Dossier projet Chrono-cross

Revue finale

version 1.0

Sommaire

Introduction

Dans le cadre de ma deuxième année de BTS SN IR j'ai travaillé sur le projet chrono-cross: chronométrage de course à pied. Il m'a été confié le module de gestion de manifestation que je vais présenter ci-après.

Présentation générale

Le projet Chrono-Cross consiste à mettre en place une automatisation presque complète d'une course à pied ou d'un cross. Il doit permettre la configuration d'une manifestation et des courses et d'autre part l'inscription des coureurs. Il doit aussi réaliser l'affichage des temps sur un panneau lumineux, lors du passage des coureurs devant la ligne d'arrivée. La détection est faite grâce à un capteur infrarouge. L'affichage des résultats (classement des coureurs) est accessible par l'organisateur et les coureurs, sur grand écran, avec la possibilité d'imprimer les résultats.

Présentation individuelle

Mon objectif est de faire une application qui permettra à l'organisateur du cross de :

- Créer, modifier ou supprimer une manifestation (nom et date de l'évènement);

Une fois la manifestation créée, il pourra :

- Créer, modifier ou supprimer des courses (nom, heure de départ, et la longueur de la course)

Durant la course :

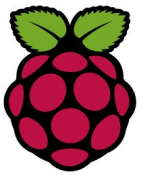
- Afficher les résultats sur grand écran via une carte Raspberry Pi

Une fois la course terminée l'application permettra de :

- Afficher les résultats, et/ou
- Imprimer les résultats.

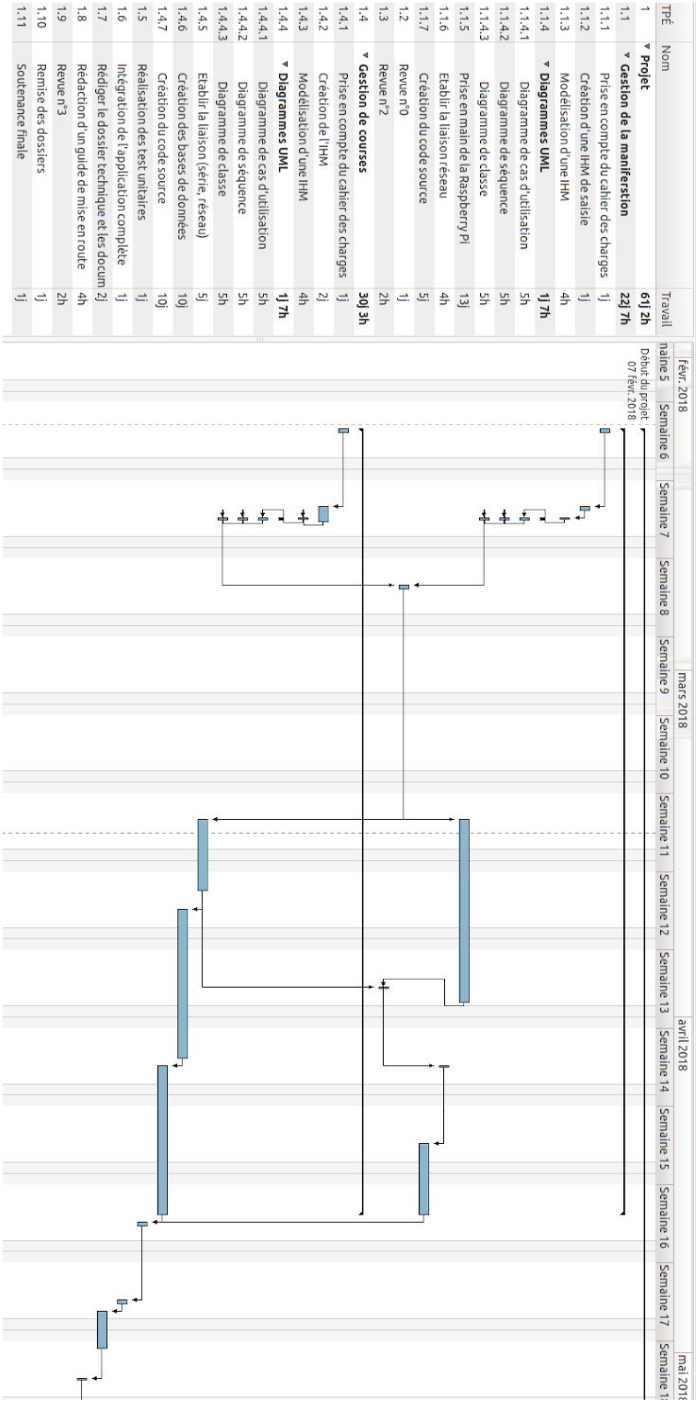
Logiciels et matériels utilisés

Le diagramme de Gantt a été réalisé avec Planner.
L'élaboration des diagrammes a été effectuée avec BOUML 7.4.
Les IHM ont été fait avec Qt Designer, et le codage avec Qt Creator en langage de programmation C++ avec l'API QT 4.8.
La rédaction du dossier : Google Doc.
Pour la réalisation du diaporama j'ai utilisé Google Slide.
Une carte Raspberry Pi pour l'affichage des résultats sur la télévision.



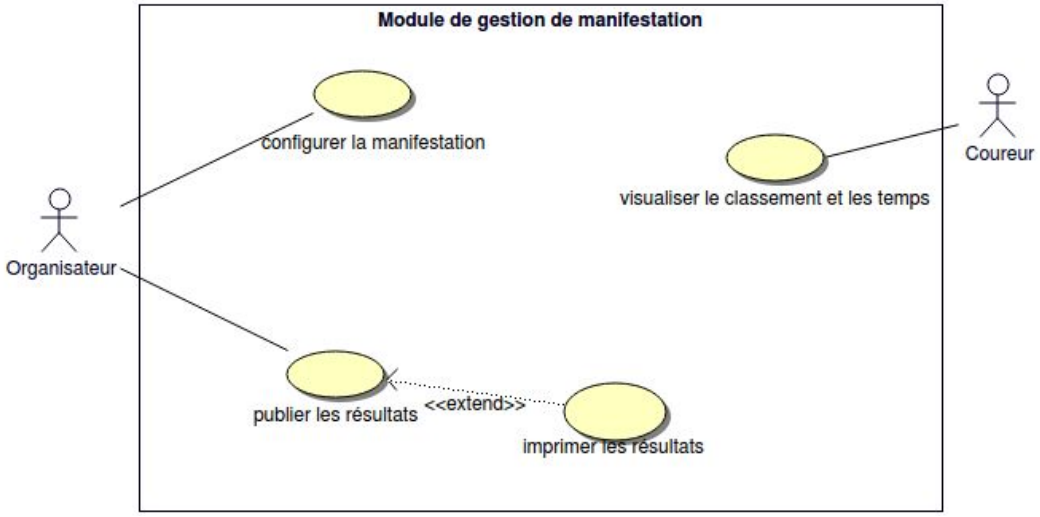

Planification des tâches

Les tâches à réaliser sont effectuées dans l'ordre chronologique suivant. Pour analyser le système, je commence par étudier le cahier des charges pour repérer les différentes tâches à réaliser. La conception débute par la réalisation des diagrammes : de cas d'utilisation, de séquence, de classe et de déploiement. La réalisation du code source permet de configurer des manifestations et l'affichage des résultats. Un exécutable est implémenté sur le PC course de l'organisateur. Une carte Raspberry Pi est reliée à la télévision par un câble HDMI pour afficher les résultats. Les tests unitaires permettent de vérifier la fonctionnalité des applications.



Analyse du système

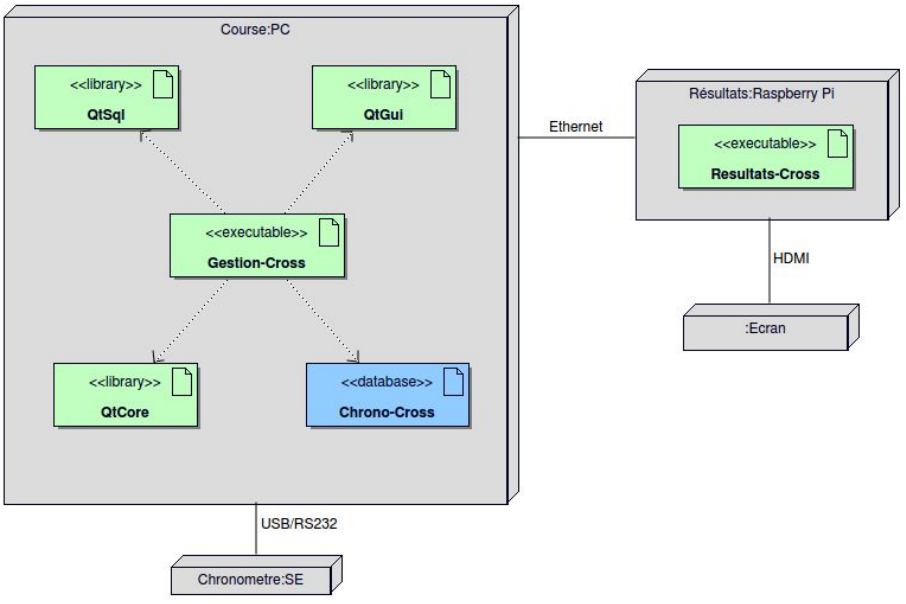
Diagramme de cas d'utilisation



Le diagramme de cas d'utilisation permet de pointer les fonctionnalités majeures du projet qui sont :

- La configuration d'une manifestation,
- La possibilité de publier les résultats ainsi que l'option pour l'organisateur de les imprimer,
- La possibilité de visualiser les résultats sur grand écran par les coureurs.

Diagramme de déploiement



Les deux applications “Gestion-Cross” et “Resultats-Cross” ont été développées en C++ avec l’API QT 4.8. La base de données MySQL “Chrono-Cross” est installée sur le PC Course et accessible aussi à partir de la Raspberry Pi.

Le système de chronométrage est relié via un adaptateur USB/RS232.

Diagramme de classes de l'application Gestion-Cross

Ce diagramme de classes est une retranscription des attributs et méthodes des classes utilisés dans le projet Gestion-Cross.

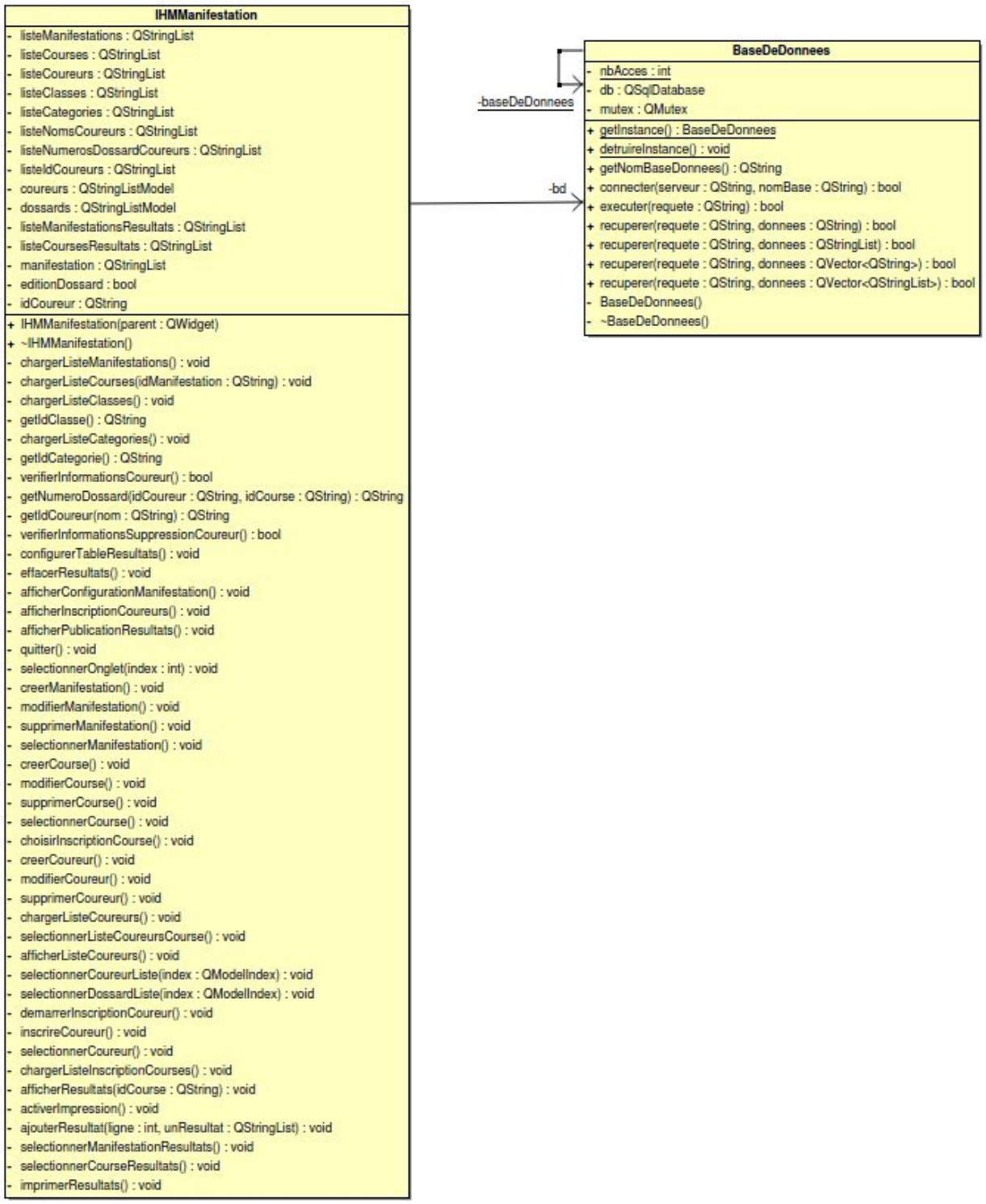
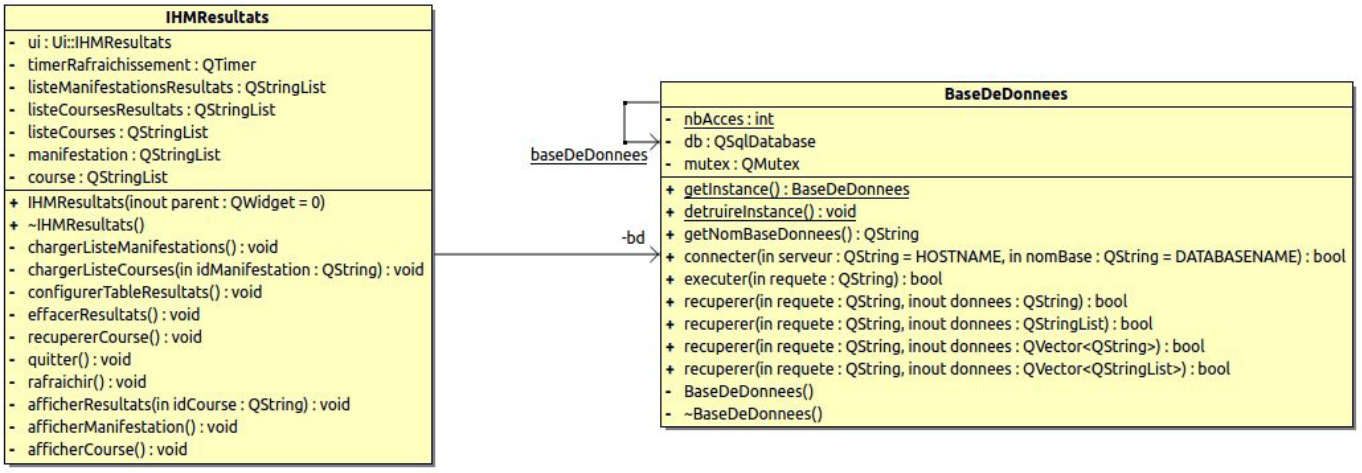


Diagramme de classes de l'application Resultats-Cross

Ce diagramme de classe est une retranscription des attributs et méthodes des classes utilisés dans le projet Resultats-Cross.



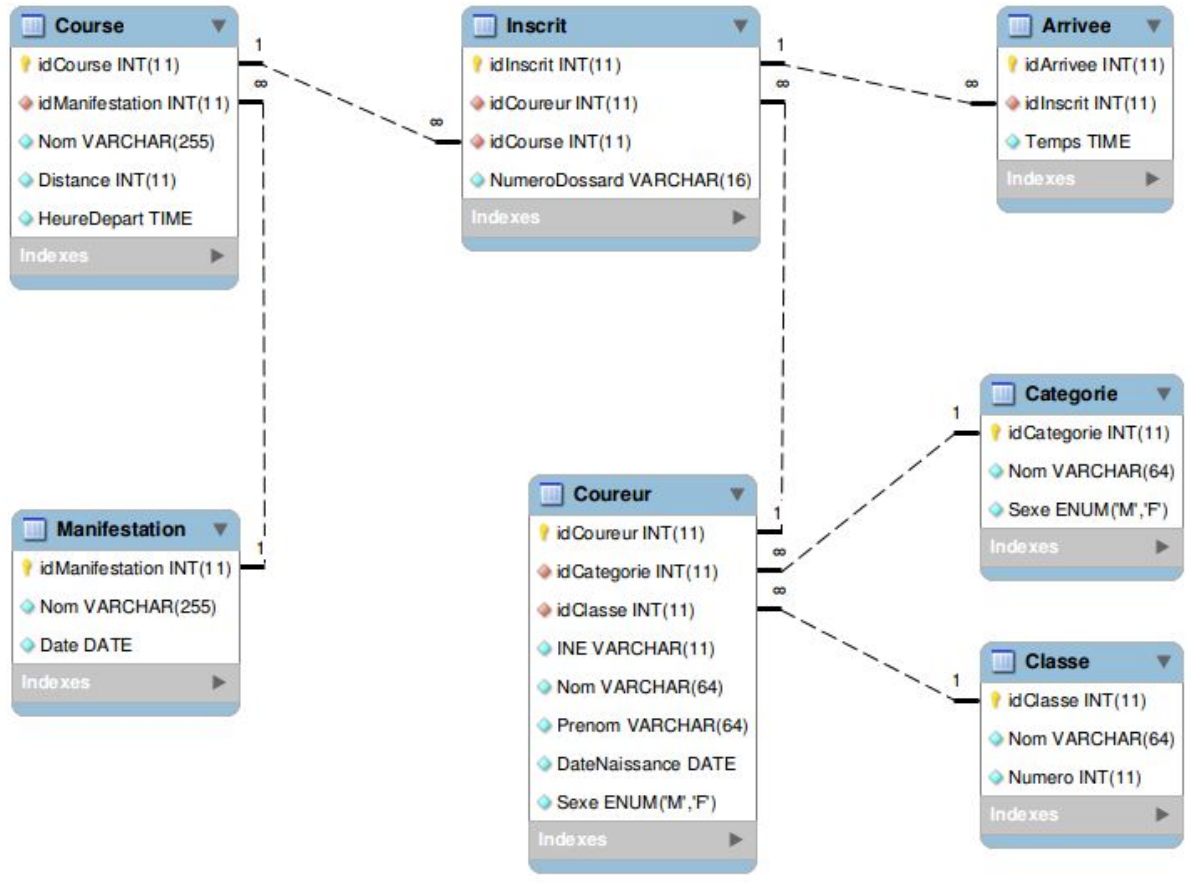
Base de données

La base de données Chrono-cross contient les informations relatives aux :

- manifestations,
- courses,
- inscriptions à la course,
- coureurs,
- temps d'arrivée des coureurs,
- catégories des coureurs,
- classe du coureur.

Schéma relationnel

Le schéma ci-dessous précise les liaisons entre les tables et les multiplicités.



Pour gérer la base de données Chrono-Cross les requêtes SQL utilisées sont :

- **INSERT** est une commande SQL qui ajoute un ou plusieurs tuple(s) dans la table d'une base de données relationnelles.
- **UPDATE** est une commande SQL qui modifie un ou plusieurs tuple(s) dans la table d'une base de données relationnelles.
- **DELETE** est une commande SQL qui supprime un ou plusieurs tuple(s) dans la table d'une base de données relationnelles.

- **SELECT** est une commande SQL qui permet d'extraire des données des tables d'une base de données relationnelles.

Exemples de requêtes SQL

- Requête **INSERT** :

```
INSERT INTO Manifestation(Nom,Date)
VALUES('" + ui->lineEditNomManifestation->text() + "','" +
ui->dateEditEditionManifestation->date().toString("yyyy-MM-dd") + "')
```

Cette requête insère dans la table "Manifestation" dans le champ "Nom" et "Date" les valeurs contenues dans les variables "lineEditNomManifestation" et "dateEditEditionManifestation".

- Requête **UPDATE** :

```
UPDATE Manifestation SET Nom = '" + ui->lineEditNomManifestation->text() + "', Date = '" +
ui->dateEditEditionManifestation->date().toString("yyyy-MM-dd") + "'
WHERE idManifestation = '" + idManifestation + "'
```

Cette requête modifie dans la table "Manifestation" les données du champ "Nom" par les données contenues dans la variable "lineEditNomManifestation", dans le champ "Date" les données contenue dans la variable "dateEditEditionManifestation".

- Requête **DELETE** :

```
DELETE FROM Manifestation WHERE idManifestation = '" + idManifestation + "'
```

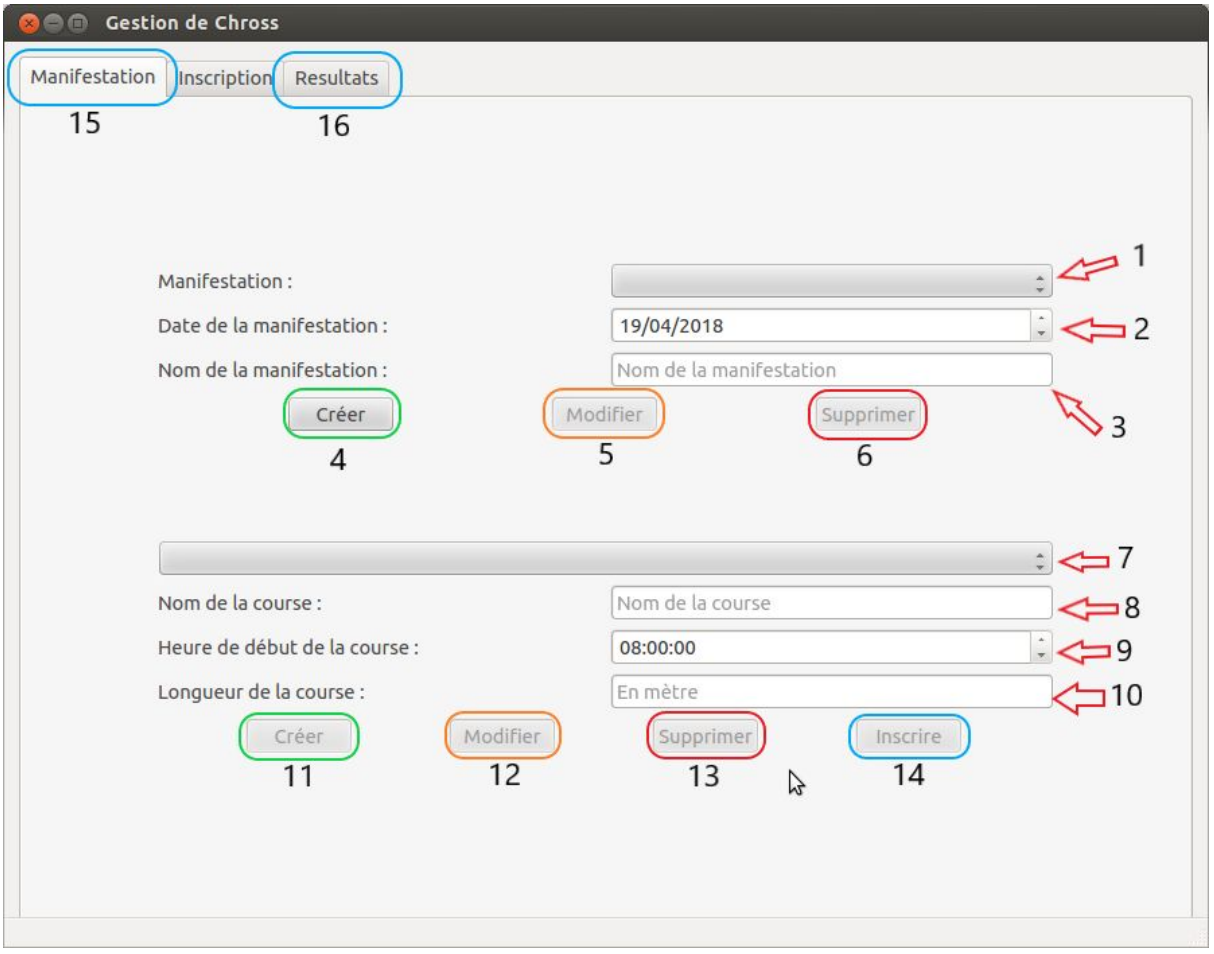
Cette requête supprime dans la table "Manifestation" le "Nom" et la "Date" de la manifestation identifiée sous le numéro "idManifestation".

- Requête **SELECT** :

```
SELECT * FROM Manifestation
ORDER BY Date ASC;
```

Cette requête récupère les données de tous les champs de la table "Manifestation" classées par date.

L'application d'édition de manifestation



Partie manifestation :

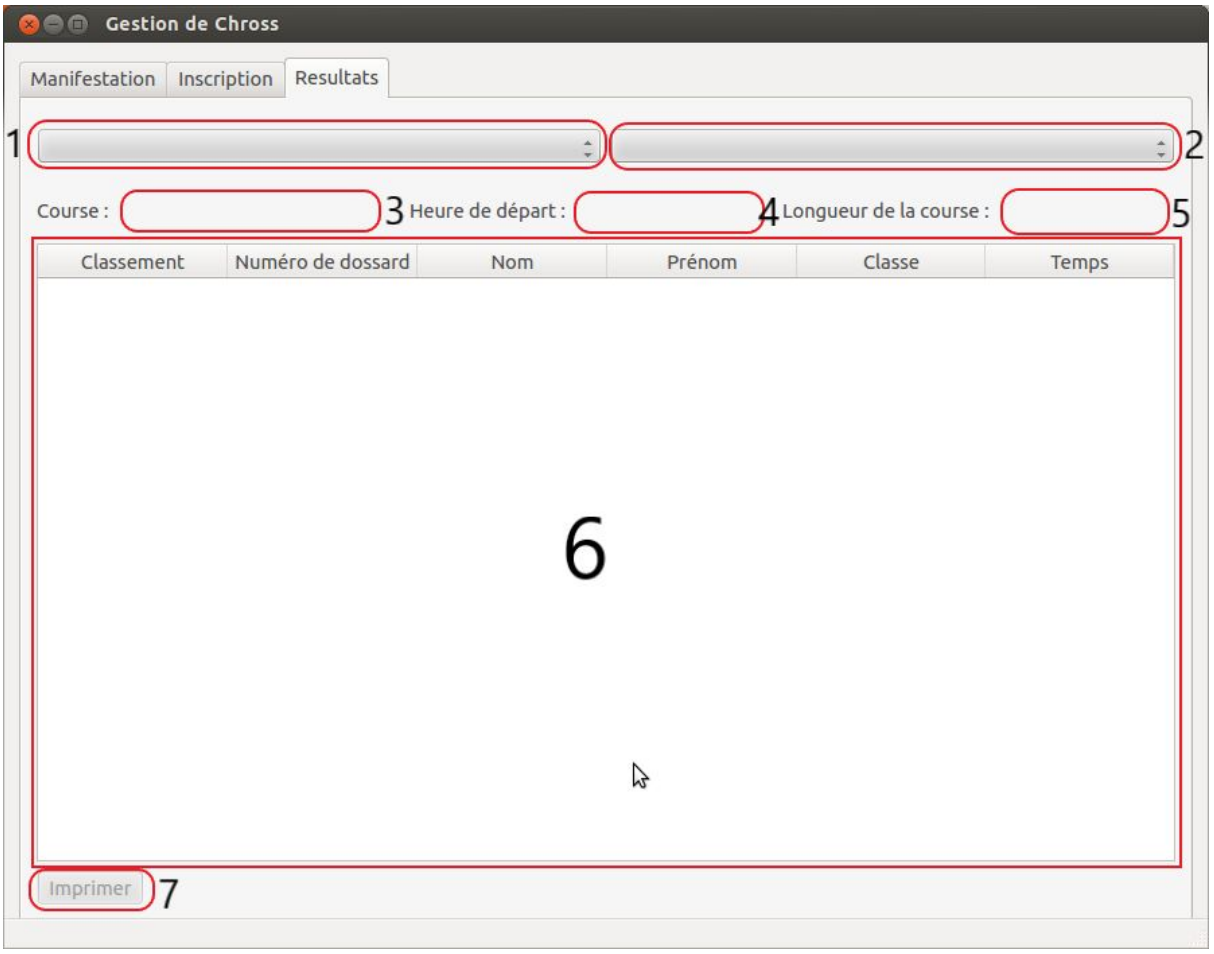
1 : Liste déroulante des manifestations
2 : Ligne de saisie de la date
3 : Ligne de saisie du nom
4 : Bouton de création
5 : Bouton de modification
6 : Bouton de suppression

Partie course :

7 : Liste déroulante des courses
8 : Ligne de saisie du nom
9 : Ligne de saisie de l'heure de départ
10 : Ligne de saisie de la distance
11 : Bouton de création
12 : Bouton de modification
13 : Bouton de suppression
14 : Bouton vers l'onglet "Inscription"

15 : Onglet "Manifestation"
16 : Onglet "Resultats"

L'affichage des résultats :

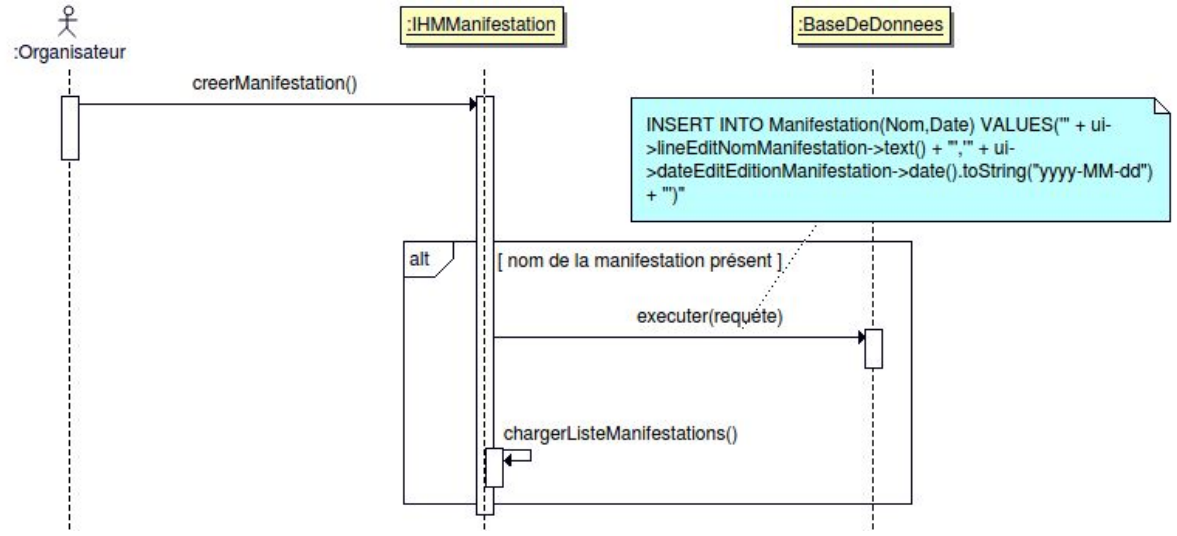


1 : Liste déroulante des manifestations
2 : Liste déroulante des courses
3 : Afficheur du nom de la course
4 : Afficheur de l'heure de départ
5 : Afficheur de la longueur de la course
6 : Tableau des résultats de la course
7 : Bouton d'impression

Cas d'utilisation : configurer la manifestation

Scénarios de manifestation

Scénario de création de manifestation

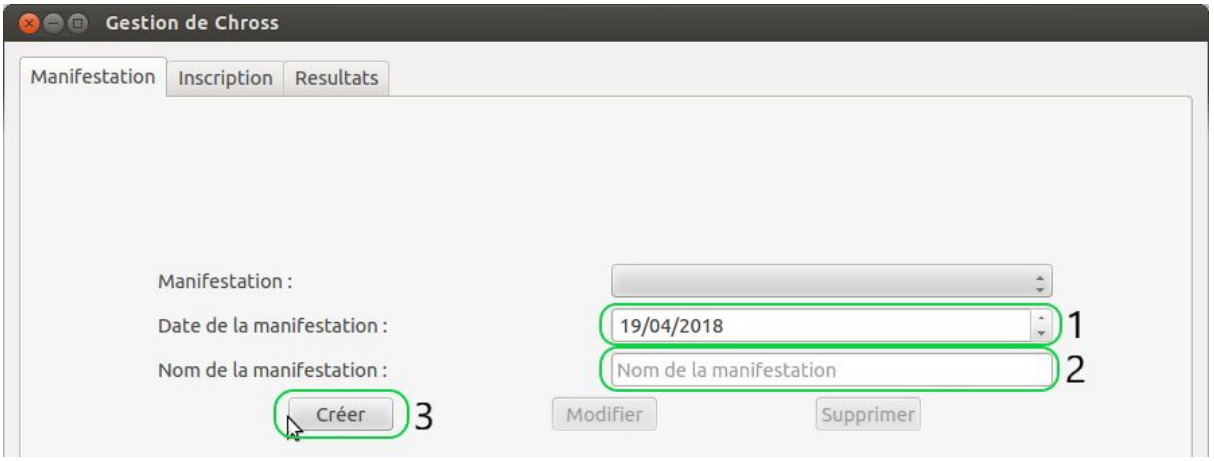


Pour la création d'une manifestation la méthode appelée est :

```
void IHMManifestation::creerManifestation()
{
        bool retour = false;

        // Vérifications
        if(ui->lineEditNomManifestation->text().isEmpty())
        {
                return;
        }
        QString requete = "INSERT INTO Manifestation(Nom,Date) VALUES('" + ui->lineEditNomManifestation->text() + "','" + ui->dateEditEditionManifestation->date().toString("yyyy-MM-dd") + "')";
        retour = bd->executer(requete);
        if(retour)
        {
                chargerListeManifestations();
        }
}
```

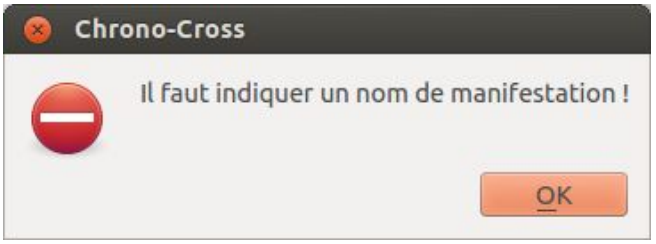
La déclaration d'une variable "retour" de type booléen initialisée en false.
La condition "if" pour vérifier la présence d'une saisie dans la ligne d'édition.
La création de la variable requete qui contient le code SQL pour une insertion.
L'appel de la méthode executer() pour réaliser l'exécution de la requête. Elle retourne true si la requête a été exécutée avec succès sinon false.
Si la requête s'est bien exécutée alors on recharge la liste des manifestations pour mettre à jour la liste déroulante.

Pour créer une manifestation il suffit de modifier la date en saisissant la date souhaitée au clavier ou en utilisant le bouton de *scroll*. La date affichée par défaut est celle du jour. Puis il faut entrer le nom de la manifestation, et appuyer sur le bouton “Créer”. Une fois que les manipulations sont effectuées, les paramètres de la manifestation sont enregistrés dans la base de données.

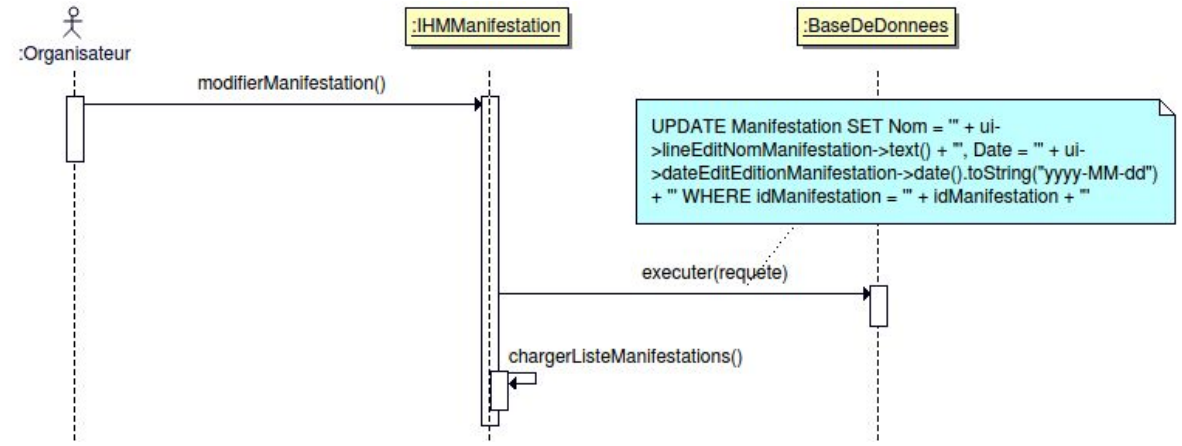


1 : Ligne de saisie de la date 2 : Ligne de saisie du nom
3 : Bouton de création

Si lors de la création de la manifestation l’organisateur oublie de saisir le nom de la manifestation, un message d’alerte prévient l’organisateur :



Scénario de modification de manifestation

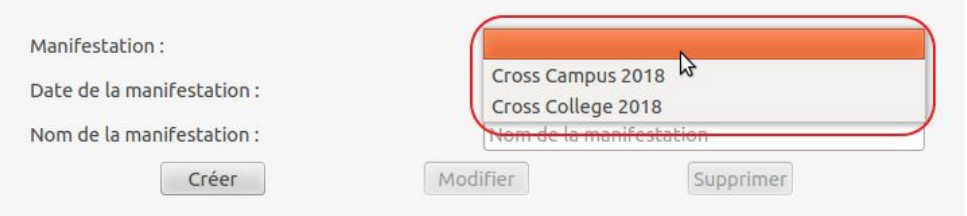


Pour la modification d'une manifestation la méthode appelée est :

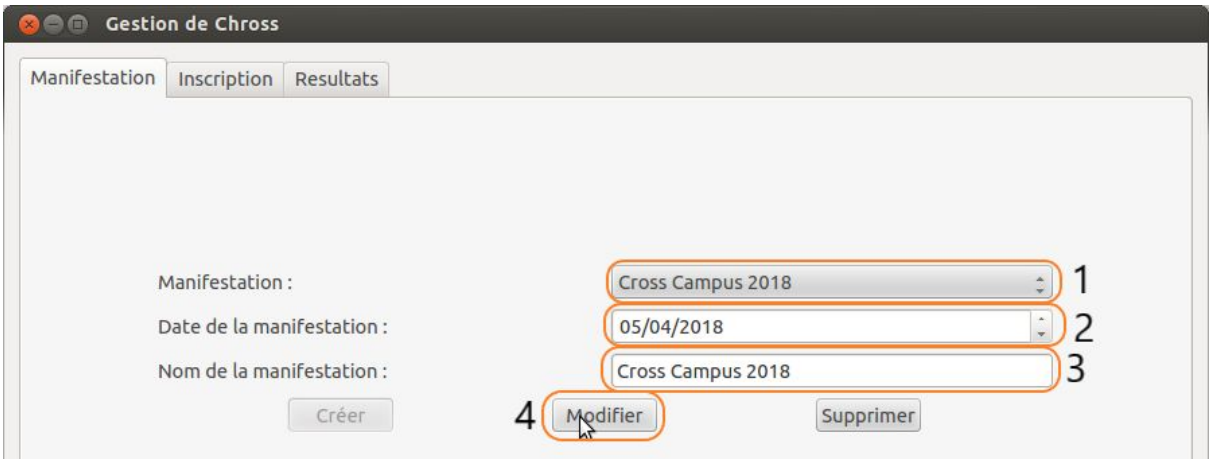
```
void IHMManifestation::modifierManifestation()
{
        bool retour = false;

        // Vérifications
        if(ui->lineEditNomManifestation->text().isEmpty())
        {
                return;
        }

        // Récupère l'identifiant de la manifestation
        QString idManifestation = listeManifestations.at(ui->listeCreationManifestations->currentIndex()-1).at(0);

        QString requete = "UPDATE Manifestation SET Nom = '" + ui->lineEditNomManifestation->text() + "', Date = '" +
ui->dateEditEditionManifestation->date().toString("yyyy-MM-dd") + "' WHERE idManifestation = '" + idManifestation + "'";

        retour = bd->executer(requete);

        if(retour)
        {
                chargerListeManifestations();
        }
}
```

La déclaration d'une variable "retour" de type booléen initialisée en false.
La condition "if" pour vérifier la présence d'une saisie dans la ligne d'édition.
La création d'une variable idManifestation qui contient la valeur de l'identifiant SQL.
La création de la variable requete qui contient le code SQL pour une modificaion.
L'appel de la méthode executer() pour réaliser l'exécution de la requête. Elle retourne true si la requête a été exécutée avec succès sinon false.
Si la requête s'est bien exécutée alors on recharge la liste des manifestations pour mettre à jour la liste déroulante.

Pour modifier une manifestation, l'organisateur sélectionne, dans la liste déroulante des manifestations, celle qu'il souhaite modifier :

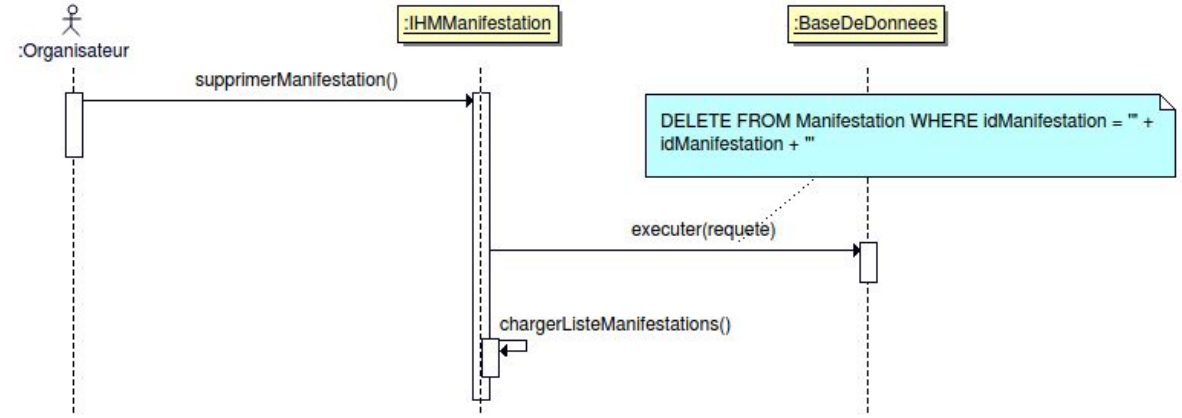


Après modification de la manifestation, l'organisateur peut modifier la date si nécessaire, en respectant la même procédure que celle utilisée lors de la création d'une manifestation (cf. page 12). Lorsque la date est actualisée, il suffit d'entrer le nouveau nom choisi pour la manifestation puis appuyer sur le bouton "Modifier". Une fois que les manipulations sont effectuées les modifications apportées aux paramètres de la manifestation sont enregistrées dans la base de données.



1 : Liste déroulante des manifestations
2 : Ligne de saisie de la date
3 : Ligne de saisie du nom
4 : Bouton de modification

Scénario de suppression de manifestation

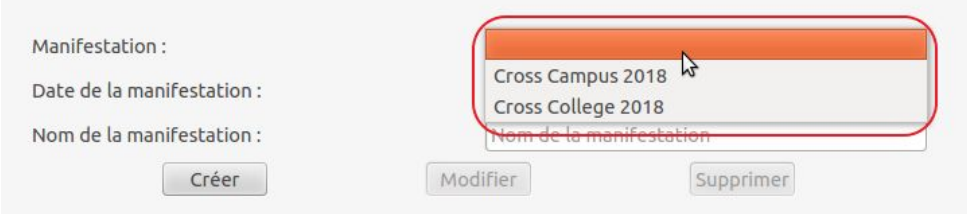


Pour la suppression d'une manifestation la méthode appelée est :

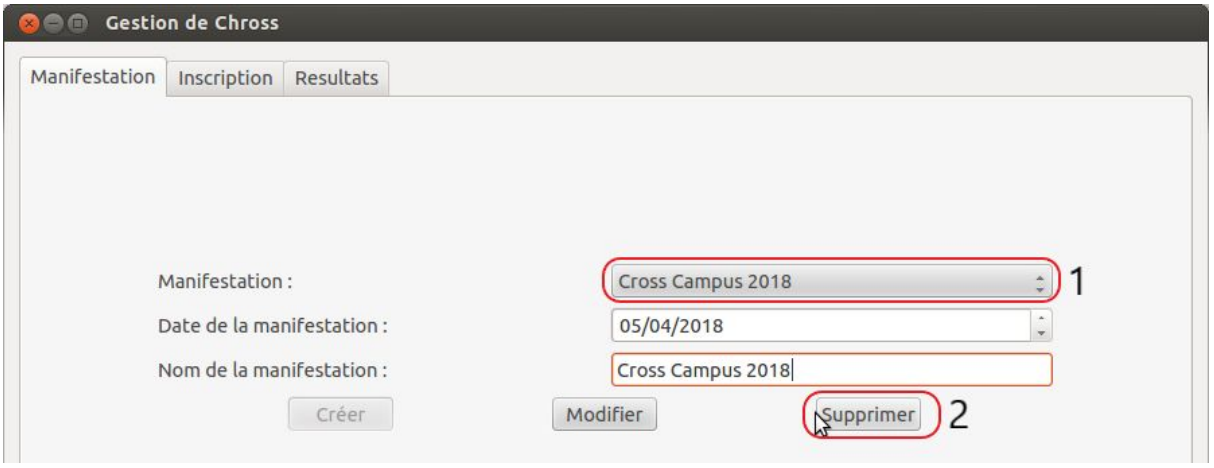
```
void IHMManifestation::supprimerManifestation()
{
        bool retour = false;

        // Récupère l'identifiant de la manifestation
        QString idManifestation = listeManifestations.at(ui->listeCreationManifestations->currentIndex()-1).at(0);

        QString requete = "DELETE FROM Manifestation WHERE idManifestation = '" + idManifestation + "'";

        retour = bd->executer(requete);

        if(retour)
        {
                chargerListeManifestations();
        }
}
```

La déclaration d'une variable "retour" de type booléen initialisée en false.
La création d'une variable idManifestation qui contient la valeur de l'identifiant SQL.
La création de la variable requete qui contient le code SQL pour une suppression.
L'appel de la méthode executer() pour réaliser l'exécution de la requête. Elle retourne true si la requête a été exécutée avec succès sinon false.
Si la requête s'est bien exécutée alors on recharge la liste des manifestations pour mettre à jour la liste déroulante.

Pour supprimer une manifestation de la base de données, il est nécessaire que l'organisateur sélectionne dans la liste déroulante des manifestations, celle qu'il souhaite supprimer.



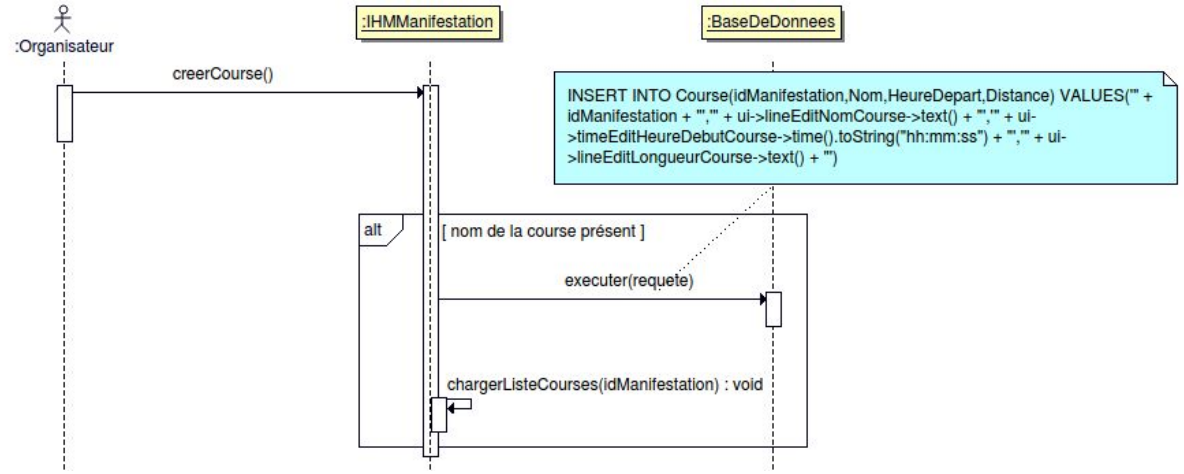
Lorsque la manifestation est identifiée, il suffit d'appuyer sur le bouton "Supprimer".



1 : Liste déroulante des manifestations 2 : Bouton de suppression

Scénarios de course

Scénario de création de course

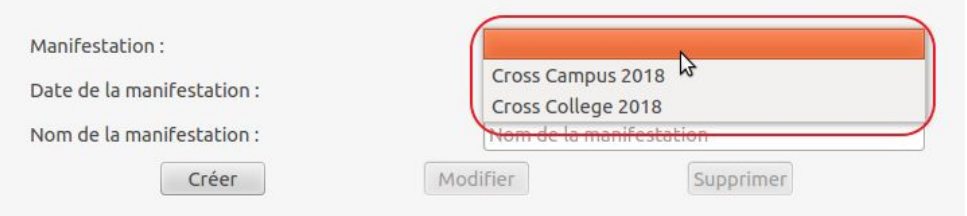


Pour la création d'une course la méthode appelée est :

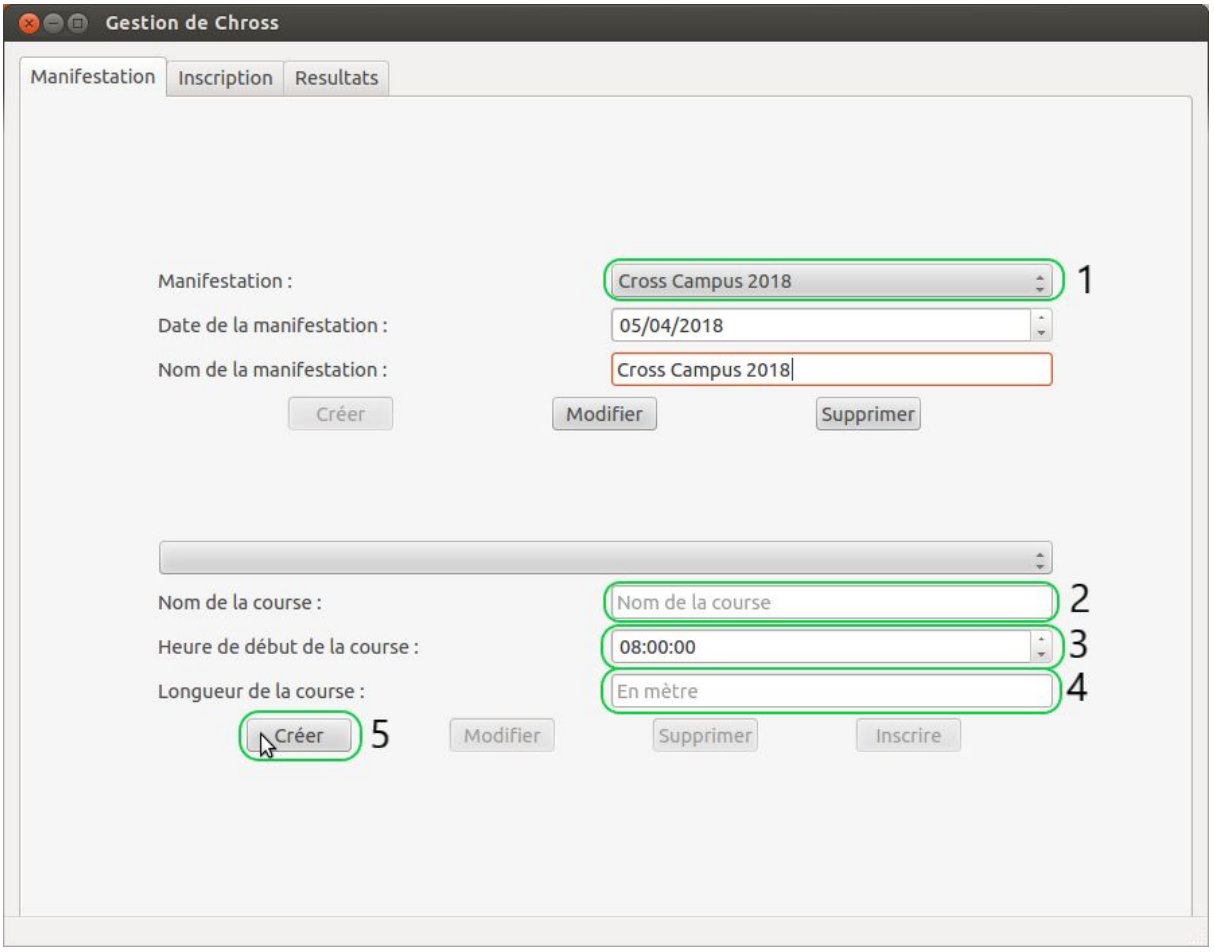
```
void IHMManifestation::creerCourse()
{
        bool retour = false;

        // Vérifications
        if(ui->lineEditNomCourse->text().isEmpty())
        {
                return;
        }

        QString idManifestation = listeManifestations.at(ui->listeCreationManifestations->currentIndex()-1).at(0);

        QString requete = "INSERT INTO Course(idManifestation,Nom,HeureDepart,Distance) VALUES('" + idManifestation + "','" + ui->lineEditNomCourse->text() + "','" + ui->timeEditHeureDebutCourse->time().toString("hh:mm:ss") + "','" + ui->lineEditLongueurCourse->text() + "')";

        retour = bd->executer(requete);

        if(retour)
        {
                chargerListeCourses(idManifestation);
        }
}
```

La déclaration d'une variable "retour" de type booléen initialisée en false.
La condition "if" pour vérifier la présence d'une saisie dans la ligne d'édition.
La création d'une variable idManifestation qui contient la valeur de l'identifiant SQL.
La création de la variable requete qui contient le code SQL pour une insertion.
L'appel de la méthode executer() pour réaliser l'exécution de la requête. Elle retourne true si la requête a été exécutée avec succès sinon false.
Si la requête s'est bien exécutée alors on recharge la liste des manifestations pour mettre à jour la liste déroulante.

Pour créer une course, il est nécessaire que l'organisateur sélectionne dans la liste déroulante des manifestations, la manifestation dans laquelle il veut créer la course.

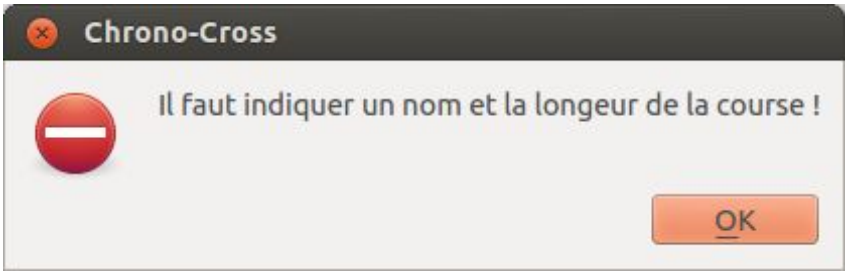


Une fois sélectionnée il suffit de saisir le nom de la course, modifier l'horaire en saisissant l'heure souhaitée au clavier ou en utilisant le bouton de scroll. L'heure affichée par défaut est 8 heures du matin . Lorsque l'heure est paramétrée, il suffit d'entrer la longueur de la course et d'appuyer sur le bouton "Créer". Les paramètres de la course sont alors enregistrés dans la base de données.

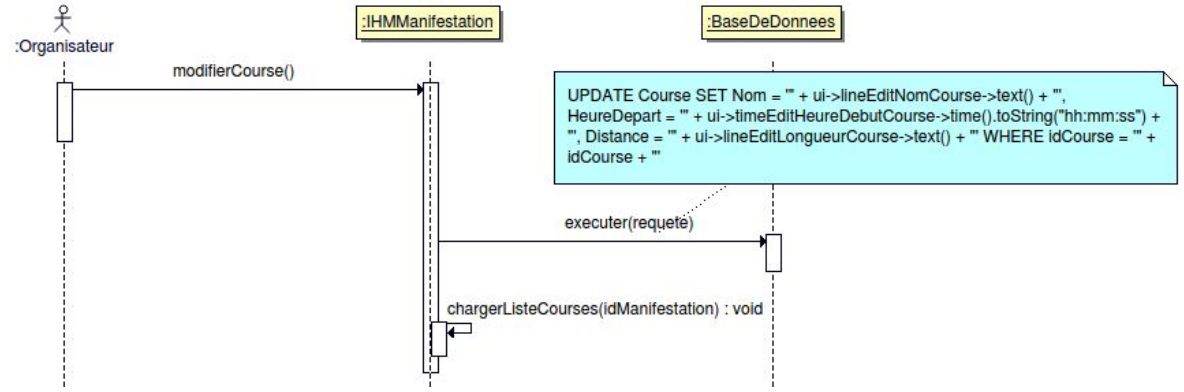


1 : Liste déroulante des manifestations
2 : Ligne de saisie du nom
3 : Ligne de saisie de l'heure de départ
4 : Ligne de saisie de la distance
5 : Bouton de création

Si lors de la création de la course l'organisateur oublie de saisir le nom de la manifestation ou la longueur de la course, il y a un message d'alerte qui le prévient :



Scénario de modification de course

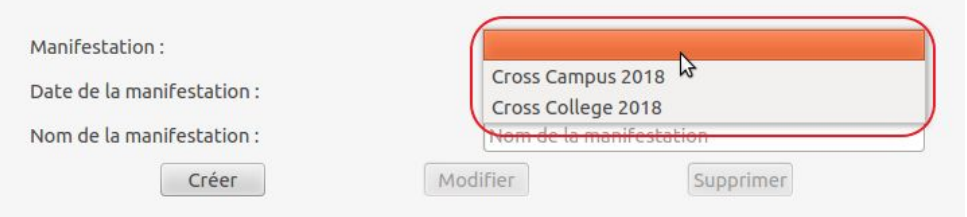


Pour la modification d'une course la méthode appelée est :

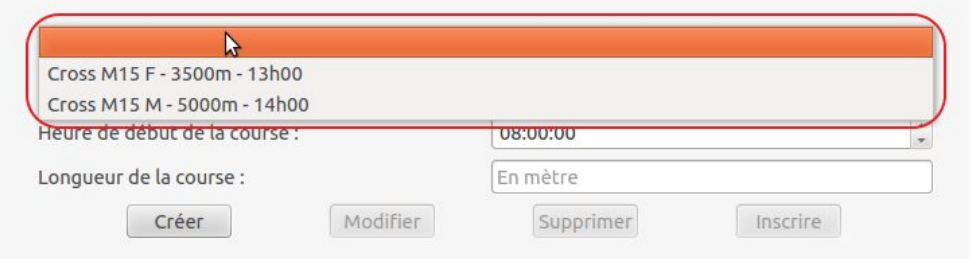
```
void IHMManifestation::modifierCourse()
{
        bool retour = false;

        // Vérifications
        if(ui->lineEditNomCourse->text().isEmpty())
        {
                return;
        }

        // Récupère l'identifiant de la course
        QString idCourse = listeCourses.at(ui->listeCreationCourses->currentIndex()-1).at(0);

        QString requete = "UPDATE Course SET Nom = '" + ui->lineEditNomCourse->text() + "', HeureDepart = '" +
ui->timeEditHeureDebutCourse->time().toString("hh:mm:ss") + "', Distance = '" + ui->lineEditLongueurCourse->text() + "' WHERE idCourse = '" +
idCourse + "'";

        retour = bd->executer(requete);

        if(retour)
        {
                chargerListeCourses(listeManifestations.at(ui->listeCreationManifestations->currentIndex()-1).at(0));
        }
}
```

La déclaration d'une variable "retour" de type booléen initialisée en false.
La condition "if" pour vérifier la présence d'une saisie dans la ligne d'édition.
La création d'une variable idCourse qui contient la valeur de l'identifiant SQL.
La création de la variable requete qui contient le code SQL pour une modification.
L'appel de la méthode executer() pour réaliser l'exécution de la requête. Elle retourne true si la requête a été exécutée avec succès sinon false.
Si la requête s'est bien exécutée alors on recharge la liste des manifestations pour mettre à jour la liste déroulante.

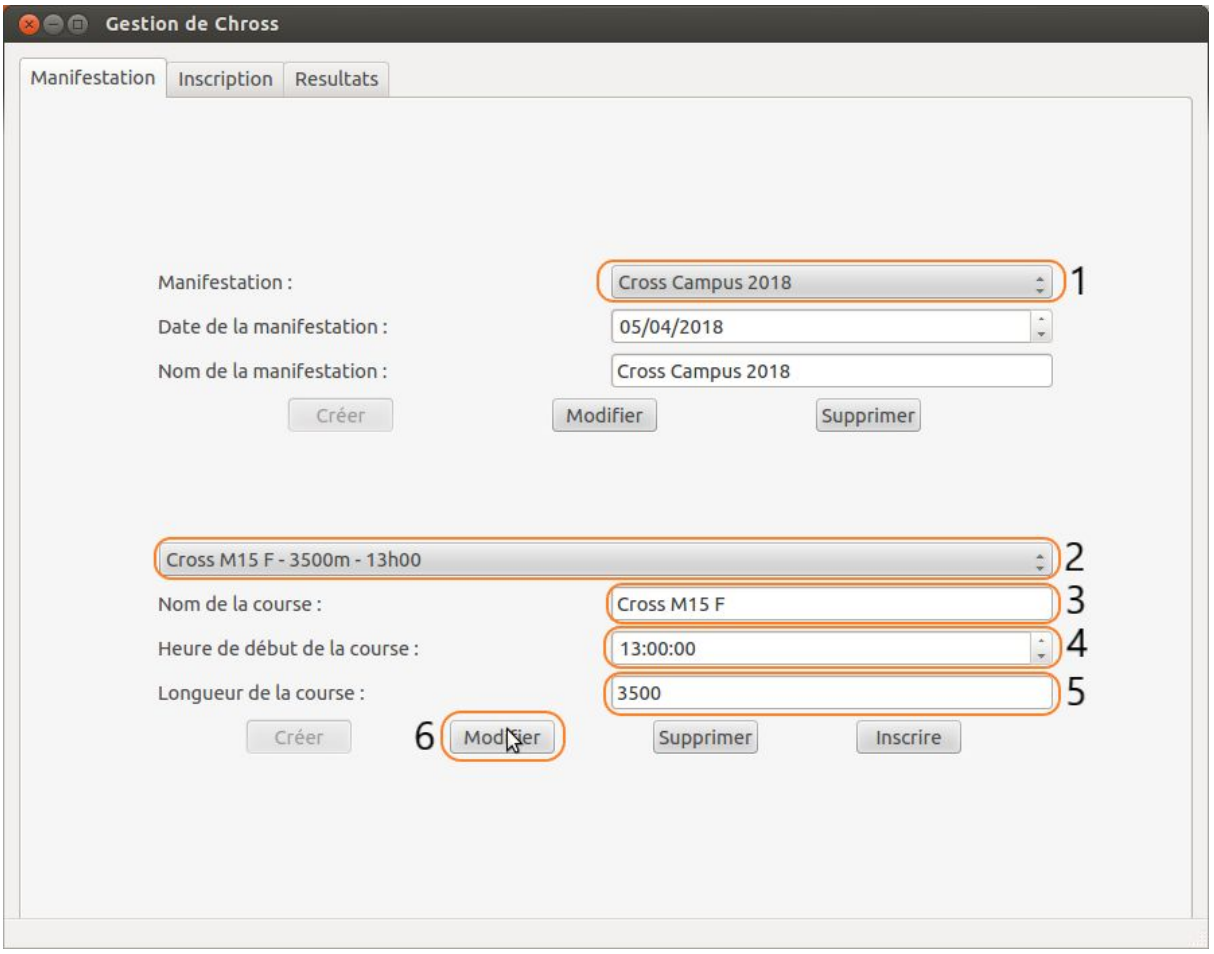
Pour modifier une course, il est nécessaire que l'organisateur sélectionne dans la liste déroulante des manifestations, celle qu'il veut modifier.



Une fois la manifestation sélectionnée il peut choisir la course à modifier.

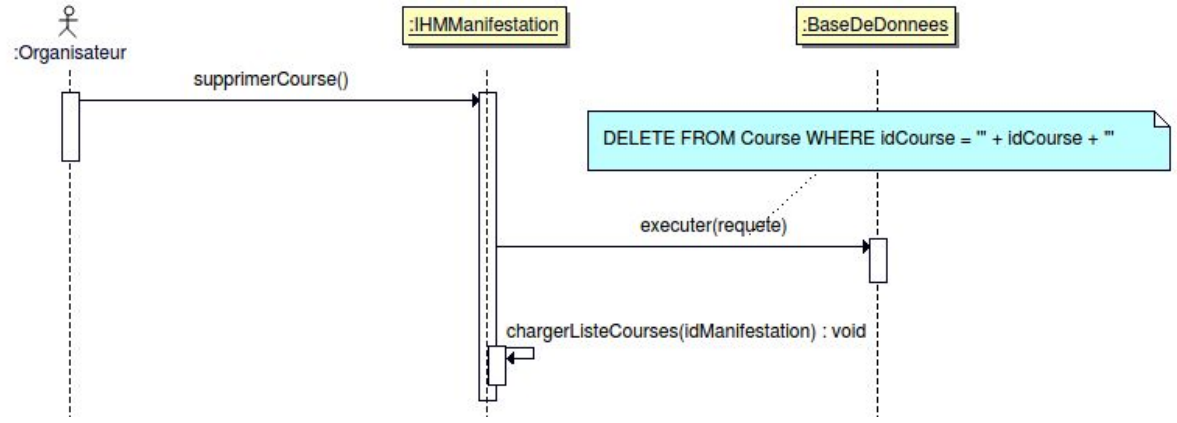


L'organisateur peut dès lors modifier le nom, l'heure et la longueur de la course si nécessaire, puis appuyer sur le bouton "Modifier". Une fois que les manipulations sont effectuées, les modifications apportées aux paramètres de la course sont enregistrées dans la base de données.



1 : Liste déroulante des manifestations
2 : Liste déroulante des courses
3 : Ligne de saisie du nom
4 : Ligne de saisie de l'heure
5 : Ligne de saisie de la distance
6 : Bouton de modification

Scénario de suppression de course

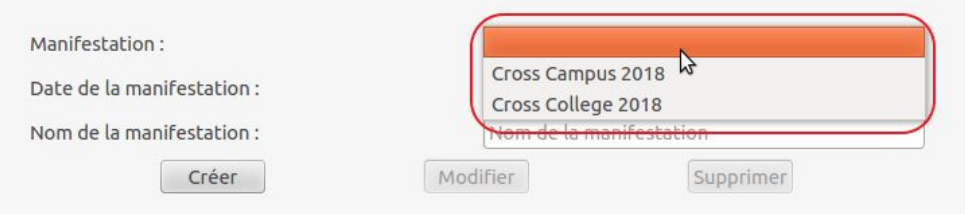


Pour la suppression d'une course la méthode appelée est :

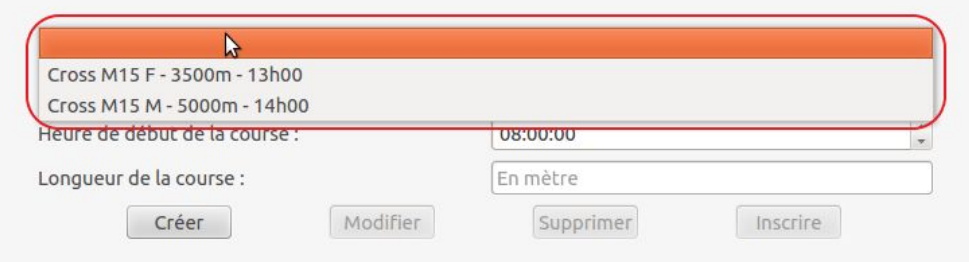
```
void IHMManifestation::supprimerCourse()
{
        bool retour = false;

        // Récupère l'identifiant de la course
        QString idCourse = listeCourses.at(ui->listeCreationCourses->currentIndex()-1).at(0);

        QString requete = "DELETE FROM Course WHERE idCourse = '" + idCourse + "'";

        retour = bd->executer(requete);

        if(retour)
        {
                chargerListeCourses(listeManifestations.at(ui->listeCreationManifestations->currentIndex()-1).at(0));
        }
}
```

La déclaration d'une variable "retour" de type booléen initialisée en false.
La création d'une variable idCourse qui contient la valeur de l'identifiant SQL.
La création de la variable requete qui contient le code SQL pour une suppression.
L'appel de la méthode executer() pour réaliser l'exécution de la requête. Elle retourne true si la requête a été exécutée avec succès sinon false.
Si la requête s'est bien exécutée alors on recharge la liste des manifestations pour mettre à jour la liste déroulante.

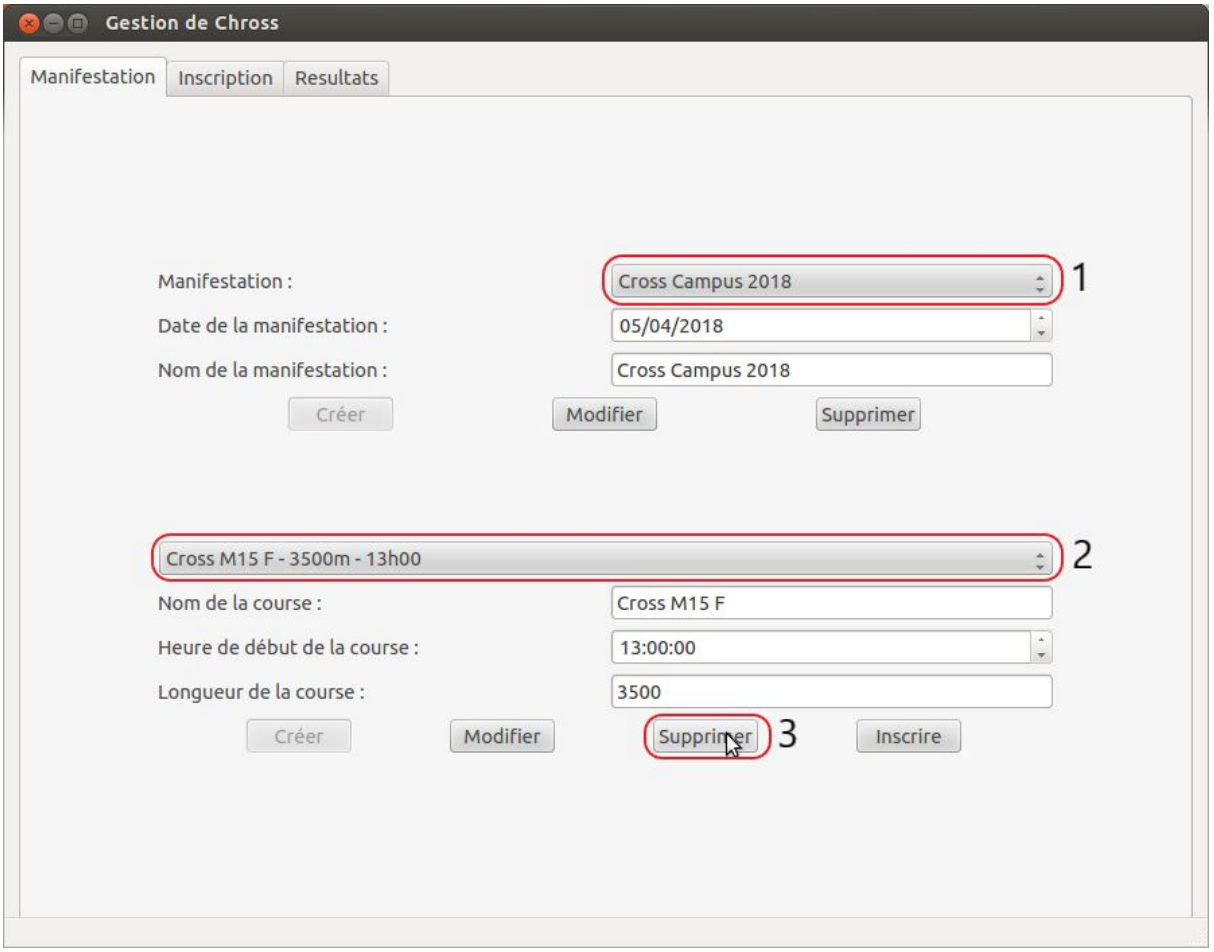
Pour supprimer une course, il est nécessaire que l'organisateur sélectionne dans la liste déroulante des manifestations, celle dans laquelle il veut supprimer la course.



Une fois que la manifestation est sélectionnée, il peut choisir la course à supprimer.



Puis il appuie sur le bouton “Supprimer”. Lorsque les manipulations sont effectuées, la manifestation est supprimée de la base de données.

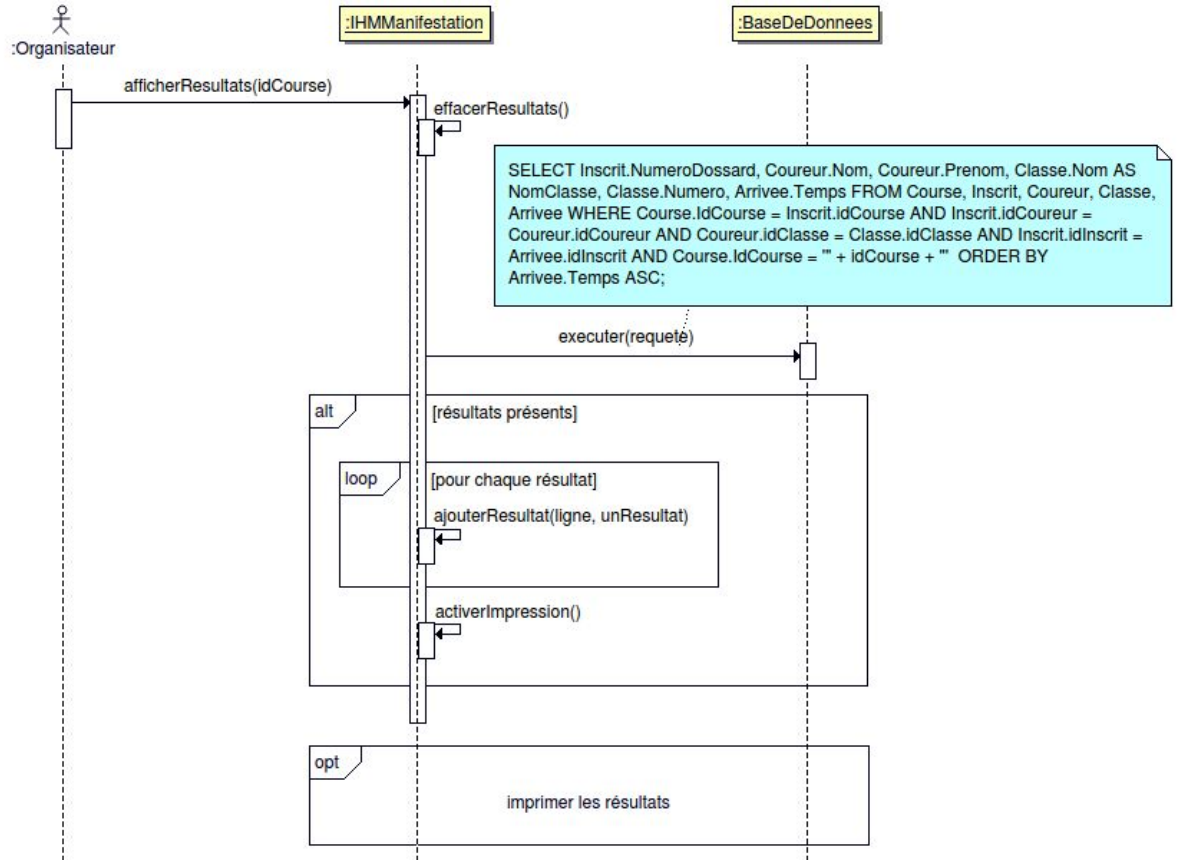


1 : Liste déroulante des manifestations 2 : Liste déroulante des courses
3 : Bouton de suppression

Grâce à ces fonctionnalitées il est aisément possible pour l'organisateur de gérer les manifestations et les courses. Lorsque les manifestations et courses sont terminées, il a la possibilité de rejoindre l’onglet “Inscription” à l’aide du bouton Inscrire.

Cas d'utilisation : publier les résultats

Scénario de l'affichage des résultats



Pour la publication des résultats sur l'application Gestion-Cross les méthodes appelées sont :

- La méthode effacerResultats() est appelé afin de vider le tableau des résultats potentiellement déjà présent.

```
void IHMManifestation::effacerResultats()
{
        int nb = ui->tableWidgetResultats->rowCount();

        // on efface les lignes du tableau une par une
        for(int i = 0; i < nb; i++)
        {
                ui->tableWidgetResultats->removeRow(0);
        }

        ui->boutonImprimerResultats->setEnabled(false);
}
```

La déclaration d'une variable "nb" de type *int*.
La boucle "for" pour ajouter le nombre de lignes nécessaire dans le tableau.
Rend indisponible le bouton "Imprimer".

- La méthode afficherResultats() est appelé afin d'insérer les résultats dans le tableau.

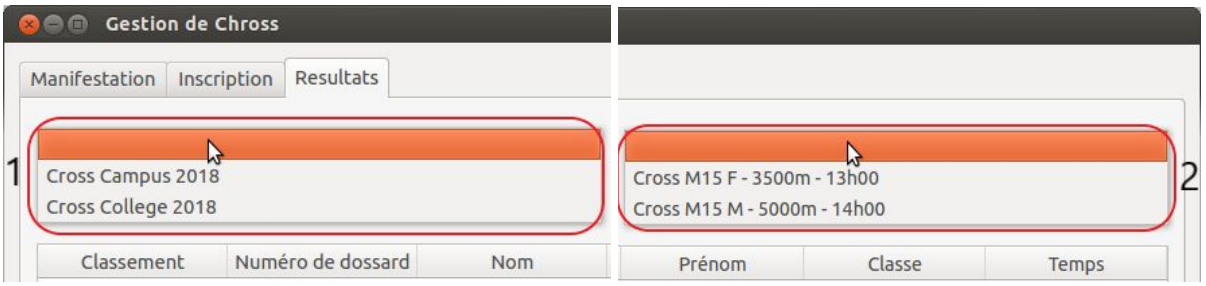
```
void IHMManifestation::afficherResultats(QString idCourse)
{
        // On commence par effacer le contenu précédent
        effacerResultats();

        // Récupérer les résultats d'une course (Requête SQL)
        QVector<QStringList> resultats;
        QString requete = "SELECT Inscrit.NumeroDossard, Coureur.Nom, Coureur.Prenom, Classe.Nom AS NomClasse, Classe.Numero,
Arrivee.Temps FROM Course, Inscrit, Coureur, Classe, Arrivee WHERE Course.IdCourse = Inscrit.idCourse AND Inscrit.idCoureur = Coureur.idCoureur
AND Coureur.idClasse = Classe.idClasse AND Inscrit.idInscrit = Arrivee.idInscrit AND Course.IdCourse = '" + idCourse + "'  ORDER BY Arrivee.Temps
ASC;";
        bool retour = bd->recuperer(requete, resultats);
        if(retour != false)
        {
                QStringList unResultat;

                // Ajouter chaque élément dans le TableWidget
                for(int ligne = 0; ligne < resultats.size(); ligne++)
                {
                        unResultat = resultats.at(ligne);

                        ajouterResultat(ligne, unResultat);
                }

                activerImpression();
        }
}
```

L'appel de la méthode "effacerResultats" avant d'insérer les résultats dans le tableau.
Déclare une liste de type *vector* appelé "resultats".
La création de la variable requete qui contient le code SQL pour une sélection.
L'appel de la méthode recuperer() pour récupérer des information de l'exécution de la requête et les informations des résultats. Elle retourne true si la requête a été exécutée avec succès sinon false.
La condition "if" pour vérifier la présence de résultats pour les insérer dans le tableau.
Déclare la liste "unResultat".
Boucle "for" qui ajoute les résultats dans le tableau.
Exécute la méthode "activerImpression()".

- La méthode activerImpression() est appelé afin de rendre le bouton "Imprimer" accessible.

```
void IHMManifestation::activerImpression()
{
        ui->boutonImprimerResultats->setEnabled(true);
}
```

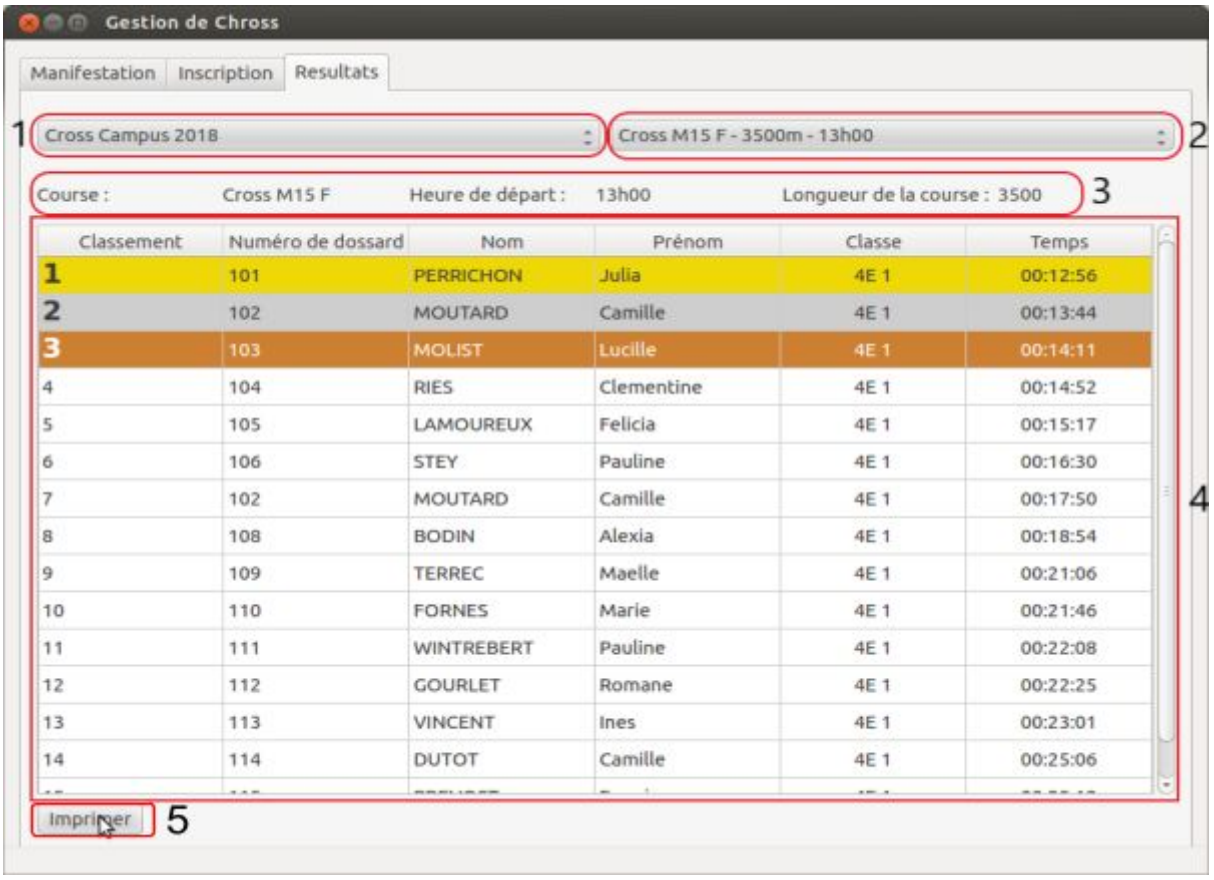
Pour l'affichage des résultats l'organisateur doit sélectionner l'onglet "Resultats". Cet onglet permet à l'organisateur de visionner les résultats d'une course qui s'est terminée, et de pouvoir, s'il le souhaite, les imprimer sous format papier ou format PDF. Pour y parvenir il doit sélectionner, dans la liste déroulante des manifestations, la manifestation, puis dans la liste déroulante des courses, la course qu'il souhaite visionner.



1 : Liste déroulante des manifestations

2 : Liste déroulante des courses

Une fois la course sélectionnée, le nom de la course saisi, l'heure de départ et la longueur de la course s'affichent au dessus du tableau de résultats. Le tableau classe les coureurs par ordre d'arrivée. Les 3 premières personnes arrivées sont affichées en couleur (Or, Argent, Bronze), les suivantes dans un affichage classique. Le tableau des résultats indique : le classement, le numéro de dossard, le nom, le prénom, la classe et le temps des élèves.

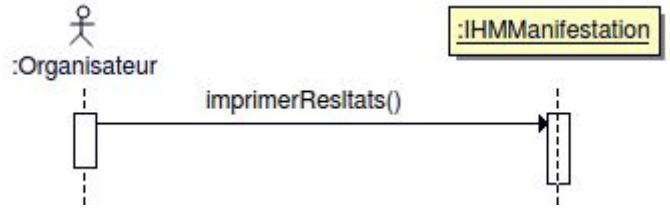


Classement	Numéro de dossard	Nom	Prénom	Classe	Temps
1	101	PERRICHON	Julia	4E 1	00:12:56
2	102	MOUTARD	Camille	4E 1	00:13:44
3	103	MOLIST	Lucille	4E 1	00:14:11
4	104	RIES	Clementine	4E 1	00:14:52
5	105	LAMOUREUX	Felicia	4E 1	00:15:17
6	106	STEY	Pauline	4E 1	00:16:30
7	102	MOUTARD	Camille	4E 1	00:17:50
8	108	BODIN	Alexia	4E 1	00:18:54
9	109	TERREC	Maelle	4E 1	00:21:06
10	110	FORNES	Marie	4E 1	00:21:46
11	111	WINTREBERT	Pauline	4E 1	00:22:08
12	112	GOURLET	Romane	4E 1	00:22:25
13	113	VINCENT	Ines	4E 1	00:23:01
14	114	DUTOT	Camille	4E 1	00:25:06

1 : Liste déroulante des manifestations
2 : Liste déroulante des courses
3 : Affichage des paramètres de la course
4 : Tableau de résultats
5 : Bouton d'impression

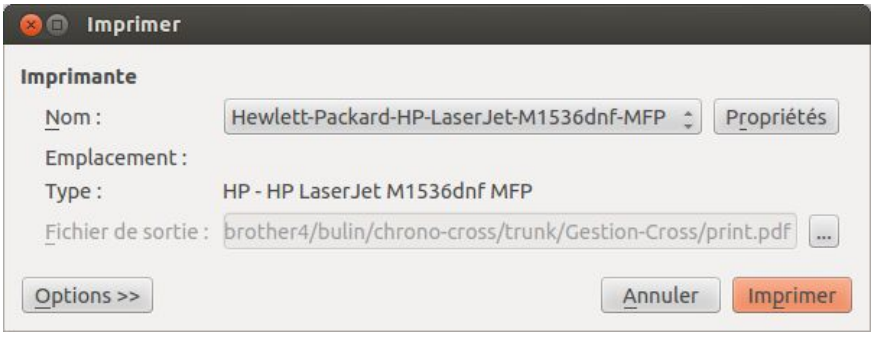
Cas d'utilisation : imprimer les résultats

Scénario d'impression



La méthode imprimerResultats() est appelé afin d'imprimer le tableau de résultats.

Quand les résultats de la course sont affichés il est possible de les imprimer sous différents formats (papier, PDF) grâce à une fenêtre de dialogue.



Exemple d'impression en format PDF :

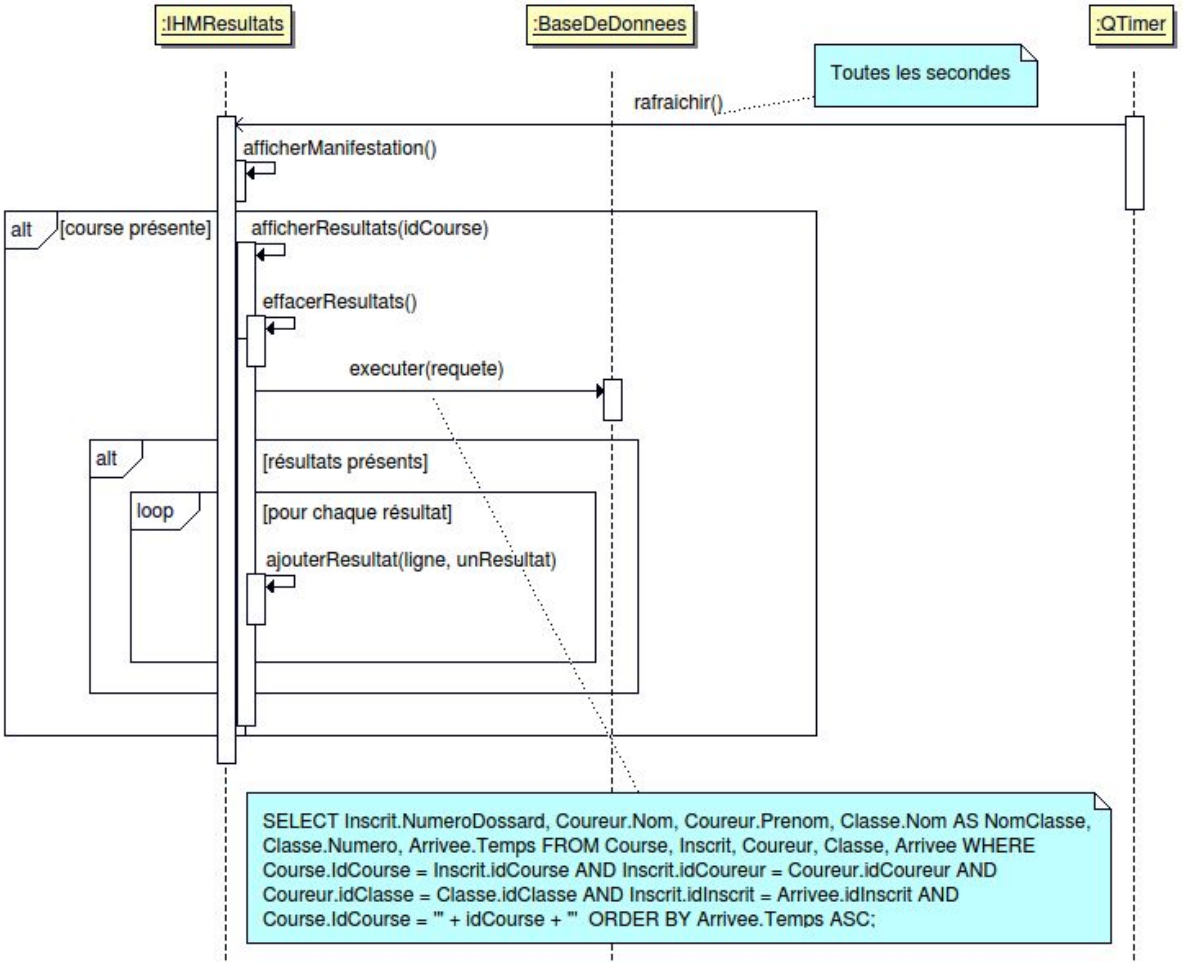
Manifestation : Cross Campus 2018 - Date : 24-05-2018

Course : Cross M15 F - Heure de départ : 11h00 - Longueur de la course : 3500 m

Classement	Numéro de dossard	Nom	Prénom	Classe	Temps
1	101	PERRICHON	Julia	4E 1	00:12:56
2	102	MOUTARD	Camille	4E 1	00:13:44
3	103	MOLIST	Lucille	4E 1	00:14:11
4	104	RIES	Clementine	4E 1	00:14:52
5	105	LAMOUREUX	Felicia	4E 1	00:15:17
6	106	STEY	Pauline	4E 1	00:16:30
7	107	BIRE-HESLOUIS	Maele	4E 1	00:17:50
8	108	BODIN	Alexia	4E 1	00:18:54
9	109	TERREC	Maelle	4E 1	00:21:06
10	110	FORNES	Marie	4E 1	00:21:46
11	111	WINTREBERT	Pauline	4E 1	00:22:08
12	112	GOURLET	Romane	4E 1	00:22:25
13	113	VINCENT	Ines	4E 1	00:23:01
14	114	DUTOT	Camille	4E 1	00:25:06
15	115	PREVOST	Emmie	4E 1	00:29:12

L'application "Resultats-Cross"

Cas d'utilisation : afficher les résultats



Le code qui affiche les résultats sur cet application est similaire au code page 27 sans les parties relatives à l'impression (effacerResultats(), ajouterResultats()).

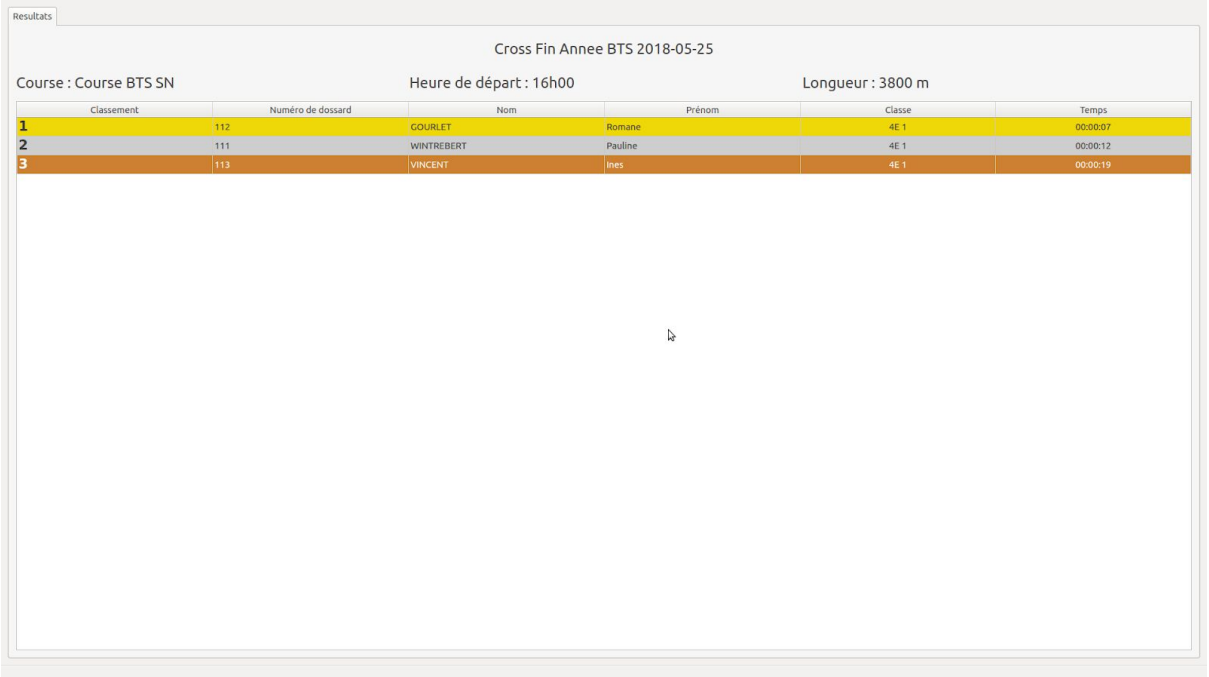
La méthode supplémentaire est la méthode rafraichir()

```
void IHMResultats::rafraichir()
{
        afficherManifestation();

        // Récupérer et affiche les résultats d'une course (Requête SQL)
        if(!course.isEmpty())
        {
                QString idCourse = course.at(COURSE_ID_COURSE);
                afficherResultats(idCourse);
        }
}
```

Appel de la méthode afficherManifestation().
Condition "if" pour vérifier la présence d'une course, dans ce cas elle affiche les résultats.

Cette IHM est en plein écran. Elle affiche les paramètres de la manifestation et la course en cours sur un grand écran TV.

Le tableau comporte les mêmes colonnes que l'onglet "Resultats" de l'IHM de Gestion-Manifestation (Classement, Numéro de dossard, Nom, Prénom, Classe, Temps).



L'application s'actualise toute les secondes et affiche les informations de la manifestation et la course la plus récente sur l'écran télé. Une fois la course terminé les classements s'affichent. En voici un exemple ci-dessus.

Prise en main de commandes utiles pour l'utilisation de la carte Raspberry Pi

Mon professeur a mis à ma disposition une carte Raspberry Pi, un écran d'ordinateur et un clavier me permettant de commencer à utiliser la carte. Nous avons eu besoin d'installer Raspbian qui est un OS de type Debian pour la RaspberryPi. Puis nous avons paramétré la carte, téléchargé Qt et la bibliothèque MySQL nécessaire au programme et le client subversion pour accéder au dépôt du projet.

```
$ sudo raspi-config

$ apt-get qt4-dev-tools
$ apt-get qt-default
$ apt-get libqt4-sql-mysql

$ apt-get subversion
```

La Raspberry Pi doit être configurée en mode *Kiosk*.

Recette

Tests	Oui	Non
La création d'une manifestation est possible	**X**	
La création des courses pour une manifestation est possible	**X**	
L'affichage des informations pendant une course est fonctionnel	**X**	
L'affichage du classement d'une course est fonctionnel	**X**	
L'impression des résultats est possible	**X**	

Bilan :

Ce projet m'a permis d'utiliser l'environnement de développement Qt, dans le langage de programmation C++. Cela m'a permis aussi d'approcher des attentes d'un milieu professionnel par les limites de temps, le travail en équipe et les contraintes attendu par le client.

PROJET CHRONO-CROSS

REVUE FINALE

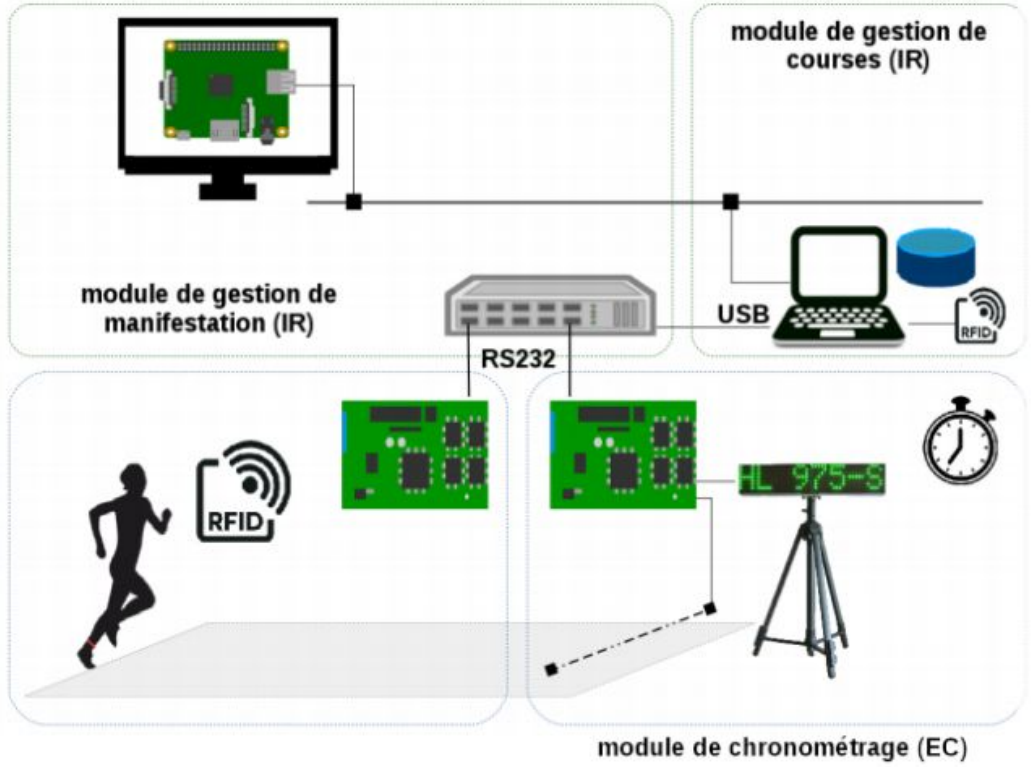
version 1.0

Étudiant n°3 : Pellizzoni Corentin (IR)

Sommaire

Présentation générale

Le projet Chrono-Cross consiste à mettre en place une automatisation presque complète d'une course à pied ou d'un cross. Avec une interface graphique il permet de configurer et d'afficher les résultats de la course en détectant les arrivées par une cellule infrarouge.



Présentation individuelle

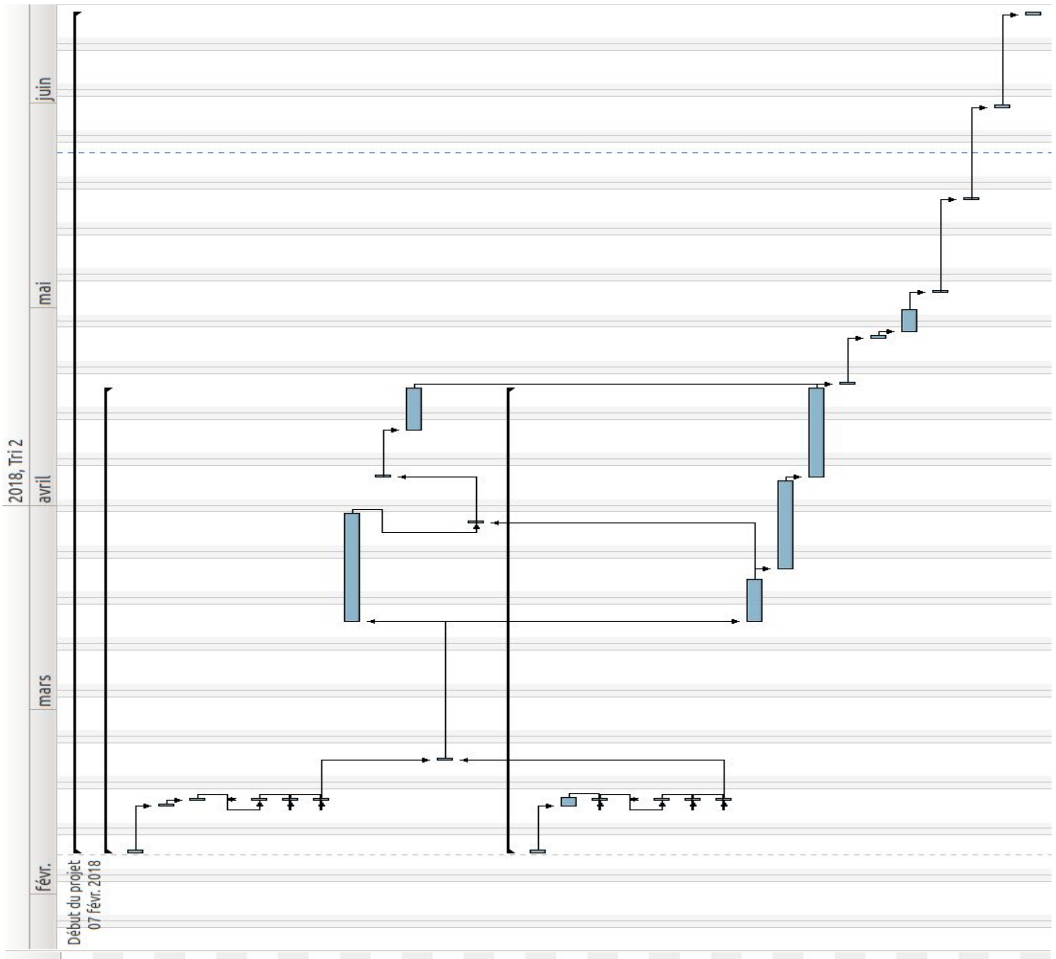
Ma partie consiste dans un premier temps à réaliser une application permettant au client de pouvoir gérer une manifestation sportive dans notre cas le projet chrono-cross.

Cette application demande à l'organisateur d'inscrire des coureurs à une course. Pour cela il faut entrer le numéro de dossard de l'élève, le nom de l'élève, son prénom, sa classe puis la catégorie dans laquelle l'élève a choisi de participer.

Pour ma part, je dois assurer la saisie des données entrées sur l'application, l'enregistrement dans une base de données et l'affichage dans l'IHM. Pour la base de données, je dois écrire toute les requêtes SQL. Je dois aussi récupérer les temps des coureurs à l'aide du module de chronométrage fourni par l'étudiant EC et les enregistrer.

Planification

Le diagramme de Gantt permet de visualiser et d'établir chronologiquement les tâches effectuées lors de la réalisation du projet.



TPÉ	Nom
1	**Projet**
1.1	**Gestion de la maniferstion**
1.1.1	Prise en compte du cahier des charges
1.1.2	Création d'une IHM de saisie
1.1.3	Modélisation d'une IHM
1.1.4	**Diagrammes UML**
1.1.4.1	Diagramme de cas d'utilisation
1.1.4.2	Diagramme de séquence
1.1.4.3	Diagramme de classe
1.1.5	Prise en main de la Raspberry Pi
1.1.6	Etablir la liaison réseau
1.1.7	Création du code source
1.2	Revue n°0
1.3	Revue n°2
1.4	**Gestion de courses**
1.4.1	Prise en compte du cahier des charges
1.4.2	Création de l'IHM
1.4.3	Modélisation d'une IHM
1.4.4	**Diagrammes UML**
1.4.4.1	Diagramme de cas d'utilisation
1.4.4.2	Diagramme de séquence
1.4.4.3	Diagramme de classe
1.4.5	Etablir la liaison (série, réseau)
1.4.6	Création des bases de données
1.4.7	Création du code source
1.5	Réalisation des test unitaires
1.6	Intégration de l'application complète
1.7	Rédiger le dossier technique et les documents relatif au module
1.8	Rédaction d'un guide de mise en route
1.9	Revue n°3
1.10	Remise des dossiers
1.11	Soutenance finale

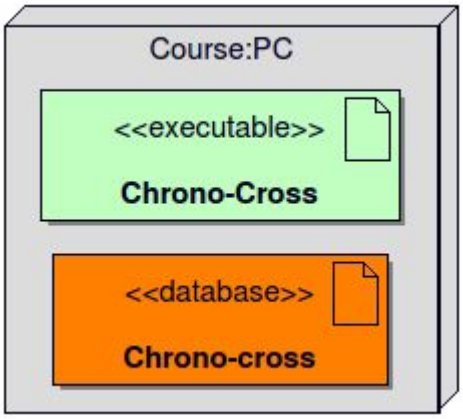
Ressources logicielles

Système d'exploitation du PC « Course »	GNU Linux
Environnement de développement (IR)	Qt Creator et Qt Designer
API GUI PC « Course »	Qt 4.8
Compilateurs	GNU g++ for Linux
Système de gestion de bases de données relationnelles	MySQL
Gestion et administration de bases de données	phpMyAdmin
Atelier de génie logiciel (IR)	bouml 7.5
Logiciel de gestion de versions (IR)	Client subversion
Générateurs de documentation (IR)	Doxygen version 1.8.11
Gestionnaire de projet (IR)	Planner

Travail personnel

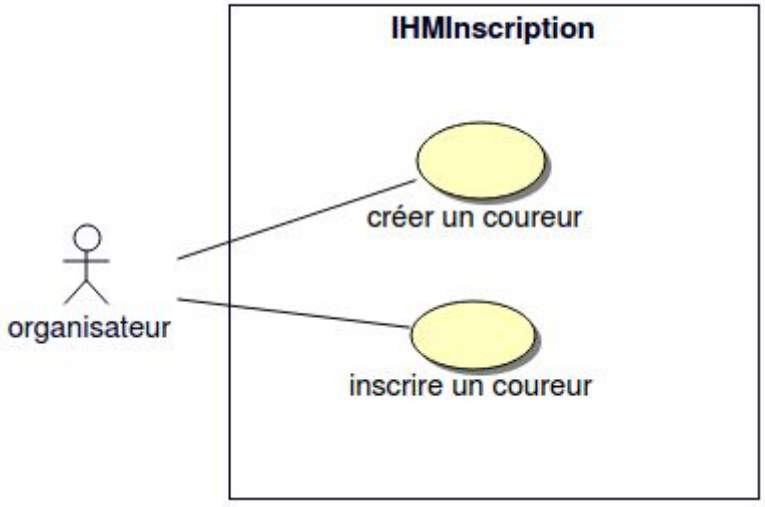
Application Gestion-Cross

Diagramme de déploiement



Le PC « Course » héberge le serveur de base de données (MySQL)

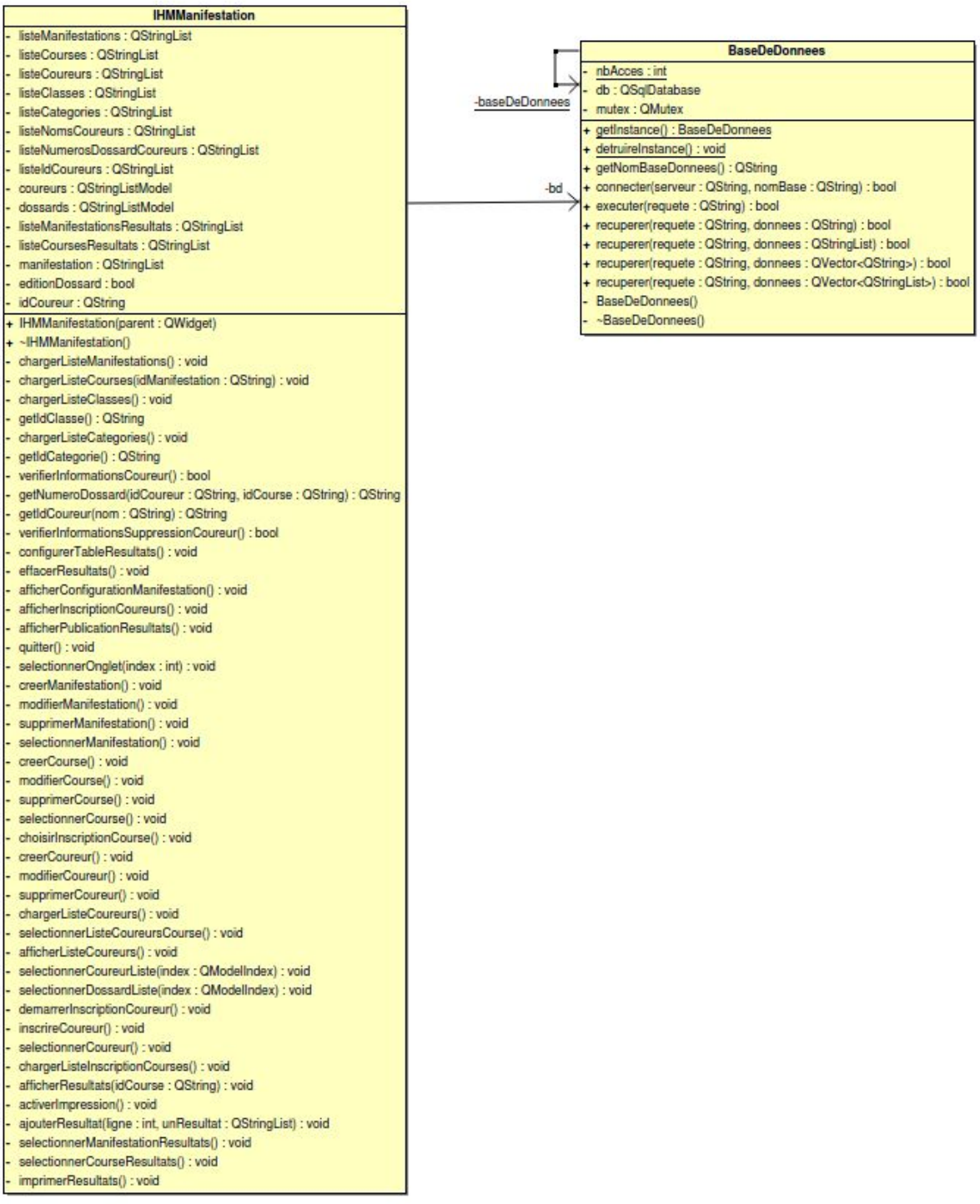
Diagramme de cas d'utilisation Gestion-Cross



L'acteur humain de ce système est : "Organisateur". Il prépare les coureurs et inscrit les coureurs.

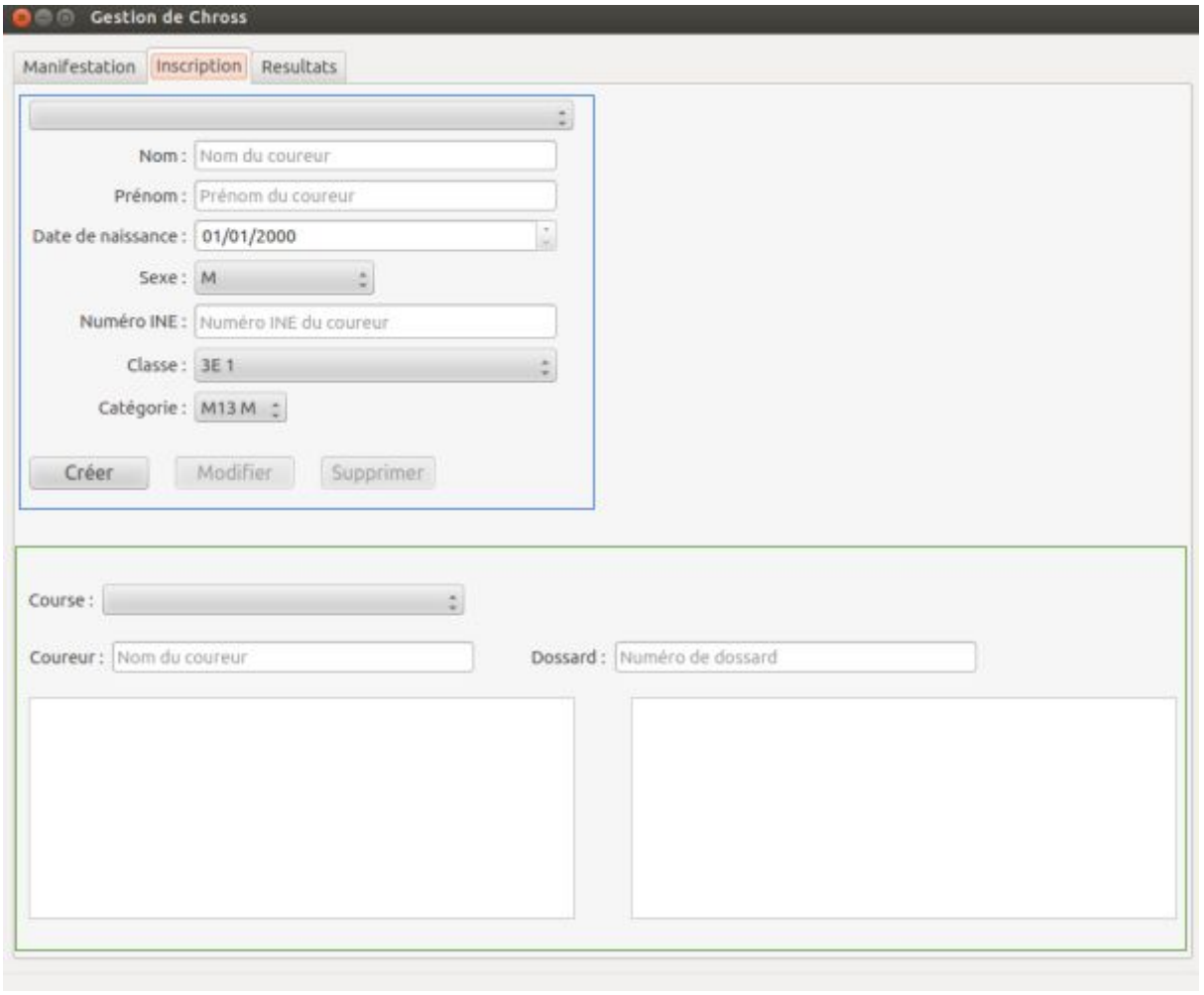
Diagramme de classe

Ce diagramme de classe retranscrit tous les attributs et toutes les méthodes de l'application Gestion-Cross.



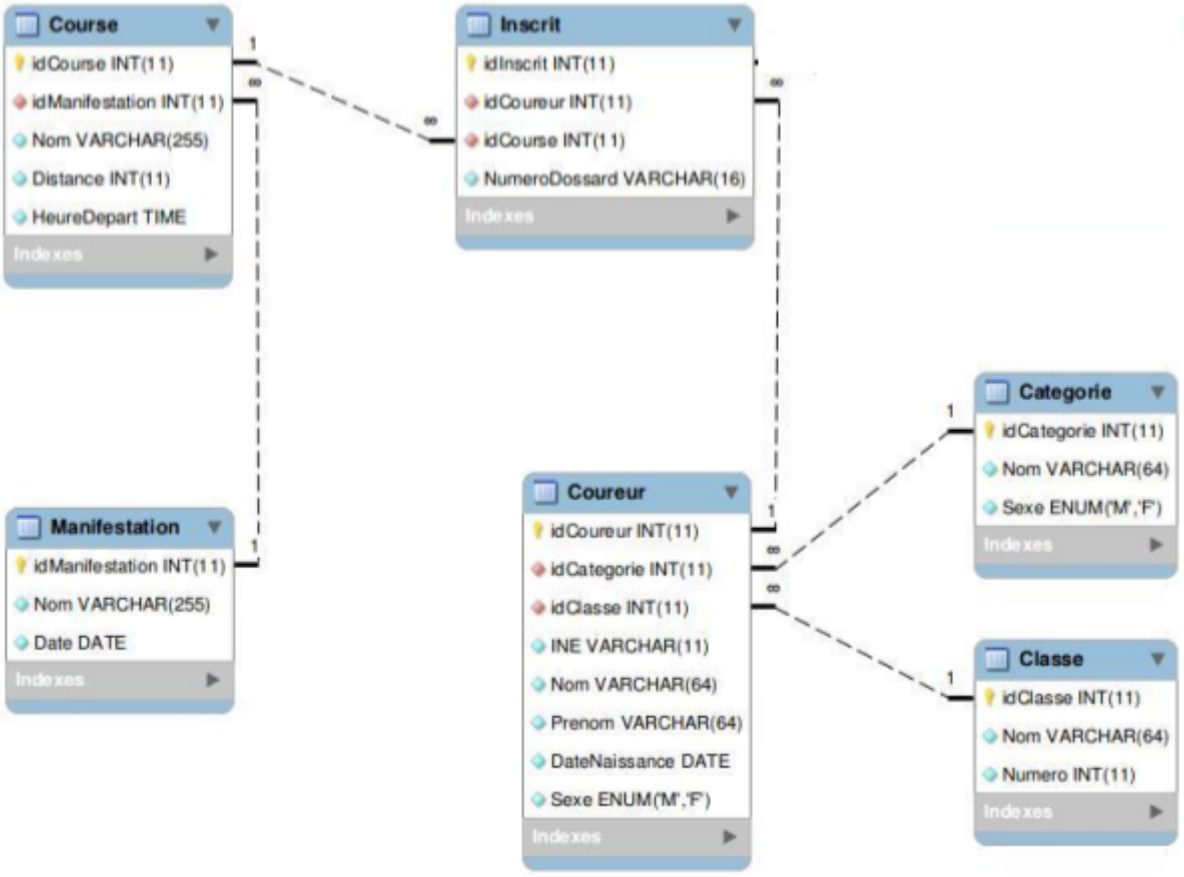
IHM

Ici l'application GestionCross avec l'onglet "Inscription" permettant la création et l'inscription d'un coureur à une course :



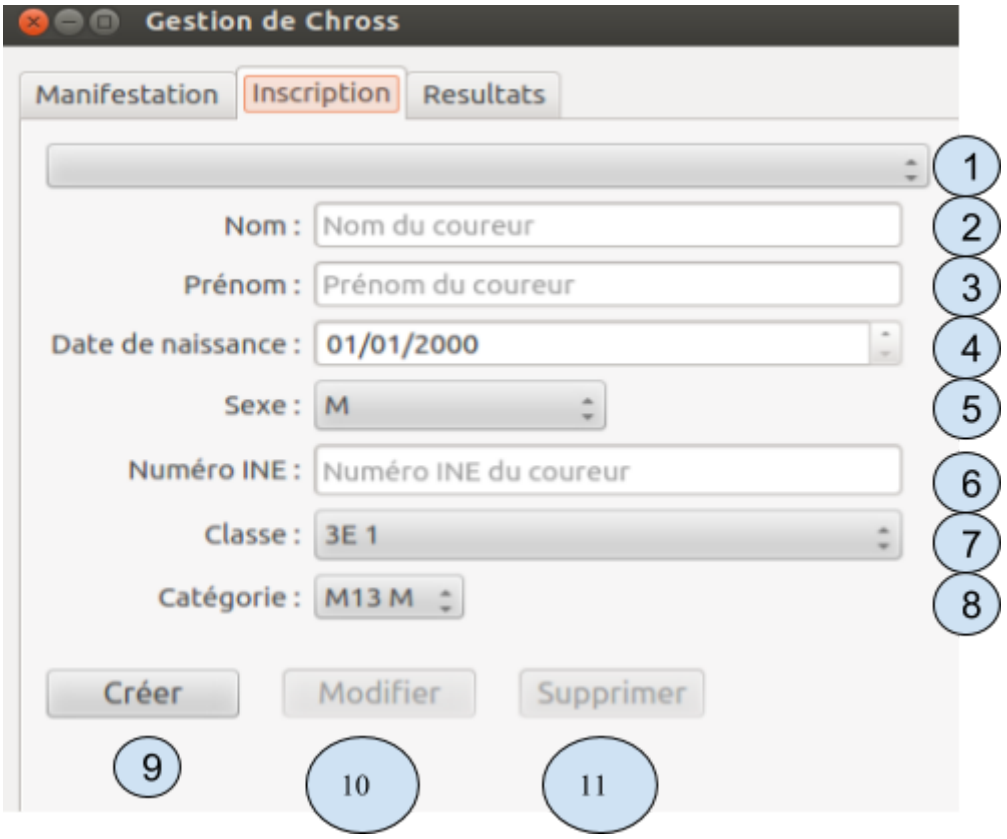
L'application Gestion-Cross a deux utilités la première partie sert à la création d'un coureur ainsi qu'à sa modification ou sa suppression. La deuxième partie sert à inscrire un coureur à une course en lui associant un numéro de dossard.

Base de données



Cette base de données est composée de 6 tables, Manifestation, Course, Inscrit, Coureur, Catégorie et Classe pour l'application Gestion-Cross, la première partie de l'application va utiliser les tables Coureur, Catégorie et Classe et la deuxième partie va utiliser Course, Inscrit et Coureur.

Cas d'utilisation : créer un coureur

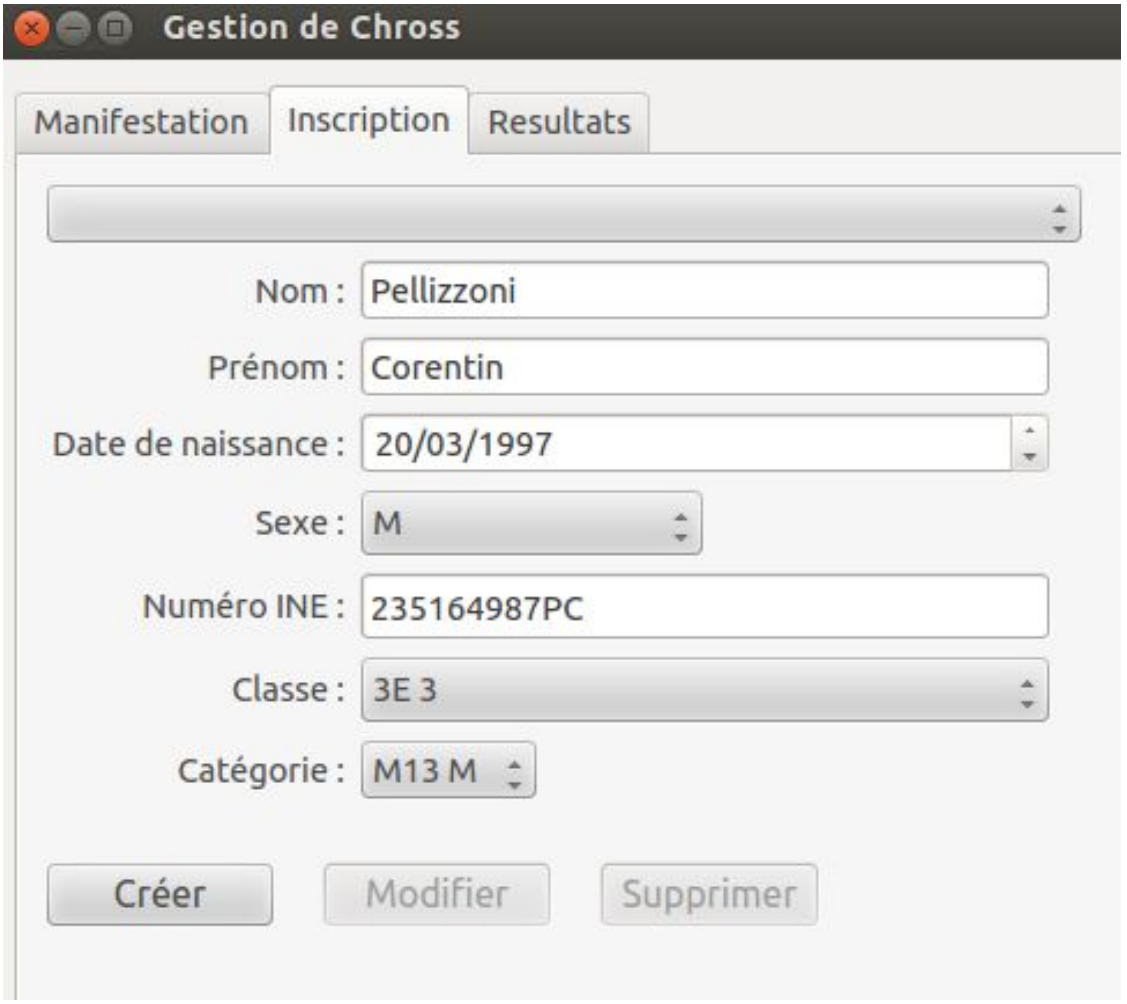


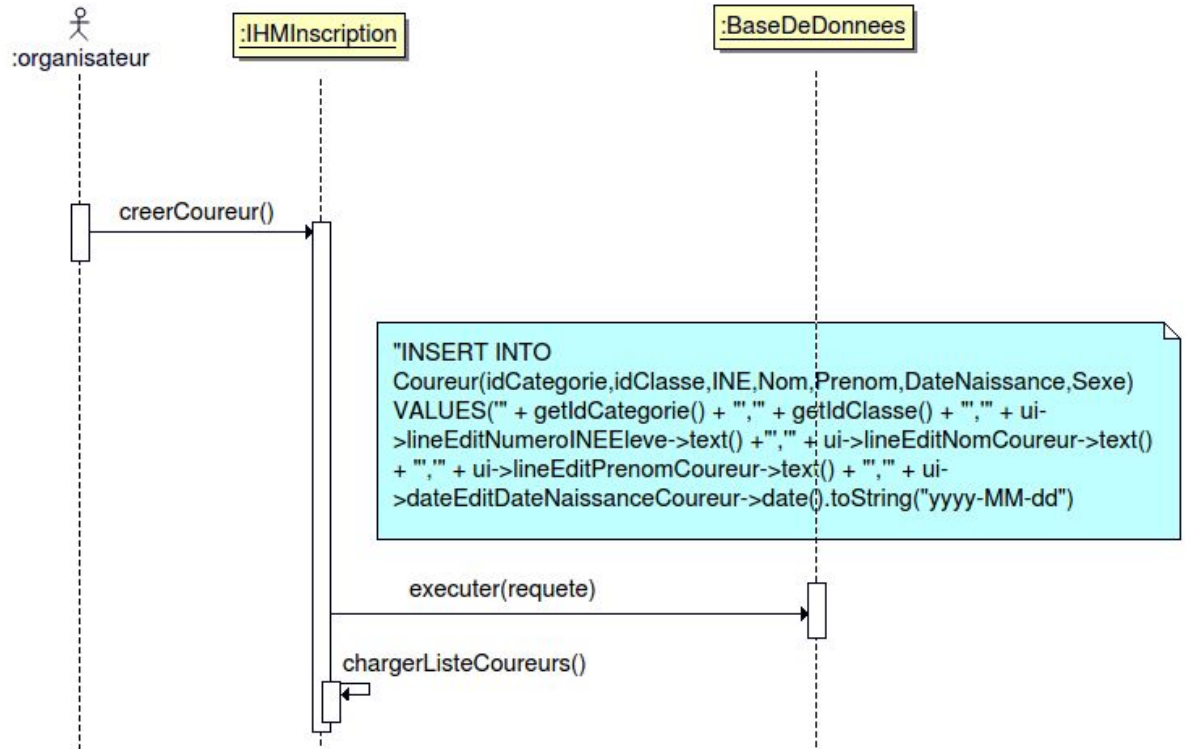
1 - Il s'agit d'une liste déroulante avec tous les coureurs préalablement enregistrés pour pouvoir poursuivre sur la procédure de modification ou de suppression .
2- Nom du coureur .
3- Prénom du coureur.
4- Date de naissance du coureur
5- Sexe du coureur (option préalablement configurer pour deux choix unique Mâle ou Femelle)
6- Numéro INE (Identifiant National Etudiant) composé de 9 chiffres et de deux lettres en majuscules.
7-Classe du coureur.
8-Catégorie du coureur.
9-Bouton de création du coureur.
10-Bouton de modification d'un coureur préalablement sélectionné dans la liste (1).
11-Bouton de suppression d'un coureur préalablement sélectionné dans la liste (1).

La première partie va permettre de configurer toutes les données pour l'inscription d'un coureur qui seront ensuite envoyés dans une base de données dans laquelle sera stockées toutes les données .Ces dernières sont stockées grâce à une requête SQL tant pour la création, modification ou suppression.

Scénario Création d'un coureur

Pour la création d'un coureur, il faut remplir tous les champs nécessaires par la suite appuyer sur le bouton Créer. Une requête SQL va enregistrer toutes les infos rentrées dans une base de données Chrono-Cross.





Vue de la base de données : nous voyons bien que le coureur Pellizzoni n'existe pas dans la base de données.

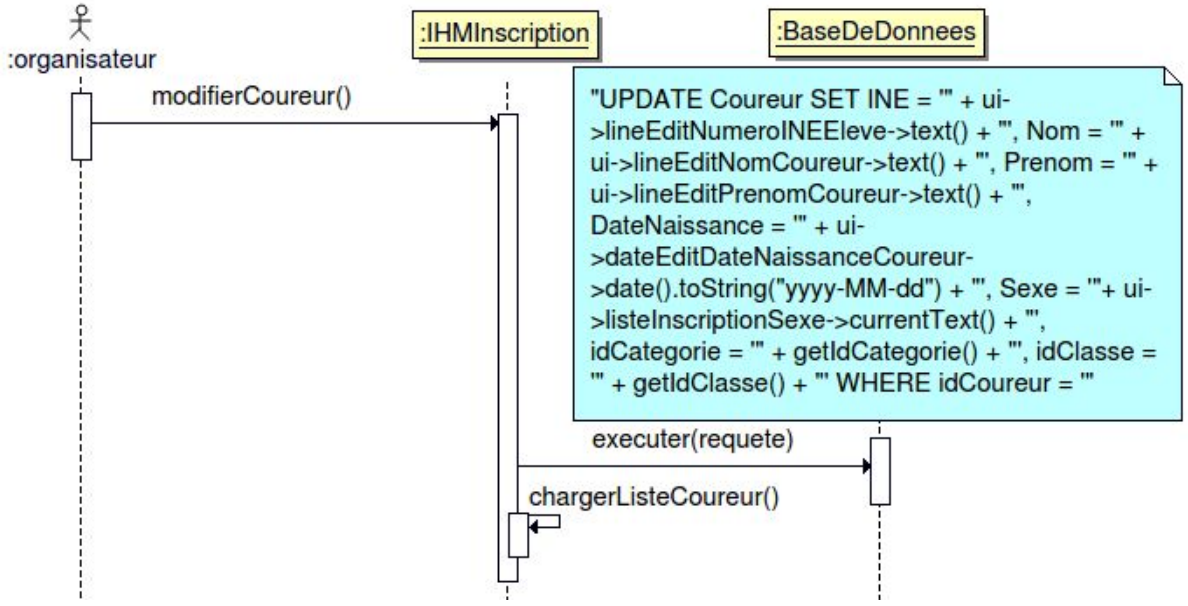
2	4	7	123456789BB	MOUTARD	Camille	2004-01-08	F
16	3	7	234567891AA	PELIOT	Julien	2004-02-16	M
1	4	7	123456789AA	PERRICHON	Julia	2004-04-16	F
29	3	7	234567891NN	PIGNON	Jean	2004-10-20	M
15	4	7	123456789OO	PREVOST	Emmie	2004-01-02	F
4	4	7	123456789DD	RIES	Clementine	2004-06-16	F
19	3	7	234567891DD	RIOUX	Clement	2004-07-07	M

Ici nous voyons que le coureur Pellizzoni existe dans la base de données.

2	4	7	123456789BB	MOUTARD	Camille	2004-01-08	F
16	3	7	234567891AA	PELIOT	Julien	2004-02-16	M
96	1	12	235164987PC	PELLIZZONI	Corentin	1997-03-20	M
1	4	7	123456789AA	PERRICHON	Julia	2004-04-16	F
29	3	7	234567891NN	PIGNON	Jean	2004-10-20	M

Scénario Modification d'un coureur

Pour modifier un coureur, il faut sélectionner le coureur dans la liste déroulante toute les informations vont s'afficher et il faudra modifier la donnée voulant l'être et cliquer sur le bouton modifier, une requête SQL est envoyé à la base de données Chrono-Cross.



La modification sera porté sur la catégorie du coureur .

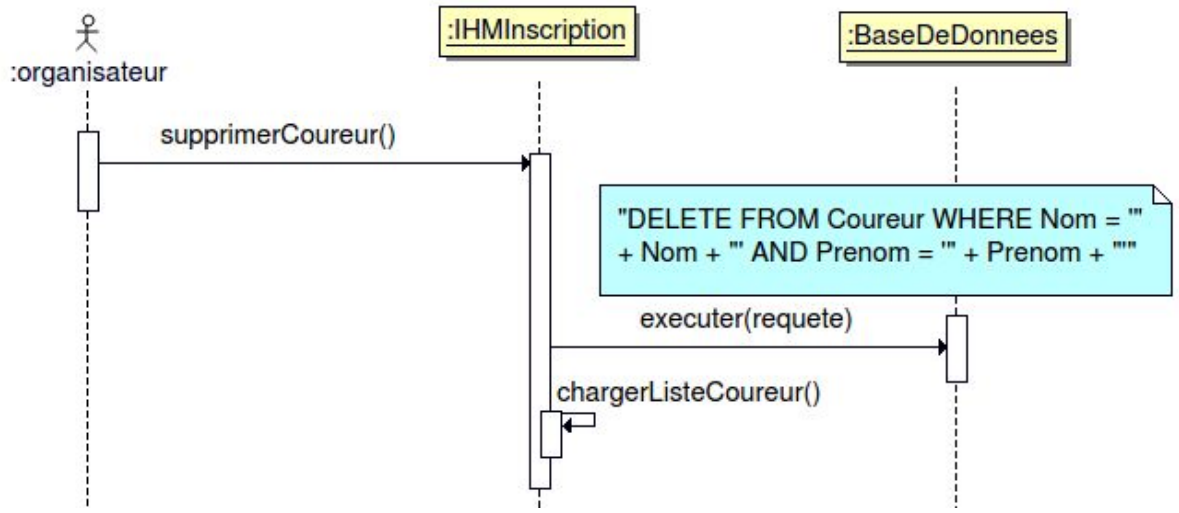
96	5	12	235164987PC	PELLIZZONI	Corentin	1997-03-20	M

Nous pouvons constater que dans la base la catégorie a était modifié.

96	1	12	235164987PC	PELLIZZONI	Corentin	1997-03-20	M

Scénario Supprimer un coureur

Pour supprimer un coureur, il faut sélectionner le coureur dans la liste déroulante toute les informations vont s'afficher et il faudra cliquer sur le bouton supprimer, une requête SQL est envoyé à la base de données Chrono-Cross.

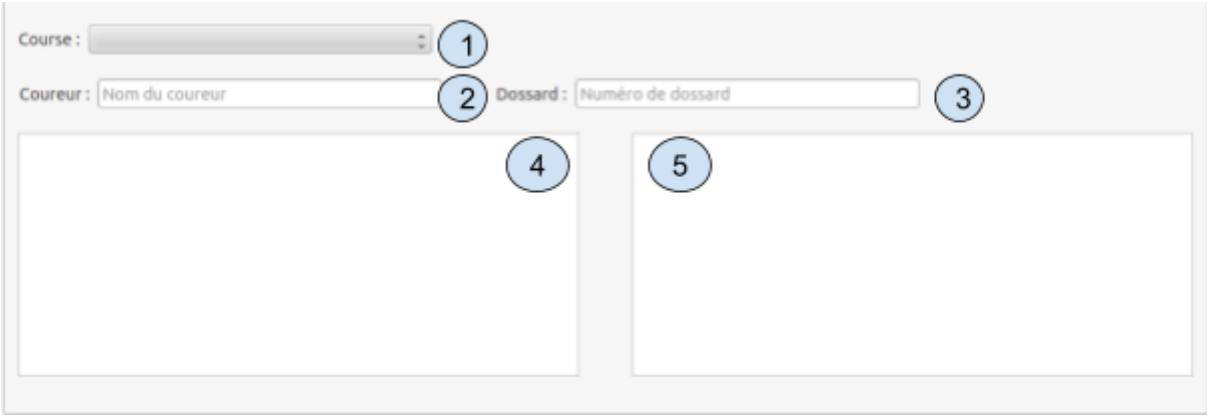


Nous voyons bien que le coureur PELLIZZONI a été supprimé de la base de données.

17	3	7	234567891BB	MOULARD	Charles	2004-03-08	M
2	4	7	123456789BB	MOUTARD	Camille	2004-01-08	F
16	3	7	234567891AA	PELIOT	Julien	2004-02-16	M
1	4	7	123456789AA	PERRICHON	Julia	2004-04-16	F
29	3	7	234567891NN	PIGNON	Jean	2004-10-20	M
15	4	7	1234567890O	PREVOST	Emmie	2004-01-02	F
4	4	7	123456789DD	RIES	Clementine	2004-06-16	F

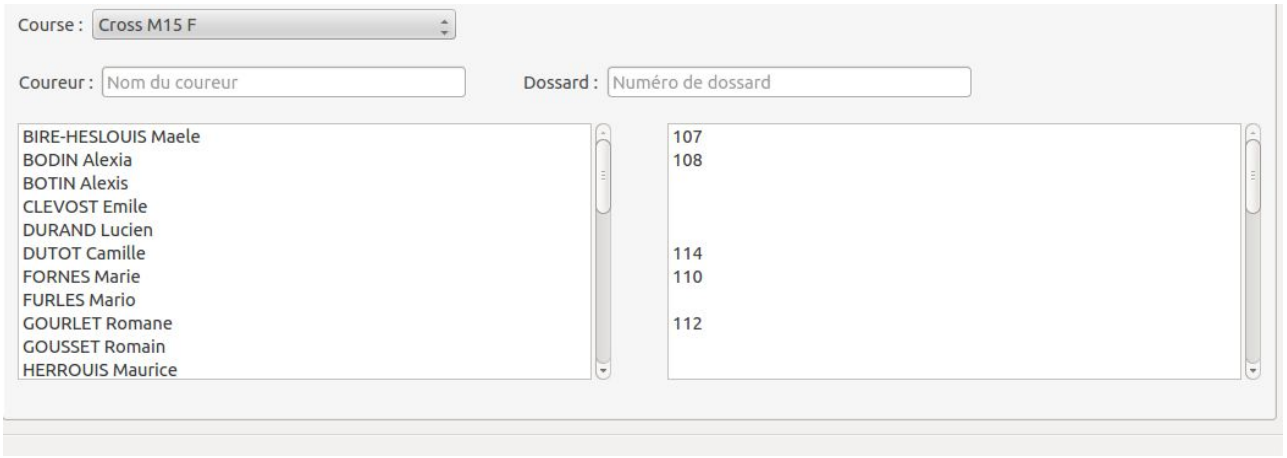
Cas d'utilisation : inscrire un coureur à une course

La deuxième partie va permettre d'inscrire un coureur à une course et de pouvoir lui associer un numéro de dossard pour la course sélectionnée.



1- Liste déroulante avec toutes les courses
2- Nom du coureur
3- Numéro de dossard du coureur
4- Vue de tous les coureurs
5- Vue des numéros de dossard

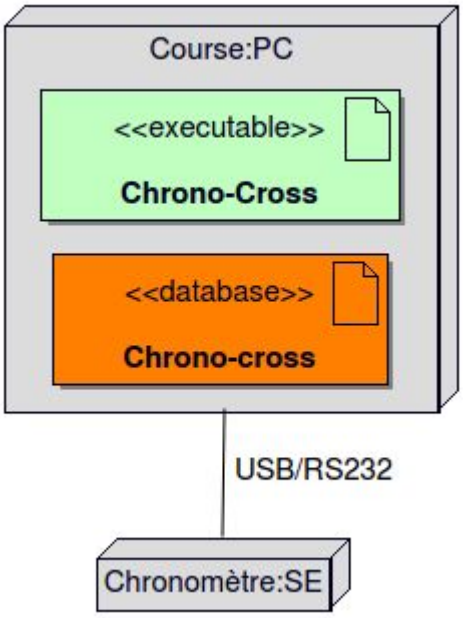
Prenons comme exemple la course moins de 15 ans féminines.



Nous pouvons voir que les participants à cette course sont bien toutes féminines,
Nous constatons que chaque coureur a un numéro de dossard attribué pour cette course.

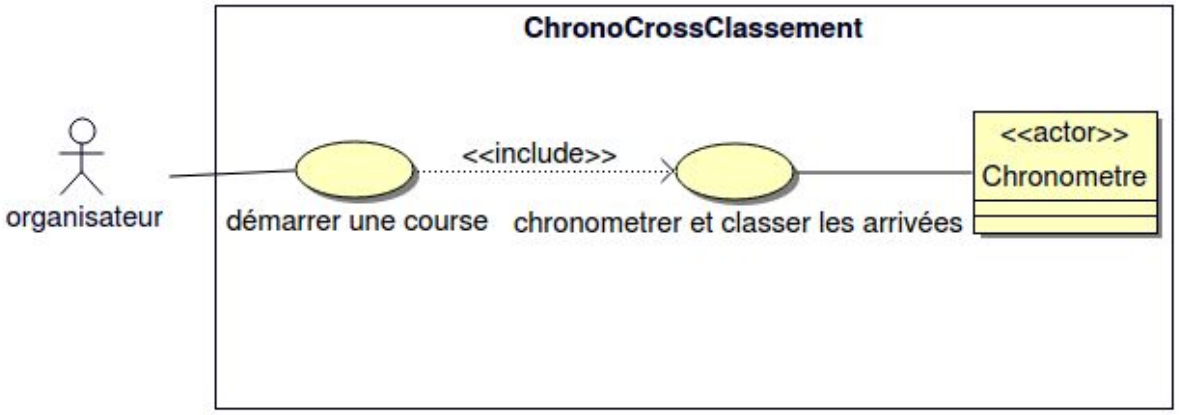
Application ChronoCrossClassement

Diagramme de déploiement



Le PC « Course » héberge le serveur de base de données (MySQL), le système embarqué Chronomètre est relié au PC-Course avec une liaison USB/RS232 , le Chronomètre est géré dans la partie EC.

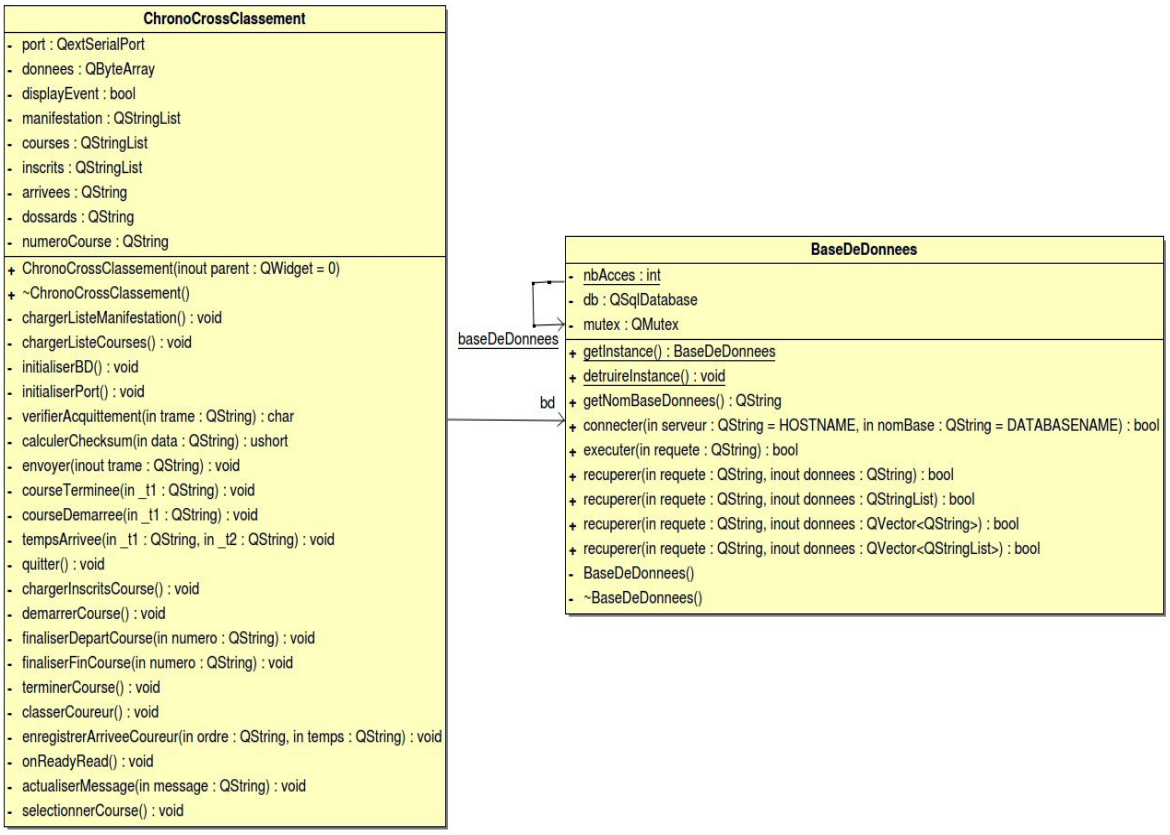
Diagramme de cas d'utilisation ChronoCrossClassement



L'acteur humain de ce système est : "Organisateur". Il démarre la course et classe les arrivées. L'acteur matériel « Chronomètre » interagit avec le système par l'échange de trames qui permettent d'obtenir les temps et l'ordre des arrivées.

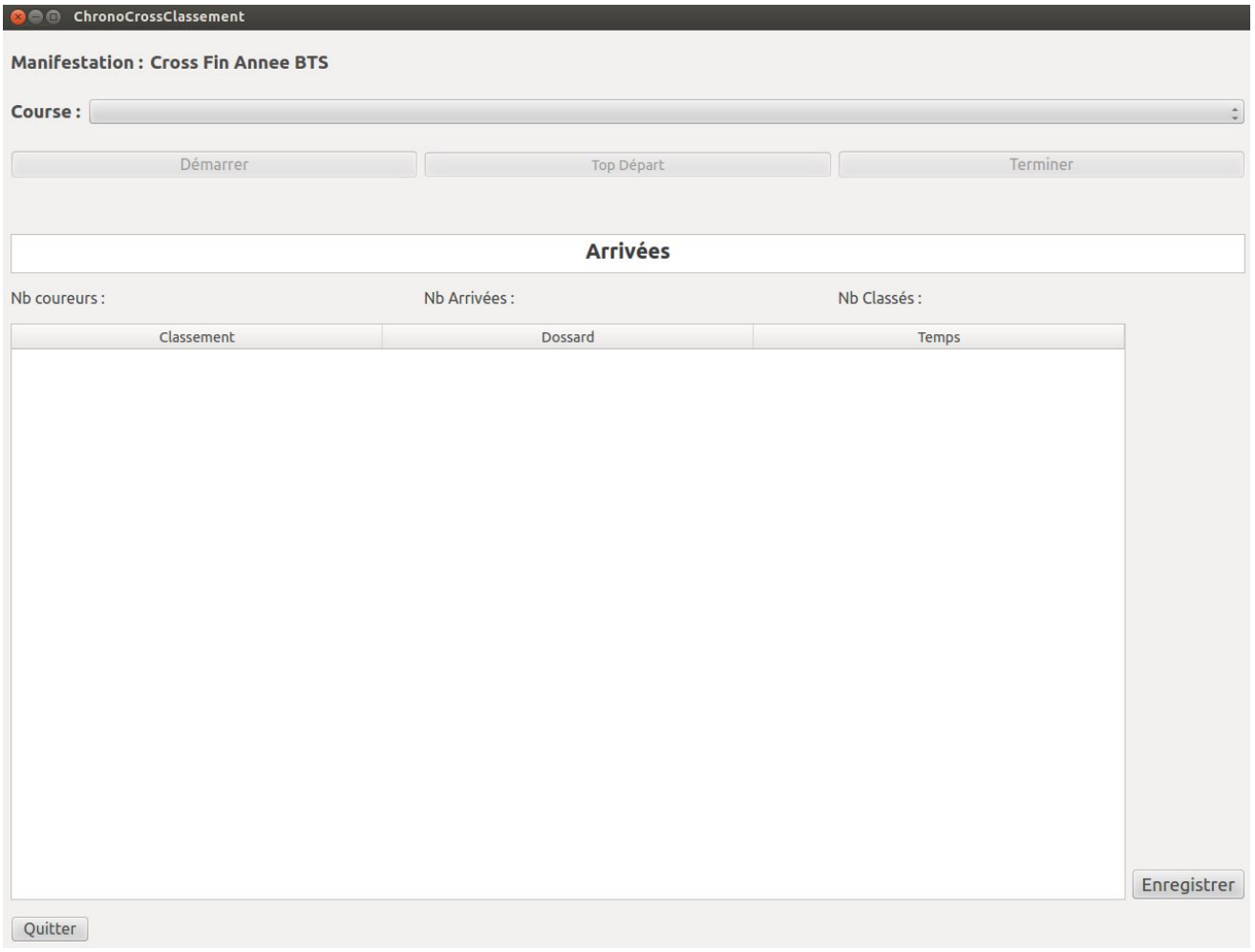
Diagramme de classe

Ce diagramme de classe retranscrit tous les attributs et toutes les méthodes de l'application Gestion-Cross



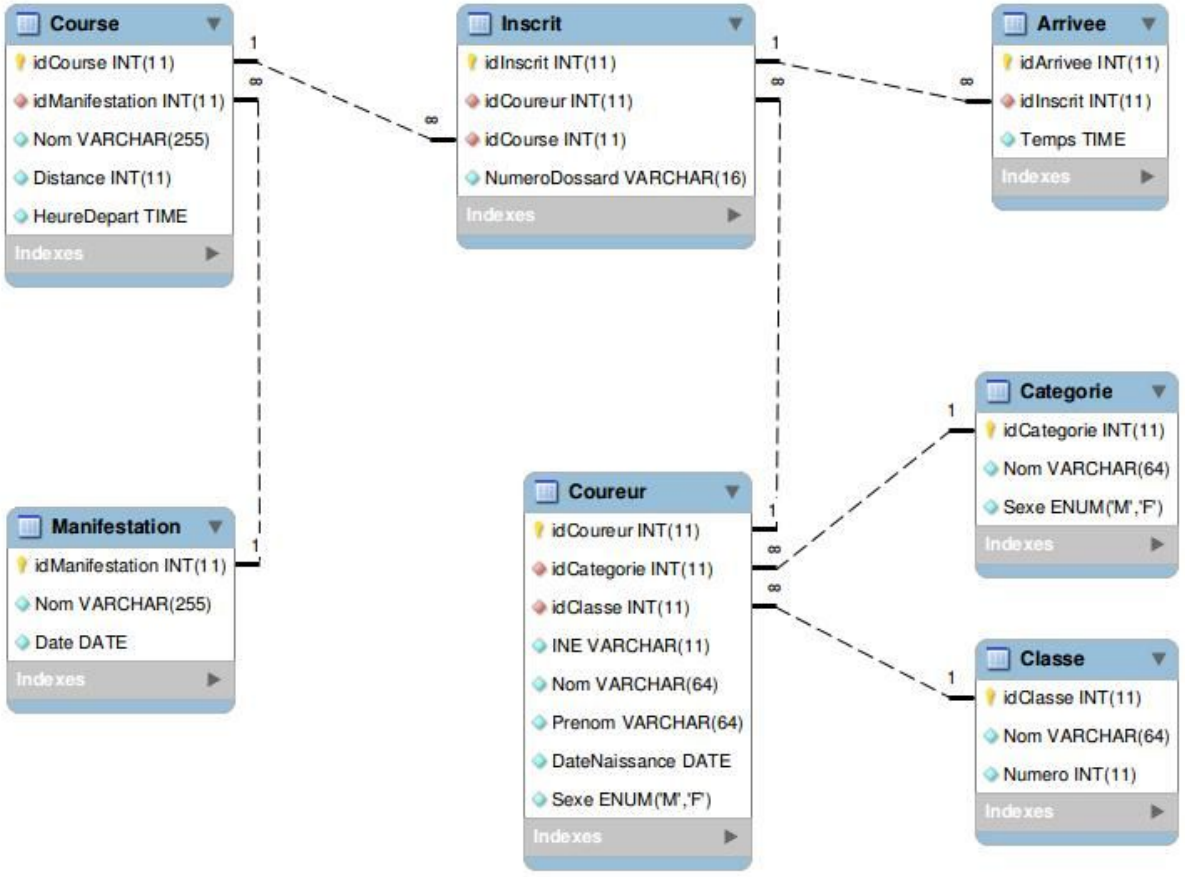
IHM

Ici l'application ChronoCrossClassement permettant de démarrer une course et de classer les arrivées des coureurs :



La première partie sert à sélectionner puis de démarrer une course et la deuxième partie sert à afficher le nombre de coureurs inscrit à la course le nombre d'arrivées et le nombre de coureurs classés, afficher les temps reçu sur un tableau et ainsi pouvoir associer le numéro de dossard avec le temps correspondant.

Base de données



La même base de données va être utilisé mais en ajoutant la table **Arrivee**, ce qui va permettre d'enregistrer les temps d'arrivés des coureurs pour ensuite les attribués au numéro de dossard associés au Coureur.

Module Chronomètre

Pour la partie détection de temps j'utilise un chronomètre TAG heuer hl975. Ce chronomètre utilise le protocole THCOM08, celui qui sera implanté dans le module de l'étudiant EC.

Avec l'aide des documents fournis et d'un programme de test j'ai pu comprendre comment fonctionne le chronomètre tout d'abord j'ai paramétré la liaison entre le chronomètre et le PC avec une liaison USB/RS232. J'ai paramétré par la suite une connexion au port automatique. Une fois tout cela effectué, j'ai testé avec l'outil cutecom.

On utilise l'outil cutecom sur le fichier de périphérique /dev/hl975 et la configuration suivante :
* 9600 bits/s
* 8 bits de données
* 1 bit de stop
* pas de parité

Format

Le format d'une trame est le suivant : `[Data] + TAB + CS16 + CR + LF`

- `Data` : voir les messages décrits ci-dessous
- `TAB` : délimiteur du *Checksum* (code ASCII `0x09`)
- `CS16` : *Checksum* sur 16 bits (4 chiffres codés en hexadécimal)
- `CR + LF` : délimiteur de fin de trame (codes ASCII `0x0D` + `0x0A`)

Le format d'une trame sans *Checksum* sera le suivant : `[Data] + TAB + CR + LF`

Acquittement

Pour chaque trame envoyée vers l'appareil (*Device*), celui-ci répond à la machine émettrice (*Host*) par une trame AK d'acquittement qui a le format suivant : `AK X`

X = 'C' accepted, 'F' rejected, 'R' not supported

Messages

Serial number (ID) : `ID NNNNN`

N = Serial number (0 - 65535)

Exemple : `ID 01145`

Serial number request (#ID) : `#ID`

Serial number + Device type + soft version (SN) : `SN_NNNNN_TTTTT_VVVV`

N = Serial number (0 - 65535)
T = Device type (CP540, HL440, HL940)
V = Software version (example: VA05)

Exemple : `SN 01145 HL975 VA10`

Serial number and device type request (#SN) : `#SN`

Opening of a new Run (OP) : `OP RR TAA XXXXXXXXXXXXXXXXXXX`

R = Run number (1 - 99)
T = 'T' if the added Run is itself the addition of two other Runs '_' otherwise.
A = Added Run (1 - 99)
X = Name of the Timing Mode (Max 19 chars)

Closing of a Run (CL) : `CL RR`

R = Run number (1 - 99)

Time (TN) : `TN SSSS CC HH:MM:SS.FFFFF DDDDD`

S = Sequential number (0 - 9999)
C = Channel number (1 - 99) in case of manual entry (M1 - M4)
H = Hours (0 - 23)
M = Minutes (0 - 59)
S = Seconds (0 - 59)
F = decimal part (0 - 99999)
D = Days (0 - 32767) counting from 01.01.2000

Exemples :

```
TN          1   2       3.71300    365
TN          2   2       8.00500    365
TN          3   2       1:01.48200    365
```

Read a parameters (#RP) : `#RP III`

`I` = parameter ID (000 – 999)

trame = "#RP 003"; // &P 003 XX YY 00:00:00 0

Parameter 003 : Run number

```
#RP 003
&P 003 XX YY HH:MM:SS DDDDD
```

X = Run number (00-99). 00 = no run open
Y = Number of the next Run(00-99).
H = Hours
M = Minutes
S = Seconds
D = Date

Exemple : `&P 003 89 90 00:00:00 0`

Write a parameters (#WP) : `#WP III`

`I` = parameter ID (000 – 999)

Parameter 120 : HL975 Operating mode
#WP 120 X
&P 120 X

X = 0 -> PTB (Base de temps)
X = 5 -> Clock (Horloge)
Exemple :

```
#WP 120 5
&P 120 5
```

Write a command (#WC) : `#WC III`

Command 001 : Close a Run

```
#WC 001
```

Command 007 : Start a new synchro

```
#WC 007 TT HH:MM DD/XX/YY
```

T = Synchro Type
2 -> Manual Synchro

If Manual Synchro is chosen:
H = Hour
M = Minute
D = Day
X = Month
Y = Year

Exemple : `#WC 007 02 00:00 00/00/01`

Command 008 : Trigger a manual input pulse

```
#WC 008 XX
```

X = Input number ('1'-'2' for HL975)

Exemple : `#WC 008 01`

Liste des messages

*Standard messages ID, from **Device to Host** :*

- AK Acknowledge of a received command
- ID Serial number (optionnel)
- SN Serial number + device type + soft version (optionnel)
- OP Opening a Run
- CL Closing a Run
- TN New time
- &P Parameter

*Standard messages ID (commands), from **Host to Device** :*

- #ID Get serial number (optionnel)
- #SN Get serial number + device type (optionnel)
- #RP Read parameter
- #WP Write parameter
- #WC Write Command

Échanges

Démarrage d'une course

Host *Device*

Mode PTB

```
trame = "#WP 120 0"
---------------------------------------------->
```

Synchro manuel

```
trame = "#WC 007 02 00:00 00/00/01"
---------------------------------------------->
```

La synchronisation du temps sera déclenchée par l'entrée numéro 1. Il est possible de la déclencher logiciellement en envoyant la trame :

```
trame = "#WC 008 01"
---------------------------------------------->
   trame = "TS 00:00:00 00/00/01"
<----------------------------------------------
```

Il est aussi possible de demander le numéro de course :

```
trame = "#RP 003"
---------------------------------------------->
trame = "&P 003 89 90 00:00:00 0"
<----------------------------------------------
```

Ici, le numéro de course reçu est le 89. Si le numéro de course est 00, alors la course n'a pas encore été démarrée.

Temps à l'arrivée

Les temps à l'arrivée sont déclenchés par l'entrée numéro 2.

Host *Device*

```
trame = "TN 1 2      3.71300  365"
<----------------------------------------------
```

etc ...

Terminer une course

Host *Device*

Close a Run

```
trame = "#WC 001"
---------------------------------------------->
                  trame = "CL 89"
<----------------------------------------------
```

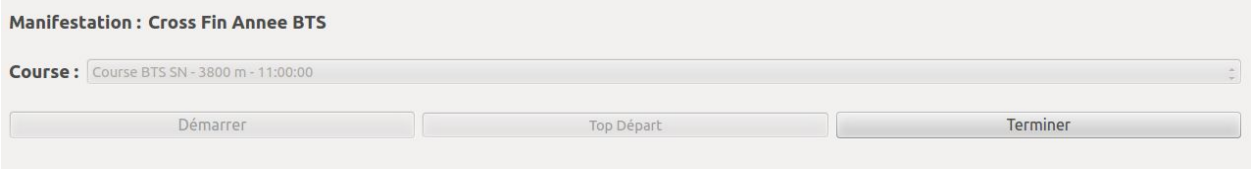
Cas d'utilisation : Démarrer une course

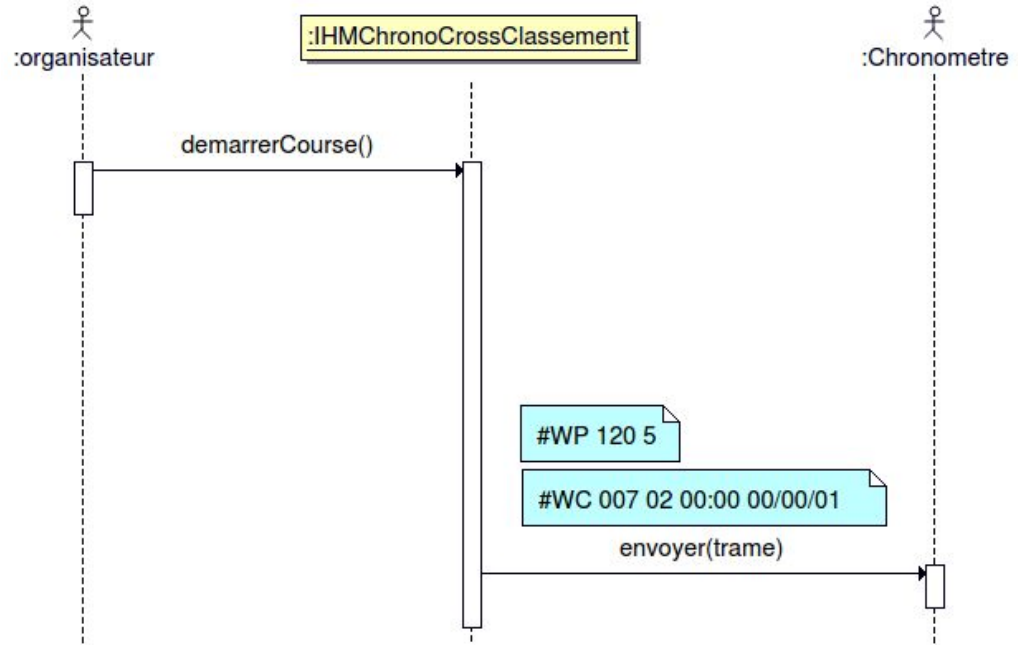
1- Manifestation du jour
2- Liste déroulante avec toutes les courses
3- Démarre la course
4- Démarre la course à partir de l'application
5- Termine la course

La première partie va permettre de sélectionner une course dans la liste déroulante par rapport à la manifestation du jour. toutes les courses sont récupérés dans la base de données avec une requête SQL.Nous pouvons alors démarrer la course et la terminer .

Scénario Démarrer une course manuellement

Pour démarrer une course suite à l'action sur le bouton Démarrer une trame va être envoyée au chronomètre et va débuter la course .
La course Cross moins de 15 féminins à démarrer.

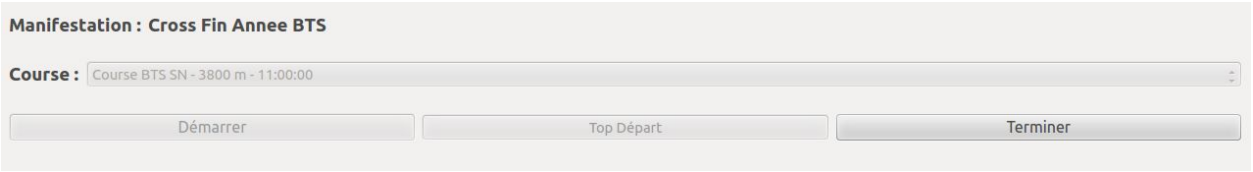




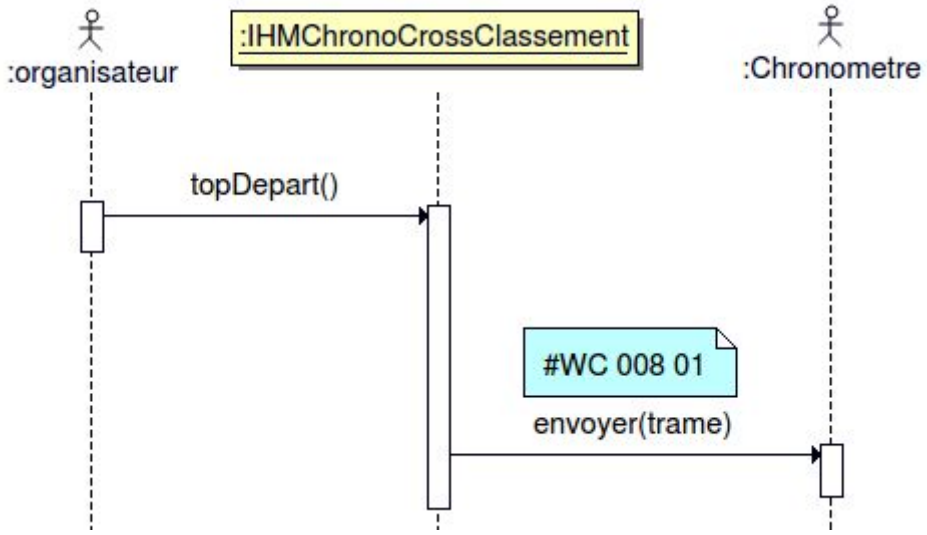
La trame `#WP 120 5` va configurer le chronomètre et la trame,
`#WC 007 02 00:00 00/00/01` va paramétrer le chronomètre au début de la course.

Suite à l'envoi de la trame le chronomètre affiche le début du temps annonçant le départ de la course.

Scénario Démarrer une course logiciellement

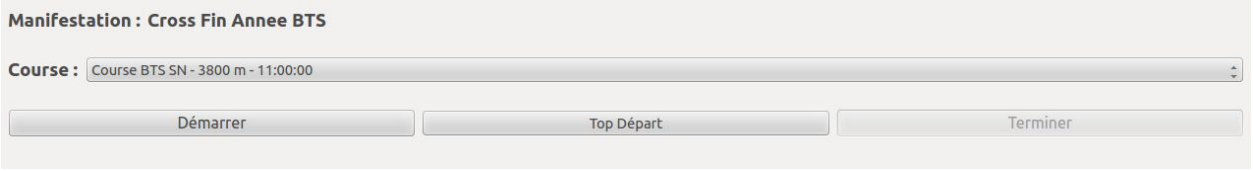


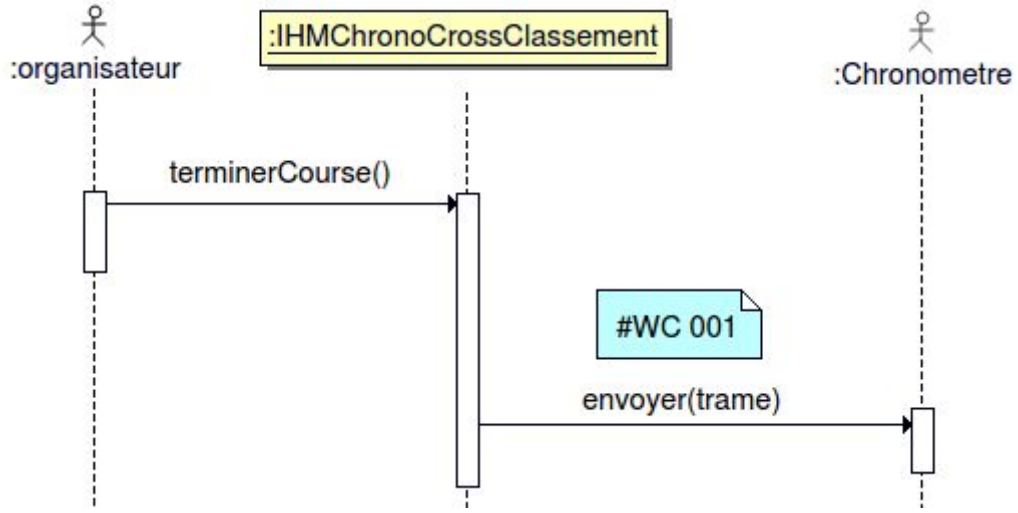
Le bouton Top Départ va lancer le chronomètre "automatiquement"



Scénario Terminer une course

Pour terminer une course suite à l'action sur le bouton Terminer une nouvelle trame va être envoyée au chronomètre et donc va clôturer la course donc plus aucune données ne va être enregistrées par l'application.





la trame `#WC 001` va clôturer la course active

Cas d'utilisation : Chronométrer et classer les arrivées

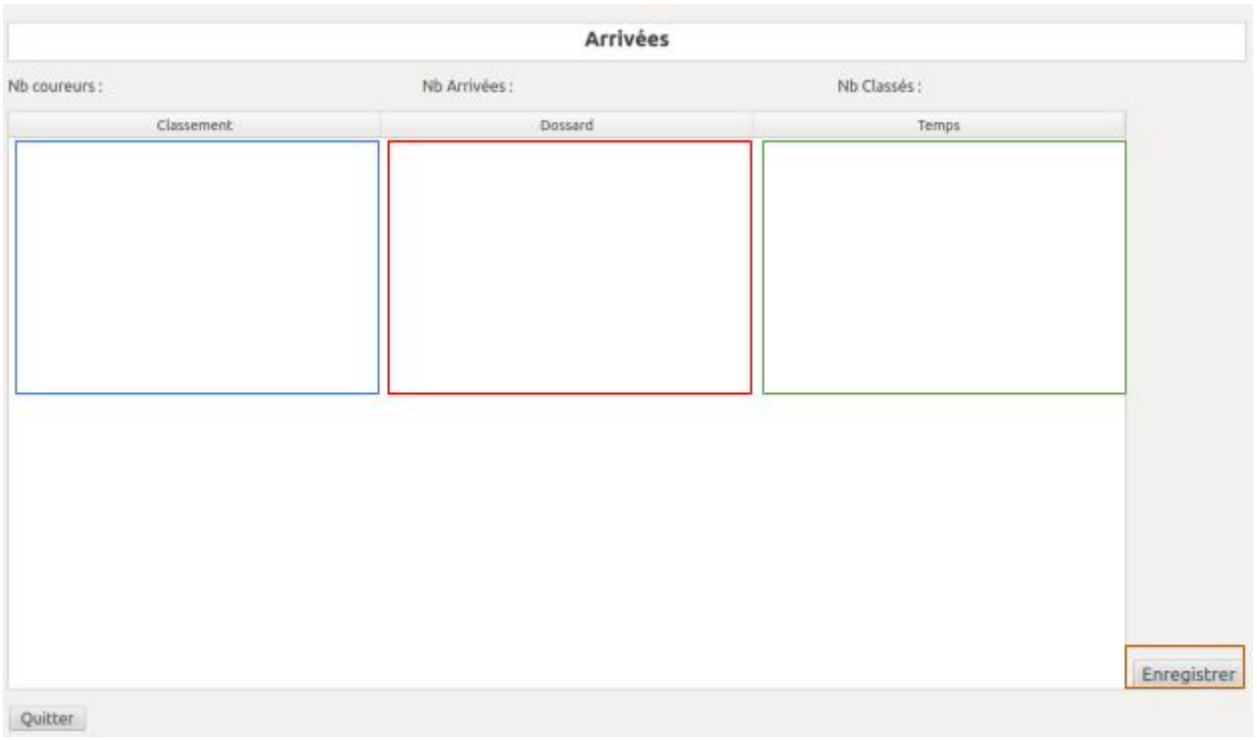
1- Affiche le nombre de coureurs
2- Affiche le nombre de coureurs arrivés
3- Affiche le nombre de coureurs classés
4- Affiche le classement des coureurs
5- Affiche le numéro de dossard
6- Affiche le temps au passage du capteur infrarouge
7- Enregistre

la deuxième partie va permettre de classer les arrivées des coureurs. A chaque passage devant le module infrarouge (module EC) , une trame va être envoyée avec le temps puis le temps s'affiche dans la colonne 6 , le numéro 2 va alors passé à 1 car il y aura eu une arrivée, ensuite il faut entrer le numéro de dossard qui associera au temps dans l'ordre.

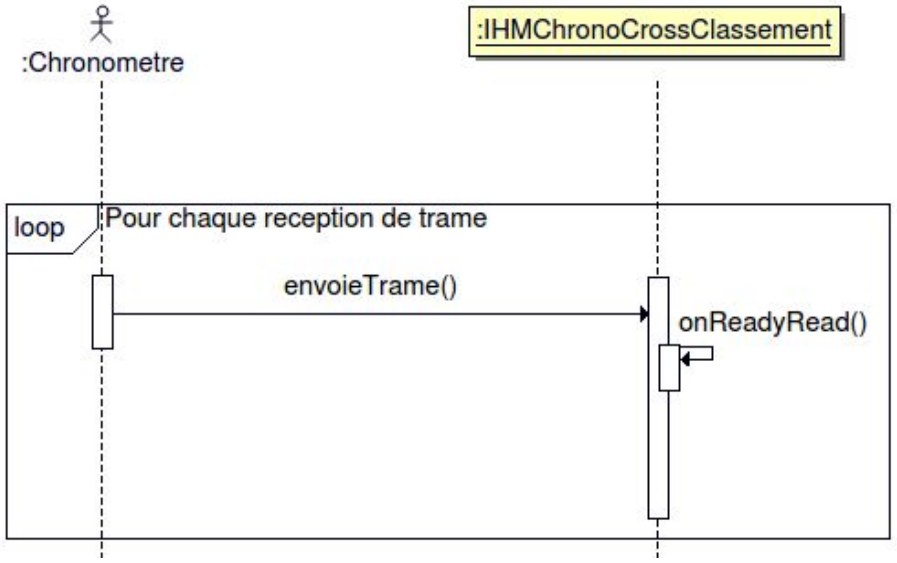
Pour afficher le nombre de coureurs la requête SQL suivante est utilisé :

```
"SELECT idInscrit, NumeroDossard FROM Inscrit WHERE idCourse = "
```

Scénario Arrivée des coureurs



Une fois que le coureur franchit le capteur infrarouge (module EC), le chronomètre envoie une trame avec le temps correspond à la traversé du capteur infrarouge, le classement est fait dans l'ordre d'arrivée des temps, pour les dossards il faut sélectionner la case et entrez le numéro de dossard et valider et une requête SQL enverra alors le temps le numéro de dossard et le classement du coureur dans la base de données.

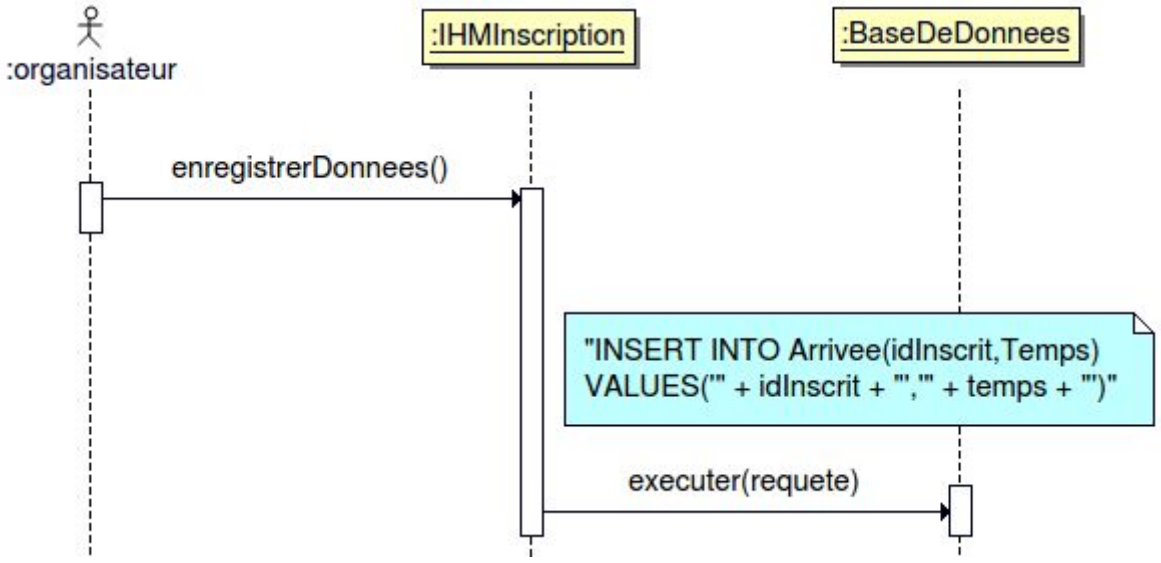


Classement	Dossard	Temps
1	109	00:00:43
2	103	00:00:46
3	105	00:00:48
4	101	00:00:50

Le chronomètre va envoyer la trame suivante pour le temps:

"TN 2 2 46.25400 36505D7"

La partie entourée dans la trame correspond au temps reçu au passage du capteur infrarouge. une fois le dossard associés au temps une requête SQL va envoyer les données dans la base de données avec le bouton Enregistrer



puis le deuxième élève IR va pouvoir afficher et imprimer les résultats.

Recette

Tests	OUI	NON
l'inscription des coureurs est possible	**X**	
les associations coureurs/transpondeurs sont stockées dans la base de données	**X**	
les temps d'arrivée et le classement sont affichés sur l'écran	**X**	
les temps d'arrivée et le classement sont stockées dans la base de données		**X**
le démarrage d'une course est possible	**X**	

Bilan

Ce projet m'a permis d'aborder une première expérience d'un travail collaboratif sur la plateforme subversion et QT avec le langage C++.Cela ma permis aussi d'approcher les attentes d'une entreprise tant en qualité de communication que de gestion du temps et du travail .

A ce jour, l'application permet d'inscrire des coureurs à une course.
Démarrer et terminer une course est fonctionnel.
La base de données fonctionne parfaitement et toutes les informations peuvent être stockés.
Plusieurs petites améliorations peuvent être apportées aux applications mais leurs fonctionnalités principales sont respectés par rapport au cahier des charges.

Annexes

Ressources

- http://www.reliableracing.com/downloads/THCOM08.pdf Chapitre 13 page 52
- http://tvaira.free.fr/projets/activites/activite-chrono-tagheuer.html

Mise en oeuvre du chronomètre TAG heuer hl975

Détection

```
$ dmesg
...
[12736.539217] USB Serial support registered for FTDI USB Serial Device
[12736.539450] ftdi_sio 4-2:1.0: FTDI USB Serial Device converter detected
[12736.539545] usb 4-2: Detected FT232RL
[12736.539551] usb 4-2: Number of endpoints 2
[12736.539557] usb 4-2: Endpoint 1 MaxPacketSize 64
[12736.539562] usb 4-2: Endpoint 2 MaxPacketSize 64
[12736.539567] usb 4-2: Setting MaxPacketSize 64
[12736.542416] usb 4-2: FTDI USB Serial Device converter now attached to ttyUSB0
[12736.542462] usbcore: registered new interface driver ftdi_sio
[12736.542467] ftdi_sio: v1.6.0:USB FTDI Serial Converters Driver

$ lsusb
...
Bus 004 Device 002: ID 0403:6001 Future Technology Devices International, Ltd FT232
USB-Serial (UART) IC

$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 mars  15 14:01 /dev/ttyUSB0
```

Prise en charge automatique

```
$ sudo vim /etc/udev/rules.d/51.ttyusb.rules
# Adaptateur pour Mini Display HL 975
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", MODE="0666",
SYMLINK+="hl975"

$ ls -l /dev/ttyUSB0

crw-rw-rw- 1 root dialout 188, 0 mars  15 14:27 /dev/ttyUSB0

pellizzonic@BTS-SN-ZUSE:~$ ls -l /dev/hl975
lrwxrwxrwx 1 root root     7 mars  15 14:27 /dev/hl975 -> ttyUSB0
```

Programme de test du chronomètre