Projet e-stock 2019

Dossier Technique

version 1.0

Thomas MACHON - Hadrien GIMENEZ - Nathan WAGINAIRE
Robin GAUTHIER

BTS SN-IR - LaSalle Avignon

Sommaire

Présentation Générale 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 ? En combien de temps ?)
- 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.

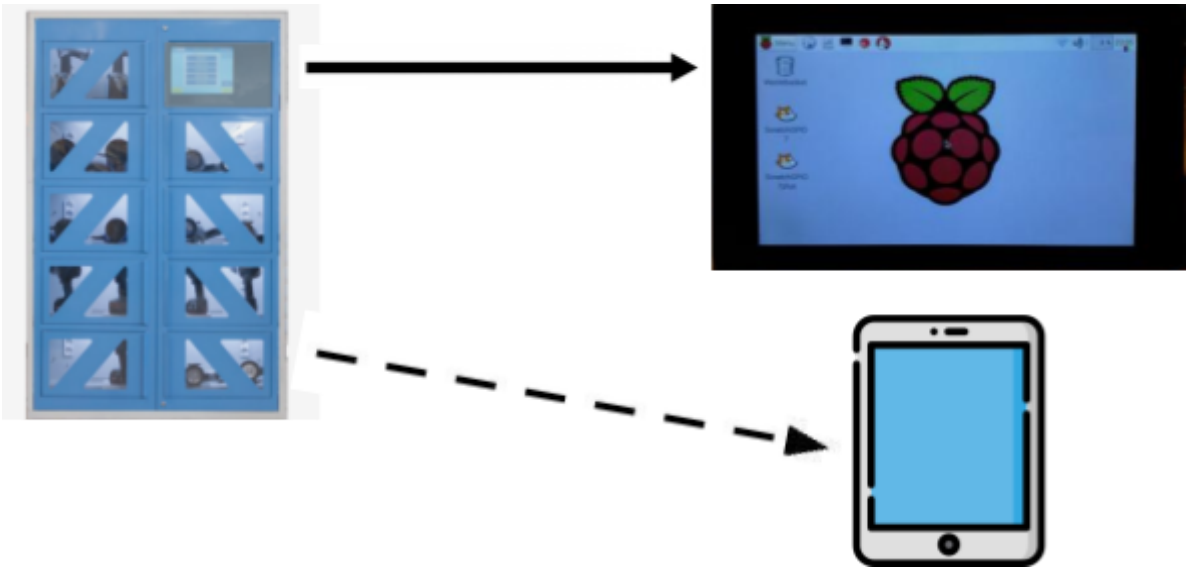
Le système devra permettre en fin d'activité de savoir :

- combien de consommable a été utilisé par chaque élève ;
- si l'élève a bien emprunté puis rangé son matériel ;
- combien temps un élève a emprunté un appareil.

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

Le système devra :

- Contrôler et gérer l'utilisation de produits stockés dans les armoires
- Assurer la traçabilité de l'attribution du matériel et des consommables stockés
- Sécuriser l'accès par un contrôle d'accès par badge RFID

Les armoires seront composées 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 déterminée 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 soit de l'écran tactile intégré soit par un terminal mobile.
vue d'ensemble du système

Architecture du système

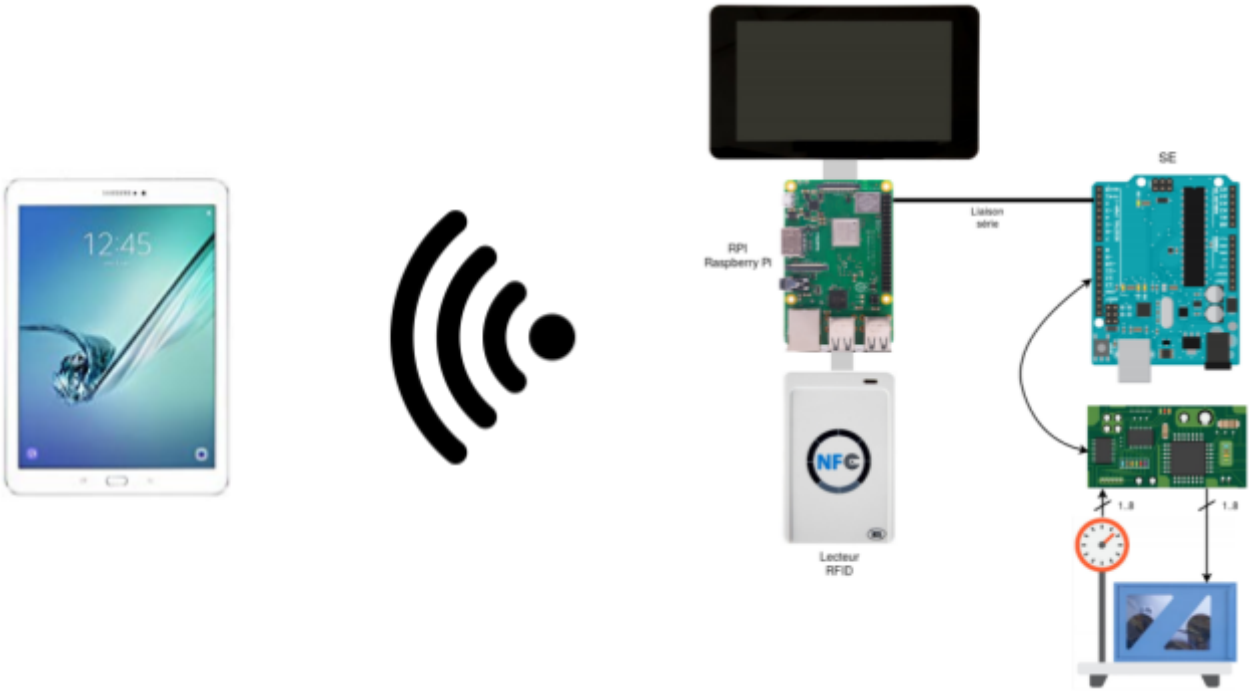
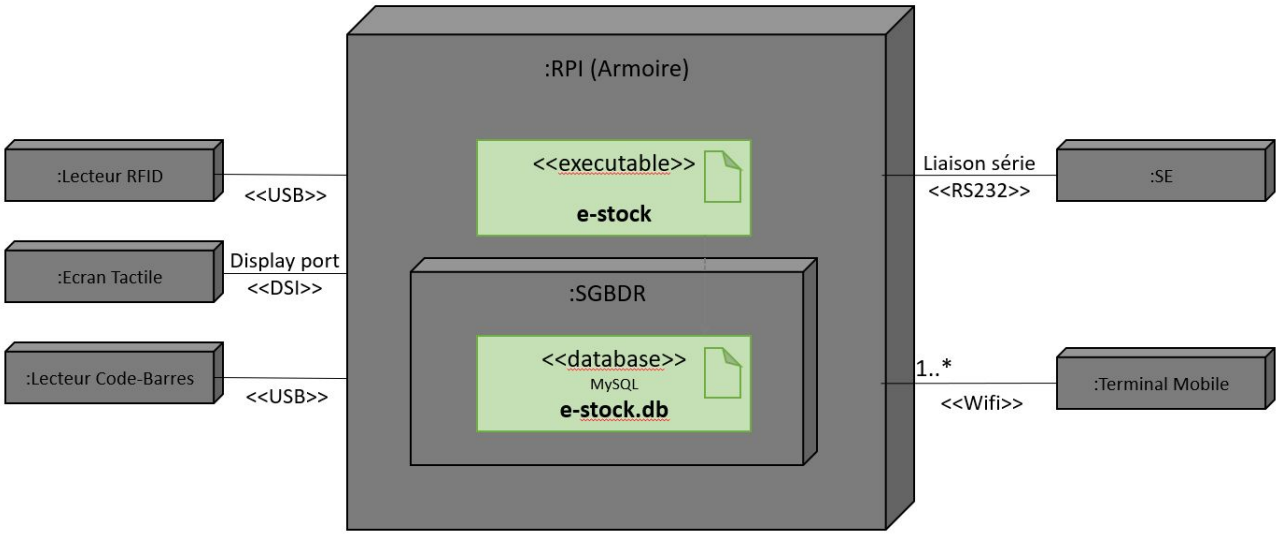


Diagramme de déploiement de l'armoire



Une armoire e-stock est architecturée autour :

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

La base de données **MySQL** sera intégrée au Raspberry Pi.

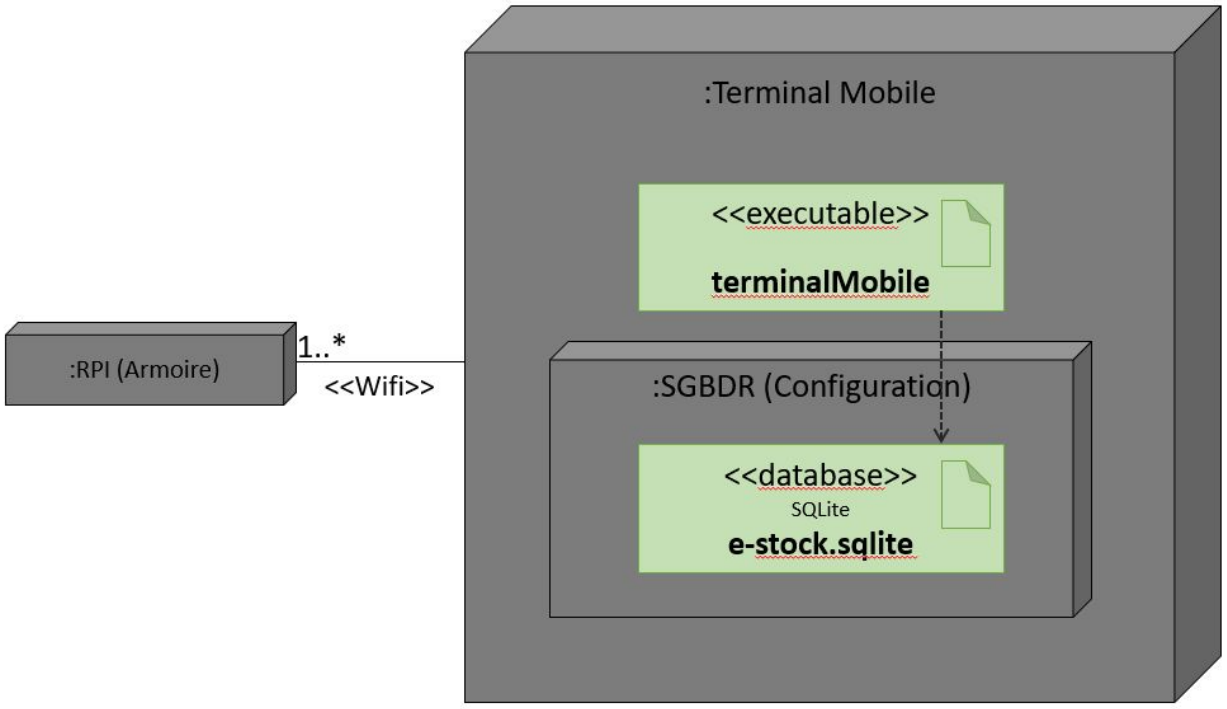
Un écran tactile sera associé à l'armoire sur lequel on pourra se connecter soit avec un badge grâce à un lecteur RFID (Radio Frequency IDentification) soit avec un compte utilisateur grâce à un clavier virtuel.

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.

Le terminal mobile pourra accéder à la base de données de l'armoire par le réseau WiFi.

Diagramme de déploiement du terminal mobile



Le terminal mobile contient une base de données locale **SQLite** contenant les informations d'accès aux armoires (Nom et Adresse IP). Elle permet à l'application cliente de proposer une liste d'armoires parmi laquelle l'utilisateur doit choisir celle sur laquelle il se connectera.

Répartition des tâches

Étudiant EC :

- ❏ Clément Martin-Fert (étudiant 1) : Commander l'ouverture/fermeture des casiers, Mesurer le poids du conteneur des casiers et Communiquer avec la RPI

Étudiants IR :

- ❏ Machon Thomas (étudiant 2) : Authentifier avec ou sans Badge RFID, Prendre et rapporter un article et Rechercher un article
- ❏ Waginaire Nathan (étudiant 3) : Gérer les articles et les utilisateurs, Gérer le stock en assurant le comptage automatique et Gérer le lecteur code-barres
- ❏ Gimenez Hadrien (étudiant 4) : Récupérer les pesées des casiers, Gérer les groupes et Alerter par email
- ❏ Gauthier Robin (étudiant 5) : Terminal mobile (Gérer les armoires)

Objectifs attendus

Gauthier Robin

- ➢ Une application informatique fonctionnelle ;
- ➢ Un modèle UML complet de la partie à développer ;
- ➢ Le code source commenté de l'application ;
- ➢ Les documentations associées au module.

Machon Thomas

- ➢ La lecture d'un badge RFID est réalisé ;
- ➢ L'authentification par badge est fonctionnelle ;
- ➢ Une autorisation ou un interdiction d'accès est signalée visuellement ;
- ➢ Prendre ou restituer un article ;
- ➢ La communication avec le SE permet l'ouverture/fermeture d'un casier

Gimenez Hadrien

- ➢ La création modification suppression de groupe
- ➢ Une configuration du système est possible
- ➢ La communication avec la SE permet la récupération des pesées
- ➢ La gestion des balances est fonctionnel
- ➢ L'envoi de l'état du stock par mail est fonctionnel

Martin-Fert Clément

- ➢ 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

Partie personnelle : Gauthier Robin

Assurer la mise en oeuvre d'une application Android pour un terminal mobile afin de gérer un ensemble d'armoires e-stock.

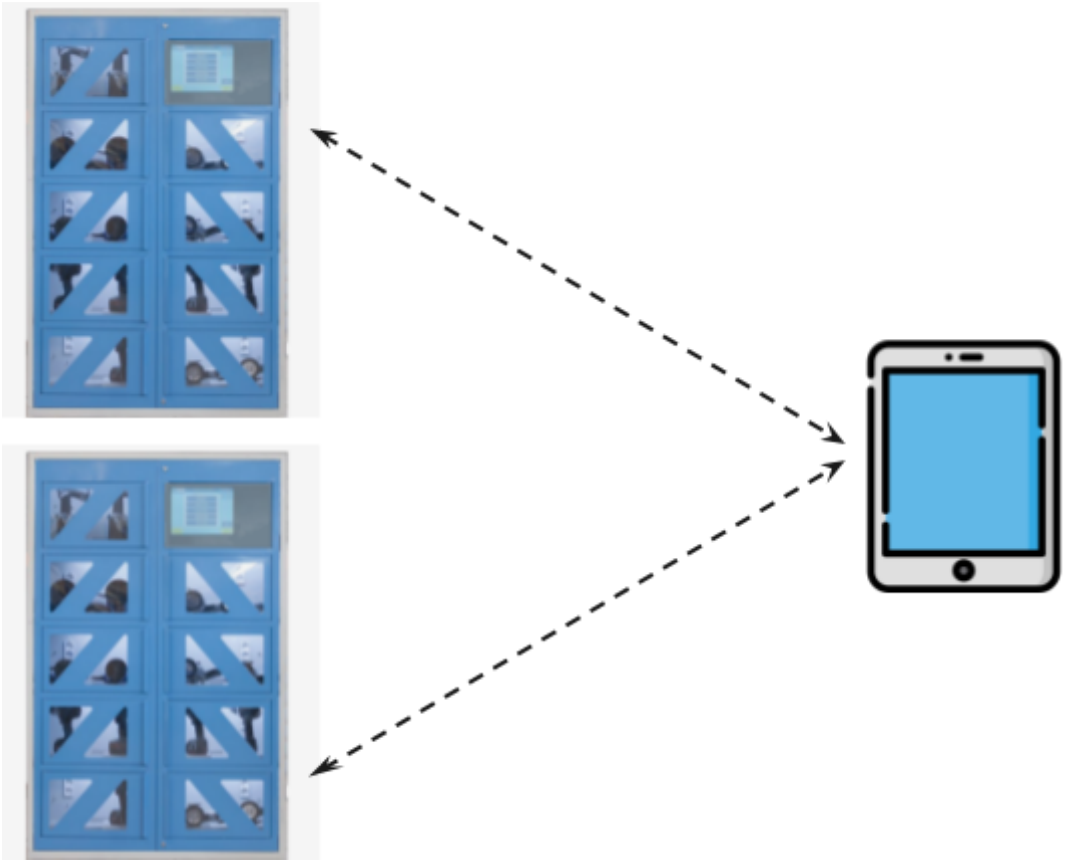
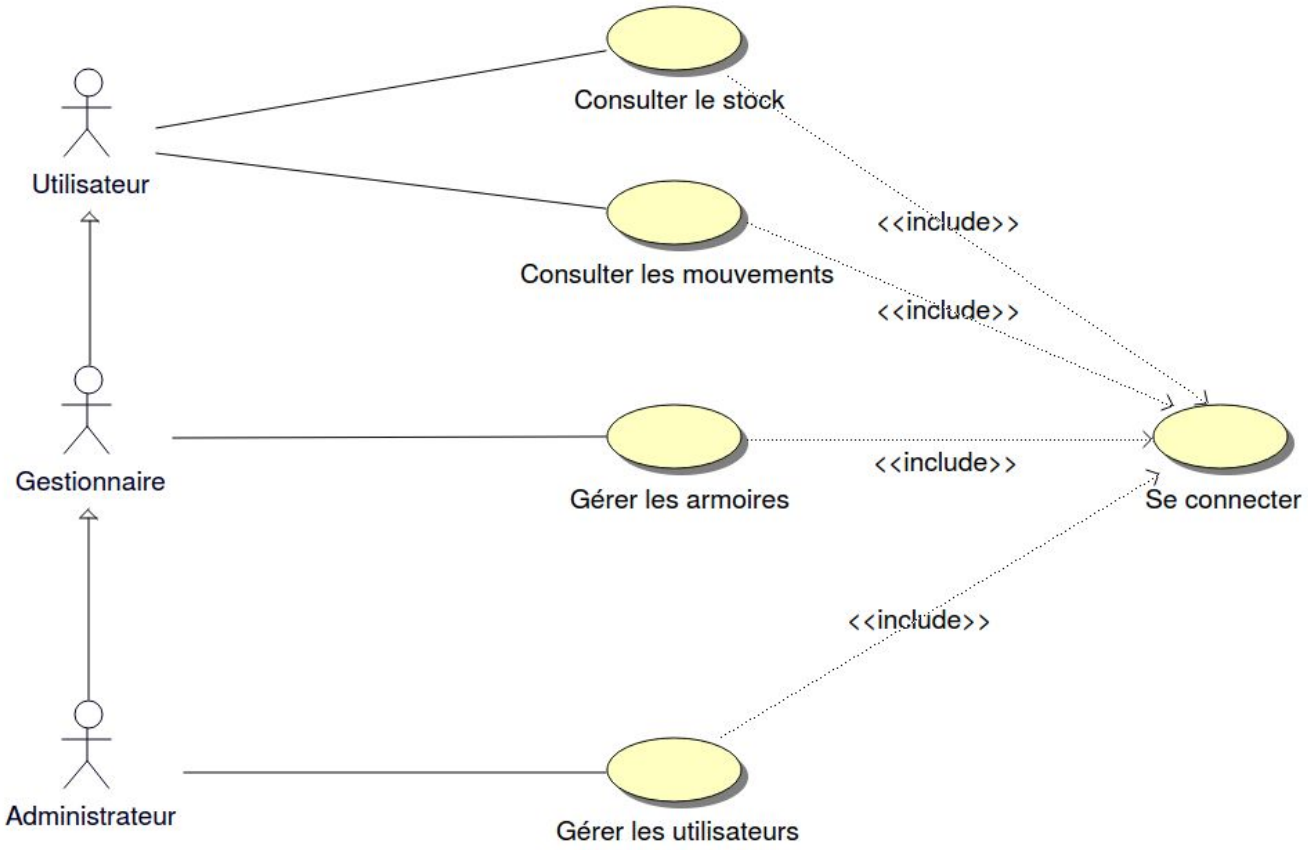


Diagramme de cas d'utilisation pour le terminal mobile



Pour l'application terminal mobile, les cas d'utilisation sont :

- ➔ Consulter le stock : consulter tous les matériels ou consommables présents actuellement dans l'armoire.
- ➔ Consulter les mouvements : consulter dans l'ordre chronologique tous les matériels ou consommables qui ont été empruntés ou remis (sortie ou entrée) dans l'armoire.
- ➔ Gérer les armoires : ajouter, ou supprimer les matériels ou consommables dans un casier d'une armoire.
- ➔ Gérer les utilisateurs : ajouter, modifier ou supprimer les utilisateurs d'une armoire.

On distingue 3 acteurs :

- ❖ L'**utilisateur** peut consulter le stock et consulter les mouvements d'une armoire.
- ❖ Le **gestionnaire** peut gérer les casiers d'une armoire.
- ❖ L'**administrateur** peut gérer les utilisateurs d'une armoire.

Dans tous les cas, il faudra procéder à une authentification.

Tests de validation

Désignation	Démarche à suivre	Résultat obtenu	Fonctionnel
Consulter le stock	Cliquer sur le bouton “Consulter le stock”	Possibilité de consulter le stock de l’armoire depuis le terminal mobile	Oui
Consulter les mouvements	Cliquer sur le bouton “Consulter les mouvements”	Possibilité de consulter les mouvements de l’armoire depuis le terminal mobile	Oui
Gérer les utilisateurs	Cliquer sur le bouton “Gérer les utilisateurs”	Possibilité de gérer les utilisateurs de l’armoire depuis le terminal mobile	Oui
Gérer les armoires	Cliquer sur le bouton “Consulter le stock” puis cliquer sur le casier	Possibilité de gérer les casiers de l’armoire depuis le terminal mobile	Non
Se connecter	Entrer le nom d’utilisateur et le mot de passe	Connexion à l’armoire	Oui

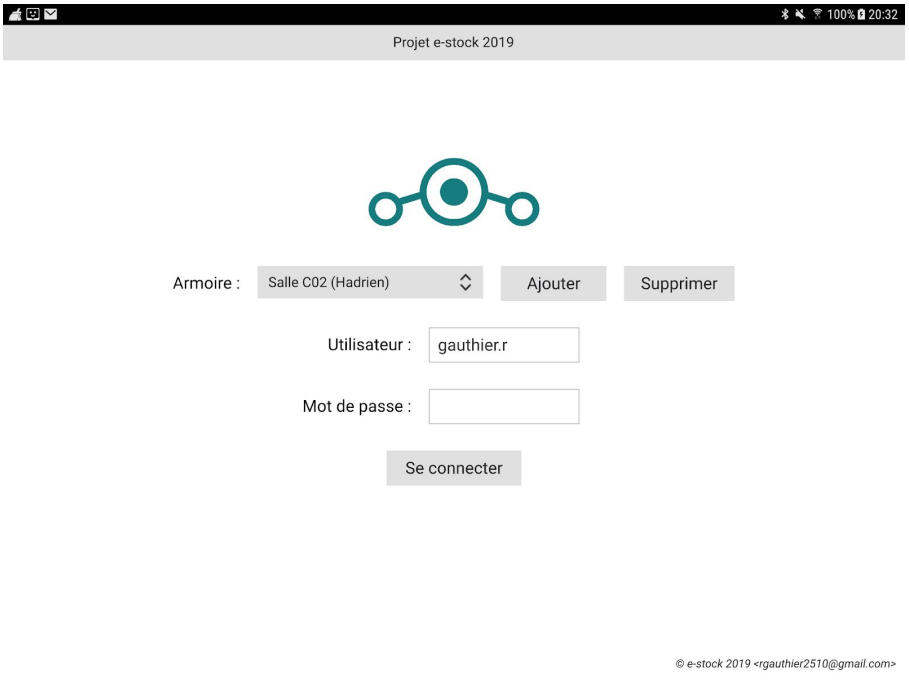
Maquette IHM terminal mobile

L'utilisateur (Utilisateur simple ou Gestionnaire ou Administrateur) doit au lancement de l'application **s'authentifier** pour pouvoir effectuer une action sur une armoire.

Pour s'authentifier l'utilisateur doit fournir :

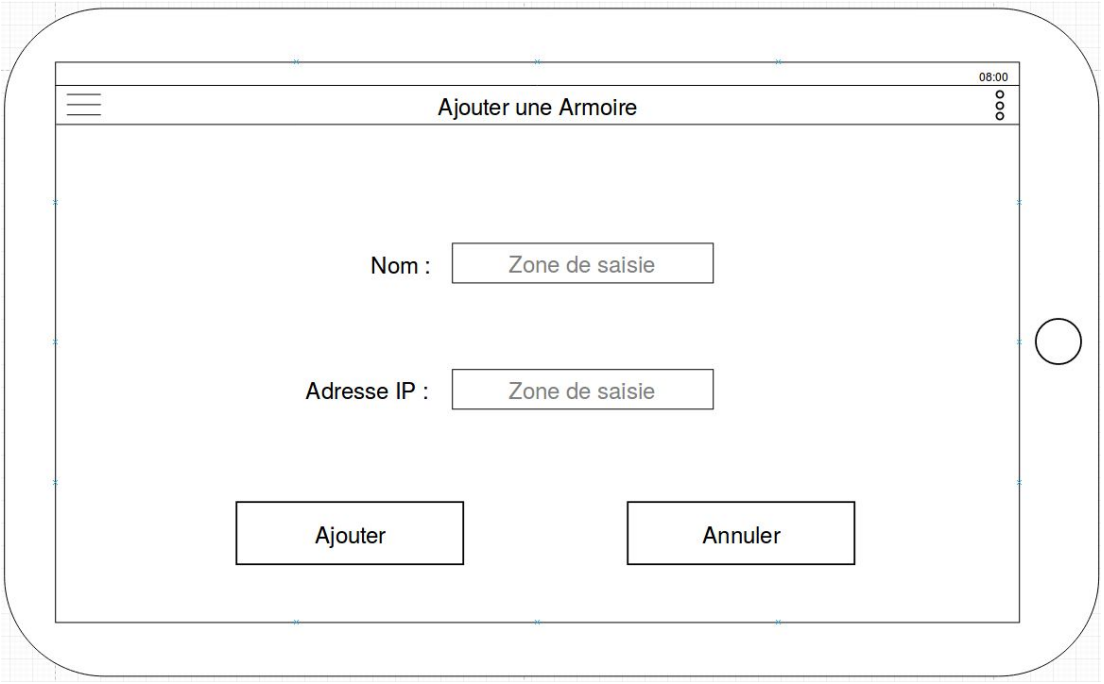
- ❏ l'adresse IP d'une armoire
- ❏ son nom d'utilisateur
- ❏ son mot de passe

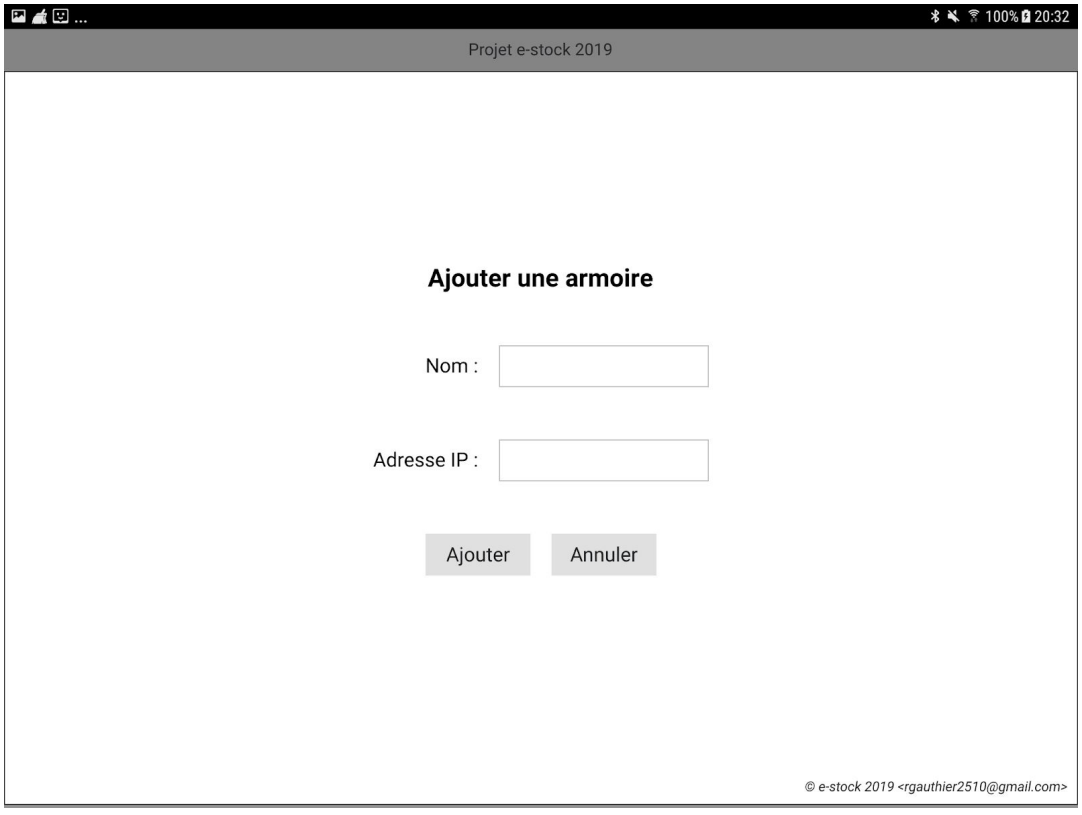
Se connecter :



Il pourra garder en mémoire via une base de données SQLite interne au terminal mobile les armoires avec leurs adresses IP et leurs Nom et de les afficher dans une liste déroulante.

Ajout d'une Armoire :

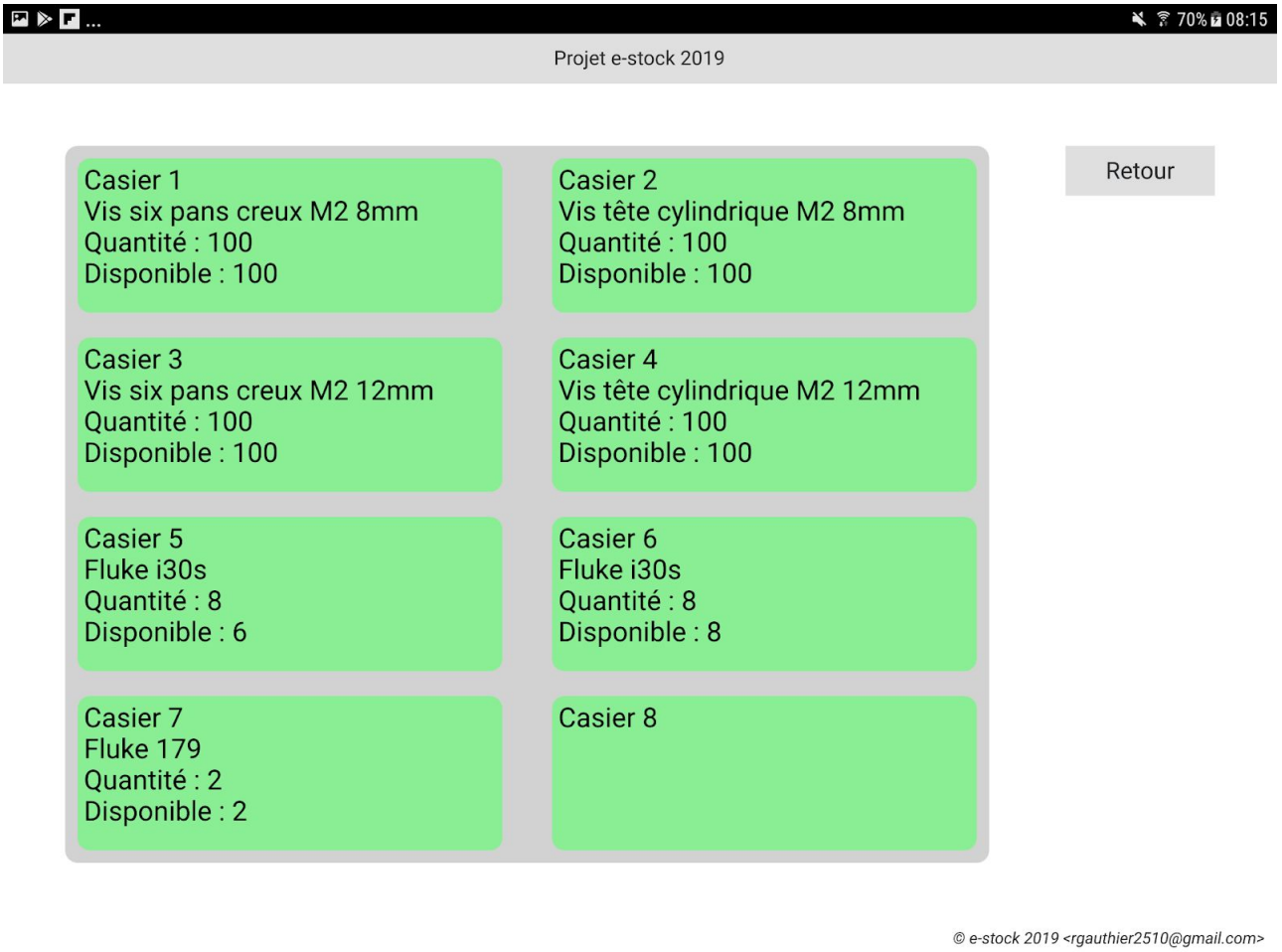




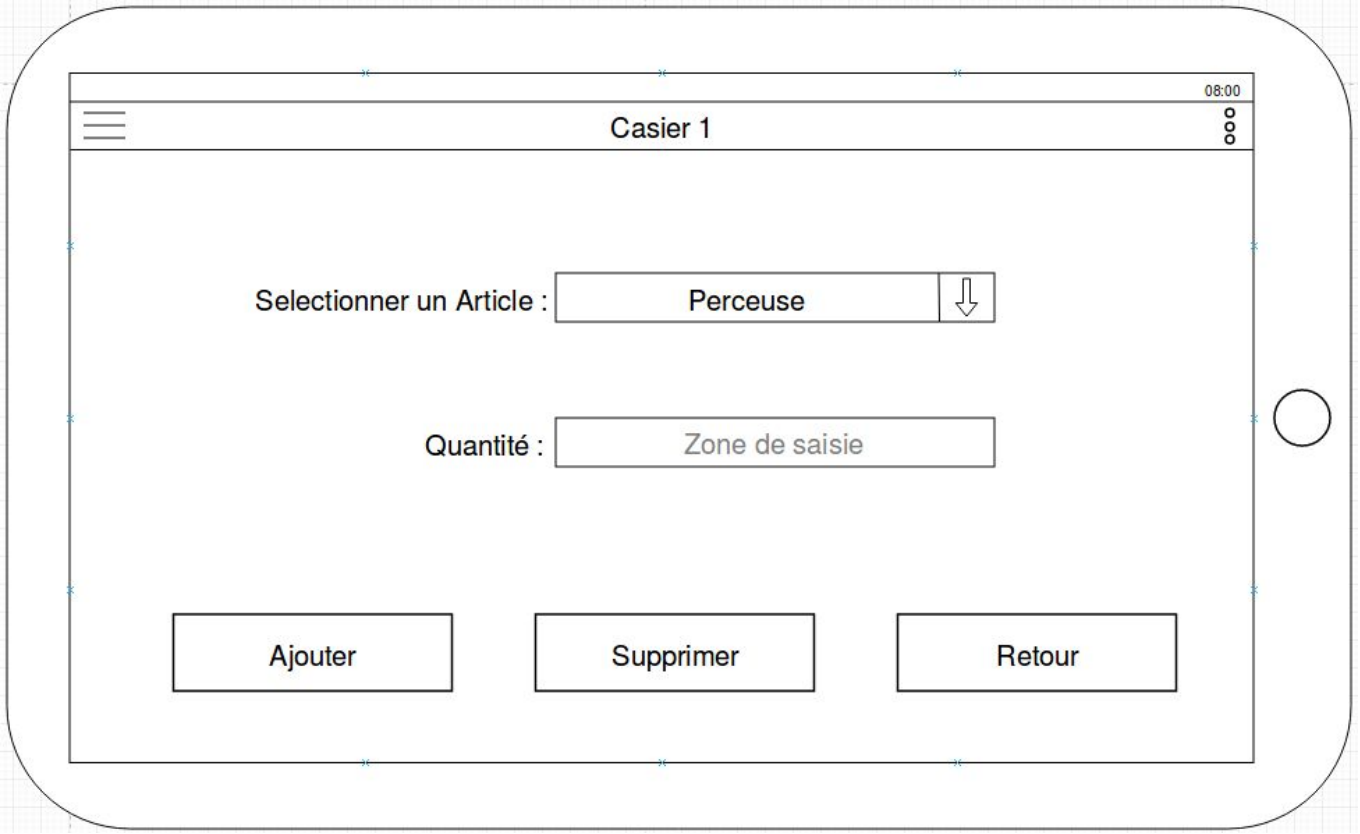
Ensuite l'utilisateur peut choisir quelle action effectuer : consulter les mouvements, consulter le stock, gérer les utilisateurs et gérer les casiers.

Menu Armoire :

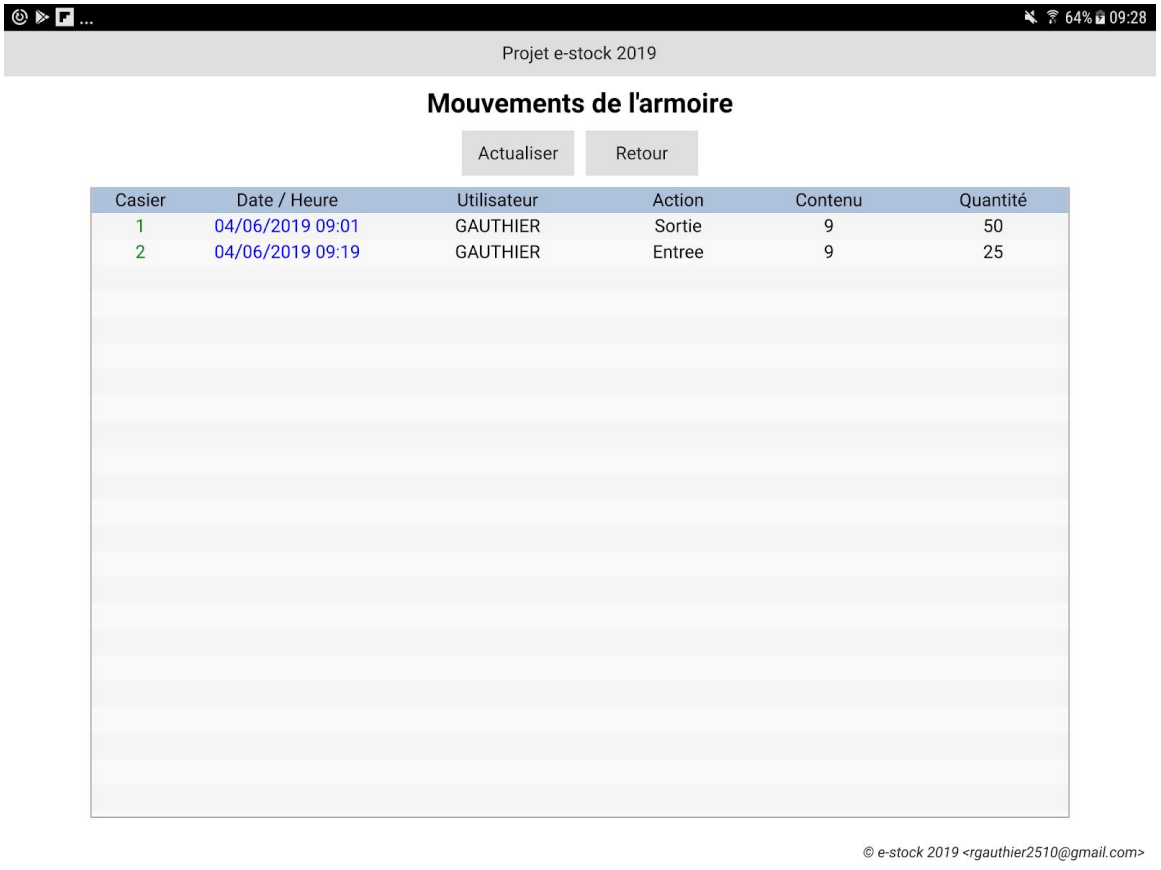
Consulter le stock :



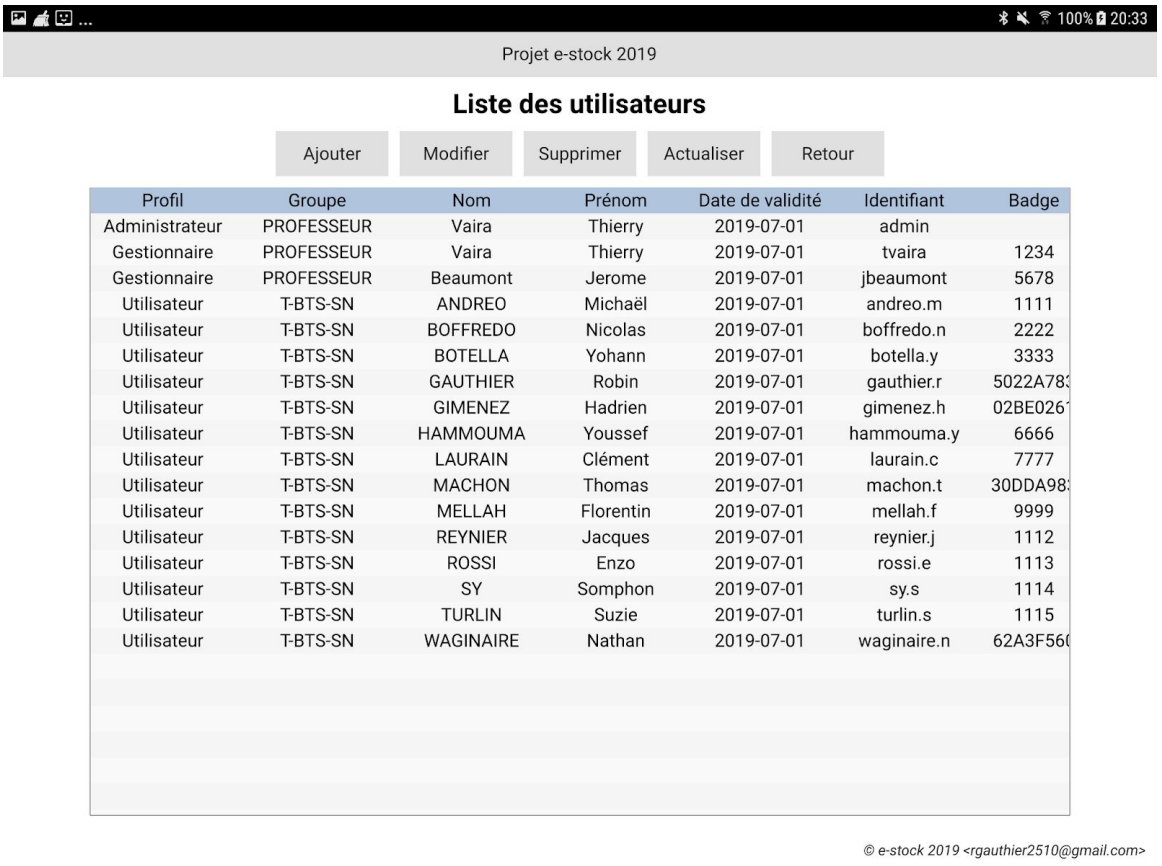
Gestion Casier:



Consulter les mouvements :



Gérer les utilisateurs :



Ressources logicielles et matérielles

Désignation	Caractéristiques
Terminal mobile	Tablette sous Android
Environnement de développement	Qt Creator et Qt Designer
API	Qt 5.10
Système d'exploitation du terminal mobile	Android 7.0
Interface binaire-programme	arm64-v8a
Logiciel de gestion de versions	subversion (RiouxSVN)
Système de gestion de bases de données relationnelles	MySQL, SQLite
Atelier de génie logiciel	Bouml version 7.8

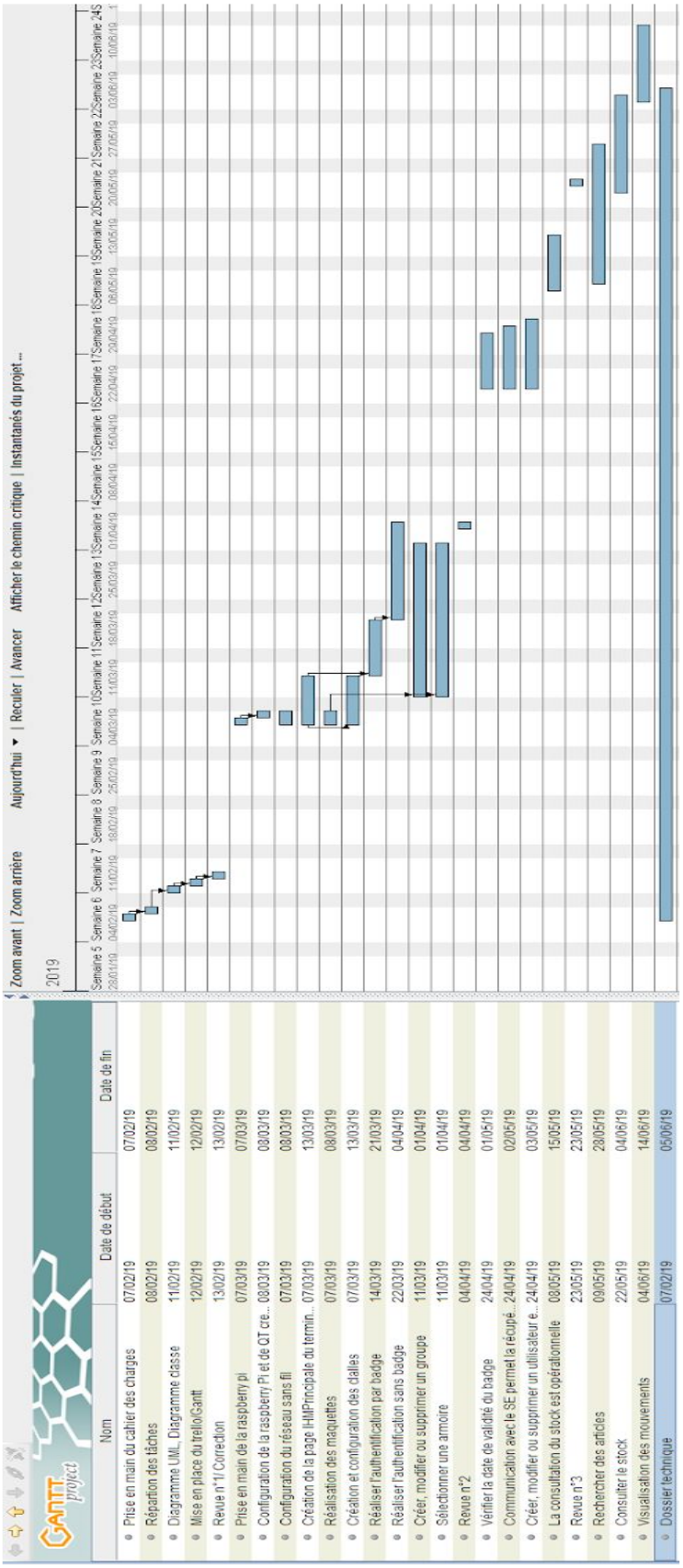
Planification

Répartition

Tâches à réaliser	Priorité	Itération
Consulter le Stock	haute	2
Consulter les mouvements	moyenne	1
Gérer les Casiers	moyenne	3
Gérer les Utilisateurs	basse	1
Se Connecter	basse	1

Remarque : Les cas d'utilisation Gérer les utilisateurs et Se connecter ont été placés dans l'itération 1 pour des raisons d'architecture logicielle. Cela a permis une prise en main plus facile de l'environnement Qt/Android.

Diagramme de Gantt



Le diagramme de Gantt permet de visualiser dans le temps les diverses tâches composant le projet. Les blocs définissent les tâches à réaliser, les jalons définissent le début et la fin d'une tâche. Certaine tâches ne peuvent pas être commencé avant d'en avoir fini une autre (exemple: il n'est pas possible de commencer le projet avant de s'être réparti les tâches).

IHM QML

L'IHM est réalisée en QML. QML est un langage déclaratif permettant de décrire des IHM avec des composants visuels. L'extension des fichiers est .qml. Il est possible d'intégrer du langage Javascript pour assurer l'interactivité avec l'utilisateur.

Exemple d'un **bouton** en QML :

```
Button {
   id: boutonAjouterArmoire
   text: qsTr("Ajouter")
   width: 50
   onClicked: {
      // ouvre la boîte de dialogue AjoutArmoire
      dialogueAjoutArmoire.open()
   }
}
```

La fenêtre principale sera construite à partir de l'élément **ApplicationWindow** :

```
ApplicationWindow {
    id: window
    title: ("E-stock")
    width: Screen.desktopAvailableWidth
    height: Screen.desktopAvailableHeight
    visible: true
    header: ToolBar {
        Label {
            text: ("Projet e-stock 2019")
            anchors.centerIn: parent
        }
    }
    ...
    footer: Label {
        width: parent.width
        horizontalAlignment: Qt.AlignRight
        padding: 10
        text: qsTr("(c) e-stock 2019 <rgauthier2510@gmail.com>")
        font.pixelSize: Qt.application.font.pixelSize * 0.8
        font.italic: true
    }
}
```

Les autres vues de l'application seront réalisées à partir du composant **Page** :

```
Page {
    width: Screen.desktopAvailableWidth
    height: Screen.desktopAvailableHeight
    title: qsTr("Menu Armoire")
    ...
}
```

L'utilisation d'élément visuel dans une IHM en QML nécessite un système de placement. QML propose un système d'ancres (*anchors*) et un ensemble de conteneurs et de layout.

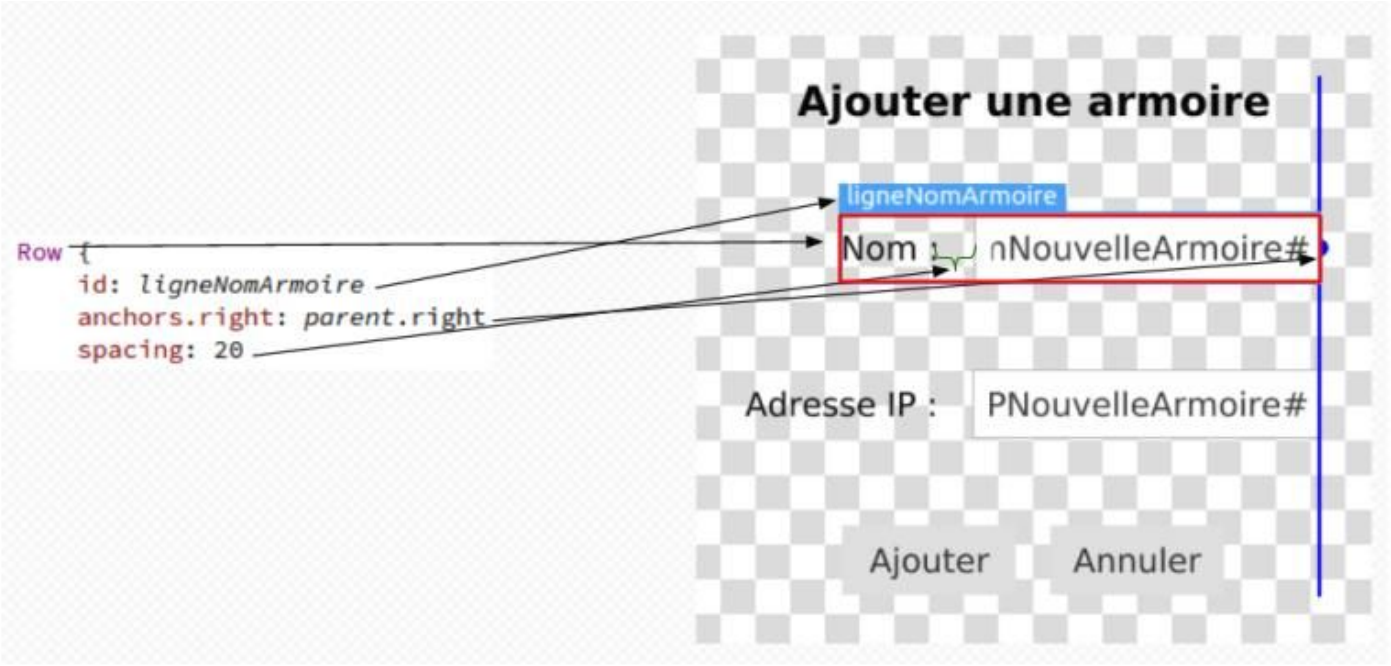
Les éléments peuvent se positionner par leur attributs x et y mais ce n'est évidemment pas pratique.

La technique est donc de positionner les éléments les uns par rapport aux autres.

On peut le faire par un système d'ancres (*anchors*). Une ancre se rattache à un autre élément par son point haut (*top*), bas (*bottom*), droit (*right*) ou gauche (*left*). Comme les éléments sont souvent emboîtés les uns dans les autres, on fera souvent référence à son élément parent (*parent*).

Un élément emboîté dans un autre élément peut aussi utiliser (remplir) tout son espace : `anchors.fill: parent`. On peut aussi appliquer des marges (`anchors.margins`) ou spécifiquement (`anchors.leftMargin`, etc ...). Les ancres permettent aussi de se centrer horizontalement (`anchors.horizontalCenter`) et/ou verticalement (`anchors.verticalCenter`).

Il existe aussi des éléments conteneurs qui assurent un positionnement spécifique automatiquement. C'est le cas des éléments **Row** (qui place automatiquement les éléments en ligne) et **Column** (automatiquement en colonne). Attention : les éléments placés dans un **Row** et/ou un **Column** ne peuvent plus se positionner par leurs ancres.



Row : désigne la ligne qui contient le label et la zone de saisie

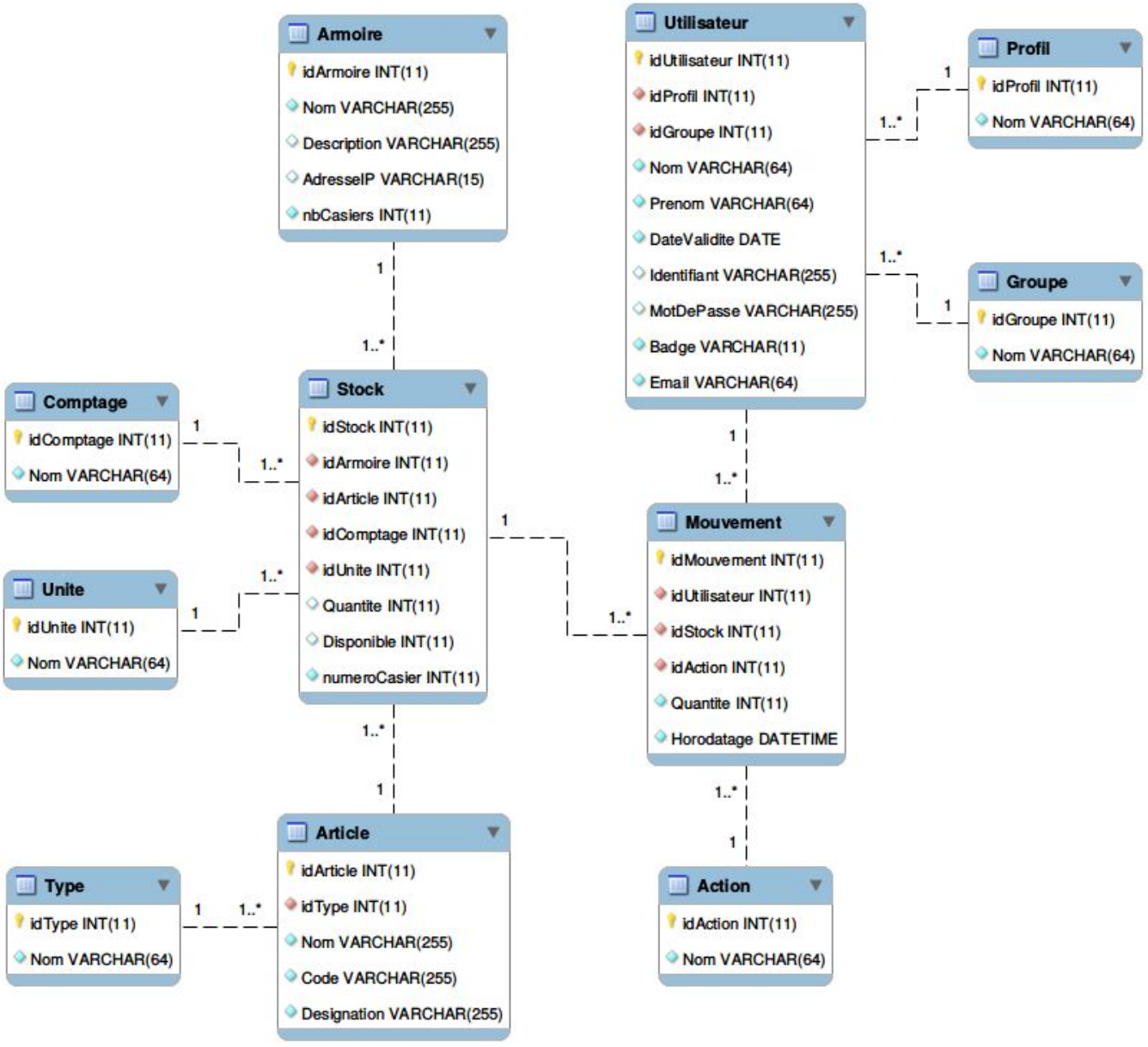
id : identifiant du conteneur Row

anchors.right : positionne le conteneur Row à son parent

spacing : espacement entre les différents éléments dans le conteneur Row (ici entre la zone de saisie et le label)

Base de données

Le schéma de la base de données MySQL d'une armoire est le suivant :



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

- ➔ un champ Nom
- ➔ un champ adresseIP
- ➔ un champ nbCasiers (défini par par défaut à 8)

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

La table **Groupe** contient un champ Nom qui détermine le groupe d'un utilisateur (par exemple: Professeur, 1-BTS-SN ou T-BTS-SN).

La table **Utilisateur** contient :
- ➔ une clé étrangère idProfil qui précise son profil
- ➔ une clé étrangère idGroupe qui indique son groupe d'appartenance
- ➔ un champ Nom
- ➔ un champ Prenom
- ➔ un champ DateValidite
- ➔ un champ Identifiant
- ➔ un champ MotDePasse
- ➔ un champ Badge
- ➔ un champ Email

La table **Type** précise le type d'un article par un champ Nom (équipement ou consommable).

La table **Article** définit un article par :
- ➔ une clé étrangère idType qui précise son type
- ➔ un champ Nom
- ➔ un champ Code (code barre)
- ➔ un champ Designation

La table **Comptage** contient un champ Nom qui indique le type de comptage (Aucun, Automatique ou par CodeBarre).

La table **Unite** contient un champ Nom (mètres, pièces, pourcentage, g (grammes), kg (kilogrammes)).

La table **Stock** contient :
- ➔ une clé étrangère idArmoire qui l'associe à une armoire
- ➔ une clé étrangère idArticle qui indique l'article stocké
- ➔ une clé étrangère idComptage qui précise le type de comptage
- ➔ une clé étrangère idUnite qui indique l'unité utilisée pour le compter
- ➔ un champ Quantite qui fournit le stock de départ pour cet article
- ➔ un champ Disponible qui comptabilise la présence de l'article dans le stock actuellement
- ➔ un champ NumeroCasier

La table **Action** contient un champ Nom (Entrée, Sortie).

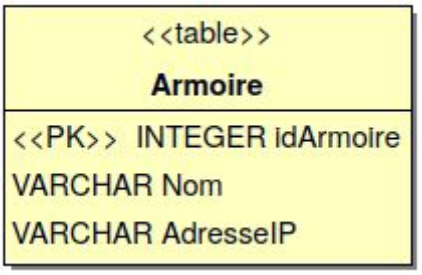
La table **Mouvement** contient l'ensemble des entrées/sorties des articles dans l'armoire :
- ➔ une clé étrangère idUtilisateur qui indique l'utilisateur qui a effectué le mouvement
- ➔ une clé étrangère idStock qui associe le mouvement dans le stock
- ➔ une clé étrangère idAction qui fournit l'action réalisée
- ➔ un champ Quantité
- ➔ un champ Horodatage

Exemple de requête SQL permettant de récupérer les caractéristiques des utilisateurs d'une armoire :

```
SELECT Profil.Nom, Groupe.Nom, Utilisateur.Nom, Utilisateur.Prenom,
Utilisateur.DateValidite, Utilisateur.Identifiant, Utilisateur.Badge FROM
Utilisateur
INNER JOIN Groupe ON Utilisateur.idGroupe = Groupe.idGroupe
INNER JOIN Profil ON Utilisateur.idProfil = Profil.idProfil
```

Cette requête est un exemple de jointure avec la table **Groupe** et la table **Profil**.

Actuellement, la base de données SQLite du terminal mobile ne contient qu'un seule table :



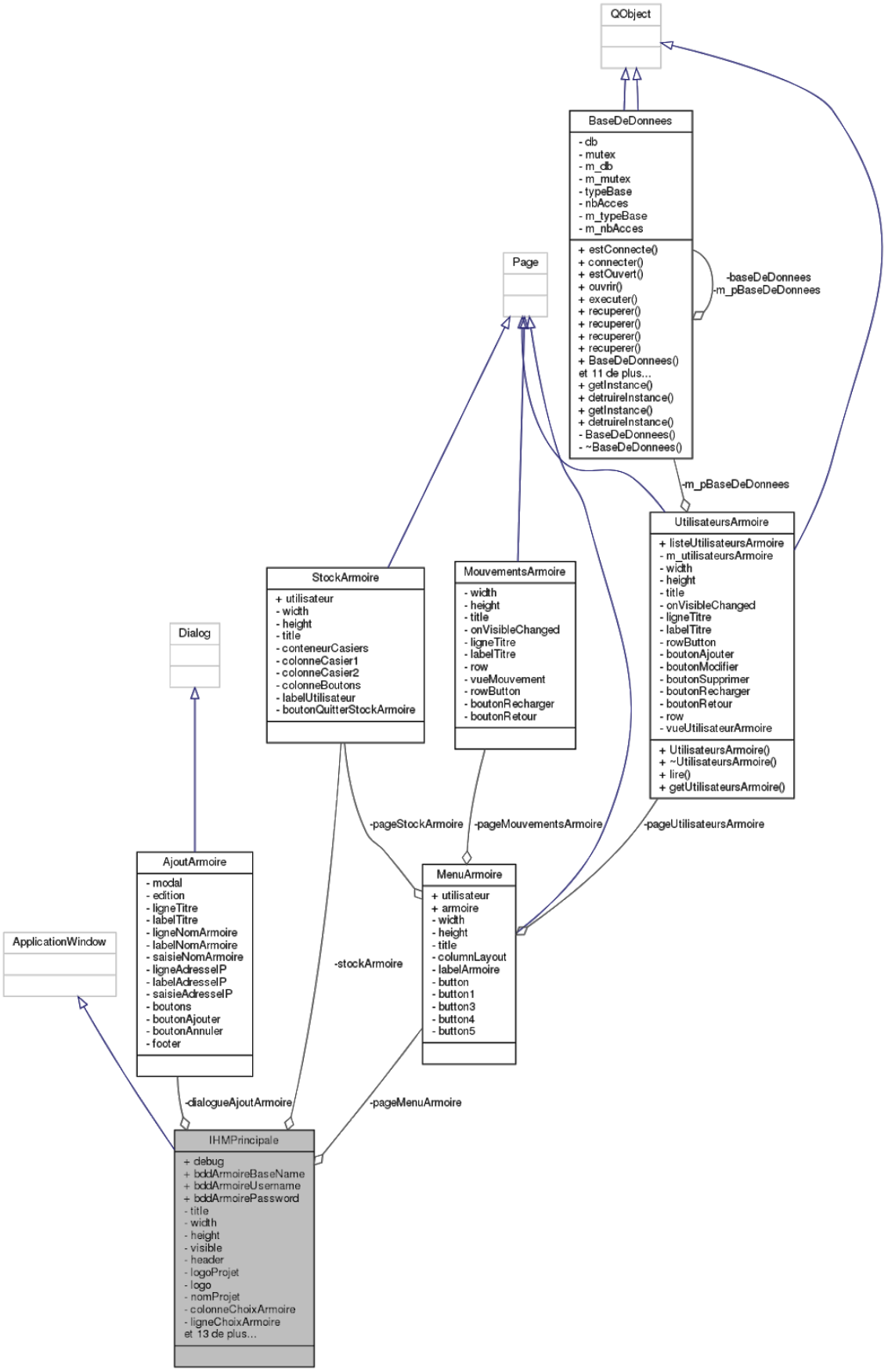
Cette table **Armoire** permet de mémoriser un ensemble d'armoires. L'adresse IP de l'armoire permettra de se connecter à la base de données hébergée dans l'armoire.

Pour récupérer la liste des armoires connues, on effectue la requête SQL : `SELECT Nom, AdresseIP FROM Armoire;`

L'application du terminal mobile permet aussi :

- d'ajouter une nouvelle armoire : `INSERT INTO Armoire (Nom, AdresseIP) VALUES ('" + p_nom + "','" + p_adresseIP + "');"`
- de supprimer une armoire existante : `DELETE FROM Armoire WHERE Nom = '" + p_nom + "';`

Diagramme de classes (IHM/QML)



Les différentes classes de ce diagramme sont des éléments QML :

- ➔ **IHMPrincipale** : page d'accueil qui permet de se connecter à une armoire, mais aussi d'ajouter une armoire et/ou supprimer une armoire.
- ➔ **MenuArmoire** : page qui permet de choisir les différentes actions (consulter les stocks, consulter, ajouter, modifier ou supprimer un utilisateur, consulter les mouvements).
- ➔ **StockArmoire** : page qui permet de consulter les stocks actuel dans l'armoire.
- ➔ **UtilisateursArmoire** : page qui permet de consulter, ajouter, modifier ou supprimer un utilisateur.
- ➔ **MouvementsArmoire** : page qui permet de consulter les mouvements dans l'ordre chronologique.
- ➔ **AjoutArmoire** : page qui permet d'ajouter une armoire.

La classe **BaseDeDonnees** est une classe C++ qui permet d'effectuer des requêtes SQL sur une base de données SQLite et/ou MySQL.

Diagramme de navigation

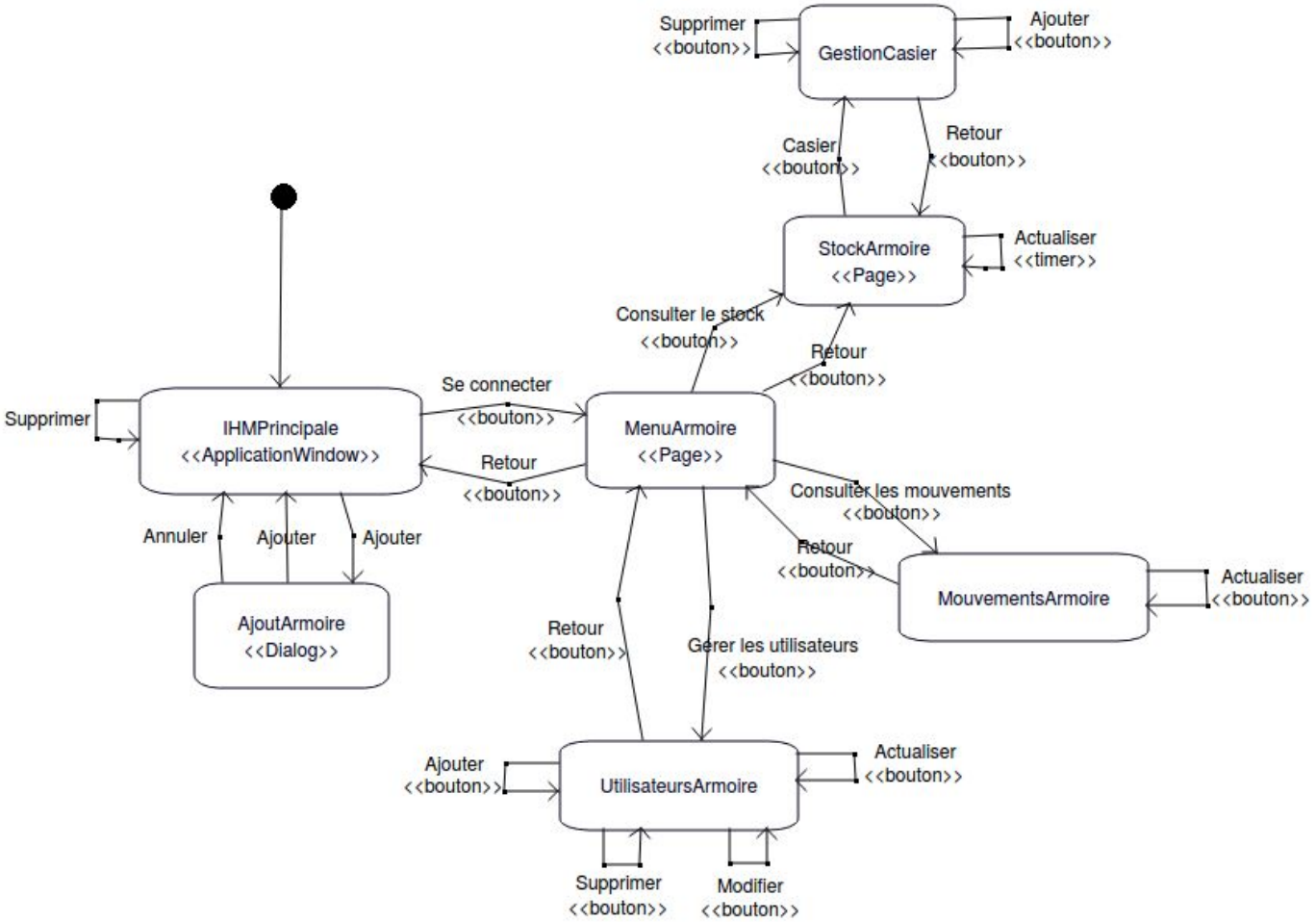
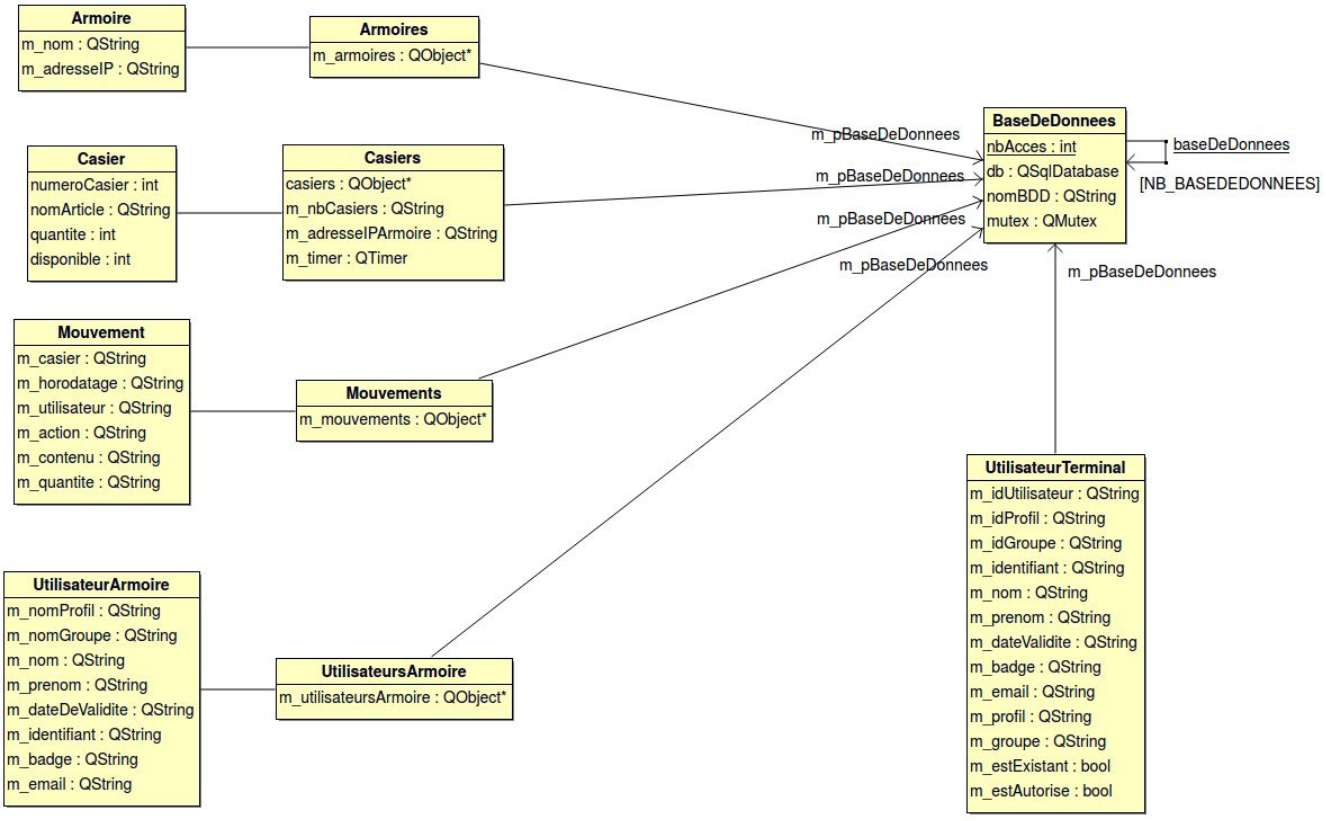


Diagramme de classes (C++)

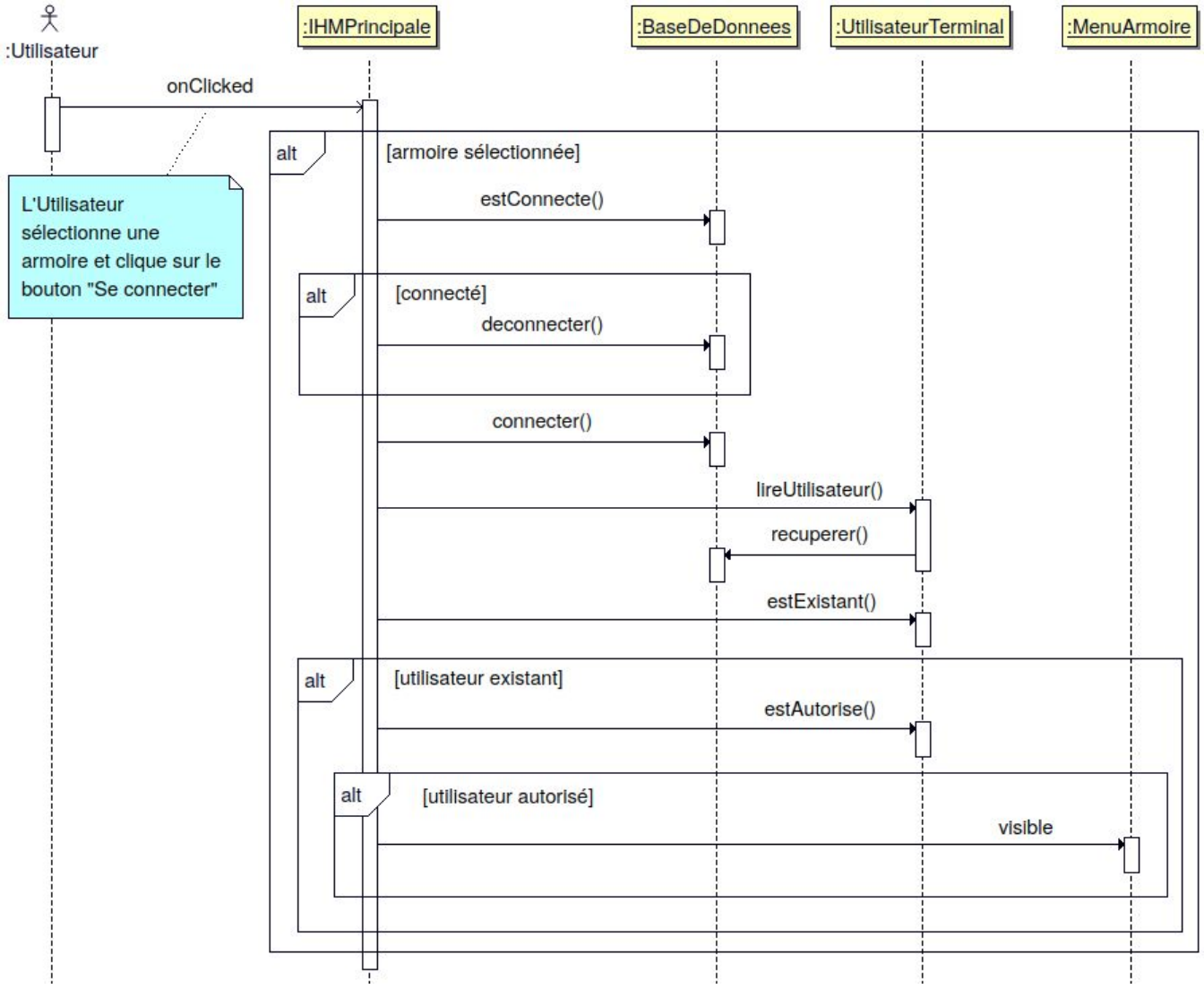


Les différentes classes de ce diagramme :

- ➔ **Armoire** : permet de mémoriser un ensemble d'armoires.
- ➔ **Armoires** : permet d'afficher les différentes armoires dans une liste déroulante. Possibilité d'actualiser la liste des armoires, ajouter ou supprimer une armoire dans la liste déroulante.
- ➔ **Mouvement** : permet de mémoriser les différents mouvements.
- ➔ **Mouvements** : permet d'afficher les différents mouvements dans un Tableau. Possibilité d'actualiser les données.
- ➔ **Casier** : permet de mémoriser les différents Contenu du casier.
- ➔ **Casiers** : permet d'afficher les différents équipements ou consommables dans un casier. L'actualisation des données se fait toutes les 0.5 secondes.
- ➔ **UtilisateurArmoire** : permet de mémoriser les différents Utilisateurs de l'armoire.
- ➔ **UtilisateursArmoire** : permet d'afficher les différents Utilisateurs dans un Tableau. Possibilité d'actualiser les données, Ajouter, Modifier ou Supprimer des utilisateurs.
- ➔ **UtilisateurTerminal** : Vérifie si l'utilisateur est bien existant et si il est autorisé

La classe **BaseDeDonnees** est une classe C++ qui permet d'effectuer des requêtes SQL sur une base de données SQLite et/ou MySQL.

Diagrammes de séquence : Connexion



Tout d'abord l'utilisateur sélectionne une armoire dans la liste déroulante puis clique sur le bouton "Se connecter".On interroge la base de données pour savoir si l'utilisateur qui vient de se connecter est existant. Puis on interroge la base de données pour savoir si l'utilisateur est autorisé. Si les deux conditions sont remplies alors la fenêtre MenuArmoire passe en visible.

Communication sans fil

Le terminal mobile doit pouvoir communiquer à distance avec une armoire. Actuellement, les terminaux mobiles disposent naturellement de deux types de communication sans fil : le **WiFi** et le **Bluetooth**. De l'autre côté, l'armoire est équipée d'un Raspberry Pi 3 disposant lui aussi de base des communications WiFi et Bluetooth.

Le **WiFi** est un ensemble de protocoles de communication sans fil régis par les normes IEEE 802.11. Un réseau WiFi (WLAN) permet de relier par ondes radio des équipements informatiques (ordinateur, terminal mobile, etc.) au sein d'un réseau informatique afin de permettre la transmission de données entre eux. Le WiFi a été conçu pour pouvoir utiliser les bandes de fréquences de 2,4 GHz et/ou 5 GHz.

Le **Bluetooth** est une norme de communications permettant l'échange bidirectionnel de données à très courte distance en utilisant des ondes radio sur une bande de fréquence de 2,4 GHz.

Remarque : la fréquence relativement élevée de ces ondes (2,4 GHz) les font mal traverser les murs.

Critères de choix :

La portée

La portée est un critère important dans ce projet car elle conditionne l'usage de l'application mobile par les utilisateurs.

Le **WiFi** a une portée de plusieurs dizaines de mètres, voire de plusieurs centaines de mètres en extérieur. Cela permettra un usage sans de déplacer à proximité de l'armoire. Pour le professeur, cela lui permettra par exemple de consulter le stock ou les mouvements à partir de son bureau.

Le **Bluetooth** est une technologie sans fil de proximité, à la portée limitée à quelques mètres seulement (10 à 15 mètres). Cela oblige de se trouver à proximité de l'armoire.

Avantage : **WiFi**

Le débit

Le débit est un critère peu important dans ce projet car la quantité des données transférée est faible (requêtes SQL).

En fonction de la norme utilisée, le débit du **WiFi** varie entre de 11 Mbit/s (802.11b), 54 Mbit/s (802.11a ou 802.11g) et 600 Mbit/s (802.11n). La norme 802.11ac offre même un débit 1 300 Mbit/s.

A partir de la version 3, le **Bluetooth** peut atteindre un débit théorique de 24 Mbit/s. Suivant les protocoles utilisés, le débit peut être limité à 360 kbit/s par exemple, pour le service RFCOMM sur les téléphones mobiles.

Remarque : le Bluetooth consommant moins d'énergie, il ne permet pas d'atteindre la même portée ni le même débit que le WiFi.

Avantage : **WiFi**

Coût

Le coût est souvent un critère important dans un projet.

Le **WiFi** est disponible de base de base à la fois dans le terminal mobile et le Raspberry Pi 3 mais peut nécessiter la présence d'un point d'accès suivant le mode de mise en réseau utilisé (Infrastructure ou Ad hoc). On peut supposer que, vue l'utilisation du projet dans le cadre d'un établissement scolaire, qu'un réseau WiFi sera disponible.

Le **Bluetooth** n'engendre aucun coût supplémentaire car la technologie est présente de base à la fois dans le terminal mobile et le Raspberry Pi 3. On peut aussi remarquer que le Bluetooth engendrera une consommation moindre que le WiFi.

Consommation de puissance d'un smartphone Nokia N95 :

- Émission : 432 mW (Bluetooth) et 1 629 mW (WiFi)
- Réception : 425 mW (Bluetooth) et 1 375 mW (WiFi)
- Connexion : 67 mW (Bluetooth) et 868 mW (WiFi)

Avantage : **Bluetooth**

Accès aux données

Dans ce projet, c'est le critère le plus important. La communication sans fil doit permettre de se connecter à une base de données MySQL hébergée dans chaque Raspberry Pi.

La base de données MySQL étant un serveur TCP, seule une communication directe en **WiFi** est possible. En **Bluetooth**, il faudrait envisager d'écrire un programme serveur spécifique dans le Raspberry Pi afin de relayer l'exécution des requêtes envoyées par le terminal mobile.

Avantage : **WiFi**

Conclusion

Dans ce projet en raison de ces nombreux avantages, le choix d'une communication en **WiFi** s'impose.

Partie personnelle : Machon Thomas

Authentifier avec ou sans Badge RFID, Prendre et rapporter un article et Rechercher un article

- ➢ La lecture d'un badge RFID est réalisé ;
- ➢ L'authentification par badge est fonctionnelle ;
- ➢ Une autorisation ou un interdiction d'accès est signalée visuellement ;
- ➢ Prendre ou restituer un article ;
- ➢ La communication avec le SE permet l'ouverture/fermeture d'un casier

Ressources logicielles et matérielles

Désignation	Caractéristiques
RPI	Raspberry PI 3
MINI-ECRAN	Écran tactile 800x480 7" relié sur la RPI
BADGE	Badge RFID 13,56 MHz
Système d'exploitation de la RPI	GNU/Linux Raspbian
Système de gestion de bases de données relationnelles	MySQL,
Atelier de génie logiciel (IR)	BOULM v.7.8
Logiciel de gestion de versions	subversion (RiouxSVN)
API GUI	Qt creator (Enterprise) v.5.11.2

Planification

Répartition

Tâches à réaliser	Priorité	Itération
S'authentifier avec badge	1	0.1
S'authentifier sans badge	1	0.2
Récupérer des articles	2	0.2
Déposer des articles	2	0.2
Rechercher des articles	3	0.3

Gantt

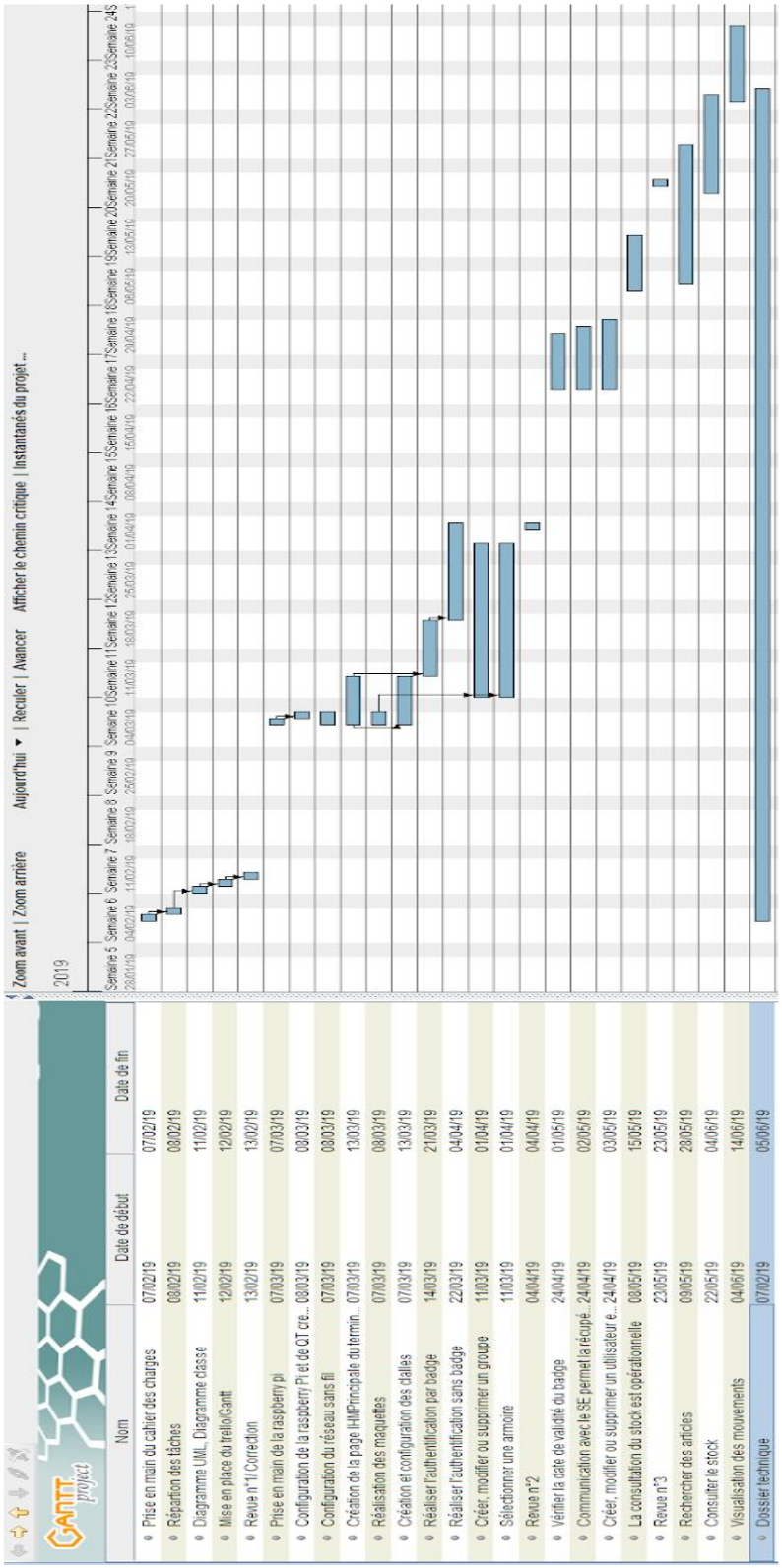
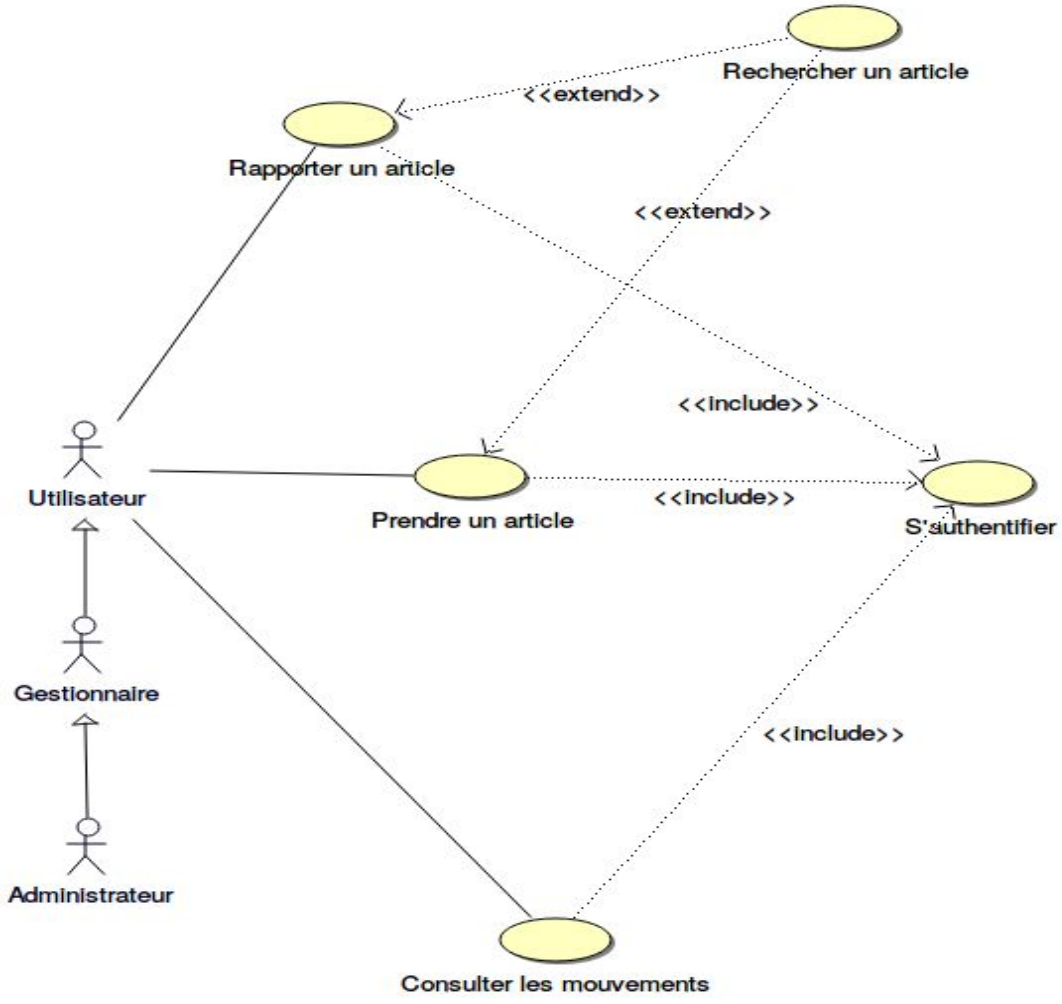

GANTT project
Zoom avant | Zoom arrière Aujourd'hui | Reculer | Avancer Afficher le chemin critique | Instantanés du projet ...
2019
Nom
Date de début
Date de fin
Prise en main du cahier des charges 07/02/19 07/02/19
Répartion des tâches 08/02/19 08/02/19
Diagramme UML, Diagramme classe 11/02/19 11/02/19
Mise en place du trello/Gantt 12/02/19 12/02/19
Revue n°1/ Correction 13/02/19 13/02/19
Prise en main de la raspberry pi 07/03/19 07/03/19
Configuration de la raspberry Pi et de QT cre... 08/03/19 08/03/19
Configuration du réseau sans fil 07/03/19 08/03/19
Création de la page IHMPrincipale du termin... 07/03/19 13/03/19
Réalisation des maquettes 07/03/19 08/03/19
Création et configuration des dalles 07/03/19 13/03/19
Réaliser l'authentification par badge 14/03/19 21/03/19
Réaliser l'authentification sans badge 22/03/19 04/04/19
Créer, modifier ou supprimer un groupe 11/03/19 01/04/19
Sélectionner une armoire 11/03/19 01/04/19
Revue n°2 04/04/19 04/04/19
Vérifier la date de validité du badge 24/04/19 01/05/19
Communication avec le SE permet la récupé... 24/04/19 02/05/19
Créer, modifier ou supprimer un utilisateur e... 24/04/19 03/05/19
La consultation du stock est opérationnelle 08/05/19 15/05/19
Revue n°3 23/05/19 23/05/19
Rechercher des articles 09/05/19 28/05/19
Consulter le stock 22/05/19 04/06/19
Visualisation des mouvements 04/06/19 14/06/19
Dossier technique 07/02/19 05/06/19

Diagramme de cas d'utilisation

L'acteur de ce diagramme est l'utilisateur.



L'utilisateur devra tout d'abord s'authentifier :

- soit avec un identifiant et un mot de passe (un clavier virtuel s'affiche pour chaque saisie)
- soit avec un badge sans contact RFID

Une fois authentifié, l'utilisateur pourra prendre et rendre des articles. Il pourra aussi visualiser ses mouvements.

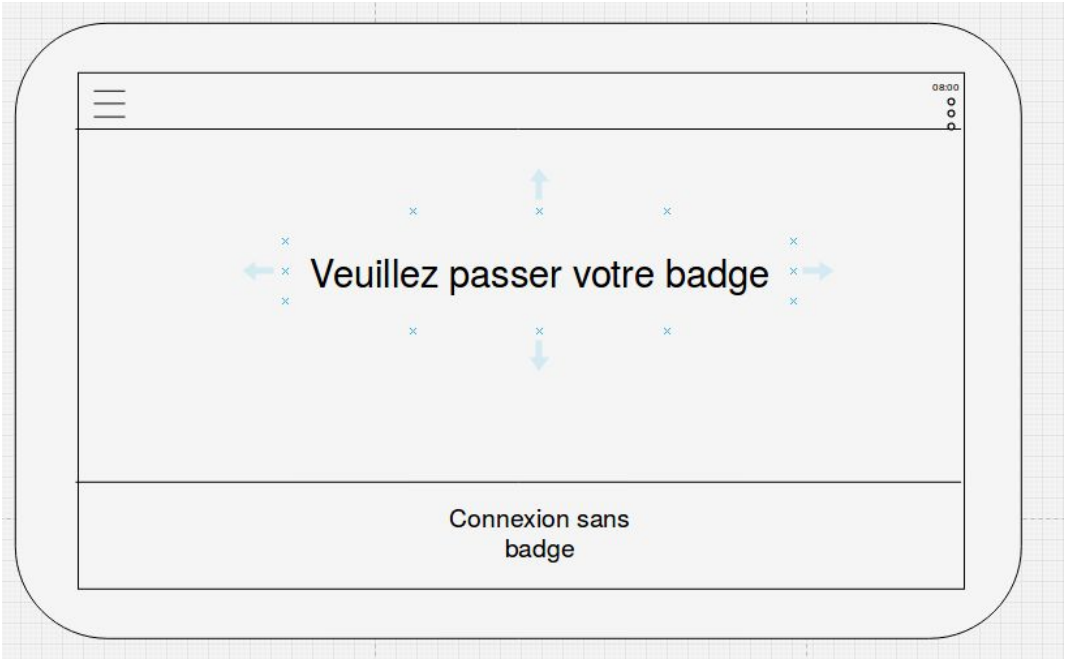
Pour faciliter l'utilisation, il sera possible de rechercher un article.

Maquette IHM

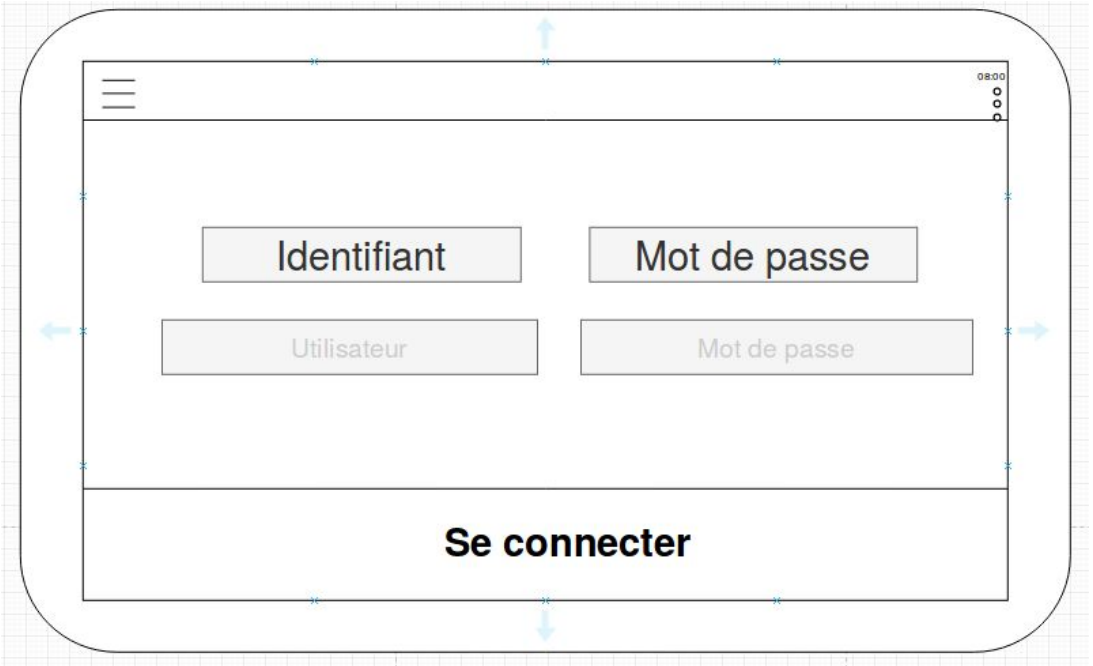
L'utilisateur (Utilisateur simple ou Gestionnaire ou Administrateur) doit au lancement de l'application s'authentifier. Pour s'authentifier l'utilisateur a 2 solution :

- Un badge RFID
- Un compte utilisateur

S'authentifier avec badge :



S'authentifier sans badge :



IHM

Authentification avec badge

Authentification sans badge

Remarque : un clavier virtuel sera alors intégré à l'application.

Lecteur de badge RFID

Les utilisateurs disposeront d'un badge RFID 13.56 MHz de type MIFARE.

Pour assurer l'authentification par badge des utilisateurs, l'armoire intègre un lecteur de badge Omnikey 5427 CK.

Le Lecteur Omnikey 5427 CK est un lecteur sans contact bi-fréquence 13.56 MHz et 125 kHz. Cela lui permet de prendre en charge les badges ou cartes à puce sans contact suivants :

- HID Prox
- MIFARE Classic, MIFARE DESFire EV1,
- iCLASS et iCLASS SE
- Autres cartes compatibles SIO

Le lecteur dispose de la fonctionnalité d'**émulation clavier** qui permet de récupérer les données du badge et de les transférer à l'application par émulation de la séquence clavier. Cette fonction permet aussi de le rendre indépendant du système d'exploitation.

Le lecteur dispose d'une connectique en **USB**.

NFC (Near Field Communication, communication en champ proche) est une technologie de communication sans-fil à courte portée et haute fréquence, permettant l'échange d'informations entre des périphériques jusqu'à une distance d'environ 10 cm. Cette technologie est une extension de la norme ISO/CEI 14443 standardisant les cartes de proximité utilisant la radio-identification (RFID).

MIFARE est une des technologies de carte à puce sans contact les plus répandues dans le monde avec 3,5 milliards de cartes et 40 millions de modules de lecture/encodage. La marque, lancée par Philips, est propriété de la société NXP. MIFARE est fondée (partiellement ou complètement selon les modèles) sur l'un des standards ISO décrivant les cartes à puce sans contact : l'ISO 14443 de

Type A fonctionnant à 13,56 MHz. La technologie est intégrée à la fois dans les cartes et dans les lecteurs/encodeurs.

En fait, le nom MIFARE englobe plusieurs types de cartes à puce sans contact très différents.

Les cartes ou badges MIFARE Classic sont des cartes mémoires disposant d'un numéro de série (appelé aussi UID ou CSN) sur 32 bits (pour la MIFARE Classic) ou de 56 bits (pour la "MIFARE Ultralight" et la "MIFARE Classic EV1") pré-encodé et d'un espace de stockage découpé en segments de données puis en blocs de données avec des mécanismes de sécurité simples.

Remarque : Les UID des cartes Mifare Classic codés sur 32 bits permettent environ 4 milliards de valeurs possibles. Il y a donc potentiellement des risques de doublons. De plus, il existe des clones d'origine chinoise compatibles avec les produits MIFARE dont les numéros de série sont probablement assignés aléatoirement ou séquentiellement.

Badge RFID

Mémoire de la puce MIFARE ®
En version standard, le badge MIFARE ® dispose de 1024 octets de mémoire, structurée en 16 secteurs de 4 blocs chacun. Chaque bloc de la puce offre 16 octets de mémoire, qui peuvent être encodés puis lus à volonté.
Badge MIFARE ® de contrôle d’accès
Le badge MIFARE ® en version 1Ko ou 4Ko de mémoire est idéal pour le contrôle d’accès en entreprise. Les badges 13,56Mhz communiquent avec le lecteur MIFARE à une distance comprise entre 1 et 6cm : leur utilisation est facile, agréable et rapide.
Numéro de série MIFARE ®
Chaque puce de badge MIFARE comprend un numéro de série unique, gravé d’origine dans la puce. L’UID (User IDentifier) peut être de 4 octets ou de 7 octets pour éviter les doublons avec des badges fournis dans le passé.

Scénario : quand on s'authentifie avec le badge

Le lecteur fonctionne en mode **QWERTY**. L'UID est récupéré sous la forme d'une chaîne de caractère :

(àééQè_"

&	é	"	'	(	-	è	_	ç	à
q	w	e	r	t	y	u	i	o	p
a	s	d	f	g	h	j	k	l	:
|	z	x	c	v	b	n	m	<	>

On retranscrit l'UID en hexadécimal avec une correspondance en mode **AZERTY** :

"5022A783"

1	2	3	4	5	6	7	8	9	0
a	z	e	r	t	y	u	i	o	p
q	s	d	f	g	h	j	k	l	:
<	w	x	c	v	b	n	:	!	

L'authentification a été réalisée et l'utilisateur "MACHON Thomas" reconnu, on affiche l'IHM gestion armoire

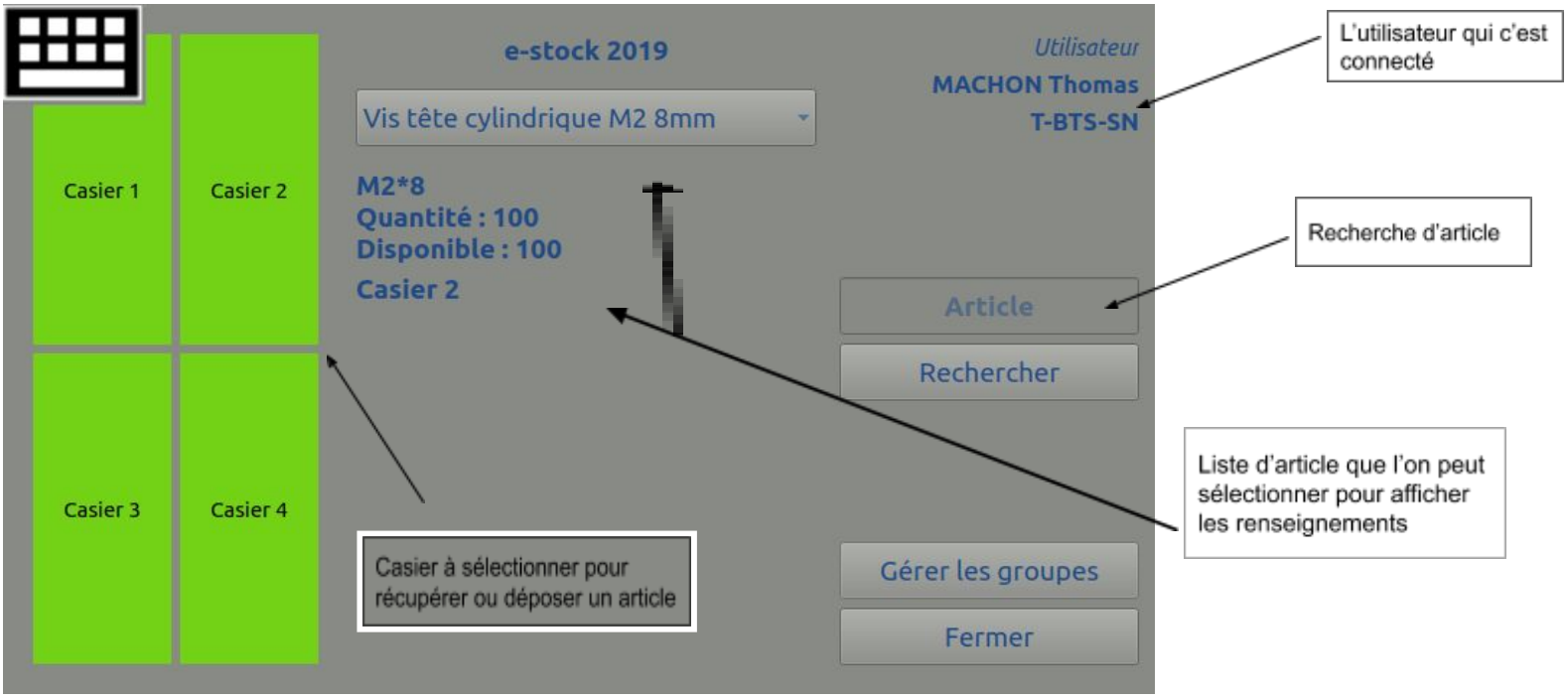
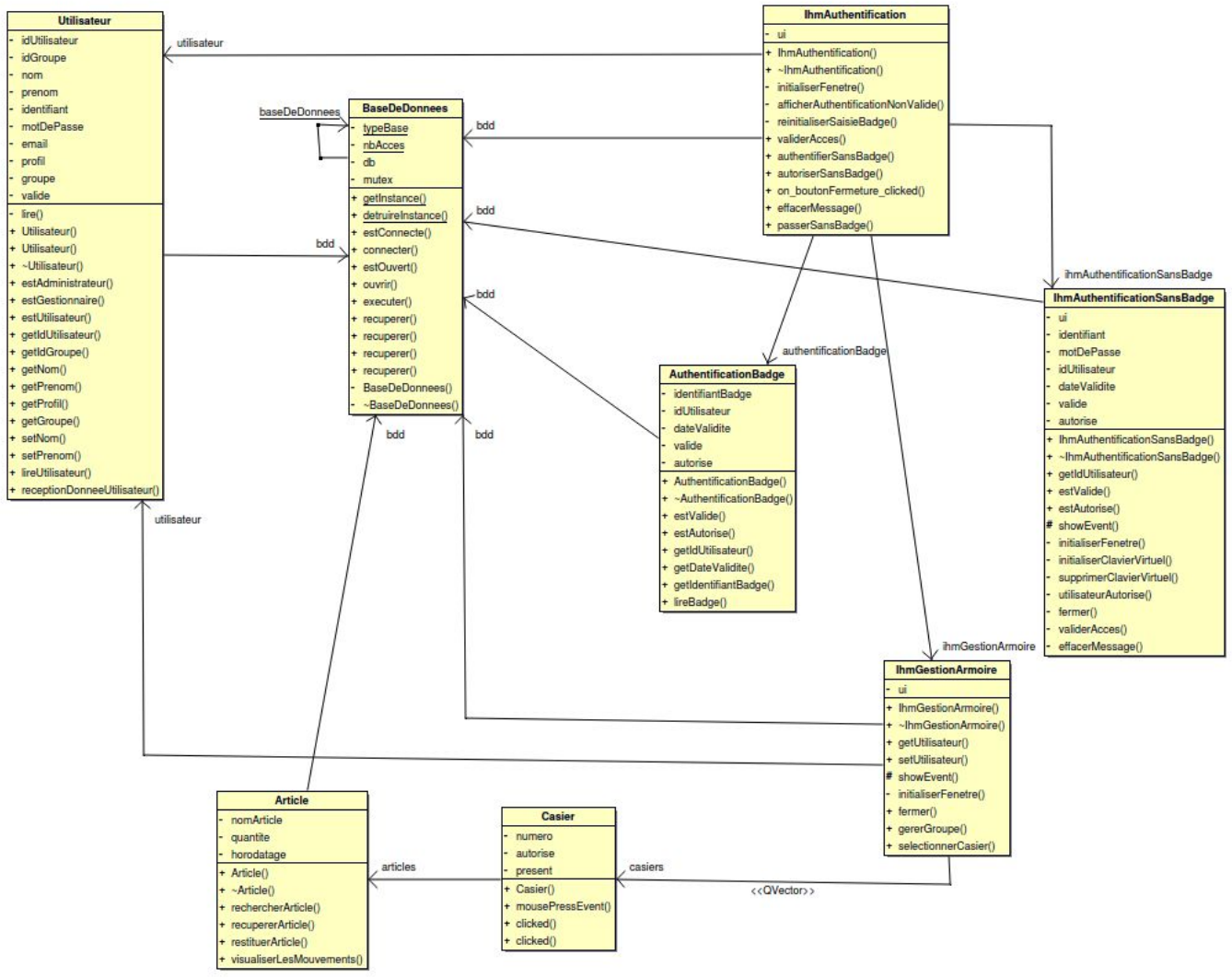


Diagramme de classes



Pour la partie “Authentification”, on dispose de deux classes pour assurer l’affichage des fenêtres :

- la classe **IhmAuthentification** qui va correspondre à la fenêtre d’accueil et permettra à l’utilisateur de s’authentifier avec un badge.
- la classe **IhmAuthentificationSansBadge** qui elle affichera une fenêtre de saisie de l’identifiant et mot de passe d’un utilisateur ne disposant pas de badge.

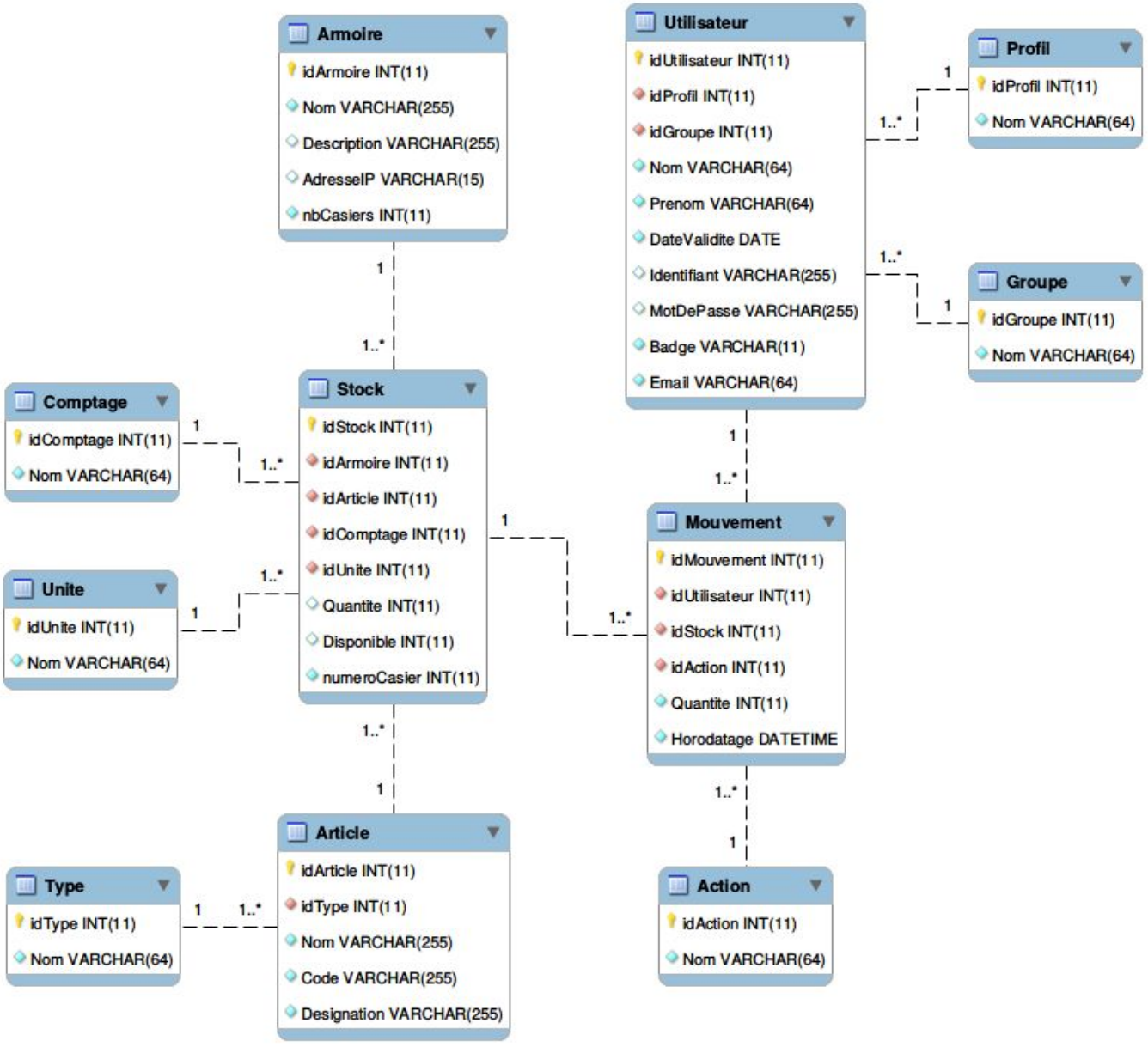
La classe **IhmAuthentification** utilise les services de la classe **AuthentificationBadge** qui va s’assurer de récupérer l’identifiant du badge et de sa validité.

Si l’authentification réussie, on aura besoin de la classe **Utilisateur** qui contiendra toutes les information sur l’utilisateur.

Une fois authentifié que ce soit avec ou sans badge, on affichera la fenêtre principale à partir de la classe **IhmGestionArmoire**.

La classe **Casier** va permettre à l’utilisateur de choisir un numéro de casier dans lequel il va pouvoir récupérer les articles. Et la classe **Article** va permettre à l’utilisateur d'interagir avec l’objet ou l’outil souhaité.

Base de données



Les tables qui concerne ma partie son : **Utilisateur, Profil, Article, Stock, Mouvement**

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

La table **Utilisateur** contient :

- ➔ une clé étrangère idProfil qui précise son profil
- ➔ une clé étrangère idGroupe qui indique son groupe d'appartenance
- ➔ un champ Nom
- ➔ un champ Prenom
- ➔ un champ DateValidite

- ➔ un champ Identifiant
- ➔ un champ MotDePasse
- ➔ un champ Badge
- ➔ un champ Email

La table **Article** définit un article par :

- ➔ une clé étrangère idType qui précise son type
- ➔ un champ Nom
- ➔ un champ Code (code barre)
- ➔ un champ Designation

La table **Stock** contient :

- ➔ une clé étrangère idArmoire qui l'associe à une armoire
- ➔ une clé étrangère idArticle qui indique l'article stocké
- ➔ une clé étrangère idComptage qui précise le type de comptage
- ➔ une clé étrangère idUnite qui indique l'unité utilisée pour le compter
- ➔ un champ Quantite qui fournit le stock de départ pour cet article
- ➔ un champ Disponible qui comptabilise la présence de l'article dans le stock actuellement
- ➔ un champ NumeroCasier

La table **Mouvement** contient l'ensemble des entrées/sorties des articles dans l'armoire :

- ➔ une clé étrangère idUtilisateur qui indique l'utilisateur qui a effectué le mouvement
- ➔ une clé étrangère idStock qui associe le mouvement dans le stock
- ➔ une clé étrangère idAction qui fournit l'action réalisée
- ➔ un champ Quantité
- ➔ un champ Horodatage

Voici la structure de la table Utilisateur dans la base données avec son contenu :

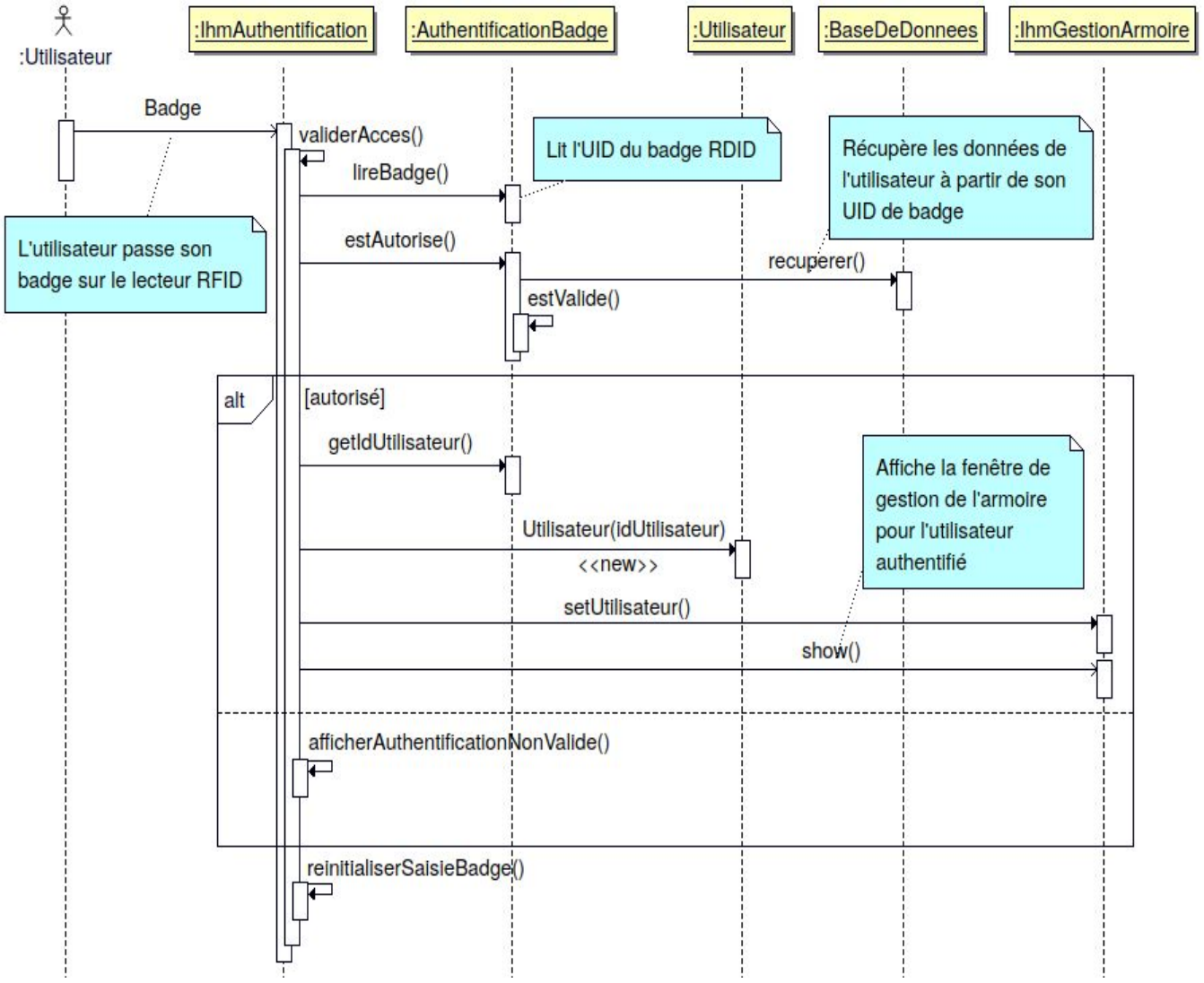

```sql
CREATE TABLE `Utilisateur` (
  `idUtilisateur` int(11) NOT NULL,
  `idProfil` int(11) NOT NULL,
  `idGroupe` int(11) NOT NULL,
  `Nom` varchar(64) NOT NULL,
  `Prenom` varchar(64) NOT NULL,
  `DateValidite` date NOT NULL,
  `Identifiant` varchar(255) DEFAULT NULL,
  `MotDePasse` varchar(255) DEFAULT NULL,
  `Badge` varchar(11) NOT NULL,
  `Email` varchar(64) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```sql
INSERT INTO `Utilisateur` (`idUtilisateur`, `idProfil`, `idGroupe`, `Nom`,
`Prenom`, `DateValidite`, `Identifiant`, `MotDePasse`, `Badge`, `Email`) VALUES
(1, 1, 1, 'Vaira', 'Thierry', '2019-07-01', 'admin',
'5f4dcc3b5aa765d61d8327deb882cf99', '', 'tvaira@free.fr'),
(2, 2, 3, 'Vaira', 'Thierry', '2019-07-01', 'tvaira',
'5f4dcc3b5aa765d61d8327deb882cf99', '1234', 'tvaira@free.fr'),
(3, 2, 3, 'Beaumont', 'Jerome', '2019-07-01', 'jbeaumont', '', '5678',
'beaumont@lasalle84.org'),
(4, 3, 5, 'ANDREO', 'Michaël', '2019-07-01', 'andreo.m', '', '1111',
'andreo.michael@outlook.fr'),
(5, 3, 5, 'BOFFREDO', 'Nicolas', '2019-07-01', 'boffredo.n', '', '2222',
'nboffredo@gmail.com'),
(6, 3, 5, 'BOTELLA', 'Yohann', '2019-07-01', 'botella.y', '', '3333',
'botellabroc.yohann@gmail.com'),
(7, 3, 5, 'GAUTHIER', 'Robin', '2019-07-01', 'gauthier.r', '', '4444',
'rgauthier2510@gmail.com'),
(8, 3, 5, 'GIMENEZ', 'Hadrien', '2019-07-01', 'gimenez.h', '', '5555',
'gimenezhadrien@gmail.com'),
(9, 3, 5, 'HAMMOUMA', 'Youssef', '2019-07-01', 'hammouma.y', '', '6666',
'yhammouma@gmail.com'),
(10, 3, 5, 'LAURAIN', 'Clément', '2019-07-01', 'laurain.c', '', '7777',
'laurain.clement.contact@gmail.com'),
(11, 3, 5, 'MACHON', 'Thomas', '2019-07-01', 'machon.t', '', '62A90261',
'thomaslasalle84@gmail.com'), ...
```

Cas d'utilisation : S'authentifier

Scénario Authentification avec badge

Ce diagramme de séquence montre l'authentification par badge.



Tout d'abord l'utilisateur présente son badge ensuite on lit l'UID du badge. On interroge la base de données pour vérifier si l'UID est correct. Si l'UID est autorisé, on crée l'utilisateur et on affiche la fenêtre IHMGestionArmoire. Si l'UID est pas autorisé alors on affiche un message d'erreur et on réinitialise la saisie.

A l'aide des qDebug, on peut visualiser les opérations réalisées ainsi que l'information contenue dans le badge (l'UID) :

```
bool AuthentificationBadge::lireBadge(QString) badge "RFIDM(àééQè_""
bool AuthentificationBadge::lireBadge(QString) identifiantBadge "5022A783"
bool AuthentificationBadge::estAutorise() valide true
bool AuthentificationBadge::estValide() requete "SELECT DateValidite FROM
Utilisateur WHERE Badge='5022A783' AND DateValidite>='2019-05-10'"
bool AuthentificationBadge::estValide() valide true dateValidite "2019-07-01"
```

Les données lues par le lecteur de badge (`"RFIDM(àééQè_"`) sont structurées de la manière suivante :

> **RFID** : début
> **M** : délimiteur (touche : d'un clavier QWERTY)
> **(àééQè_"** : l'UID du badge « saisie » par le clavier QWERTY

Ensuite on retranscrit la chaîne de caractère "**(àééQè_"**" en chaîne de caractères correspondant au touche du clavier AZERTY "**5022A783**" et on obtient l'UID sous sa forme hexadécimale.

La méthode **validerAcces()** va permettre de lire l'UID du badge et vérifier si il est autorisé et valide à l'aide des méthodes estAutorise() et estValide().

```
void IhmAuthentification::validerAcces()
{
      if(authentificationBadge->lireBadge(ui->saisieBadge->text()))
      {
            if(authentificationBadge->estAutorise())
            {
                  utilisateur = new
Utilisateur(authentificationBadge->getIdUtilisateur());
                        ihmGestionArmoire->setUtilisateur(utilisateur);
                        ihmGestionArmoire->show();
            }
            else
            {
                        afficherAuthentificationNonValide();
            }
      }
      reinitialiserSaisieBadge();
}
```

La méthode **estAutorise**() va vérifier si le badge est autorisé à partir d'une requête SQL et récupérer l'idUtilisateur et la DateValidite :

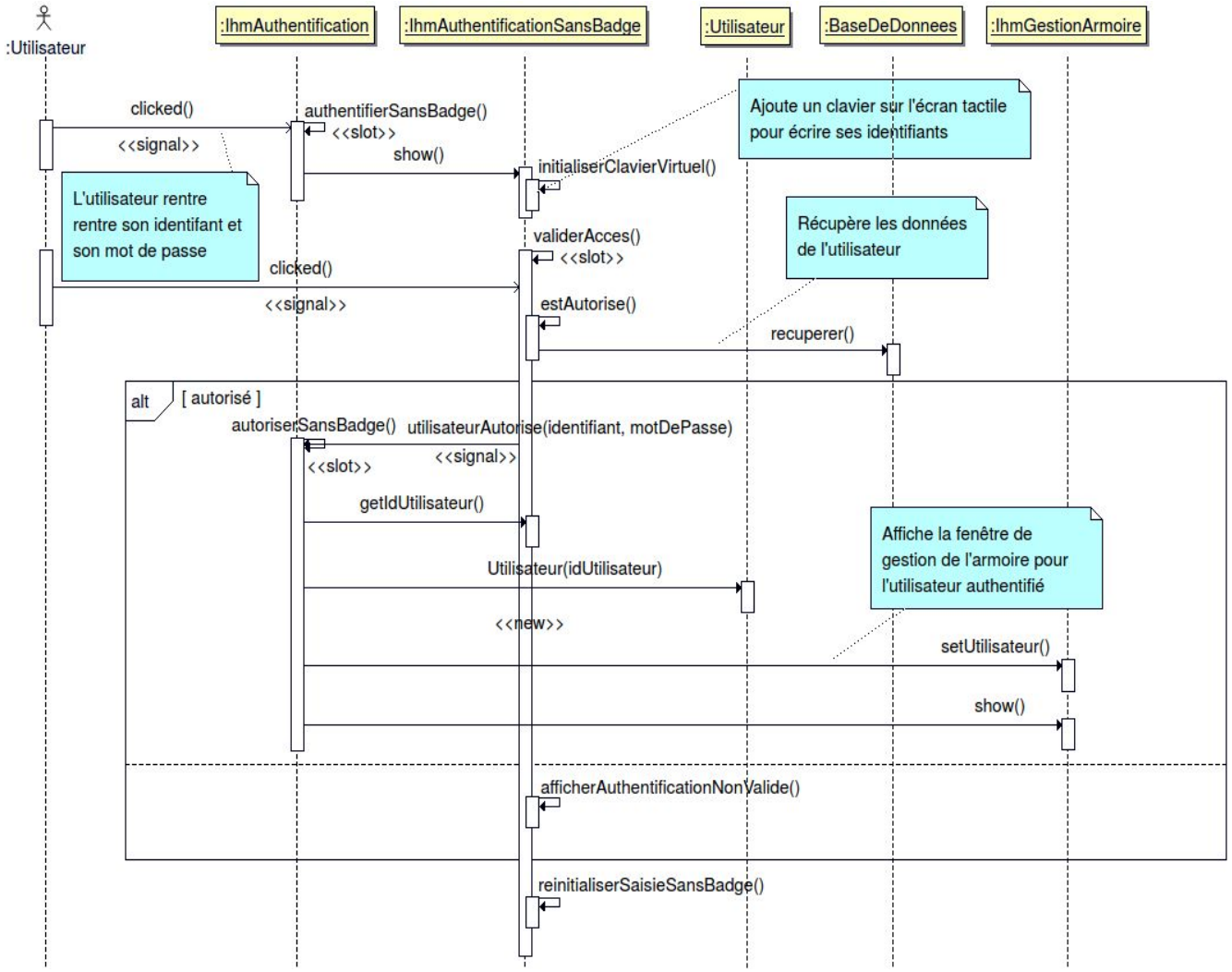
```
bool AuthentificationBadge::estAutorise()
{
   if(!valide)
       return false;
   QString requete = "SELECT idUtilisateur, DateValidite FROM Utilisateur WHERE
Badge='" + identifiantBadge + "'";
   QStringList donneesUtilisateur;
   idUtilisateur.clear();
   dateValidite.clear();
   valide = bdd->recuperer(requete, donneesUtilisateur);
   if(valide)
   {
       idUtilisateur = donneesUtilisateur.at(0);
       dateValidite = donneesUtilisateur.at(1);
       qDebug() << Q_FUNC_INFO << "valide" << valide << "idUtilisateur" <<
idUtilisateur << "dateValidite" << dateValidite;
   }
   qDebug() << Q_FUNC_INFO << "valide" << valide;
   return (valide && estValide());
}
```

La methode **estValide**() va permettre à l'aide d'une requête **SQL** de vérifier la date de validité de l'utilisateur :

```
bool IhmAuthentificationSansBadge::estValide()
{
      QString dateMaintenant =
QDateTime::currentDateTime().toString("yyyy-MM-dd");
      QString requete = "SELECT DateValidite FROM Utilisateur WHERE Identifiant='"
+ ui->lineEditAuthentification->text() + "' AND DateValidite>='" + dateMaintenant
+ "'";
      qDebug() << Q_FUNC_INFO << "requete" << requete;
      QString retour;
      valide = bdd->recuperer(requete, retour);
      qDebug() << Q_FUNC_INFO << "valide" << valide << "dateValidite" <<
dateValidite;
      return valide;
}
```

Scénario Authentification sans badge

Ce diagramme de séquence montre l'authentification sans badge.

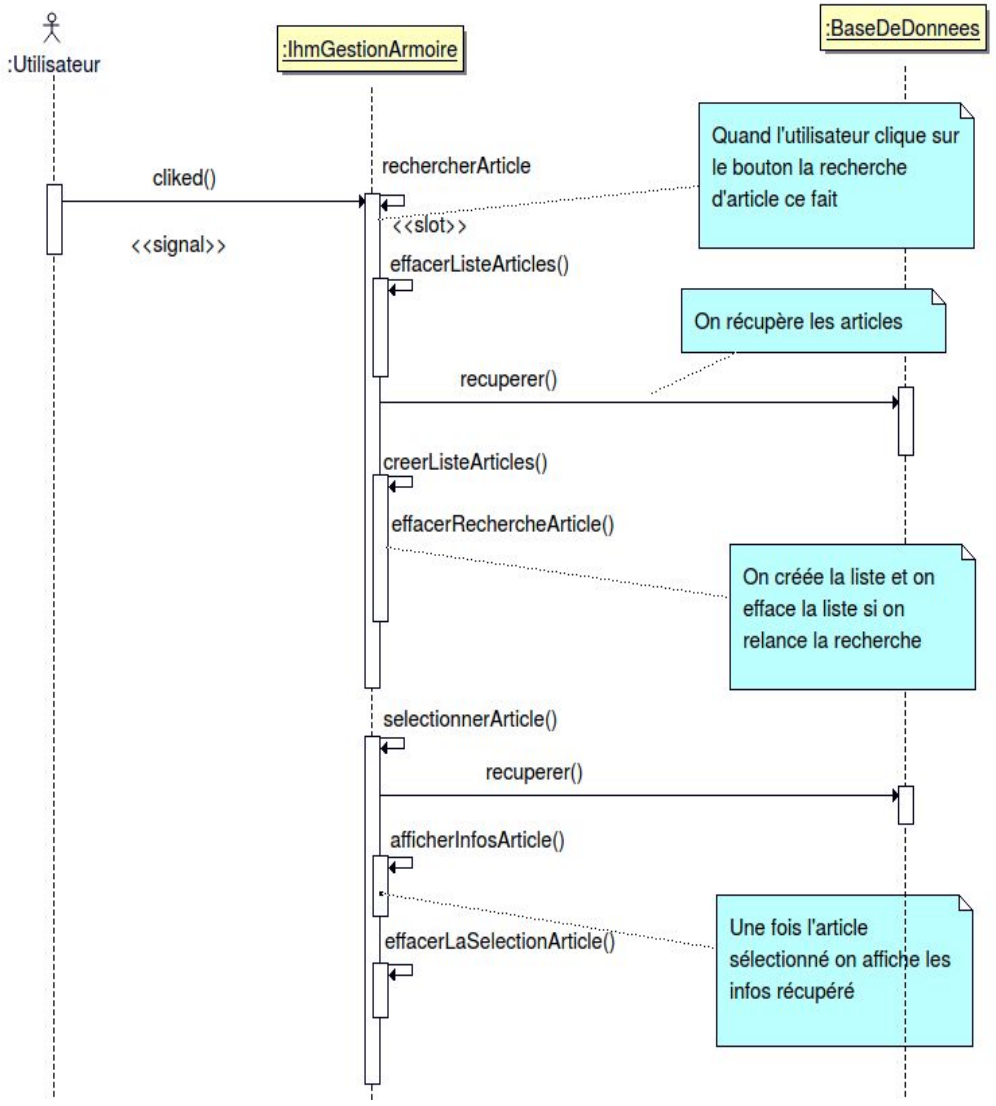


L'utilisateur clique sur le bouton sans badge. La fenêtre **ihmAuthentificationSansBadge** va alors s'afficher. L'utilisateur va devoir entrer son identifiant ainsi que son mot de passe (un clavier virtuel lui permettra d'assurer une saisie). Les données seront ensuite vérifiées avec une requête depuis la **BaseDeDonnees**. Si l'utilisateur est autorisé, on crée l'utilisateur et on affiche la nouvelle fenêtre IHMGestionArmoire. Sinon un message d'erreur s'affiche et il pourra recommencer.

Cas d'utilisation : Rechercher un article

Scénario Recherche d'un article

Ce diagramme de séquence montre la recherche d'un article et la sélection d'un article :



La méthode **rechercherArticle**() va permettre à l'utilisateur de rechercher des articles qu'il a besoin. Pour ce faire on utilise une requête **SQL** qui récupère les lettres saisies par l'utilisateur et qui va fournir tous les articles comprenant ce mot. L'ensemble des articles trouvés s'afficheront dans une liste déroulante.

```
void IhmGestionArmoire::rechercherArticle()
{
	QString mot = ui->lineEditRechercheArticle->text();
	qDebug() << Q_FUNC_INFO << mot;
	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 '%" + mot +
"%' OR Article.Code LIKE '%" + mot + "%' OR Article.Designation LIKE '%" + mot +
"%' ORDER BY Stock.NumeroCasier ASC";
	qDebug() << Q_FUNC_INFO << requete;

	effacerListeArticles();

	bdd->recuperer(requete, articlesTrouves);
	qDebug() << Q_FUNC_INFO << "articlesTrouves" << articlesTrouves.size() <<
articlesTrouves;

	creerListeArticles();

	effacerRechercheArticle();
}
```

Ce qu'on obtient quand on recherche l'intégralité des articles :

void IhmGestionArmoire::rechercherArticle() articlesTrouves
QVector(("1", "2", "Vis six pans creux M2 8mm", "", "M2*8", "1", "1", "1", "2", "2", "100", "100", "1"), ("2", "2", "Vis tête cylindrique M2 8mm", "", "M2*8", "2", "1", "2", "2", "2", "100", "100", "2"), ("3", "2", "Vis six pans creux M2 12mm", "", "M2*12", "3", "1", "3", "2", "2", "100", "100", "3"), ("4", "2", "Vis tête cylindrique M2 12mm", "", "M2*12", "4", "1", "4", "2", "2", "100", "100", "4"), ("5", "1", "Fluke i30s", "2584935", "Amperemetre AC/DC", "5", "1", "5", "2", "2", "8", "6", "5"), ("5", "1", "Fluke i30s", "2584935", "Amperemetre AC/DC", "6", "1", "5", "2", "2", "8", "8", "6"), ("6", "1", "Fluke 179", "", "Multimetre", "7", "1", "6", "3", "2", "2", "2", "7"))

En rouge on a le nom de l'article qui sera affiché dans la liste déroulante une fois la recherche réalisée
En bleu on aura le numéro du casier de l'article quand on le sélectionne dans la liste.

L'utilisateur pourra sélectionner un article dans la liste et on indiquera son numéro de casier ainsi que la quantité et le nombre disponible actuellement :

```
void IhmGestionArmoire::selectionnerArticle(int index)
{
	if(articlesTrouves.size() == 0 || index < 0)
		return;
	QString Article = articlesTrouves[index].at(12);
	ui->labelMessageArticle->setText(Article);
}
```

```
void IhmGestionArmoire::afficherInfosArticle()
{
      QString article = articlesTrouves[index-1].at(5);
      QString qte = QString::fromUtf8("Quantité : ") +
articlesTrouves[index-1].at(3);
      QString dispo = QString::fromUtf8("Disponible : ") +
articlesTrouves[index-1].at(4);
      ui->labelMessageArticle->setText(article + "\n" + qte + "\n" + dispo);
      ui->labelMessageNumeroCasier->setText("Casier " +
articlesTrouves[index-1].at(0));
}
```

En cliquant sur le casier correspondant, on pourra alors actionner l'ouverture de celui-ci pour "Prendre un article".

Tests de validation

Test	Badge / identifiant	Résultats attendus	Résultats obtenus	Valide (Oui/Non)
Badge autorisé Date valide	30DDA983	valide = true dateValidite = "2019-07-01" Affichage Fenêtre GestionArmoire	valide = true dateValidite = "2019-07-01" Affichage Fenêtre GestionArmoire	**oui**
Badge autorisé Date non valide	62A3F560	valide = false dateValidite "2018-07-01" Affichage Fenêtre IhmAuthentification avec message "Badge non valide !"	valide = false dateValidite "2018-07-01" Affichage Fenêtre IhmAuthentification avec message "Badge non valide !"	**oui**
Badge non autorisé	5022A783	valide = false dateValidite "" Affichage Fenêtre IhmAuthentification avec message "Badge non valide !"	valide = false dateValidite "" Affichage Fenêtre IhmAuthentification avec message "Badge non valide !"	**oui**
Sans badge Identifiant Mot de passe valide	machon.t 170796	bool IhmAuthentificationSansBadge::estAutorise() valide true Affichage Fenêtre GestionArmoire	bool IhmAuthentificationSansBadge::estAutorise() valide true Affichage Fenêtre GestionArmoire	**oui**
Sans badge Identifiant non valide	mocho.t 170796	bool IhmAuthentificationSansBadge::estAutorise() valide false Message “Identifiant non valide”	bool IhmAuthentificationSansBadge::estAutorise() valide false Message “Identifiant non valide”	**oui**
Sans badge Mot de passe non valide	machon.t 170799	bool IhmAuthentificationSansBadge::estAutorise() valide false Message “Identifiant non valide”	bool IhmAuthentificationSansBadge::estAutorise() valide false Message “Identifiant non valide”	**oui**
Recherche d’article	MACHON Thomas	void IhmGestionArmoire::rec	void IhmGestionArmoire::rech	

		hercherArticle() articlesTrouves QVector(("1", "2", "Vis six pans creux M2 8mm", "", "M2*8", "1", "1", "1", "2", "2", "100", "100", "1"),	ercherArticle() articlesTrouves QVector(("1", "2", "Vis six pans creux M2 8mm", "", "M2*8", "1", "1", "1", "2", "2", "100", "100", "1"),	**oui**

Partie personnel : Gimenez Hadrien.

Récupérer la pesée des casiers grâce à la liaison avec l'Arduino , Editer les groupes (Ajouter , Modifier ou Supprimer un groupe) et l'envoie de l'état du stock par mail

- ❒ On peut créer, modifier ou supprimer un groupe
- ❒ Une configuration minimale du système est possible
- ❒ La communication avec le SE permet la récupération des pesées
- ❒ La gestion des balances est fonctionnelle (visualisation des pesées, tarage)
- ❒ L'envoi de l'état du stock par email est fonctionnel

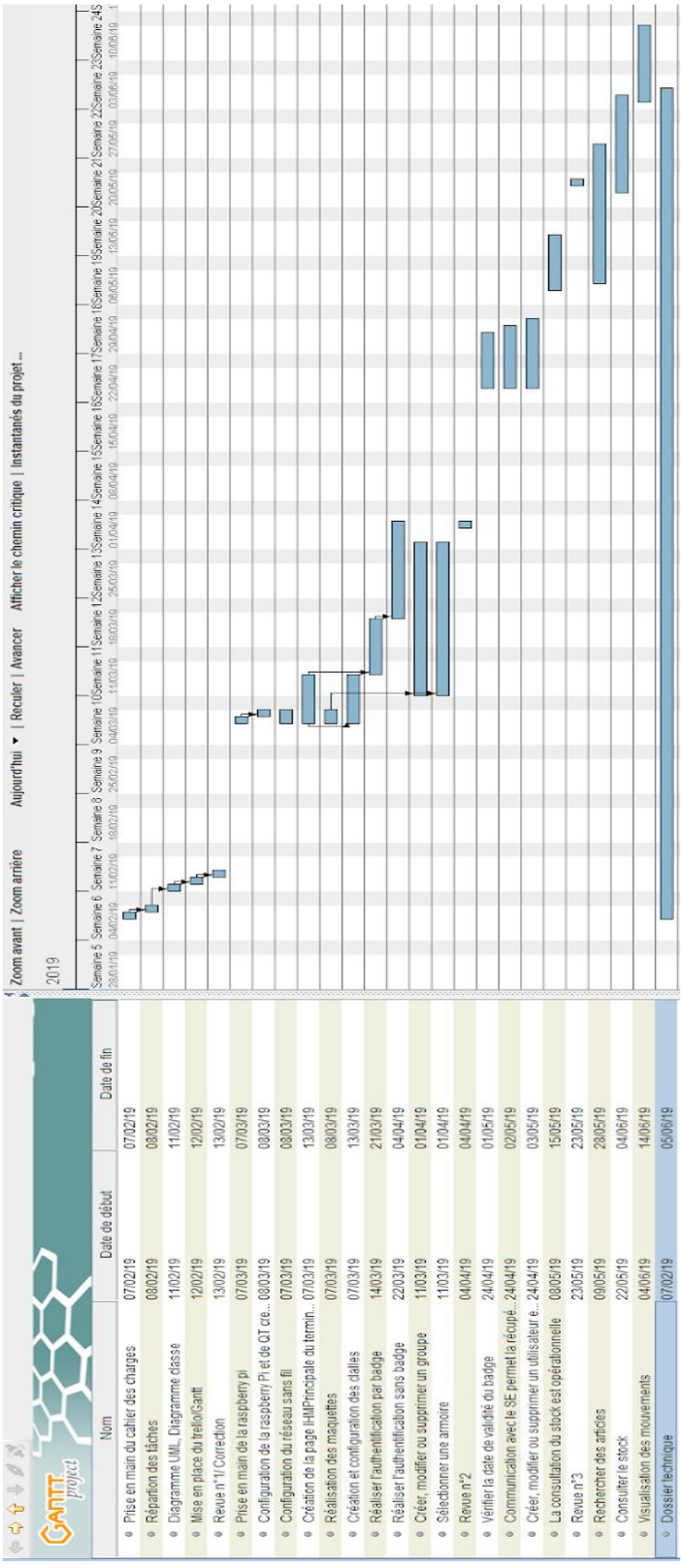
Désignation	**Caractéristiques**
RPI	Raspberry PI 3
MINI-ECRAN	Écran tactile 800x480 7" relié à la RPI
Système d'exploitation de la RPI	GNU/Linux Raspbian
Système de gestion de bases de données relationnelles	MySQL,
Atelier de génie logiciel (IR)	BOULM v.7.8
Logiciel de gestion de versions	subversion (RiouxSVN)
API GUI	Qt creator (Enterprise) v.5.11.2

Planification

Répartition des tâches

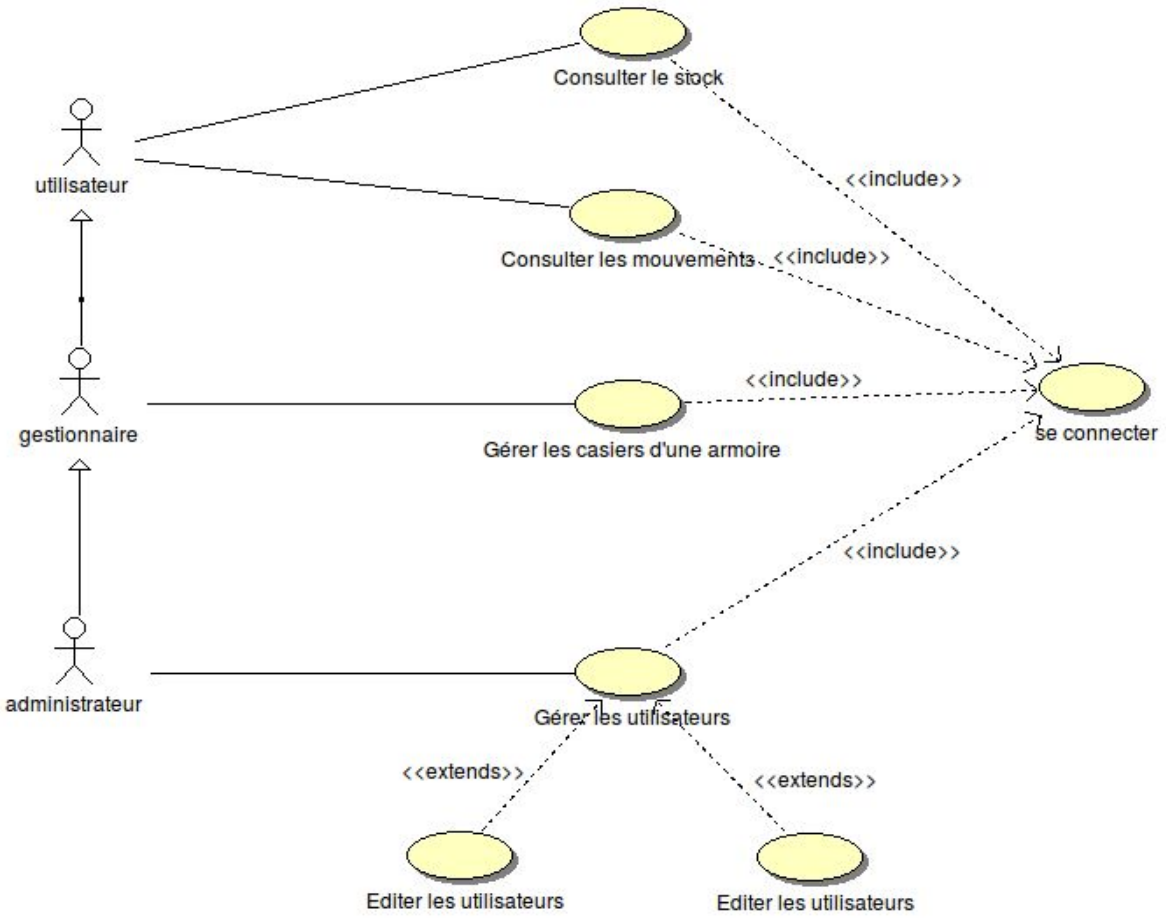
Tâches à réaliser	**Priorité**	**Itération**
Editer groupes	1	0.1
Récupérer la pesée des casiers	1	0.2
Alerter par mail	2	0.3

Diagramme de Gantt



Le diagramme de Gantt permet la planification du projet et de visualiser dans le temps les diverses tâches à accomplir.
Les blocs définissent les tâches à réaliser, les jalons définissent le début et la fin d'une tâche.
Certaine tâches ne peuvent pas être commencé avant d'en avoir fini une autre (exemple: il n'est pas possible de commencer le projet avant de s'être réparti les tâches).

Diagramme de cas d'Utilisation administrateur



L'administrateur hérite du cas d'utilisation du gestionnaire et de l'utilisateur il peux donc lui aussi consulter le stock , les mouvements dans le stock la gestion des casiers d'une armoire.

Mais sa spécificité est de pouvoir gérer les utilisateurs, c'est à dire editer , modifier , supprimer les groupes ainsi que ses utilisateurs.

IHM

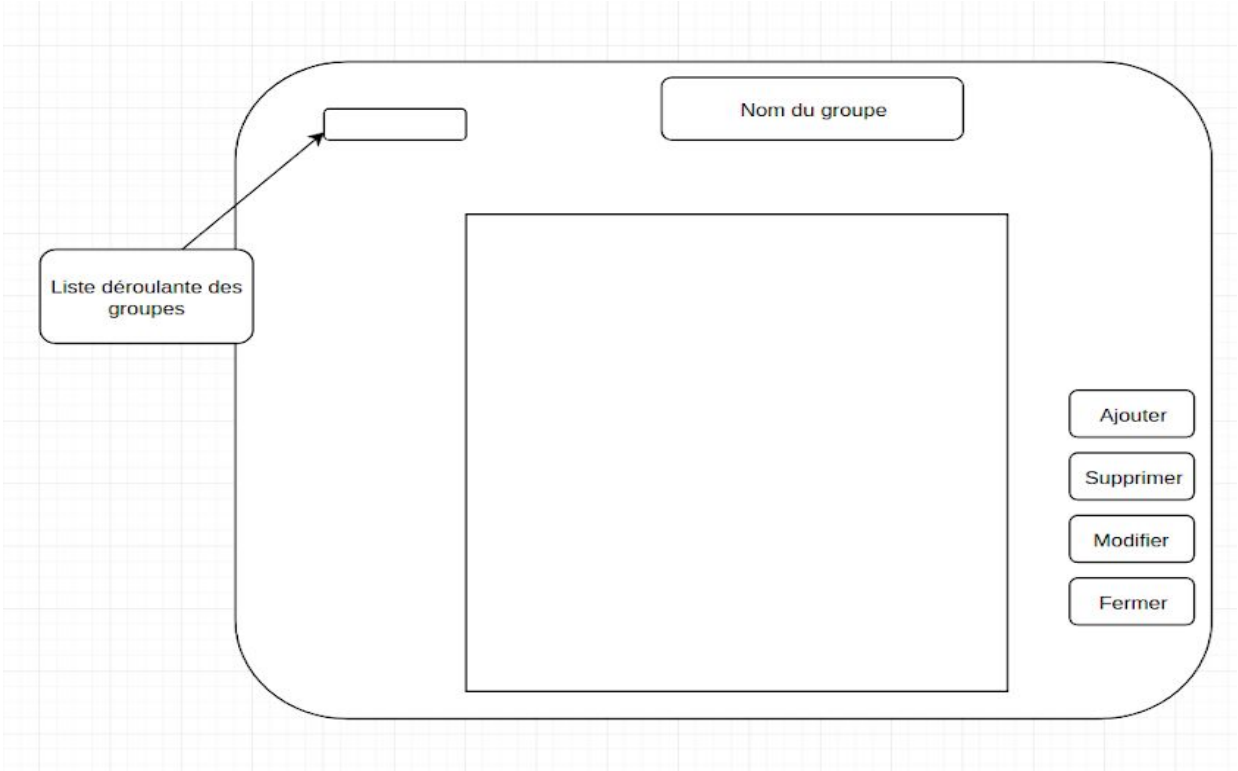
Voici l'ihm principal une fois l'authentification effectuer.

Nous pouvons voir à droite les 4 casiers sélectionnable, ainsi que au milieu la liste déroulante avec les articles.

A gauche un champ de recherche pour les articles ainsi que le bouton Gérer les groupes qui sera accessible uniquement par l'administrateur.

Maquette IHM

Gérer les groupes



Cette IHM est accessible uniquement à l'administrateur.

L'administrateur a la possibilité de gérer les groupes, c'est à dire d'ajouter modifier ou supprimer un groupe.

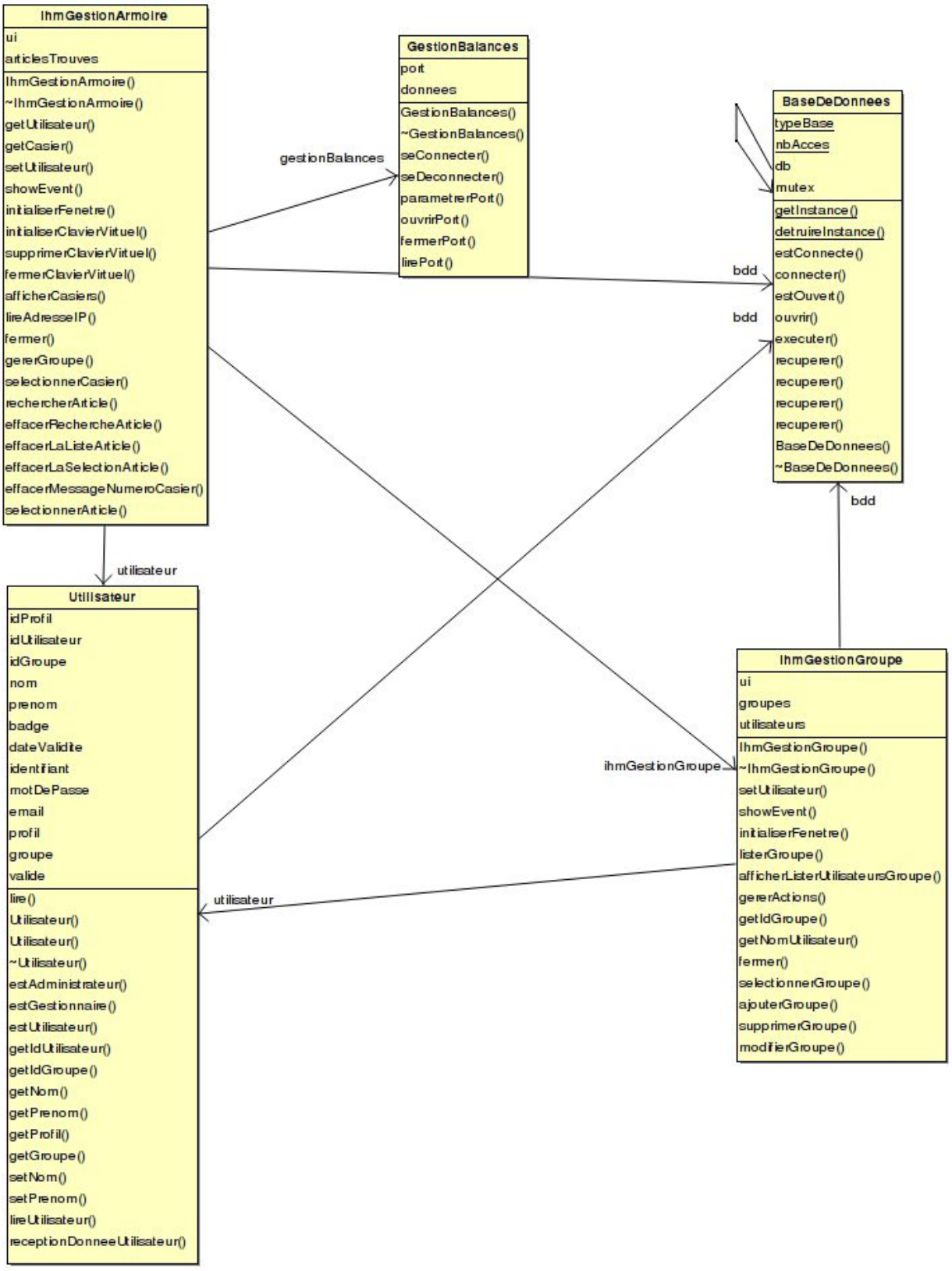
En quoi consiste un groupe ?

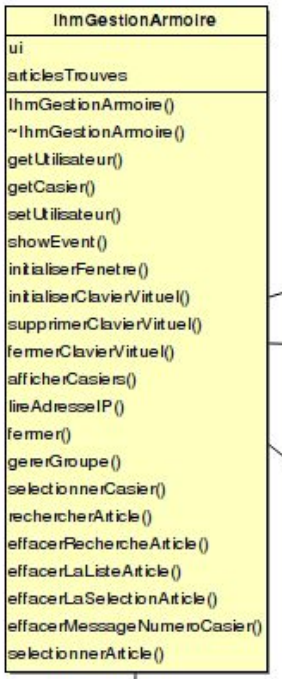
Le groupe permet de regrouper plusieurs utilisateur dans une seul catégorie pour permettre de leurs données à tous les mêmes droits sans devoir répéter l'opération.

Elle possède une liste déroulante affichant les groupes présents, un champ "nom de groupe" permettant d'entrer le nom du groupe qu'il veut ajouter.
A droite, on retrouve les boutons ajouter, supprimer, modifier et fermer.

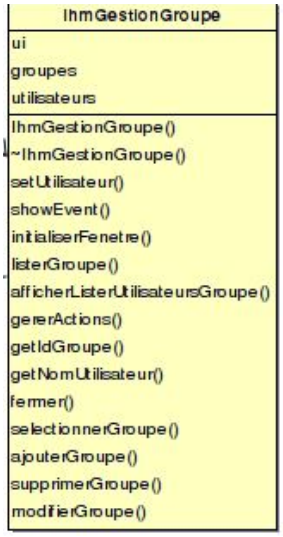
Les boutons sont connectés à des méthodes qui effectuent des requêtes SQL.

Diagramme de classes

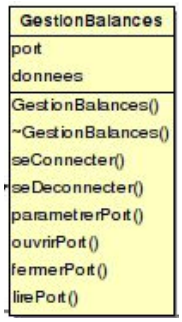




❒ La classe **IHMGestionArmoire** est la fenêtre principal une fois que l'utilisateur c'est authentifier , une fois sur cette fenêtre s'il est Administrateur ou Gestionnaire il peut accéder à la gestion de groupe géré par la classe **IHMGestionGroupe.**



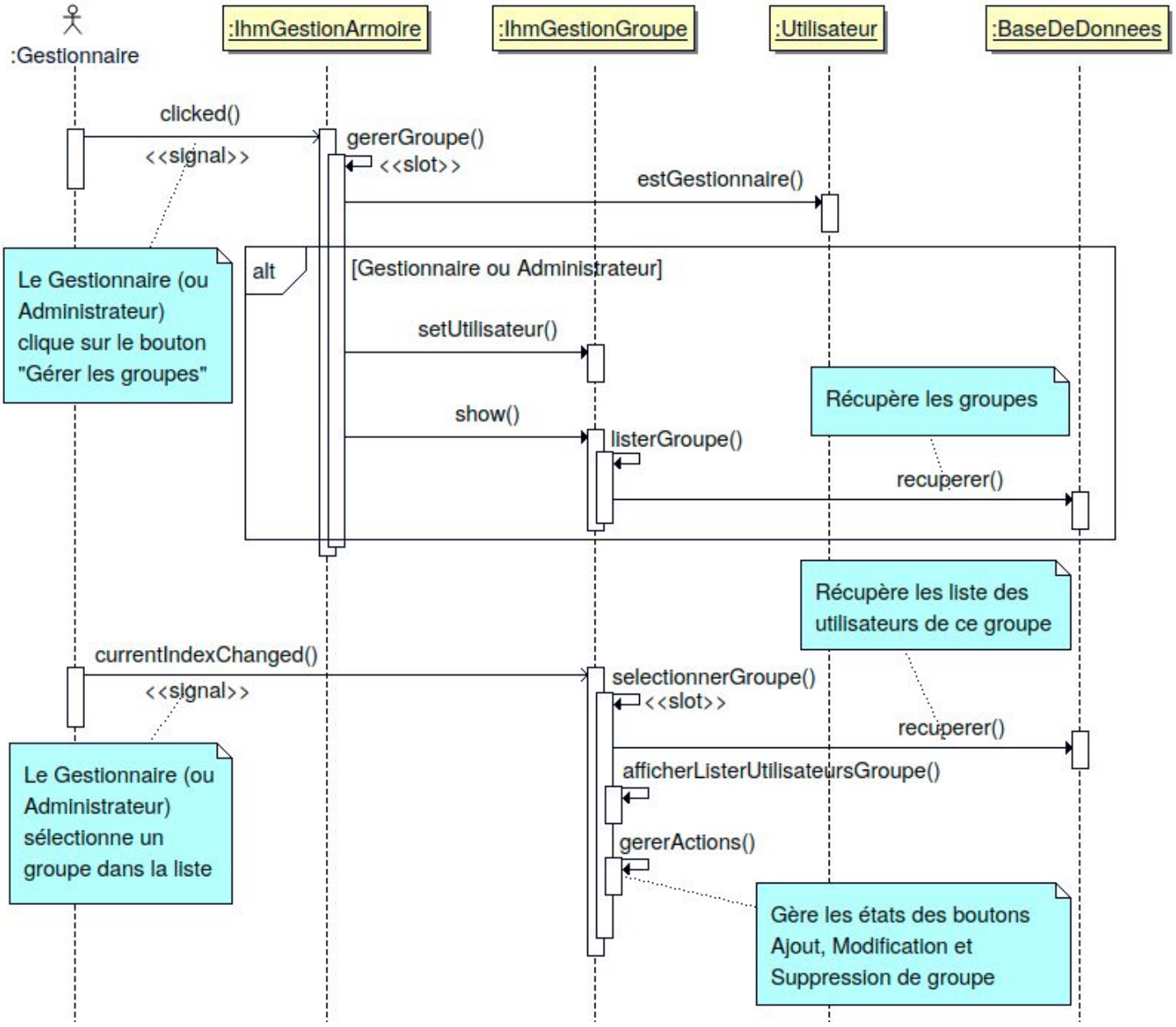
❒ La classe **IHMGestionGroupe** permet d'afficher la liste des groupes et utilisateurs
De modifier , ajouter ou supprimer un groupe ou un utilisateur qui sont eux stocker dans la base de donnée.



❒ La classe **GestionBalance** permet l'ouverture du port / la fermeture, la lecture de la trame envoyée par la carte Arduino.

❒ La classe **Utilisateur** permet d'afficher les différents Utilisateurs avec leurs données comme nom prenom etc...

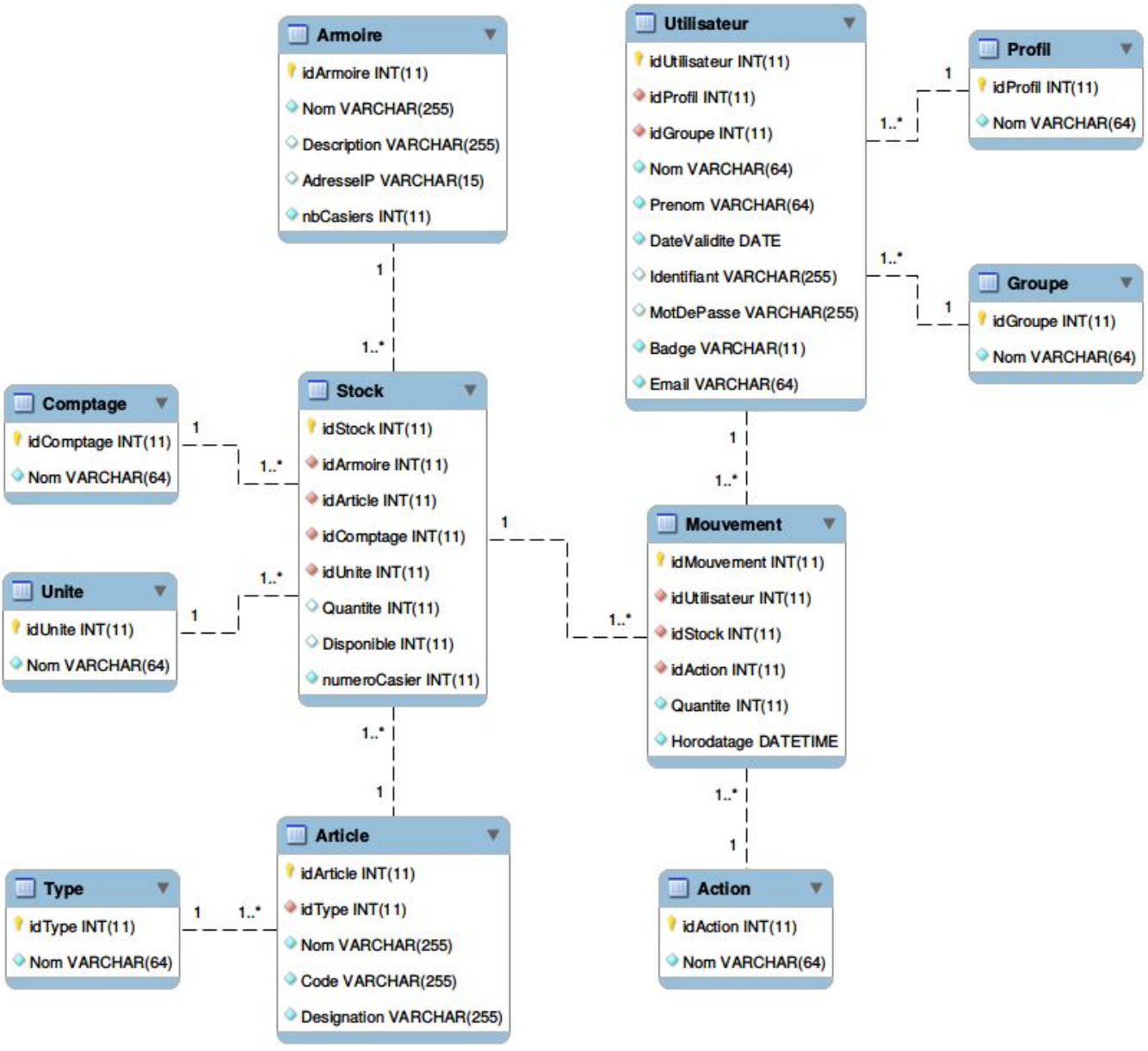
Diagramme de séquence : éditer groupe



Tout d'abord l'administrateur ou le gestionnaire clique sur le bouton "Gérer les groupes" puis on récupère les groupes dans la base de données. L'administrateur ou le gestionnaire peuvent sélectionner un groupe dans la liste récupérée et une fois le groupe sélectionné il récupère les utilisateurs de ce groupe. Ils peuvent aussi gérer les groupes l'ajout, la modification et la suppression d'un groupe ou d'un utilisateur

Base de données

Le schéma de la base de données MySQL d'une armoire est le suivant :



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

- ➔ un champ Nom
- ➔ un champ adresseIP
- ➔ un champ nbCasiers (défini par par défaut à 8)

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

La table **Groupe** contient un champ Nom qui détermine le groupe d'un utilisateur (par exemple: Professeur, 1-BTS-SN ou T-BTS-SN).

La table **Utilisateur** contient :
- ➔ une clé étrangère idProfil qui précise son profil
- ➔ une clé étrangère idGroupe qui indique son groupe d'appartenance
- ➔ un champ Nom
- ➔ un champ Prenom
- ➔ un champ DateValidite
- ➔ un champ Identifiant
- ➔ un champ MotDePasse
- ➔ un champ Badge
- ➔ un champ Email

La table **Groupe** contient :
- ➔ une clé primaire idGroupe qui précise son Groupe
- ➔ un champ Nom

La table **Unite** contient un champ Nom (mètres, pièces, pourcentage, g (grammes), kg (kilogrammes)).

La table **Stock** contient :
- ➔ une clé étrangère idArmoire qui l'associe à une armoire
- ➔ une clé étrangère idArticle qui indique l'article stocké
- ➔ une clé étrangère idComptage qui précise le type de comptage
- ➔ une clé étrangère idUnite qui indique l'unité utilisée pour le compter
- ➔ un champ Quantite qui fournit le stock de départ pour cet article
- ➔ un champ Disponible qui comptabilise la présence de l'article dans le stock actuellement
- ➔ un champ NumeroCasier

La table **Action** contient un champ Nom (Entrée, Sortie).

La table **Mouvement** contient l'ensemble des entrées/sorties des articles dans l'armoire :
- ➔ une clé étrangère idUtilisateur qui indique l'utilisateur qui a effectué le mouvement
- ➔ une clé étrangère idStock qui associe le mouvement dans le stock
- ➔ une clé étrangère idAction qui fournit l'action réalisée
- ➔ un champ Quantité
- ➔ un champ Horodatage

Voici la tables Groupe de la base de données obtenu en faisant la requête sql suivante:

```
CREATE TABLE IF NOT EXISTS `Groupe` (
  `idGroupe` int(11) NOT NULL AUTO_INCREMENT,
  `Nom` varchar(64) NOT NULL,
  PRIMARY KEY (`idGroupe`),
  CONSTRAINT Unique_Groupe UNIQUE (`Nom`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

La table a été récupéré avec l'idGroupe , le nom et le numéro du groupe.
Comme nous pouvons le voir ci-dessous l'idGroupe est présent aussi dans la table utilisateur qui permet d'identifier à quelle groupe appartient l'utilisateur

Pour modifier un groupe nous allons utiliser la méthode modiferGroupe() de la classe IhmGestion Groupe:

```
void IhmGestionGroupe::modifierGroupe()
{
   if(utilisateur != nullptr
)
   {
       qDebug() << Q_FUNC_INFO << "Nom" << utilisateur->getNom() <<
"Administrateur" << utilisateur->estAdministrateur() << "estGestionnaire" <<
utilisateur->estGestionnaire();
        //Vérifie que l'utilisateur connecté est bien un administrateur ou
        un gestionnaire
       if(utilisateur->estAdministrateur() || utilisateur->estGestionnaire())
       {
           QString nomGroupe = ui->saisieNomGroupe->text();
           if(nomGroupe.isEmpty())
               return;
           QString idGroupe = getIdGroupe();
           if(idGroupe.isEmpty())
               return;
      // La requête sql permettant de modifier dans la table
          QString requete = QString("UPDATE `Groupe` SET `Nom`='%1' WHERE
idGroupe='%2'").arg(nomGroupe).arg(idGroupe);
           qDebug() << Q_FUNC_INFO << "requete" << requete;
           bool retour = bdd->executer(requete);

      // Permet d'actualiser la liste des groupes
           if(retour)
           {
               ui->saisieNomGroupe->clear();
               listerGroupe();
           }
       }
   }
}
```

```
UPDATE 'Groupe'  => signifie modifier depuis la table groupe
SET 'Nom'= 'PROFESSEUR' => modifier le champ nom pour y mettre PROFESSEUR
Dans la fonction :
WHERE idGroupe='%1'.arg(idGroupe) => Le %1 permet d'insérer le QString idGroupe
(ici PROFESSEUR)
Dans la requête direct :
WHERE idGroupe='3' => sélectionne l'id groupe n°3 qui est celui de Professeur
```

Nous pouvons remarquer dans la base de donnée que dans la table le champ Nom “ PROFESSEUR” a était modifier et le “S” a était enlevé

idGroupe	Nom	Numero
4	1-BTS-SN	4
1	ADMINISTRATEUR	1
2	GESTIONNAIRE	2
3	PROFESSEUR	3
5	T-BTS-SN	5

Pour ajouter un groupe nous allons appeler la méthode ajouterGroupe().
Cette méthode permet d’ajouter un groupe grâce à la requête sql **INSERT INTO**.

```
INSERT INTO `Groupe` (`Nom`) => insérer dans la table groupe le nom
VALUES('%1')").arg(nomGroupe); => Avec pour valeur l'argument nomGroupe
```

```
void IhmGestionGroupe::ajouterGroupe()
{
   QString nomGroupe = ui->saisieNomGroupe->text();
   if(nomGroupe.isEmpty())
       return;
   //Requete sql permettant l'ajout du groupe
   QString requete = QString("INSERT INTO `Groupe` (`Nom`)
VALUES('%1')").arg(nomGroupe);
   bool retour = bdd->executer(requete);
   if(retour)
   {
       ui->saisieNomGroupe->clear();
       listerGroupe();
   }
}
```

Pour supprimer un groupe dans la base de donnée nous allons utiliser la requête suivante sql suivante:

La méthode permettant de supprimer un groupe :

```
void IhmGestionGroupe::supprimerGroupe()
{
   QString nomGroupe = ui->saisieNomGroupe->text();
   i
(nomGroupe.isEmpty())
       return;
   QString idGroupe = getIdGroupe();
   if(idGroupe.isEmpty())
       return;

   QString requete = QString("DELETE FROM `Groupe` WHERE
idGroupe='%1'").arg(idGroupe);
   bool retour = bdd->executer(requete);
   if(retour)
   {
       ui->saisieNomGroupe->clear();
       listerGroupe();
   }
```

Pour supprimer un groupe dans la base de donnée nous allons utiliser la requête suivante sql suivante:

```
DELETE FROM 'Groupe'  => signifie supprimer depuis la table groupe
WHERE idGroupe='%1'.arg(idGroupe) => Le %1 permet d'insérer le QString idGroupe
```

Gestion des balances:

Protocole

Délimiteur de début	$
Séparateur	;
Champs	String
Ordre	Poids
Délimiteur de fin	#

La liaison série assure la transmission de la carte Arduino vers la Raspberry.
Nous avons le choix pour la communication entre la Raspberry et l'Arduino entre la liaison RX/TX ou par l'adaptateur usb. Nous avons choisi l'adaptateur USB pour des raisons de cablage.

Nous utilisons la norme RS232 car nous avons n'avons pas besoin d'une longue distance , un gros débit n'est pas nécessaire et nous n'avons pas besoin de plusieurs récepteurs. C'est pour cela que nous avons choisi la RS232.

EIA CCITT	RS232C V24 / V28	RS422 V11 / X27	RS485 V11 / X27	Boucle de courant
Type d'interface	unipolaire	Différentiel	Différentiel	0-20 mA
Sensibilité				
Distance	15 m	1200 m	1200 m	1 à 2 km
Débit max.	19200 Bauds	10 MBds	10 MBds	19200 Bauds
Multipoint	non	oui	oui	oui
Nombre d'émetteurs	1	1	32	
Nombre récepteurs	1	10	32	
Impédance d'entrée	3 à 7 kΩ	4 kΩ	12 kΩ	

Paramétrage du port :

```
void GestionBalances::parametrerPort()
{
  port->setBaudRate(QSerialPort::Baud9600);
  port->setDataBits(QSerialPort::Data8);
  port->setParity(QSerialPort::NoParity);
  port->setStopBits(QSerialPort::OneStop);
  port->setFlowControl(QSerialPort::NoFlowControl);
}
```

Paramétrage du port qui permet de communiquer avec l'arduino et de recevoir la trame contenant le poids de la balance

Ouverture du port :

```
void GestionBalances::ouvrirPort()
{
   port->open(QIODevice::ReadOnly);
   qDebug() << Q_FUNC_INFO << "etat ouverture port" << port->isOpen();
   if(port->isOpen())
   {
       connect(port, SIGNAL(readyRead()), this, SLOT(lirePort()));
   }
}
```

La méthode ouvrirPort() permet d'ouvrir le port et d'afficher l'état de celui ci à travers un QDebug si le port est ouvert un signal est envoyé comme quoi il est prêt à être lu.

```
void GestionBalances::lirePort()
{
   if(port->canReadLine())
   {
       QByteArray donnees = port->readLine();
       qDebug() << Q_FUNC_INFO << donnees;
       if(donnees.startsWith("$"))
       {
           if(donnees.endsWith('#'))
           {
               qDebug() << Q_FUNC_INFO << donnees;
               donnees.clear();
           }
       }
   }
}
```

La méthode lirePort() est utilisé pour lire la trame reçue avec :

if(donnees.startsWith("$")) => le délimiteur de début de trame

if(donnees.endsWith('#'))=> le délimiteur de fin de trame

Cette méthode permet d'obtenir la trame suivante que nous pouvons lire grâce au QDebug dans la fonction précédente :

```
void GestionBalances::ouvrirPort() etat ouverture port true
void GestionBalances::lirePort() "Reading: 0.70 g calibration_factor:
256610.00\r\n"
```

Nous pouvons remarquer que le port est ouvert.

"Reading: 0.70 g calibration_factor: 256610.00\r\n"

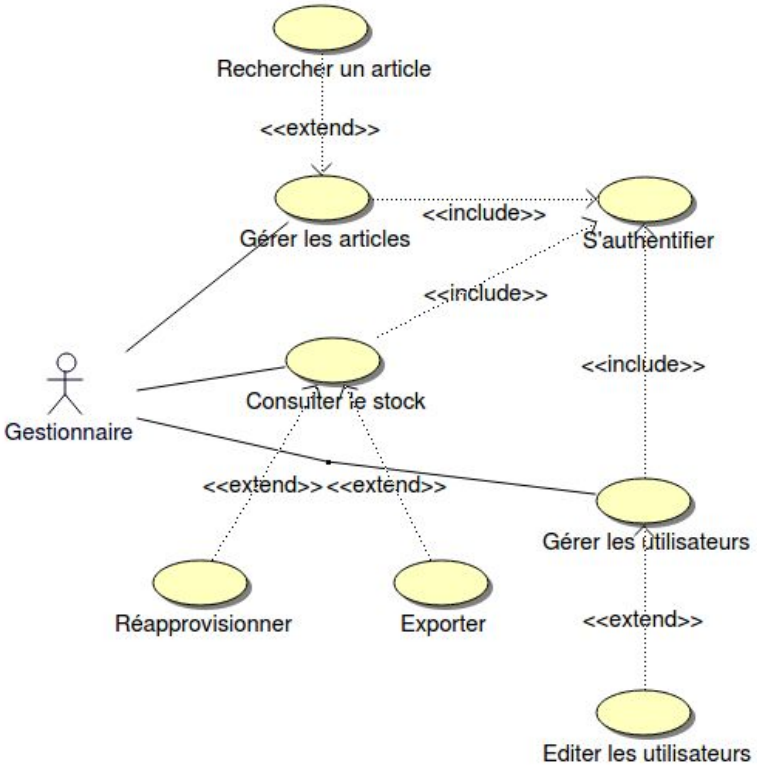
"Reading: 0.70 g" => est le poids qui est sur balance et envoyer par l'arduino
"calibration_factor" =>correspond à un coefficient multiplicateur qui pour nous est de 256610.00 pour la précision de la pesée

Test	Valide (Oui/Non)
Ajouter un groupe	**Oui**
Supprimer un groupe	**Oui**
Modifier un groupe	**Oui**
Ouverture du port	**Oui**
Reception de la trame	**Oui**

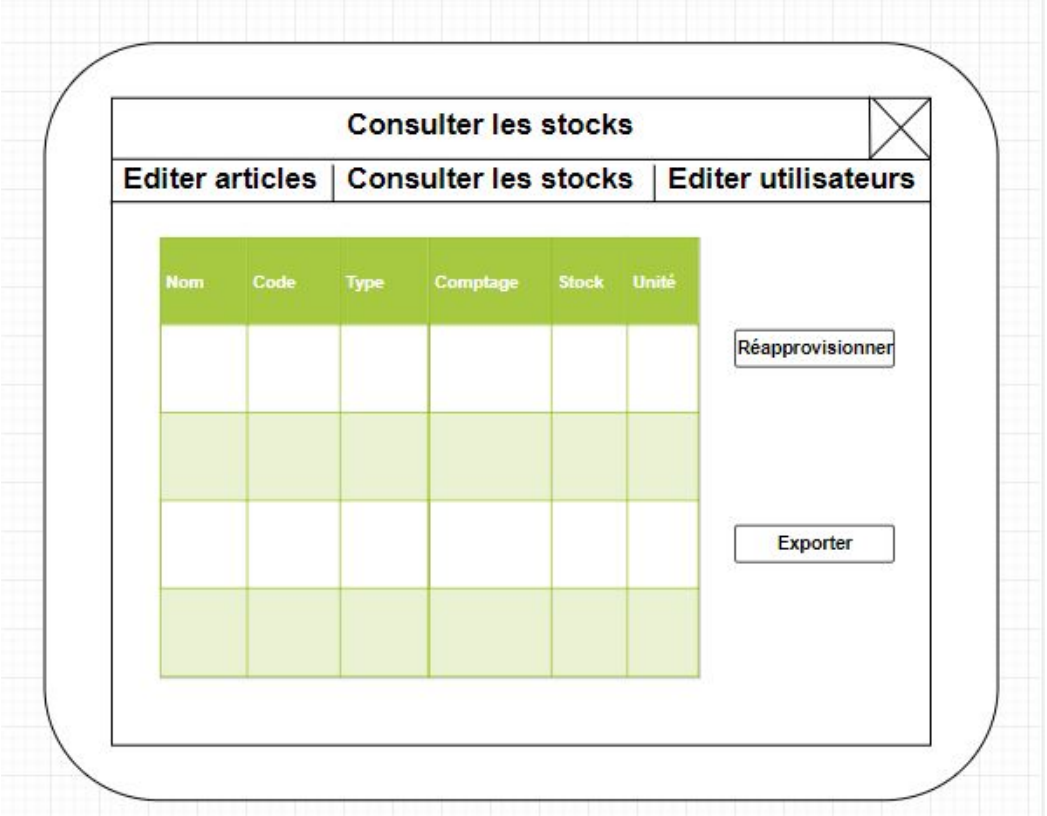
Partie personnelle : Waginaire Nathan

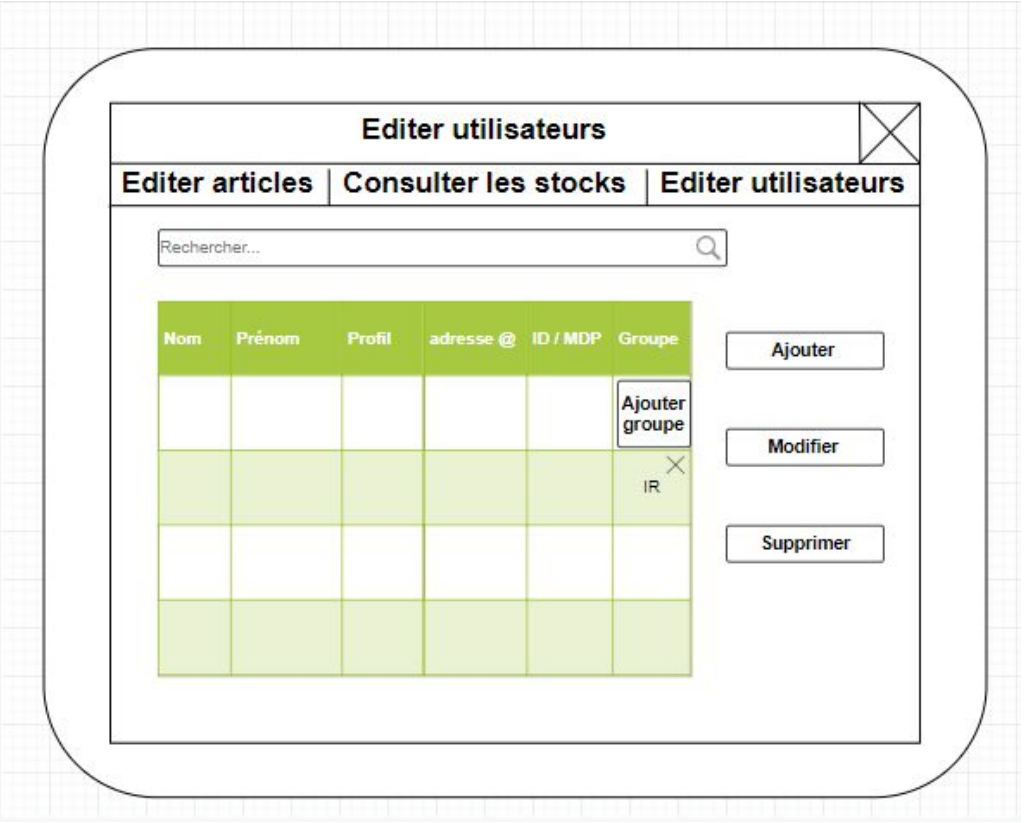
- ❏ Gérer les articles et les utilisateurs, Gérer le stock en assurant le comptage automatique et Gérer le lecteur code-barres

Diagramme des cas d'utilisation

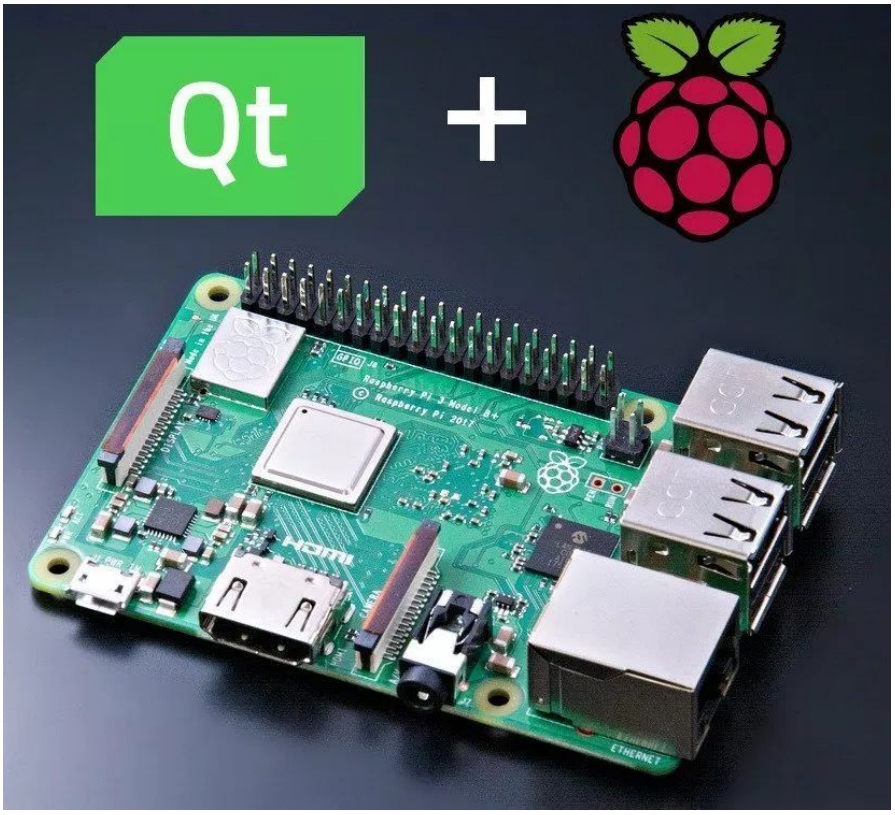


Maquettes IHM





Raspberry Pi



Annexes

Manuel d'installation Qt/Android pour terminal mobile

L'environnement de développement Qt5 (version Qt 5.10.1)

```
$ wget
https://download.qt.io/archive/qt/5.10/5.10.1/qt-opensource-linux-x64-5.10.1.run
$ chmod +x qt-opensource-linux-x64-5.10.1.run
$ ./qt-opensource-linux-x64-5.10.1.run
```

Le kit de développement Java SDK

```
$ sudo apt install openjdk-8-jdk
$ javac -version
javac 1.8.0_191
```

L'Android SDK

Pour les systèmes 64-bits, il faudra tout d'abord installer les bibliothèques 32-bits suivantes :

```
$ sudo apt-get install libstdc++6:i386 libgcc1:i386 zlib1g:i386 libncurses5:i386
```

Pour l'émulateur, il faut :

```
$ sudo apt-get install libsdl1.2debian:i386
```

Ensuite il faut télécharger Android Studio puis l'installer avec ces commandes :

```
$sudo mv android-studio-ide-171.4443003-linux.zip /usr/local/
$ cd /usr/local/
$ sudo unzip android-studio-ide-171.4443003-linux.zip
$ sudo rm android-studio-ide-171.4443003-linux.zip
```

L'Android NDK (Android Native Development Kit)

```
$ cd HOME/Android/Sdk
$ wget https://dl.google.com/android/repository/android-ndk-r17c-linux-x86_64.zip
$ unzip android-ndk-r17c-linux-x86_64.zip
```

Configuration dans Qt Creator :

- ❏ Dans Option > Appareils Mobiles > Android, indiquer l'emplacement du :
  - ❏ JDK (Java SDK)
  - ❏ SDK Android
  - ❏ NDK Android
- ❏ Dans Option > Compiler & Exécuter > Compilateur, Debugger, Qt versions et Kits vérifier que les GCC, Debuggers, armv7 et x86 soient bien présents

Manuel d'installation Qt pour Raspberry

L'environnement de développement Qt5 (version Qt 5.11.2) :

```
$ wget
http://download.qt.io/official_releases/qt/5.11/5.11.2/single/qt-everywhere-src-5.
11.2.tar.xz
$ tar xf qt-everywhere-src-5.11.2.tar.xz
```

Pour la cross compilation :

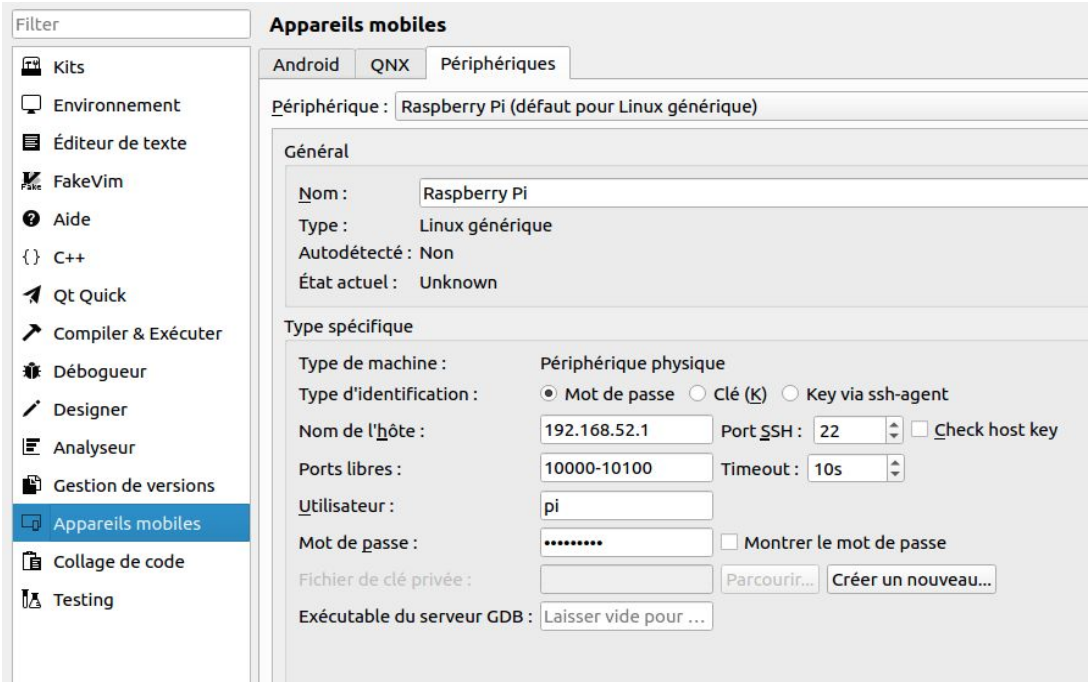
```
$ git clone https://github.com/raspberrypi/tools
```

Configuration dans Qt Creator :

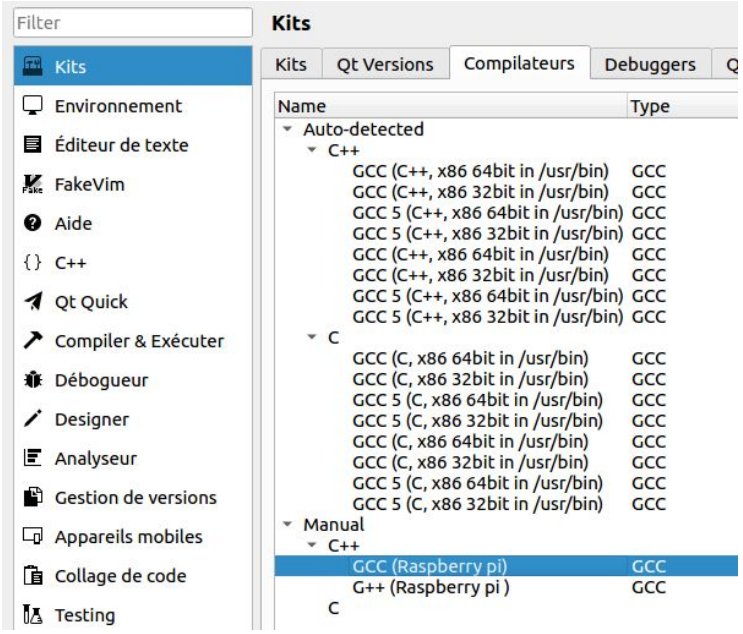
Qt Creator permet de créer, déployer, exécuter et déboguer des applications Qt directement sur la Raspberry Pi en un seul clic.

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

- Ajouter la Raspberry Pi avec son adresse Ip (192.168.52.1)



- Ajouter la compilation pour la Raspberry Pi



- Ajout du compilateur :

```
/usr/local/rpi3/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-gcc
/usr/local/rpi3/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-g++
```

- Ajout du debuggers

```
/usr/local/rpi3/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-gdb
```

- Ajout du qmake

```
/opt/Qt5.11.2/5.11.2/gcc_64/bin/qmake
```

- Ajout du kit (création du kit avec toute la configuration effectué auparavant)