IMPORTANT : L’application fonctionne, mais à noter que l’envoi des mails ne peut pas se faire depuis la connexion internet du pôle Leonard-de-Vinci, l’envoi fonctionne cependant depuis une connexion internet « normale » (c’est probablement un problème de configuration réseau, nous utilisons gmail et smtp pour l’envoi mais n’avons pas eu le temps de contacter le service informatique du pôle pour de plus amples informations ni de régler le problème car nous ne l’avons repéré que ce vendredi 21 mai (le projet était déjà envoyé), étant donné que nous travaillons depuis chez nous). Ajouter une commande ou envoyer les fichiers .json/.xml depuis la connexion du pôle entrainera un *time out* du serveur (300 sec d’attente avant ..) mais ne fera pas crash l’application, seulement la bloquer durant ce laps de temps.

A picture containing text

Description automatically generated

Rapport de code

probleme *BDD & intéropérabilité*

Régis de LAGREVOL – François COUTAU – Yanis DAHMOUCHE | TD Q | BDD | 22/05/2021

# Introduction

Notre avons réalisé une application WPF fonctionnelle à 3 et conformément aux règles, avons utilisé des concepts non vus en TD.

# Organisation de la BDD

On peut considérer que la base de la BDD est la table Pieces. Chaque piece est unique et est référencée dans 2 autres tables, la table Assemblages et la table Commandes. En effet un client peut effectuer une commande d’un modèle et ou d’une pièce. Chaque pièce possède son fournisseur. La table Commandes est donc reliée à une piece ou à un modele ainsi qu’à un client qui l’a commandée. Un client possède un idFidelio qui correspond au détail de son offre Fidelio. Chaque acteur de la BDD (Clients et Fournisseurs) possède une Adresse.

La BDD utilise 3 Triggers :

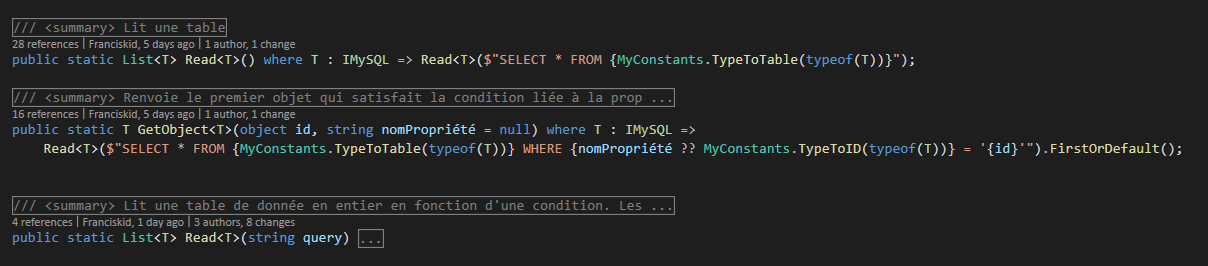
1. Un premier qui supprime toutes les pièces liées à un fournisseur quand on supprime ce fournisseur de la BDD
2. Un second qui décrémente la quantité de pièces en stock lorsque on effectue une commande (modèle ou pièce) en fonction de la quantité commandé
3. Un troisième qui commande une pièce dont la quantité est tombée à 0 automatiquement auprès du fournisseur concerné pour l’entreprise velomax.

On utilise aussi un Event qui permet d’ajouter automatiquement à la BDD la quantité de pièces commandées par l’entreprise velomax une fois la date de livraison passée.

# Option de codage

Pour développer à 3 un projet aussi lourd, nous avons utilisé Github. Au début difficile à prendre en main, cet outil s’est avéré indispensable et nous a fait économiser un gain de temps considérable.

Pour lire la BDD nous avons manuellement associé chaque table de la BDD à une classe qui hérite de l’interface IMySQL, interface qui nous permet de lire et d’ajouter/modifier/supprimer très facilement les objets concernés.

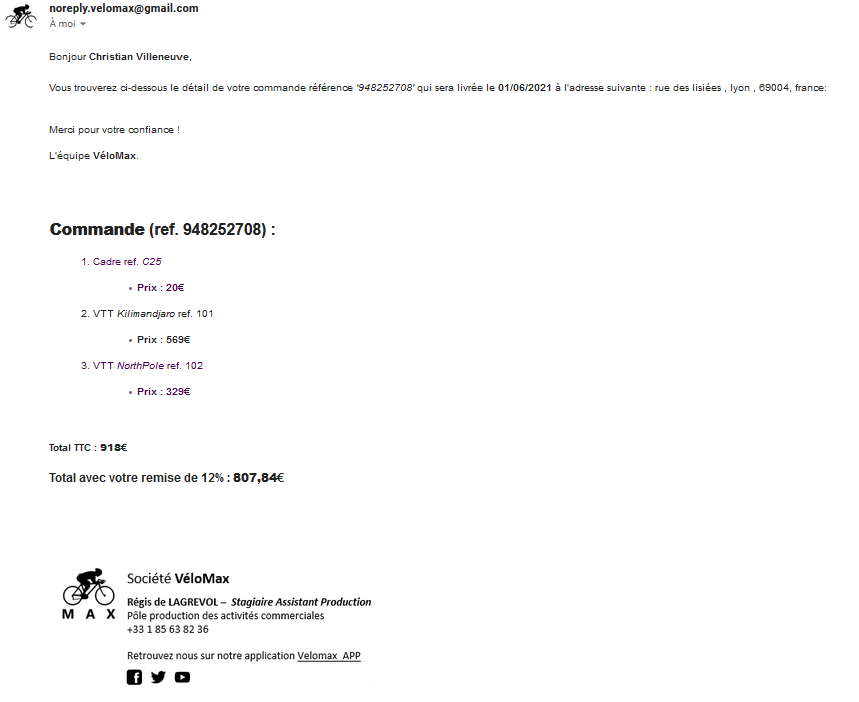
La fonction Read<T>() prend en paramètre un type générique IMySQL et renvoie la liste des objets de la table de la BDD associé à ce type. Par exemple :

* List<Adresse> ad = BDDReader.Read<Adresse>() , ad possèdera tous les objets de la table ‘Adresse’. Ceci est valable pour tous les objets héritant de IMySQL

BDDWriter permet, de la même manière, d’insérer, modifier, supprimer n’importe quel objet IMySQL.

# Développements complémentaires

Notre application envoie automatiquement un mail au client une fois la commande passée (lire la note de début de rapport pour éviter un possible disfonctionnement indépendant de notre volonté). Un mail est aussi envoyé lors de l’export des données en .xml ou .json, il faut alors préciser le mail. Si le champ pour entrer son mail est laissé vide alors aucun mail ne sera envoyé.

Ceci est le template de base du mail lors de l’envoi d’une commande à un client. Nous utilisons du code HTML pour la mise en page et avons essayé de recréer un template le plus réaliste possible selon nos moyens et notre temps.

Lors de l’envoi d’un fichier .xml ou .json, les fichiers sont envoyés en pièce jointe.