

P2017 : Contrôleur de ronde

Lemée Gabriel

Dossier technique du projet - partie individuelle

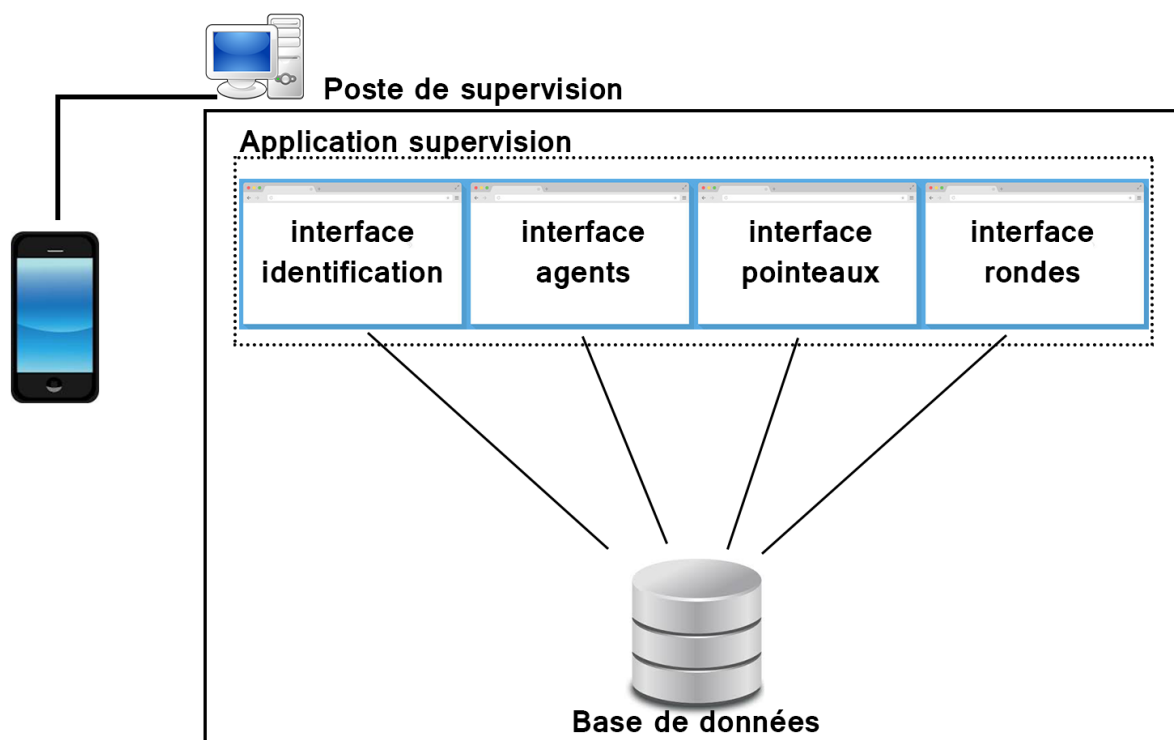
1 -SITUATION DANS LE PROJET.....	2
1.1 -SYNOPTIQUE DE LA RÉALISATION	2
1.2 -DESCRIPTION DE LA PARTIE PERSONNELLE.....	3
2 -PRÉ-REQUIS POUR LE DÉVELOPPEMENT DE L'APPLICATION.....	5
2.1 -INSTALLATION DE LA BASE DE DONNÉES MySQL ET DE MySQL WORKBENCH.....	5
2.2 -INSTALLATION DES PILOTES MySQL POUR QT 5.7.....	5
2.3 -CRÉATION DU SCRIPT D'INITIALISATION DE LA BASE DE DONNÉES.....	6
3 -STRUCTURE DE L'APPLICATION.....	8
4 -MODÉLISATION DES CLASSES TYPES.....	8
5 -DÉVELOPPEMENT DES ACCÈS À LA BASE DE DONNÉES.....	10
6 -DÉVELOPPEMENT DE L'IDENTIFICATION D'UN RESPONSABLE DE SÉCURITÉ.....	11
6.1 -INTERFACE.....	11
6.2 -DIAGRAMME DE CLASSE.....	12
7 -DÉVELOPPEMENT DE L'ADMINISTRATION D'UN AGENT.....	13
7.1 -INTERFACE.....	13
7.2 -DIAGRAMME DE CLASSE.....	14
8 -DÉVELOPPEMENT DE LA GESTION D'UN POINTEAUX.....	15
8.1 -INTERFACE.....	15
8.2 -DIAGRAMME DE CLASSE.....	16
9 -DÉVELOPPEMENT DE LA GESTION D'UNE RONDE.....	17
9.1 -INTERFACE.....	17
9.2 -DIAGRAMME DE CLASSE.....	18
9.3 -FICHE DE TEST.....	19
10 -BILAN DE LA RÉALISATION PERSONNELLE.....	21
11 -ANNEXES.....	22
11.1 -SCRIPT DE CRÉATION DE LA BASE DE DONNÉES MySQL.....	22
11.2 -DIAGRAMME DE CLASSE.....	25

1 - Situation dans le projet

1.1 - Synoptique de la réalisation

Dans le développement du système, mon but est d'intervenir lors de la création des agents, des pointeaux et des rondes afin qu'ils puissent être envoyés au smartphone. Ces informations sont stockées dans la base de données MySQL du poste de supervision.

Ma première mission a été de restreindre l'accès à l'application de supervision par le biais de l'identification du responsable de sécurité. J'ai été chargé par la suite de faire des interfaces intuitives permettant la gestion d'agents, de pointeaux et de rondes. Ces informations sont enregistrées dans la base de données et seront transférées au smartphone par l'intermédiaire de la liaison USB.

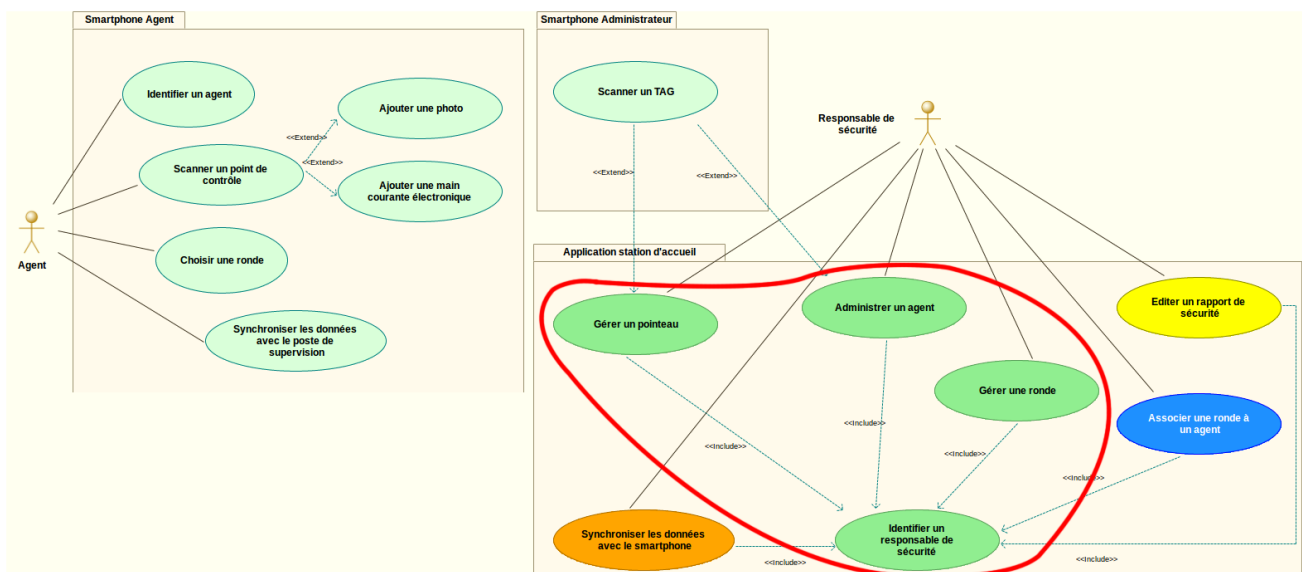


1.2 - Description de la partie personnelle

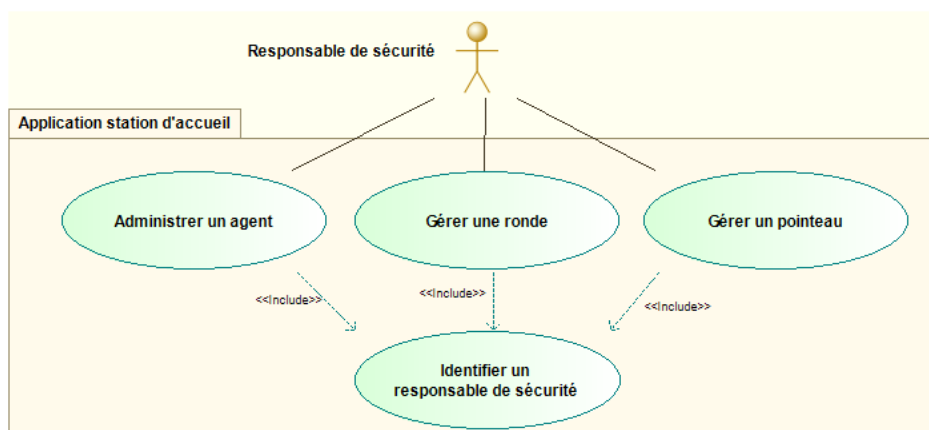
Lors de ce projet, ma partie du développement s'est concentré sur l'application du poste de supervision, les fonctionnalités que j'ai réalisé sont les suivantes :

- Identifier un responsable de sécurité
- Administrer un agent
- Gérer un pointeaux
- Gérer une ronde

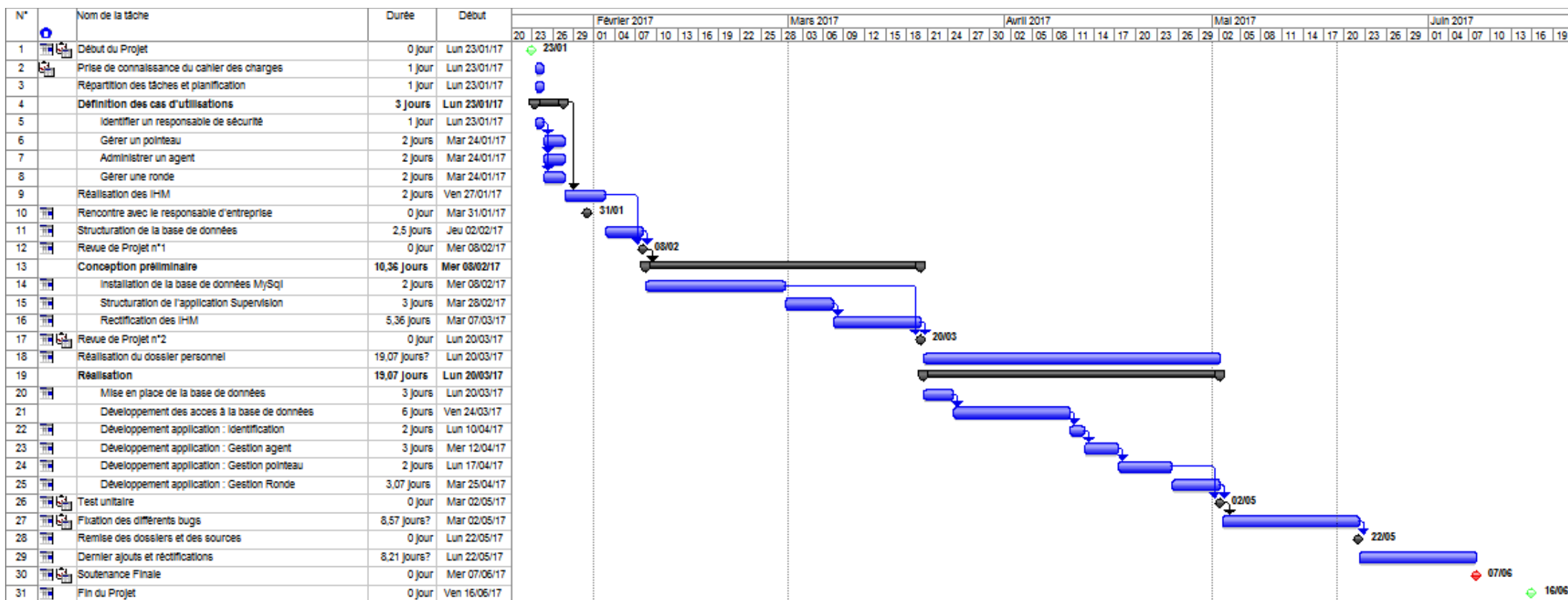
Dans le but de modéliser la répartition des tâches au sein du groupe nous avons réalisés un diagramme des cas d'utilisation.



Sur ce dernier son entourés en rouge mes cas d'utilisation , voici un diagramme restreint les représentant.



Dans l'optique de ne pas s'égarer lors de la réalisation du projet, j'ai réalisé une planification des différentes missions qu'il m'a été confié sous forme d'un diagramme de Gant.



2 - Pré-requis pour le développement de l'application

2.1 - Installation de la base de données MySQL et de MySQL Workbench

Pour mettre en place un serveur de base de données local, nous avons décidé d'utiliser MySQL dans sa version 5.7 ainsi que son utilitaire MySQL Workbench, ce dernier n'étant pas nécessaire mais facilitant grandement la manipulation de la base de données durant la phase de développement. Il peut aussi être utilisé après l'installation du système comme outil de maintenance.

Voici la procédure suivie pour l'installation de ces logiciels :

Après avoir fait l'acquisition de la dernière version de l'installateur depuis le site web de MySQL (<https://www.mysql.com/fr/>), lancer l'installation en s'assurant d'avoir les droits d'administrateur sur la machine.

Lors de l'installation les composants à installer sont le serveur MySQL 5.7 et l'utilitaire MySQL Workbench.

À la fin de l'installation de MySQL serveur, la configuration est demandée. Le mot de passe pour l'utilisateur root (administrateur) est alors demandé. Il est aussi possible à ce moment de créer d'autres utilisateurs. Ces comptes utilisateurs permettent la connexion à la base de données.

2.2 - Installation des pilotes MySQL pour Qt 5.7

Ayant comme contrainte de développer l'application sur une machine sous système d'exploitation Windows, Qt a donc besoin d'être complété avec un driver permettant à une application développée sous Qt Creator d'effectuer des accès vers une base de données MySQL.

Après l'installation du serveur MySQL, il faut se rendre dans le dossier d'installation de ce dernier (par défaut « C:\Program Files\MySQL »), puis aller dans le dossier « MySQL Server [numéro de version] ». Dans ce dossier est présent le dossier des bibliothèques nommé « lib ». Pour finir dans ce dossier vous trouverez un fichier dll nommé « libmysql.dll ». Ce fichier est le fichier dont a besoin Qt pour communiquer avec une base de données MySQL, il faut donc copier ce fichier dans Qt. Le dossier dans lequel il doit être copié se trouve dans le dossier « [dossier d'installation de Qt]\[numéro de version]\mingw53_32\bin ». Dans notre cas ce dossier était « C:\Program Files\Qt\5.7\mingw53_32\bin ».

Après cette manipulation les projets créés sous Qt Creator ont désormais accès à la base de données MySQL.

Voici le détail de création d'une table, le script complet est disponible en [annexe](#).

```
CREATE TABLE `agents` (  
  `idAgent` int(11) NOT NULL AUTO_INCREMENT,  
  `idTag` char(8) DEFAULT NULL,  
  `nom` varchar(45) NOT NULL,  
  `prenom` varchar(45) NOT NULL,  
  `agentActif` tinyint(4) NOT NULL DEFAULT '1',  
  PRIMARY KEY (`idAgent`),  
  KEY `fk_Agents_Tags1_idx` (`idTag`),  
  CONSTRAINT `fk_Agents_Tags1` FOREIGN KEY (`idTag`) REFERENCES `tags`  
  (`idTag`) ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Description :

```
CREATE TABLE `agents`
```

« **CREATE TABLE** » Le début de la création d'une table nommé agents.

```
`idAgent` int(11) NOT NULL AUTO_INCREMENT,
```

« **idAgent** » Ajout d'un champ avec pour nom idAgent.

« **int** » La valeur sera un entier (interger). 11 défini le nombre de bits réservé au stockage de cette valeur.

« **NOT NULL** » Le champ est obligatoire dans chaque ligne de la table.

« **AUTO_INCREMENT** » Si le champ n'est pas spécifié il sera automatiquement incrémenté par rapport à la ligne précédemment insérée dans la table.

```
`nom` varchar(45) NOT NULL,
```

« **varchar(45)** » Le champ nom sera une chaîne de caractères pouvant aller jusqu'à 45 caractères.

```
`agentActif` tinyint(4) NOT NULL DEFAULT '1',
```

« **tinyint(4)** » Le champ agentActif sera un booléen, une variable binaire pouvant être soit 0 soit 1.

« **DEFAULT '1'** » Si le champ n'est pas spécifié il prendra pour valeur 1.

```
PRIMARY KEY (`idAgent`),
```

« **PRIMARY KEY** » La clé primaire de la table sera le champ idAgent, ce champ doit être unique il sert d'identification entre deux lignes dans la table.

```
KEY `fk_Agents_Tags1_idx` (`idTag`),
```

Création d'une clé étrangère sur le champ idTag. Une clé étrangère est une liaison entre deux champs de deux tables.

```
CONSTRAINT `fk_Agents_Tags1` FOREIGN KEY (`idTag`) REFERENCES `tags` (`idTag`)  
ON DELETE NO ACTION ON UPDATE NO ACTION
```

Liaison entre la clé étrangère du champ idTag de la table agent et le champ idTag de la table tags. Toutes modifications ou suppression sera annulée si une liaison est présente entre ces deux champs.

```
ENGINE=InnoDB DEFAULT CHARSET=utf8
```

La table utilisera le système de gestion InnoDB et son contenu sera encodé en utf8 utilisant 8 bits pour encoder un caractère.

3 - Structure de l'application

Possédant la partie d'identification du responsable de sécurité j'ai été chargé de trouver une structure pour l'application permettant une implémentation ultérieure des parties de mes associés.

Après avoir décidé de présenter chaque interface sous forme d'un onglet à l'exception de l'identification, toutes les interfaces sont donc des classes héritant de QWidget. Cette structure rend possible l'ajout de fonctionnalités futures sans changer tout le reste de l'application. Elle permet aussi de développer individuellement chaque interface peut être créée à partir de l'outil de design de Qt puis les classes seront importées dans le projet final.

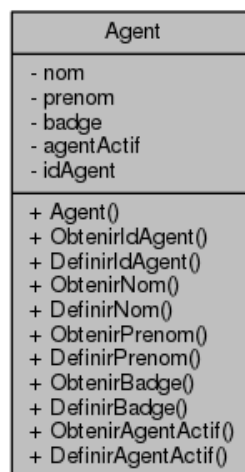
Lors de la création d'une interface avec l'outil de design Qt, le logiciel génère automatiquement une classe supplémentaire contenant tous les éléments de l'interface.

Le diagramme de classe est disponible en [annexe](#).

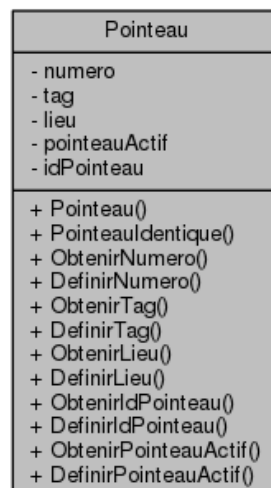
4 - Modélisation des classes types

Afin de rendre le développement plus clair et plus simple j'ai décidé de modéliser les 3 principaux objets du projet en une classe respective. Une classe modélisant un agent de sécurité, une autre pour les pointeurs et une classe représentant une ronde de surveillance. Chaque classe possède ses fonctions accesseur et mutateur afin d'obtenir et de modifier ses attributs privés. La classe ronde possède aussi une autre classe, nommée « TempsPointage », elle contient le temps minimum et maximum entre le pointeur actuel et le pointeur suivant.

Classe « Agent » :

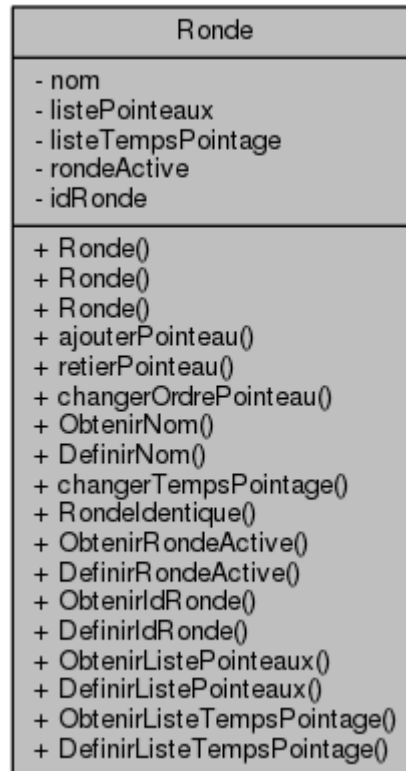


Classe « Pointeur » :

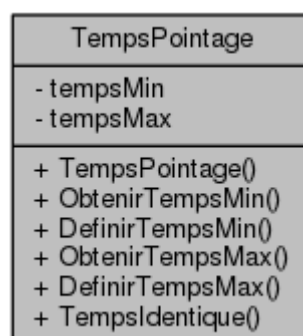


Classe « Ronde » :

La classe Ronde utilise la classe de Qt QList afin de modéliser sa liste de pointeau. Les pointeaux de cette liste sont dans l'ordre du déroulement.

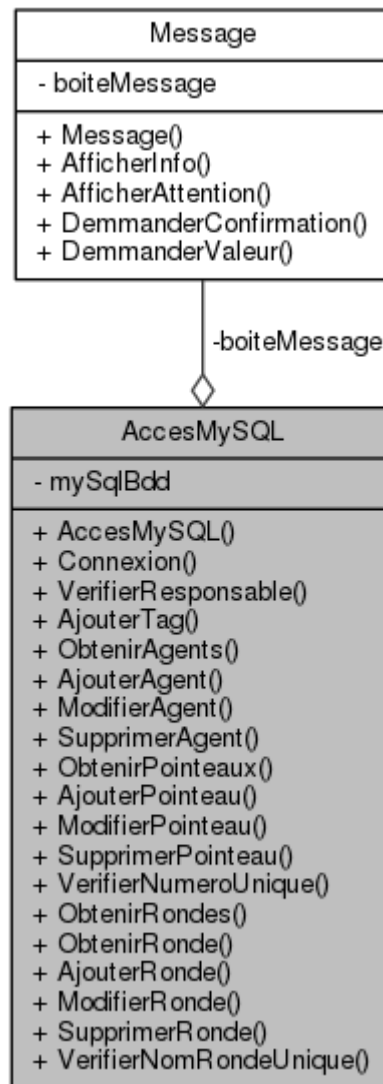


Classe « TempsPointage »



5 - Développement des accès à la base de données

La première étape de la programmation fonctionnelle a été de faire la classe effectuant tous les accès à la base de données. Cette classe est la Classe AccesMySQL.



5.1 - Gestion des agent dans la base de données

La première étapes était d'obtenir tous les agent présent dans la base de données. La méthode « ObtenirAgents » retourne un QListe d'objet « Agent » contenant tous les agent présent dans la base de données. Les agent sont récupéré grâce à la requête SQL suivante :

« ORDER BY » permet d'ordonner les agent par ordre des nom et si il y as des nom identique par ordre des prénom.

La méthode « AjouterAgent » doit uniquement vérifier si le tag de l'agent à ajouter n'est pas déjà utilisé dans la base de données. Un agent est ajouté grâce à la requête SQL suivante :



La requête doit posséder des arguments dont la valeur est contenue dans une variable de l'application, il est donc nécessaire de passer par une affectation des variables de la requête grâce au « bindValue ».

Pour modifier un agent grâce à la méthode « ModifierAgent », il est important de vérifier si le tag de l'agent a été modifié et si c'est le cas, vérifier qu'il n'est pas déjà utilisé dans la base de données par un autre agent ou un pointeau. Les informations de l'agent sont modifiées grâce à la requête suivante :



Lors de la suppression d'un agent avec la méthode « SupprimerAgent », on doit d'abord vérifier si l'agent est utilisé dans la table des historiques. Si c'est le cas, l'agent ne peut pas être supprimé car les informations peuvent être demandées lors de l'édition d'un rapport dans la partie de l'application de l'étudiant Sénechal Florian. Dans le cas où l'agent ne peut pas être retiré de la base de données, le champ « agentActif » prend la valeur false. Quand un agent est retiré, toutes les associations avec les rondes dans la table « associationagentsrondes » sont supprimées également.

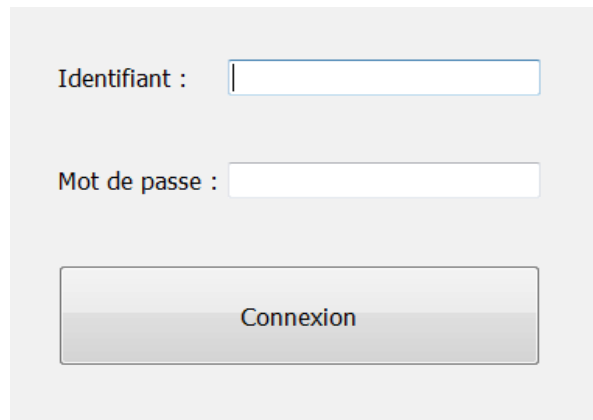
Si l'agent ne possède pas d'historique, il peut être supprimé grâce à la requête suivante :



6 - Développement de l'identification d'un responsable de sécurité

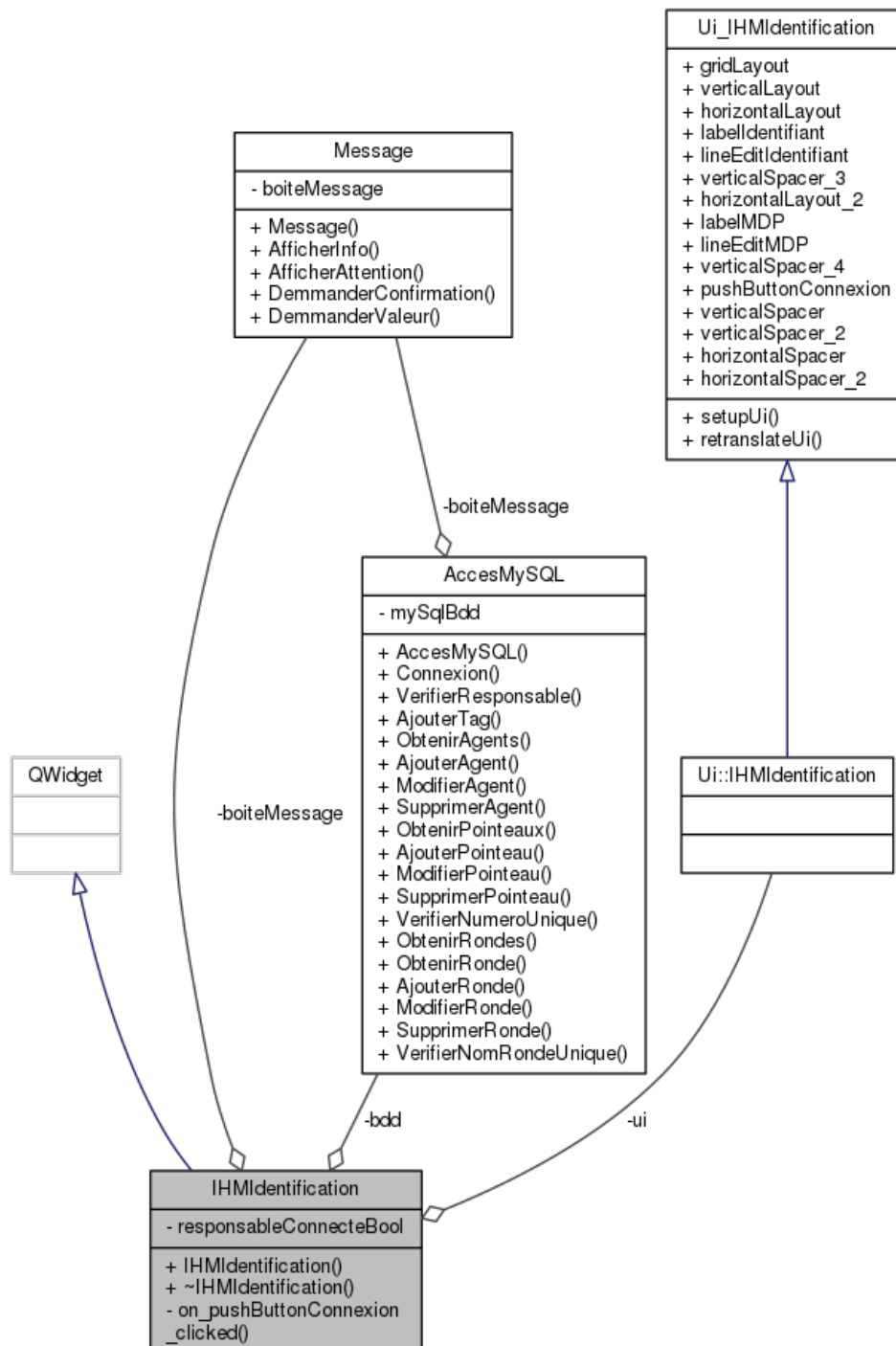
L'interface d'identification du responsable est très simple, un champ identifiant et un champ mot de passe permettant au responsable de s'identifier. Lors du clic sur le bouton connexion, on vérifie que les deux champs sont remplis et on vérifie la véracité de ces informations grâce à la méthode `VerifierResponsable` de la classe `AccesMySQL`.

6.1 - Interface



The image shows a login form with a light gray background. It contains two input fields: the first is labeled 'Identifiant :' and the second is labeled 'Mot de passe :'. Below these fields is a button with a gradient background and the text 'Connexion'.

6.2 - Diagramme de classe



7 - Développement de l'administration d'un agent

La partie de gestion des agents possède 3 modes principaux représentés par une variable nommée « selectionAgent ». C'est un entier qui prend pour valeur 0 quand aucun agent est sélectionné, 1 quand un agent déjà enregistré dans la base de données est sélectionné et 2 quand on est en mode d'édition d'un nouvel agent. Cette variable est utilisée pour savoir quelle action effectuer lors d'une interaction de l'utilisateur avec l'interface. La contrainte principale de cette interface était de rendre accessible toutes les opérations sur un agent dans une seule et même interface.

Lorsque qu'aucun agent n'est sélectionné, les champs de l'interface se désactivent. Lors de la sélection d'un agent dans la liste des agents, on vérifie si un agent est en cours d'édition et si il a été modifié sans être enregistré. Si on change d'agent alors que l'agent actuel n'est pas enregistré alors une fenêtre apparaît nous proposant de continuer sans enregistrer ou de revenir en arrière nous laissant la possibilité de sauvegarder les modifications dans la base de données.

Quand un agent est sélectionné ses informations sont alors affichées dans les champs respectifs. Lors de la modification d'un champ, les informations sont vérifiées pour voir si elles ont été modifiées comparées avec celles de la base de données. Si les informations diffèrent, le bouton « enregistrer » est alors actif. Le bouton « supprimer » est actif dès lors qu'un agent existant est sélectionné dans la liste.

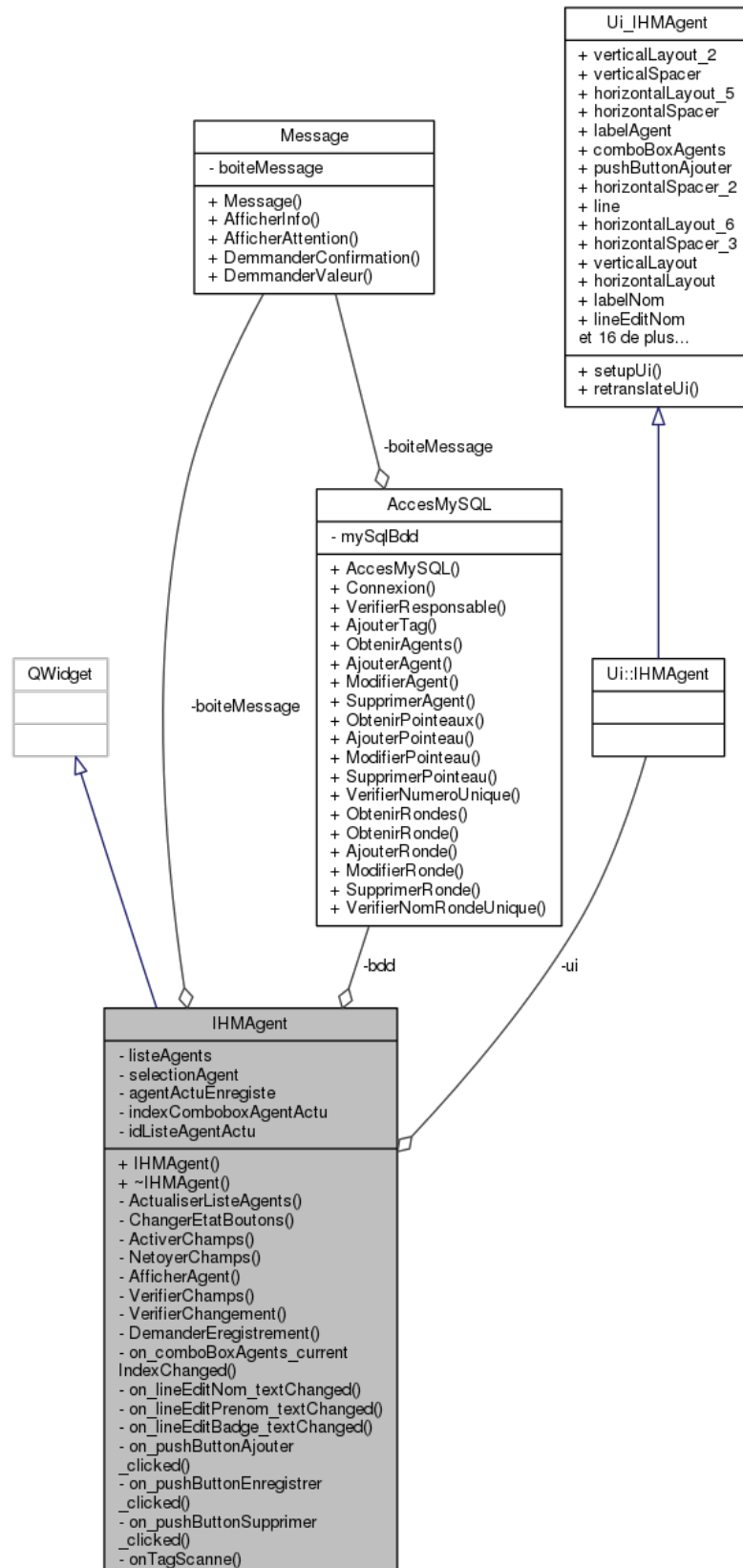
Lors du clic sur le bouton « Nouveau » une vérification de l'enregistrement actuel est effectuée comme lors de la sélection dans la liste d'agent. Si l'action est confirmée alors les champs sont vidés et la variable « selectionAgent » prend la valeur 2 pour signifier un ajout d'agent.

Durant la phase de développement le champ badge est actif et est rempli manuellement, mais l'objectif est d'intégrer la partie scanner de l'étudiant Herbron Tanguy. Le scanner remplira le champ badge lors du scan d'un tag. Cette fonctionnalité permet d'éviter les erreurs de saisie et simplifie la création d'un agent.

7.1 - Interface

The screenshot shows the 'SupervisionRondier' application window. At the top, there are three tabs: 'Gestion Agents' (selected), 'Gestion Pointeaux', and 'Gestion Rondes'. Below the tabs, there is a form for managing agents. It starts with a label 'Agent :' followed by a dropdown menu and a green '+ Nouveau' button. Below this, there are three input fields labeled 'Nom :', 'Prenom :', and 'Badge :'. At the bottom of the form, there are two buttons: 'Enregistrer' (with a floppy disk icon) and 'Supprimer' (with an 'X' icon).

7.2 - Diagramme de classe



8 - Développement de la gestion d'un pointeaux

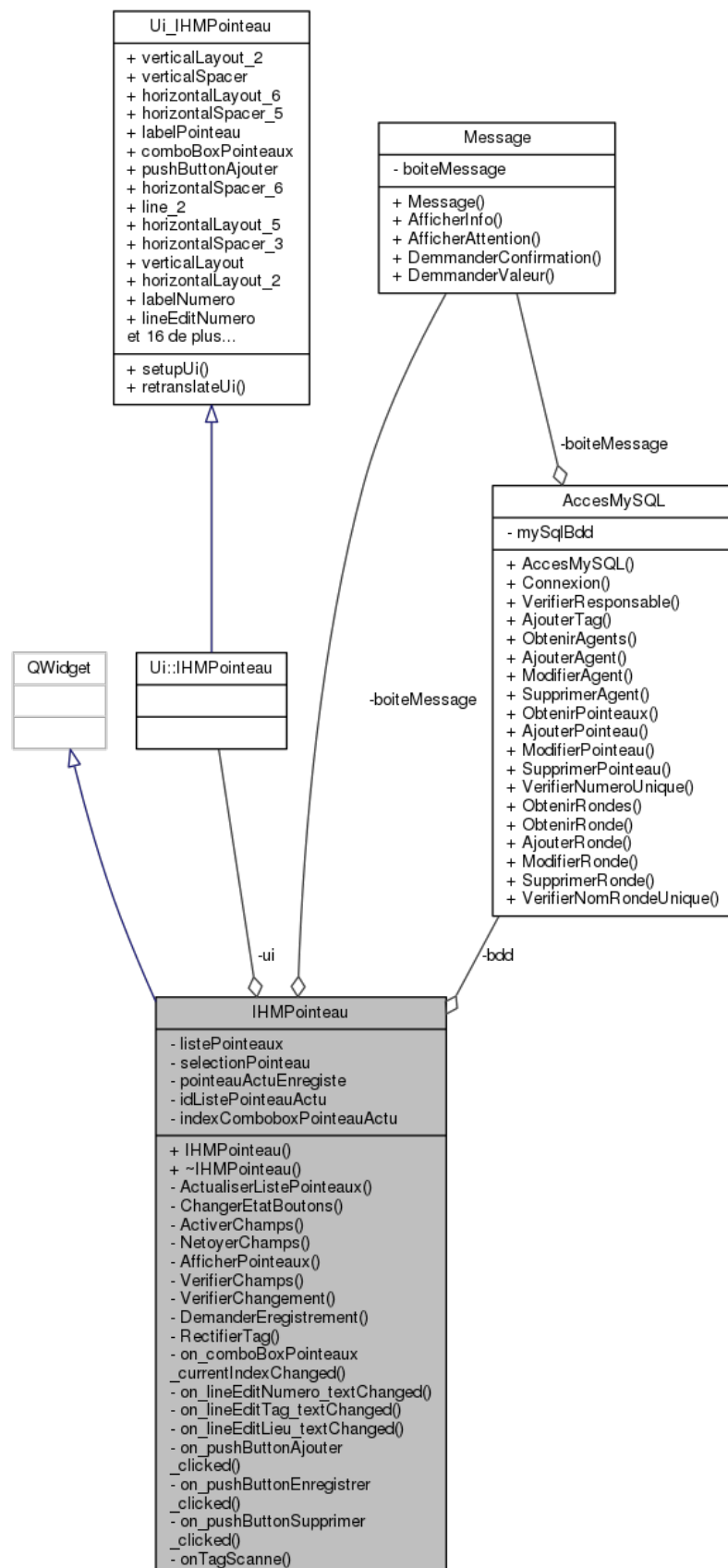
Une fois la partie des agent terminé, la partie des pointeaux a été plus simple et plus rapide car elle reprend les même fonctionnalité que l'interface de la gestion des agent, seulement elle ne fait pas appelle au même méthode de classe AccesMySQL.

Lors de la phase de développement le champs tag est actif et peu être remplis manuellement, seulement comme pour l'interface de gestion d'un agent, la saisie du tag seras remplacer par le scanne d'un tag sur le scanner.

8.1 - Interface

The screenshot displays the 'SupervisionRondier' application window. It features a top navigation bar with three tabs: 'Gestion Agents' (selected), 'Gestion Pointeaux', and 'Gestion Rondes'. The 'Gestion Pointeaux' tab is active, showing a form for managing points. The form includes a 'Pointeau :' dropdown menu with a '+ Nouveau' button next to it. Below this, there are three text input fields labeled 'Numero :', 'Tag :', and 'Lieu :'. At the bottom of the form, there are two buttons: 'Enregistrer' (with a save icon) and 'Supprimer' (with a delete icon).

8.2 - Diagramme de classe



9 - Développement de la gestion d'une ronde

Le développement de la partie ronde a été la plus longue et la plus compliquée. Le mode de sélection suis les interface agent et pointeau mais l'affichage est plus complexe et fournie.

L'onglet ronde a besoin d'être recharger a chaque changement d'onglet car les modification d'un pointeau peuvent impacter directement les rondes. La liste des pointeau est représenter avec un tableau contenant le numéro et le lieu. Seul les pointeau actif dans la base de données possédant un lieu son affichés.

La liste des ronde active dans la base de données est affiché dans une liste déroulante affichant le nom des différentes rondes.

9.1 - Interface

SupervisionRondier

Gestion Agents Gestion Pointeaux Gestion Rondes

Ronde : Atelier Nouveau

Nom de la ronde : Atelier

Liste des pointeaux :

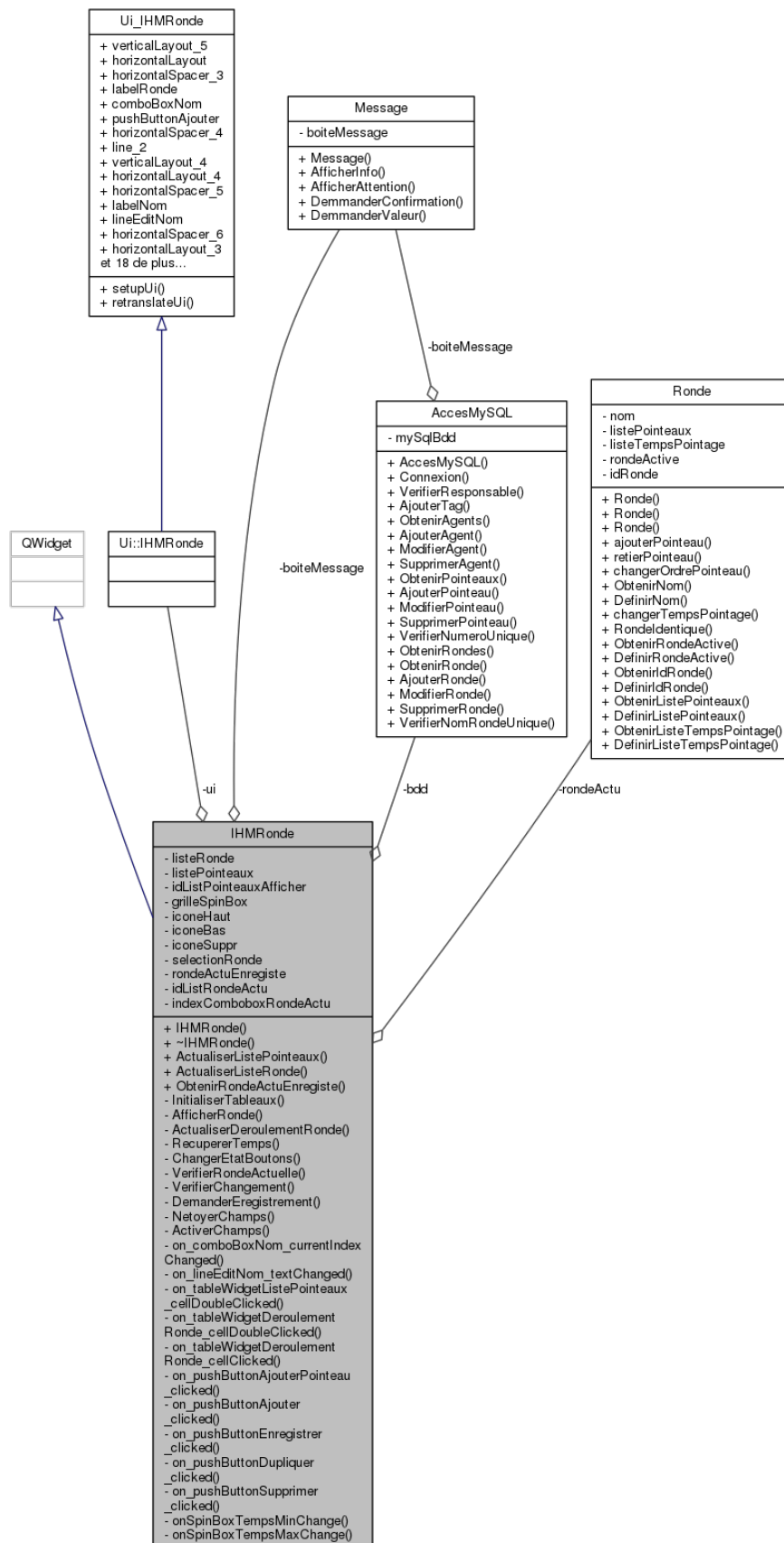
N°	Lieu
1	Salle de réunion
2	Hall principal
3	Escalier
78	Salle B108

Ajouter le pointeau : ➔

Déroulement de la ronde :

Position	N°	Lieu	Temps min	Temps max		
1	1	Salle de réunion	8	17	↓	×
2	1	Salle de réunion	5	6	↑	↓
3	1	Salle de réunion	13	15	↑	↓
4	1	Salle de réunion	3	10	↑	↓
5	2	Hall principal	5	21	↑	↓
6	1	Salle de réunion			↑	×

Dupliquer Enregistrer Supprimer



9.2 - Diagramme de classe

9.3 - Fiche de test

Projet : Contrôleur de ronde		Fiche de tests	
Nature :	Fonctionnel	Référence :	F 03
Module :	Classe IHMRonde		
Objectif :	Vérifier que la gestion de ronde est fonctionnel.		
Condition du test			
État initial du module		Environnement du test	
Programme	L'application supervision est lancée.	PC sous Windows avec base de données et Qt	
Conditions initiales			
La base de données possède les informations nécessaires, Le responsable est identifié (identifiant : admin, mot de passe : admin) et l'onglet actuel est « Gestion Ronde ».			
Procédure de test			
Créer, modifier et supprimer une ronde.			
Repère	Opérations	Résultats attendus	
1	Clic sur le bouton Ajouter.	Une erreur est affichée car le nom ne respecte pas les critères.	
2	Renseigner un nom de ronde avec plus de 30 caractères.	Aucune action.	
3	Clic sur le bouton Ajouter.	Une erreur est affichée car le nom ne respecte pas les critères.	
4	Renseigner un nom de ronde avec entre 1 et 30 caractères.	Aucune action.	
5	Clic sur le bouton Ajouter.	Une erreur est affichée car la ronde ne possède pas assez de pointeaux.	
6	Sélection d'un pointeau dans la liste des pointeaux.	Le pointeau est affiché en bleu.	
7	Clic sur le bouton ajouter le pointeau.	Le pointeau sélectionné est ajouté à la fin du déroulement de la ronde.	
8	Double clic sur un pointeau dans la liste des pointeau.	Le pointeau sélectionné est ajouté à la fin du déroulement de la ronde.	
9	Sélection d'un pointeau dans la liste déroulement.	Le pointeau est affiché en bleu.	
10	Double clic sur un pointeau dans la liste des pointeau.	Le pointeau sélectionné est ajouté au déroulement de la ronde après le pointeau sélectionné dans cette dernière.	
11	Sélection d'un pointeau dans la liste déroulement.	Le pointeau est affiché en bleu.	
12	Sélection d'un pointeau dans la liste des pointeaux.	Le pointeau est affiché en bleu.	
13	Clic sur le bouton ajouter le pointeau.	Le pointeau sélectionné est ajouté au déroulement de la ronde après le pointeau sélectionné dans cette dernière.	
14	Double clic sur un pointeau dans la liste des pointeau.	Le pointeau sélectionné est ajouté au déroulement de la ronde.	
15	Double clic sur un pointeau dans la liste des pointeau.	Le pointeau sélectionné est ajouté au déroulement de la ronde.	

16	Clic sur la flèche monté d'un pointeau dans la liste déroulement.	Le pointeau change de position avec le pointeau précédent.
17	Clic sur la flèche descendre d'un pointeau dans la liste déroulement.	Le pointeau change de position avec le pointeau suivant.
18	Clic sur la croix de suppression d'un pointeau dans la liste déroulement.	Le pointeau est supprimer du déroulement.
19	Clic sur le bouton Ajouter.	Un message s'affiche comme quoi la ronde a été ajoutée.
20	Sélectionner un ronde dans la liste des nom.	Le bouton ajouter se désactive et le bouton enregistrer et supprimer s'active. Le déroulement de la ronde s'affiche dans la liste déroulement.
21	Modifier un temps min pour qu'il soit supérieure ou égale au temps max du même pointeau.	Aucune action.
22	Clic sur le bouton enregistrer.	Un message indique que les temps min et max ne son pas valide.
23	Changer le temps max pour qu'il soit supérieure au temps min.	Aucune action.
24	Clic sur le bouton enregistrer.	Un message nous indique que la ronde a été modifiée.
25	Clic sur le bouton supprimer	Un message demande si l'on souhaite vraiment supprimer la ronde.
26	Clic sur le bouton non	Aucune action.
27	Clic sur le bouton supprimer	Un message demande si l'on souhaite vraiment supprimer la ronde.
28	Clic sur le bouton oui	La ronde est supprimée.

Compte rendu :

Repère 16 : Crash de l'application lors de l'appuie sur la flèche du premier pointeau. Le pointeau étant le premier de la liste il ne peut pas être déplacé un cran plus haut. Une solution, vérifier que le pointeau dont la flèche est cliquée n'est pas le premier pointeau pointeau ainsi que ne pas afficher la flèche.

Repère 17 : Crash de l'application lors de l'appuie sur la flèche du dernier pointeau. Le pointeau étant le dernier de la liste il ne peut pas être déplacé un cran plus bas. Une solution, vérifier que le pointeau dont la flèche est cliquée n'est pas le dernier pointeau ainsi que ne pas afficher la flèche.

10 - Bilan de la réalisation personnelle

En conclusion, l'application actuelle est fonctionnelle. L'intégration des parties de mes collègues pourras se faire facilement grâce au système de classe mis en place.

Dans un premier temps il serait nécessaire d'implémenter le fichier XML de configuration. La classe AccesXML est terminée mais n'est pas encore utilisée dans le programme.

Dans la partie de gestion de la ronde, la liste des poteaux contenue dans le déroulement est rechargée dans sont intégrité à chaque ajout ou suppression de poteau. Afin d'optimiser l'affichage et de le rendre plus fluide il faudrait modifier la façon de charger cette liste.

11 - Annexes

11.1 - Script de création de la base de données MySQL

```
CREATE TABLE `agents` (  
  `idAgent` int(11) NOT NULL AUTO_INCREMENT,  
  `idTag` char(8) DEFAULT NULL,  
  `nom` varchar(45) NOT NULL,  
  `prenom` varchar(45) NOT NULL,  
  `agentActif` tinyint(4) NOT NULL DEFAULT '1',  
  PRIMARY KEY (`idAgent`),  
  KEY `fk_Agents_Tags1_idx` (`idTag`),  
  CONSTRAINT `fk_Agents_Tags1` FOREIGN KEY (`idTag`) REFERENCES `tags`  
  (`idTag`) ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `associationagentsrondes` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `idAgent` int(11) NOT NULL,  
  `idRonde` int(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `fk_AssociationAgentsRondes_Rondes1_idx` (`idRonde`),  
  KEY `fk_AssociationAgentsRondes_Agents1_idx` (`idAgent`),  
  CONSTRAINT `fk_AssociationAgentsRondes_Agents1` FOREIGN KEY (`idAgent`)  
  REFERENCES `agents` (`idAgent`) ON DELETE NO ACTION ON UPDATE NO ACTION,  
  CONSTRAINT `fk_AssociationAgentsRondes_Rondes1` FOREIGN KEY (`idRonde`)  
  REFERENCES `rondes` (`idRonde`) ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `associationpointeauxrondes` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `idRonde` int(11) NOT NULL,  
  `idPointeau` int(11) NOT NULL,  
  `ordrePointeau` int(11) NOT NULL,  
  `tempsProchainMin` int(11) DEFAULT '5',  
  `tempsProchainMax` int(11) DEFAULT '15',  
  PRIMARY KEY (`id`),  
  KEY `fk_AssociationPointeauxRondes_Pointeaux_idx` (`idPointeau`),  
  KEY `fk_AssociationPointeauxRondes_Rondes1_idx` (`idRonde`),  
  CONSTRAINT `fk_AssociationPointeauxRondes_Pointeaux` FOREIGN KEY  
  (`idPointeau`) REFERENCES `pointeaux` (`idPointeau`) ON DELETE NO ACTION ON  
  UPDATE NO ACTION,  
  CONSTRAINT `fk_AssociationPointeauxRondes_Rondes1` FOREIGN KEY (`idRonde`)  
  REFERENCES `rondes` (`idRonde`) ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `historiquepointeau` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `idAgent` int(11) NOT NULL,  
  `idRonde` int(11) NOT NULL,  
  `idPointeau` int(11) NOT NULL,  
  `date` datetime NOT NULL,  
  `ordrePointeau` int(11) NOT NULL,  
  `numeroRonde` int(11) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `fk_HistoriquePointeau_Rondes1_idx` (`idRonde`),  
  KEY `fk_HistoriquePointeau_Pointeaux1_idx` (`idPointeau`),  
  KEY `fk_HistoriquePointeau_Agents1_idx` (`idAgent`),  
  CONSTRAINT `fk_HistoriquePointeau_Agents1` FOREIGN KEY (`idAgent`)  
REFERENCES `agents` (`idAgent`) ON DELETE NO ACTION ON UPDATE NO ACTION,  
  CONSTRAINT `fk_HistoriquePointeau_Pointeaux1` FOREIGN KEY (`idPointeau`)  
REFERENCES `pointeaux` (`idPointeau`) ON DELETE NO ACTION ON UPDATE NO  
ACTION,  
  CONSTRAINT `fk_HistoriquePointeau_Rondes1` FOREIGN KEY (`idRonde`)  
REFERENCES `rondes` (`idRonde`) ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `mainscourantes` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `texte` varchar(250) NOT NULL,  
  `idHistoriquePointeau` int(11) NOT NULL,  
  `date` datetime NOT NULL,  
  `type` int(11) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `pointeaux` (  
  `idPointeau` int(11) NOT NULL AUTO_INCREMENT,  
  `idTag` char(8) DEFAULT NULL,  
  `lieu` varchar(20) DEFAULT NULL,  
  `numero` int(11) NOT NULL,  
  `pointeauActif` tinyint(4) NOT NULL DEFAULT '1',  
  PRIMARY KEY (`idPointeau`),  
  KEY `fk_Pointeaux_Tags1_idx` (`idTag`),  
  CONSTRAINT `fk_Pointeaux_Tags1` FOREIGN KEY (`idTag`) REFERENCES `tags`  
(`idTag`) ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



```
CREATE TABLE `responsable` (  
  `idResponsable` int(11) NOT NULL AUTO_INCREMENT,  
  `login` varchar(45) NOT NULL,  
  `mdp` varchar(45) NOT NULL,  
  PRIMARY KEY (`idResponsable`)  
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `rondes` (  
  `idRonde` int(11) NOT NULL AUTO_INCREMENT,  
  `nom` varchar(30) NOT NULL,  
  `rondeActive` tinyint(4) NOT NULL DEFAULT '1',  
  PRIMARY KEY (`idRonde`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `tags` (  
  `idTag` char(8) NOT NULL,  
  PRIMARY KEY (`idTag`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Lemée Gabriel Dossier Personnel

