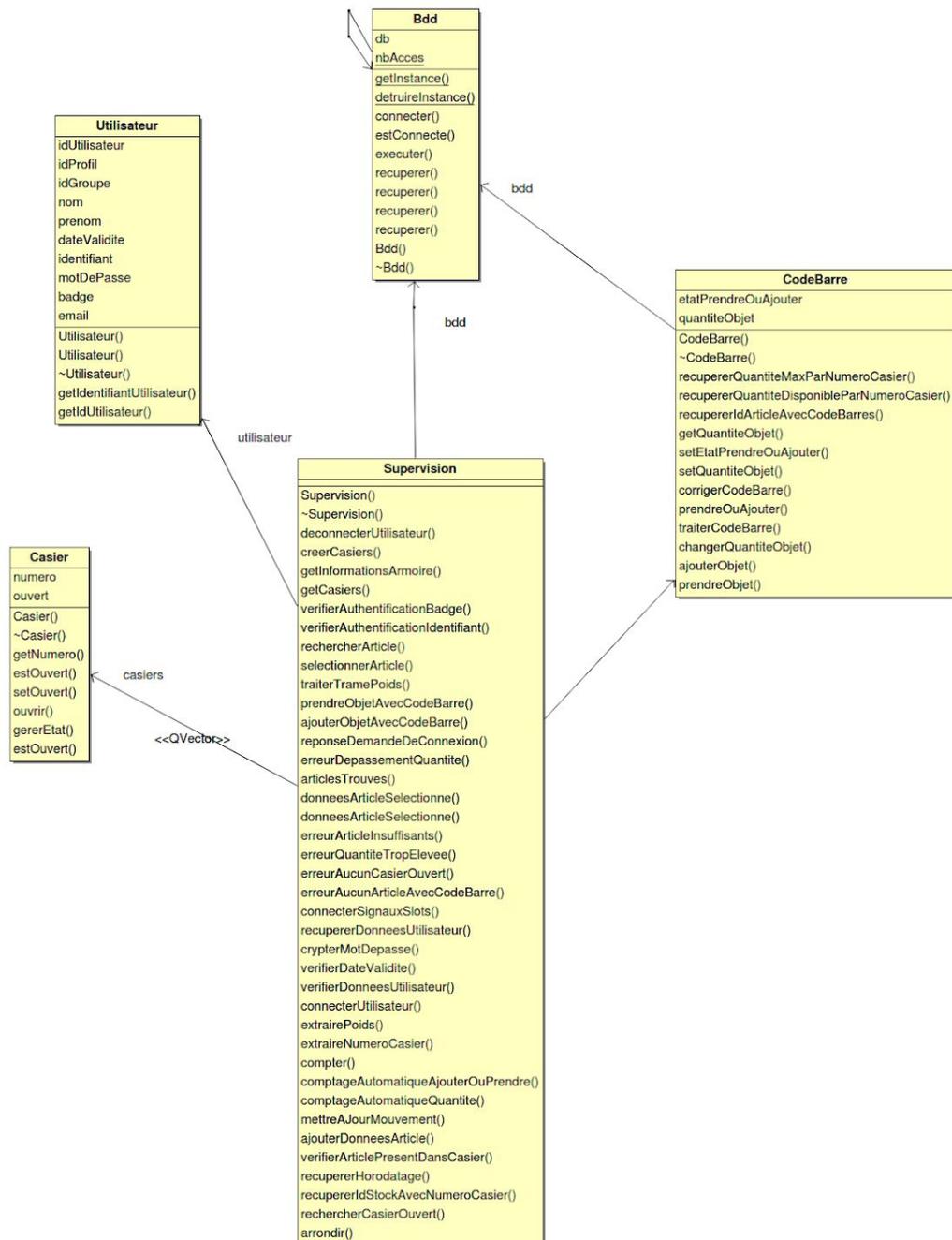
Test scanner objet

Test	données	Résultat attendu	Résultat obtenu	Valide
aucun casier ouvert	numeroCasier = ""	un message d'erreur s'affiche pour dire que aucun casier n'est ouvert	un message d'erreur s'affiche pour dire que aucun casier n'est ouvert	oui
le code-barres ne correspond pas à l'objet présent dans le casier	code barres scanner = "8452943185237" code barres attendu = "3400936486294"	un message d'erreur s'affiche pour signaler que cette objet n'est pas dans ce casier	un message d'erreur s'affiche pour signaler que cette objet n'est pas dans ce casier	oui
la quantité prise est trop élevée	quantité Disponible = "5" quantité demandée = "7"	un message d'erreur s'affiche pour signaler que la quantité demandée est trop élevée	un message d'erreur s'affiche pour signaler que la quantité demandée est trop élevée	oui
un casier est ouvert	numeroCasier = "1"	Le programme passe à la prochaine vérification	Le programme passe à la prochaine vérification	oui
le code-barres correspond à l'objet dans le casier	code barres scanner = "3400936486294" code barres attendu = "3400936486294"	le programme passe à la suite	le programme passe à la suite	oui
la quantité demandée n'est pas trop élevée	quantité Disponible = "5" quantité demandée = "2"	le programme m'est à jour les données	le programme m'est à jour les données	oui

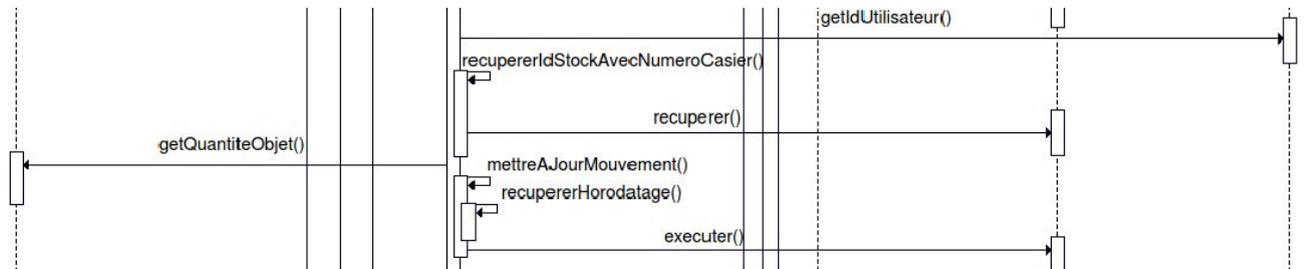
Mise à jour des mouvements

Scénario mise à jour des mouvements

Le diagramme de classe pour ce scénario est le suivant :



La mise à jour des mouvements est utilisée à plusieurs endroits dans le programme comme par exemple lors de l'ajout d'objet grâce au lecteur code-barres comme le montre cette partie du diagramme de séquence de l'ajout d'un objet avec le lecteur code-barres.



Gestion de la mise à jour des mouvements

La mise à jour des mouvements s'effectue grâce à la méthode `mettreAJourMouvements` de la classe `Supervision`.

```

void Supervision::mettreAJourMouvement(QString idUtilisateur, QString
idStock, QString idAction, QString quantite)
{
    QString horodatage = recupererHorodatage();
    QString requete = "INSERT INTO Mouvement(idUtilisateur, idStock,
idAction, Quantite, Horodatage) VALUES('" + idUtilisateur + "', '" +
idStock + "', '" + idAction + "', '" + quantite + "', '" + horodatage +
"' );";
    bdd->executer(requete);
}
  
```

Cette méthode effectue une requête SQL pour mettre à jour la base de données avec les mouvements qu'il vient d'être effectuée. Cela grâce aux paramètres qu'elle reçoit et à la récupération de l'horodatage grâce à la fonction `recupererHorodatage` de la classe `Supervision`.

```

QString Supervision::recupererHorodatage()
{
    QDate qDate(QDate::currentDate());
    QString date = qDate.toString("yyyy-MM-dd");

    QTime time(QTime::currentTime());
    QString heure = time.toString("hh:mm:ss");

    return date + " " + heure;
}
  
```

Cette fonction permet de récupérer la date et l'heure et de mettre tout cela dans le format DATETIME (YYYY-MM-DD hh:mm:ss) pour pouvoir être écrit dans la base de données.

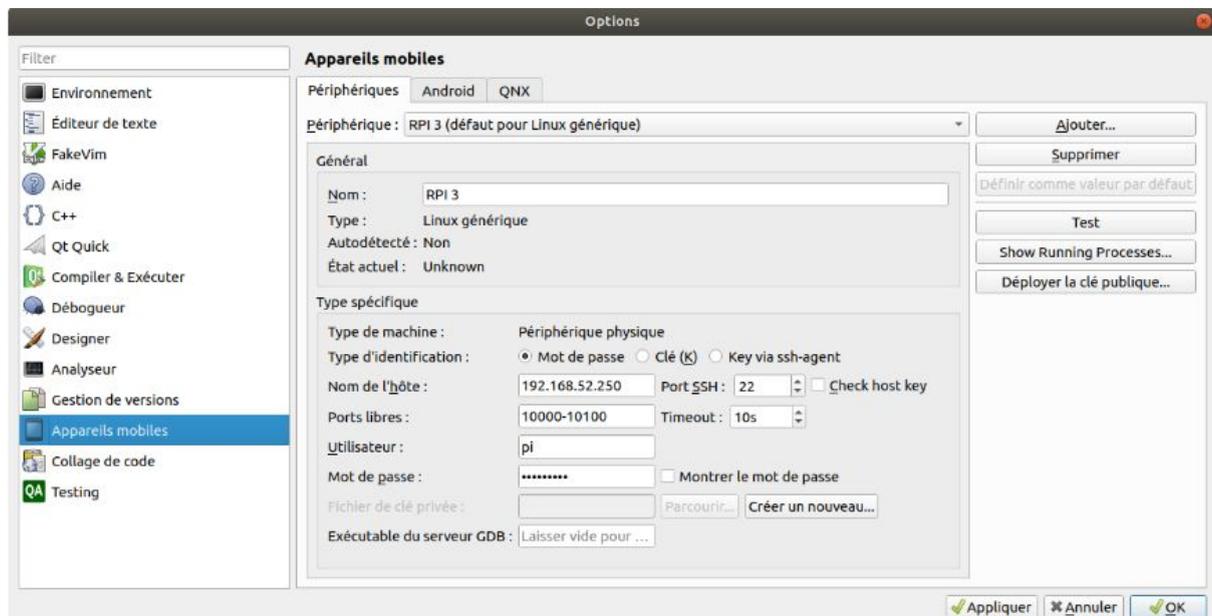
Annexes

Manuel d'installation de Qt pour Raspberry Pi

Dans Qt Creator :

Qt Creator permet de créer, déployer, exécuter et déboguer des applications Qt directement sur la Raspberry Pi en un seul clic.

Menu Options → Appareils mobiles → Périphériques : Ajouter **Périphérique Linux générique**

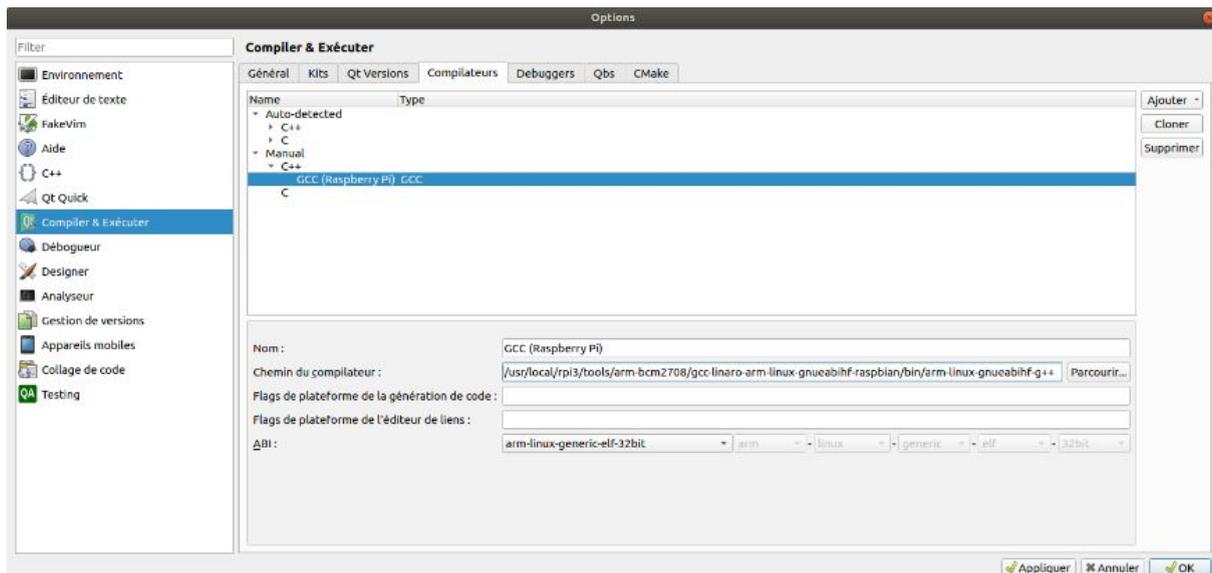


Menu Options → Compiler & Exécuter puis Compilateurs : Ajouter **GCC C++**

Chemin :

`~/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian/bin/arm-linux-gnueabi-hf-g++` ou

`/usr/local/rpi3/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian/bin/arm-linux-gnueabi-hf-g++`

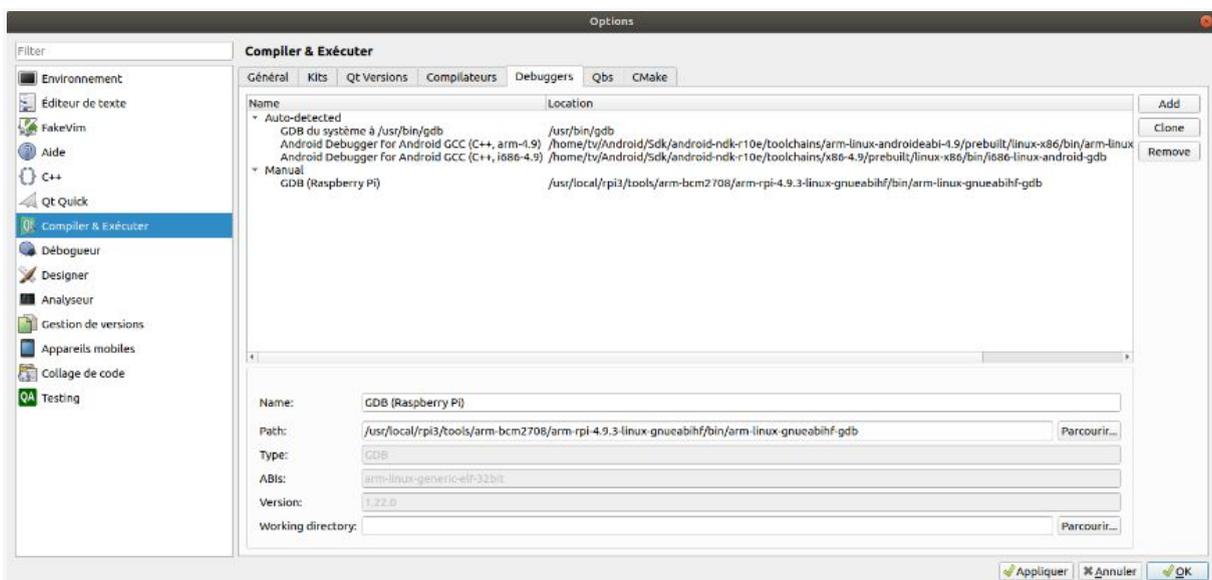


Menu Options → Compiler & Exécuter puis Debuggers : Ajouter

Chemin :

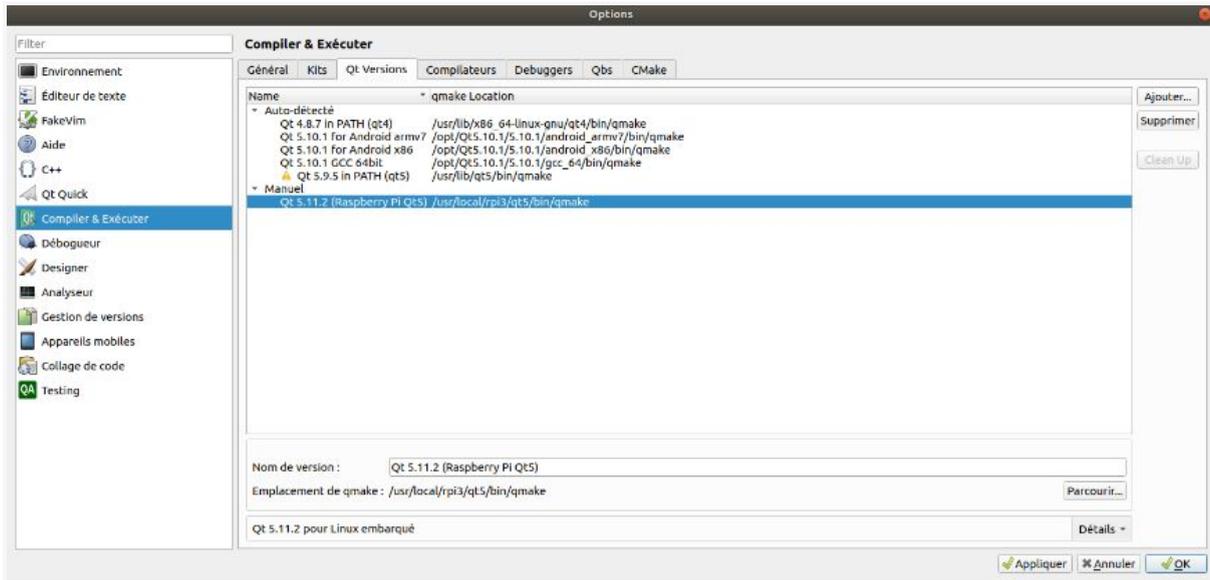
`~/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin/arm-linux-gnueabi-gdb` ou

`/usr/local/rpi3/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian/bin/arm-linux-gnueabi-gdb`



Menu Options → Compiler & Exécuter puis Qt Versions : Ajouter

Emplacement de qmake : `/usr/local/rpi3/qt5/bin/qmake`



Menu Options → Compiler & Executer puis Kits : Ajouter

Sysroot : `~/raspi/sysroot` ou `/usr/local/rpi3/raspi/sysroot`

