Etudiant C

HOUDAYER Pierre

BTS2

Dossier de conception détaillée/ réalisation

**Projet 1 :**

**Consignes et assistance vidéolink: Base de données locale**

BTS IRIS 2ème année

Immaculée Conception – LAVAL

Sommaire

[I –La création et la connexion à la base de données. 3](#_Toc421604444)

[I.A – Introduction. 3](#_Toc421604445)

[I.B – Le module QtSql et ses pilotes. 3](#_Toc421604446)

[I.C –Connexion à la base de données. 4](#_Toc421604447)

[I.D – Requêtes sans retour de données. 4](#_Toc421604448)

[I.E – Requête avec retour de données. 5](#_Toc421604449)

[I.F – La classe CSQLite\_Local\_DB. 6](#_Toc421604450)

[II- Les Interfaces Homme Machine (IHM). 6](#_Toc421604451)

[II.A – Interface de maintenance. 6](#_Toc421604452)

[II.A.1 – Diagramme de classe de l’interface de maintenance. 7](#_Toc421604453)

[II.A.2 – Gestion de la base de données. 7](#_Toc421604454)

[II.A.3 – Gestion des portes. 9](#_Toc421604455)

[II.B – Interface de Configuration du site. 12](#_Toc421604456)

# I –La création et la connexion à la base de données.

## I.A – Introduction.

Beaucoup de frameworks proposent une couche d'accès aux données, c'est-à-dire un système apportant bien souvent une certaine transparence vis-à-vis du Système de Gestion de Bases de Données (SGBD).

Nous n'avons plus à nous préoccuper du driver au niveau du code, puisque ce sera le rôle du framework.

Qt en fait partie et nous allons voir quelques-unes des très nombreuses possibilités qu'il offre. Bien évidemment, il existe d'autres frameworks, comme .Net par exemple, mais nous utiliserons Qt pour sa facilité d'utilisation, sa portabilité et l’intégration dans l’application LC.

Le langage utilisé sera C++ et le module QtSql.

## I.B – Le module QtSql et ses pilotes.

Différents drivers sont utilisables au niveau de la connexion avec le module QtSql, mais pour nous, développeurs, nous n'avons pas à nous préoccuper de ce qui se passe à si bas niveau d'abstraction. Il est cependant utile de connaître quels drivers existent, et comment les utiliser.

Certains sont disponibles dans la version open source, d'autres nécessitent une recompilation de Qt.

Les pilotes (drivers) compatible avec le module QtSql sont répertoriés dans le tableau suivant :

|  |  |  |
| --- | --- | --- |
| Pilote | Base de données | Disponible en open source |
| QDB2 | IBM DB2 version 7.1 et ultérieure | Non |
| QIBASE | Borland InterBase | Non |
| QMYSQL | MySQL | Non |
| QOCI | Oracle (Oracle Call Interface) | Oui |
| QODBC | ODBC (inclut Microsoft SQL Server) | Oui |
| QPSQL | PostgreSQL versions 6.x et 7.x | Non |
| QSQLITE | SQLite version 3 et ultérieure | Oui |
| QSQLITE2 | SQLite version 2 | Non |
| QTDS | Sybase Adaptive Server | Non |

Le pilote que nous utiliserons est naturellement QSQLITE du fait qu’il pilote une base de données avec la librairie SQLite que nous avons précédemment choisi (dernière version 3.8.10.1).

## I.C –Connexion à la base de données.

Avant toute chose nous devons modifier le .pro, avec Qt, il faut précompiler le projet avec qmake pour qu'il puisse être compilé correctement. Il faut modifier ce fichier de projet pour que les classes d'accès aux données soient accessibles. Pour ce faire il suffit d'ajouter ceci au .pro : « QT += sql », On obtient alors l’accès au module QtSql et ses pilotes.

Nous allons maintenant utiliser notre pilote QSQLITE.

Les connexions s'utilisent au travers de la classe QSqlDatabase.

Cette classe possède une méthode statique QSqlSatabase::addDatabase(constQString) renvoyant une instance de QSqlDatabase et reçoit en paramètre une chaîne de caractères correspondant au driver utilisé donc dans notre cas : QSQLITE.

Voici un petit programme simple pour créer une base de données, illustrant ce principe de création et connexion a une base de données :

#**include**<QApplication>

#**include**<QSqlDatabase>

#**include**<QSqlError>

#**include**<QMessageBox>

int**main**(intargc, char \*argv[])

{

QApplication **app**(argc, argv);

win\_Form \*window = new **win\_Form**();

QSqlDatabase db = **QSqlDatabase::addDatabase**("QSQLITE");

db.**setDatabaseName**("LC\_Database.sqlite");

if(!db.**open**())

{

**QMessageBox::critical**(0, **QObject::tr**("Database Error"), db.**lastError**().**text**());

}

window->**show**();

returnapp.**exec**();

}

## I.D – Requêtes sans retour de données.

Il y a deux différences fondamentales dans les requêtes SQL, celles qui renvoient des données, et celles qui n'en revoient pas.

La différence au niveau applicatif est qu'il faudra (ou pas) gérer un retour de données.

Dans certains frameworks, il existe plusieurs fonctions permettant d'exécuter des requêtes, certaines pour gérer des retours, d'autres non ... Cette utilisation peut paraître peu pratique aux yeux de certains. En effet, le programmeur peut se tromper de fonction et provoquer une erreur, alors que la requête est correcte.

Qt procède différemment dans le sens où il n'existe qu'un seul moyen d'envoyer la requête. Libre à nous, si on souhaite récupérer les valeurs de retour,on utilise pour cela une instance de la classe QSqlQuery en appelant sa méthode QSqlQuery::exec(constQString).

Voici le formalisme à utiliser pour envoyer des requêtes au format SQL au moteur de la base de données :

**QSqlQuery**query;

query.**exec**("INSERT INTO Table(Column\_Name\_1, Column\_Name\_2) VALUES ('Value\_ Column\_Name\_1', 'Value\_ Column\_Name\_2')");

## I.E – Requête avec retour de données.

Comme expliqué dans la partie précédente, la manière de requêter sera la même.

Nous avons seulement à parcourir le retour de la requête stocké au sein de l’instance QSqlQuery afin de récupérer et traiter une à une les lignes retournés par la requête.

Pour avoir accès aux données reçues, on utilise la méthode QSqlQuery::value(int) qui nous permet d’identifier la colonne (champ de la table) avec un son numéro en commençant par la gauche que l’on souhaite traiter.

Récupérer ces données nécessite de connaître parfaitement la construction de la table sélectionnée pour identifier les colonnes correctement.

Voici un exemple de l’utilisation de la méthode QSqlQuery::value(int) pour récupérer les données.

**QSqlQuery**query;

query.**exec**("SELECT \* FROM Table");

while(query.next())

{

intValue\_Column\_Name\_1 = query.**value**(0).**toInt**();

**QString**Value\_Column\_Name\_2 = query.**value**(1).**toString**();

}

Si c'est son premier appel la méthode QSqlQuery::next() place la lecture au premier enregistrement retourné, sinon elle positionnera la lecture à l'enregistrement suivant.

*Remarque :* Il peut sembler incohérent d’utiliser des fonctions de conversions comme « toInt() » ou« toString() »car la base de données contient des données déjà formatées mais QSqlQuery::value(int) retourne un QVariant. Il s’agit en interne d’une structure proche de l’union pour les types de données de Qt les plus communs. On peut alors l’associer à n’importe quel type, ce qui semble logique car QSqlQuery::value(int) ne connais pas le type de données qu’elle traite. Nous allons donc utiliser des fonctions de conversion pour stocker les résultats de la requête renvoyée par QSqlQuery::value(int).

## I.F – La classe CSQLite\_Local\_DB.

La conception de la classe CSQLite\_Local\_DBest basée sur les principes énoncés précédemment. Les principales méthodes de la classe sont destinées à communiquer directement avec la base de données. Celle-ci sera développée avec l’environnement Qt-creator en C++.

Nous utilisons un outil de génération de documentation automatique nommé « Doxygen ». La totalité de la description de la classe est représenté à l’adresse suivante :

<http://bts2-projet-1.github.io/html>

Menu « Classes » puis « Liste des classes » : CSQLite\_Local\_DB.

Afin de faciliter l’évolution et la future intégration de la classe dans l’application LC. Nous avons utilisé un gestionnaire de version : Git. Le dépôt local de git est ensuite associé à une interface web, GuiHub qui permet à chacun de travailler sur le projet.

# II- Les Interfaces Homme Machine (IHM).

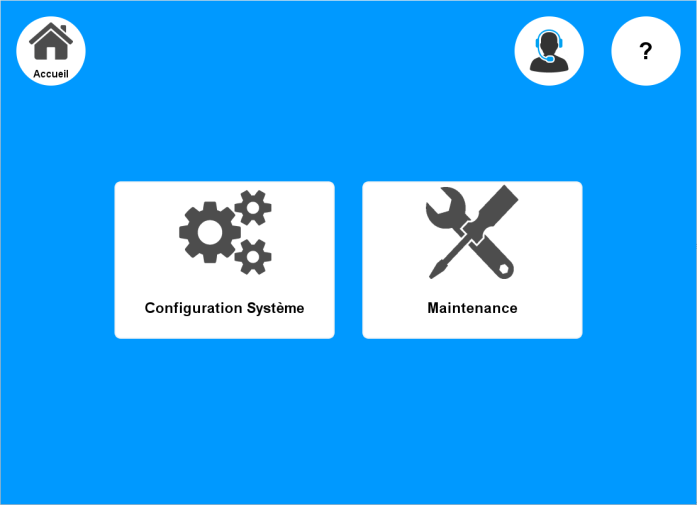
La conception des IHM sera effectué avec l’environnement de développent Qt-creator en C++ et inclus Qt-designer qui permet l’édition des interfaces avec un rendu en temps réel.

## II.A – Interface de maintenance.

L’interface de maintenance a pour but de fournir à un opérateur, un moyen simple et rapide de tester des portes de consignes et la possibilité de pouvoir les activer ou désactiver. Enfin il est possible de pouvoir gérer la base de données en reconstruction par défaut ou d’une ancienne sauvegarde par exemple.

L’interface de maintenance est disponible, après identification, par le bouton Maintenance du menu de configuration.





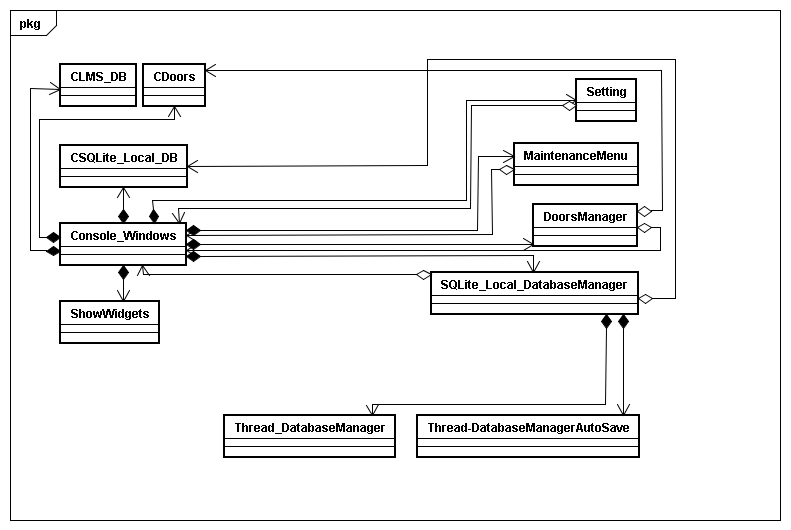
Disponible

Indisponible

Le menu de maintenance est désormais disponible. Dans l’état actuel du projet, la configuration du site et les redémarrages ne sont pas disponible.

### II.A.1 – Diagramme de classe de l’interface de maintenance.

Le diagramme de classe suivant décrit l’implémentation des classes de gestion de l’interface de maintenance dans l’application LC.



Classes de gestion des IHM.

### II.A.2 – Gestion de la base de données.

Dans cet exemple de gestion de la base de données l’opérateur sélectionne une action à réaliser sur la base de données. Tous les cas d’utilisation se déroulent de la même façon dans l’interface, seul le cas de sauvegarde manuel sera détaillé.

Dans un premier temps après avoir cliqué sur le bouton « Gestion de la base de données » dans le menu de maintenance l’opérateur est invité à effectuer une action. L’image suivante est extraite de l’application LC.

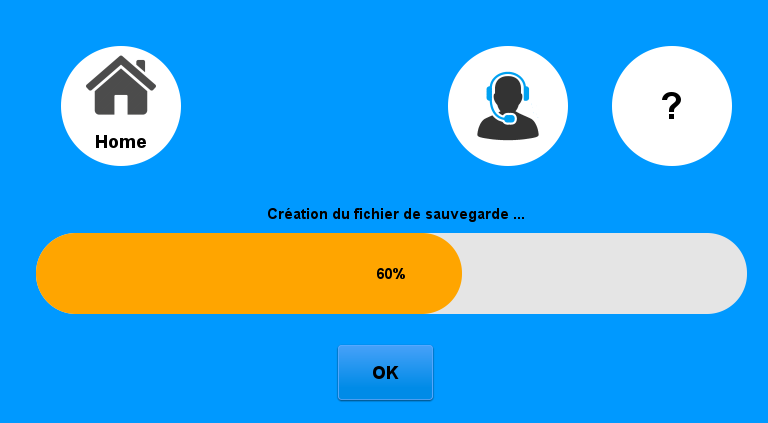


Dans cet exemple l’opérateur sélectionne d’effectuer une sauvegarde de la base de données. Une page de confirmation lui demande de valider l’action à réaliser (image suivante).



Si l’opérateur souhaite continuer (valider par OUI), La commande sélectionnée est réalisée sur la base de donnée sinon la page précédente est affichée.

Lors de la confirmation un page de chargement apparait informant l’opérateur de l’avancée du processus demandé (image suivante).



Une fois le processus demandé terminé, soit l’opérateur valide avec OK, sinon un retour automatique est exécuter (évite les oublis de déconnexion).

Les autres cas d’utilisation (Charger la dernière sauvegarde, reconstruire la base de données à l’état initial ou une base de données par défaut) se font sur le même principe :

* Choisir une action ;
* Confirmer ;
* Page de chargement ;
* Retour.

### II.A.3 – Gestion des portes.

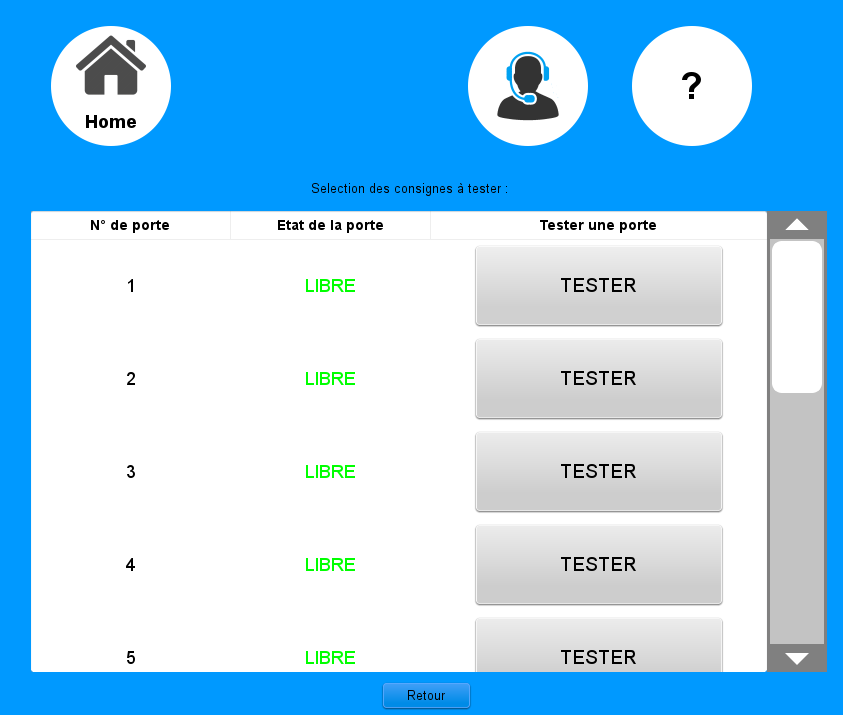
Le gestionnaire des portes est disponible dans le menu de maintenance avec le bouton « Gestion des portes ». Le menu de gestion des portes dispose de trois fonctionnalisées :

* La possibilité de pouvoir modifier la configuration du site en ajoutant ou supprimant des portes installées (Non fonctionnel).
* Pouvoir activer ou désactiver des portes suite à un dégât par exemple.
* Tester une porte (Ouverture/Fermeture et Verrouillage/Déverrouillage).



Dans un premier temps le cas du test d’une porte :

L’opérateur choisis le bouton « Tester une porte ». L’interface de sélection s’affiche pour que l’opérateur choisisse une porte parmi la liste installée.



En choisissant la porte 3 qui est libre (Sans colis) on peut effectuer des tests sur celle-ci avec l’interface suivante.



Cette interface propose à l’opérateur d’effectuer la demande de déverrouillage et d’ouverture de la consigne 3 lors de l’appui respectif des boutons. L’application remplie automatiquement le résultat qu’elle place dans les cases à cocher (Lecture de l’état des portes) si l’opérateur constate une erreur de lecture de l’application il peut cocher NON sur la ligne correspondante.

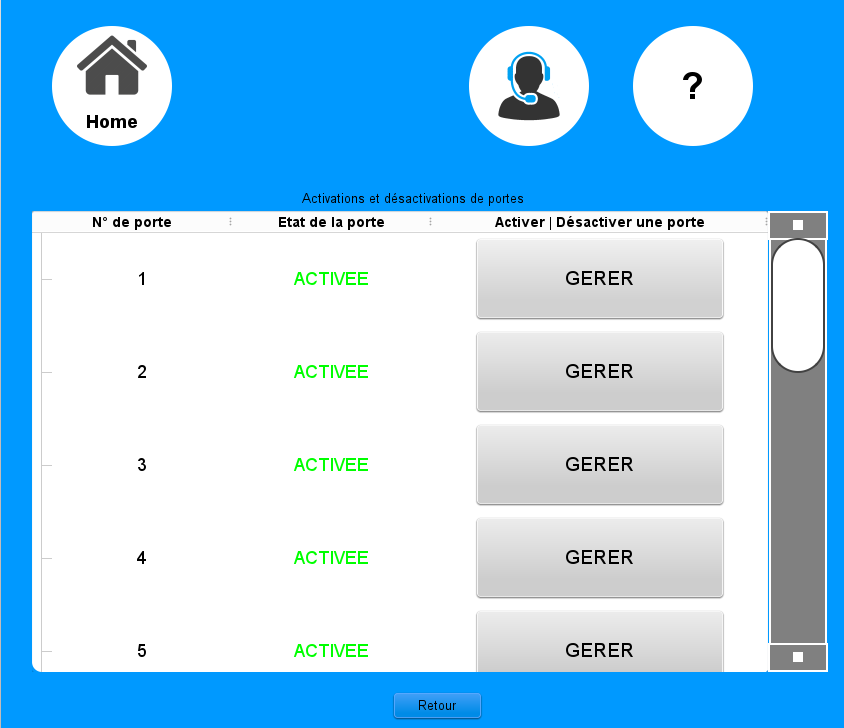
Suite à l’appui des boutons « Demander le déverrouillage » et « Demander l’ouverture » voici l’interface remplie par l’application.



Apres l’appuie sur le bouton « Envoyer » l’application analyse le contenu des cases sélectionnées et en déduit le comportement de la porte et informe l’opérateur. Dans le cas où l’une ou plusieurs cases sont cochés l’application place la porte en état de défaut et informe l’opérateur de cette action.



Dans un second temps l’activation et la désactivation manuelle de la porte. L’opérateur choisis le bouton « Activer / Désactiver une porte » du menu de gestion des portes. L’interface de sélection s’affiche pour que l’opérateur choisisse une porte parmi la liste installée.



Dans cet exemple l’opérateur choisis de désactiver la porte 2, L’interface de gestion s’affiche alors avec la gestion de la porte 2.



Grâce à cette interface l’opérateur peut désactiver la porte sélectionnée et la rendre inutilisable par les livreurs ou retourner à la page précédente si celui-ci s’est trompé de porte et ainsi éviter de désactiver une mauvaise porte.

## II.B – Interface de Configuration du site.

L’interface de configuration du site doit permettre à un opérateur authentifié de pourvoir gérer le parc de consignes configurées en base de données. Il sera possible à un opérateur d’ajouter/supprimer des consignes utilisable par l’application LC.

Cette partie du projet n’est pas encore terminée suite aux contraintes de temps.