

# Rapport du projet de génie logiciel de 3BE : Application KitBox

**Auteurs** : Soysal Deniz, Gorjon Julien, Argyrakis Yannis, Itunime Masala Rudy

---

## TABLE DES MATIERES

1	Introduction.....	3
2	Exigences du client .....	3
2.1	Interface Client .....	3
2.2	Interface Magasinier.....	3
3	Architecture des classes .....	4
3.1	Diagramme de classe .....	4
4	Choix technologiques .....	6
5	Base de données .....	7
5.1	Passer du fichier .CSV au fichier .SQL .....	7
5.2	Structure de la base de données : .....	8
5.3	Commande SQL .....	9
5.4	Intégration de SQL en C# (visual studio).....	9
6	interface client .....	10
6.1	Page d'accueil .....	10
6.2	Choix des caractéristiques .....	10
6.3	Armoires standards .....	11
6.4	Armoire sur mesure .....	12
6.5	Panier.....	13
6.6	Code.....	13
7	Interface magasinier.....	14
7.1	Accès au stock.....	15
7.2	Acces aux commandes.....	16
7.3	Relation avec la base de données/fonctionnement .....	17
8	Mode d'emploi .....	18
9	Conclusion .....	19
9.1	Bilan des objectifs.....	19
9.2	Pistes d'améliorations .....	20
9.3	Outils utilisés .....	20
9.4	Collaboration dans le groupe .....	20
10	Annexes.....	22
10.1	Annexe 1 : Diagramme d'activité.....	22
10.2	Annexe 2 : diagramme des cas d'utilisations.....	23

## 1 INTRODUCTION

---

Pour mettre en pratique les acquis accumulés durant le premier quadrimestre, nous avons développé une application pour le cours de « Projet informatique ».

KitBox, une société fictive, nous a contactés dans le cadre de son informatisation. Cette société vend des armoires en kit. Et pour ce faire, deux fichiers nous ont été transmis. Une avec la liste complète des pièces disponibles et l'autre avec la liste de ses fournisseurs.

Afin de mener ce projet à bien, en groupe de quatre via plusieurs séances de « bureau d'étude », nous avons d'abord analysé les besoins du client. Et de là, nous avons pu faire un choix d'architecture adapté pour ce projet et également choisir une méthodologie de travail. La communication entre les différents membres de l'équipe a été faite via Trello, Github et Messenger.

Ce rapport explique nos choix, nos soucis et le résultat du projet. Il contiendra entre autres une explication du fonctionnement globale de l'application, une justification de nos choix technologiques ainsi qu'une conclusion.

## 2 EXIGENCES DU CLIENT

---

Lorsque les clients remplissent eux même leurs commandes dans le catalogue de KitBox, il arrive souvent que des erreurs soient faites. Des dimensions incompatibles entre elles dans la constitution d'une armoire peuvent par exemple être choisies. Le but principal de l'application est donc d'éviter que de telles erreurs puissent arriver.

L'informatisation du catalogue étant une bonne occasion pour informatiser la gestion du stock et des commandes, la société Kitbox demande la conception de deux interfaces distinctes : une interface client et une interface magasinier.

### 2.1 INTERFACE CLIENT

Cette application doit permettre au vendeur et au client de remplir une commande d'armoires. Pour finaliser cette commande, un récapitulatif doit permettre au client de la passer en revue avant de la confirmer et de la payer. La disponibilité étant un facteur parfois déterminant dans la décision du client, elle doit être consultable. Une fois que le client a entré ses coordonnées et confirmé la commande, le système doit permettre d'en imprimer la facture.

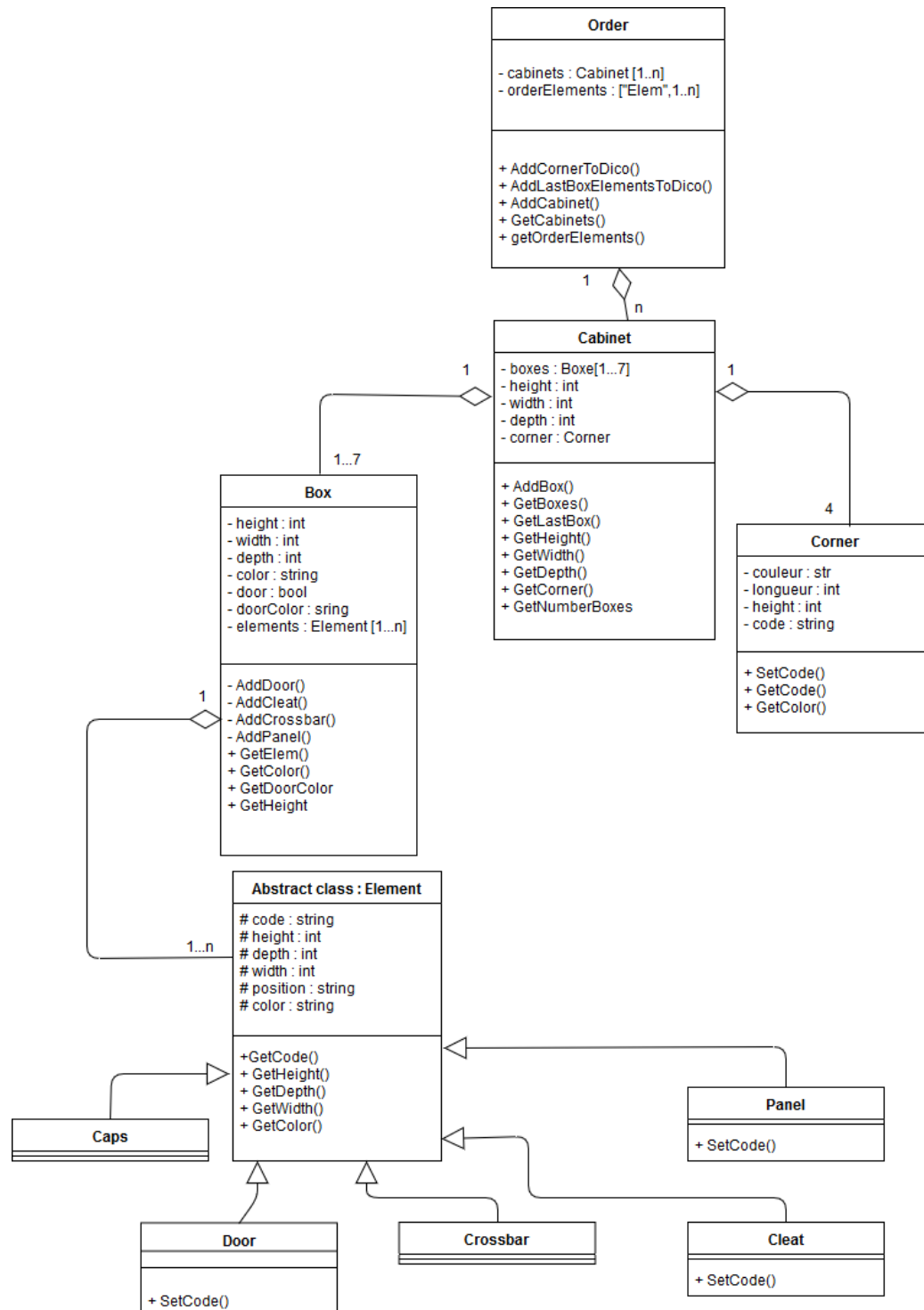
### 2.2 INTERFACE MAGASINIER

De son côté, le magasinier doit pouvoir consulter les commandes payées afin d'en imprimer la liste des pièces. Une fois ces commandes préparées, il doit finalement pouvoir les clôturer. En cas de problème, il doit pouvoir consulter les informations sur le client d'une commande afin de le contacter.

L'interface doit aussi permettre de gérer le stock. En plus de fonctions simples comme supprimer ou ajouter des pièces, un algorithme doit proposer des commandes de pièces en se basant sur les ventes effectuées durant les 6 mois qui ont précédé. Ces commandes doivent être proposées chez les fournisseurs les moins chers. C'est pourquoi une secrétaire doit pouvoir mettre à jour les prix de ceux-ci.

### 3 ARCHITECTURE DES CLASSES

#### 3.1 DIAGRAMME DE CLASSE



Nous avons réalisé l'architecture de nos classes selon le design pattern **Composite**. Ce design pattern permet de rassembler plusieurs objets ayant des fonctionnalités ou caractéristiques communes en les organisant sous forme d'un arbre grâce auquel on peut aisément manipuler un groupe d'objet. On peut ainsi constater qu'un objet de la classe « Order » se constitue d'une ou plusieurs « Cabinet » elles-mêmes constituées de « Boxe », celle-ci étant faites à partir d'éléments tels que des « Panel », des « Door »...

Ce design pattern nous permet de répondre à la problématique de la construction de l'armoire. On a notamment une classe abstraite « Element » qui définit des attributs tel que : height, depth, position,... ainsi que des méthodes utilisées pour renvoyer les valeurs de ces attributs, ce qui prend tout son sens lorsqu'on veut effectuer une opération comme une recherche sur tous les éléments d'une « Boxe » pour rechercher leurs codes barre dans la base de données par exemple.

Nous avons tâché de respecter le principe de la POO d'ouverture à l'extension et de fermeture à la modification à travers la classe « Element » car elle nous offre la possibilité de rajouter des éléments constitutifs du casier, par exemple si la société KitBox décide d'ajouter les serrures à leur casier, on pourra créer la classe « Serrure » qui sera une sous-classe de « Element » sans pour autant changer l'implémentation des autres classes.

L'encapsulation des classes a aussi été réalisée grâce aux attributs qui sont toujours privés, sauf pour la classe abstraite où l'on a choisi des attributs protégés, ce qui permet d'y accéder dans les sous classes. La classe abstraite a été préférée à l'interface car certaines méthodes peuvent s'implémenter dans « Element » étant donné qu'elles sont identiques pour la plupart, cela évite donc la duplication de code.

On constate que la classe « Boxe » possède des méthodes privées tel que « AddDoor() », « AddPanel() ». Celles-ci sont utilisées lors de la construction de chaque étage de l'armoire et s'occupent d'ajouter les éléments de chaque « Boxe » dans une liste. La construction de cette liste se fait dans le constructeur de la classe « Boxe » c'est pourquoi ces méthodes sont privées et non publiques. Nous n'avons en aucun cas besoin d'y accéder de l'extérieur de la classe.

Il est à noter que la classe « Cabinet » possède un seul attribut de type « Corner » alors qu'une armoire est constituée de 4 cornières. Étant donné qu'une armoire a toujours 4 cornières, on décide d'utiliser uniquement le facteur multiplicatif 4 lorsqu'on va manipuler les stocks dans la base de données.

La classe « Order » possède un dictionnaire contenant les code-barres des pièces détachées de la commande en cours. Ce dictionnaire avait pour but d'interagir plus facilement dans l'interface client avec la base de données car nous pensions qu'il serait plus facile de manipuler le stock en connaissant le nombre de pièces grâce au dictionnaire qui contient comme clés des codes de référence d'une pièce et en valeur le nombre de fois que la pièce apparaît. C'est pourquoi vous pouvez constater qu'une fonction « SetCode() » permet pour chaque élément de renvoyer son code bar construit de manières logiques par rapport à la valeur de ses attributs. Nous n'utilisons actuellement plus ce dictionnaire car nous nous sommes rendu compte qu'il manquait de fiabilité dans le cas où nous souhaitons rajouter des nouvelles pièces. Par exemple, KitBox crée une nouvelle traverse verte, nous allons créer le code référence « TRG » en prenant le mot clé « TR » suivi de la 1<sup>ère</sup> lettre majuscule de la couleur (en anglais G pour green). Le problème surviendrait donc si on souhaitait créer par exemple une nouvelle traverse grise qui aurait également le code « TRG », cela ne nous permettrait pas de les différencier.

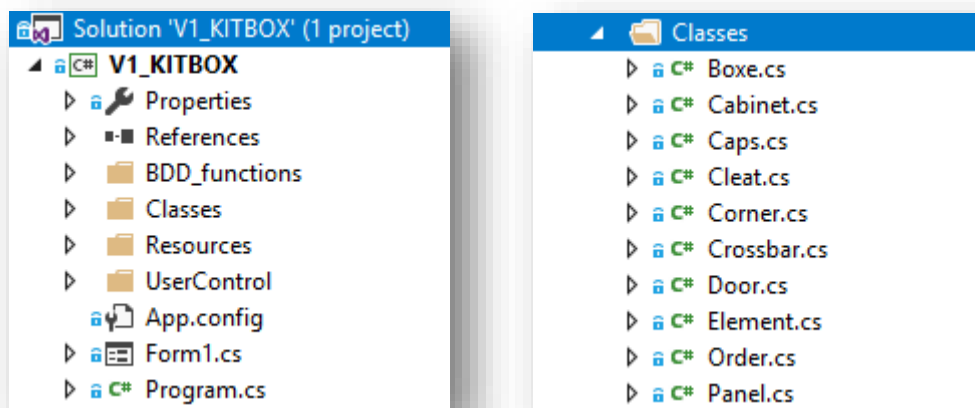
Le diagramme de classe ne couvre pas toutes les fonctionnalités de notre application, notamment les fonctions issues de la base de données car la tâche de travail a été divisé d'une part pour la réalisation de l'interface client et l'implémentation des classes et d'autre part pour la réalisation de l'interface magasinier et l'interaction avec la base de données. De ce fait, la stratégie de l'interface client est de renvoyer d'abord un objet de la classe commande et de n'agir sur la base de données qu'uniquement en toute fin de commande.

## 4 CHOIX TECHNOLOGIQUES

L'application a été réalisée sur la suite de logiciels de développement **Microsoft Visual Studio** en **C#** avec l'interface graphique **Winforms**, celle-ci a été choisie car elle est simple d'utilisation et facile à prendre en main.

Le langage C# est un langage orienté objet, adapté à la conception de design patterns tels que le pattern Composite que nous avons employé car il semblait le plus adapté à notre problématique (Voir « diagramme de classe »).

Nous avons structuré nos solutions en tentant de rendre l'organisation des fichiers la plus compréhensible pour tout programmeur comme vous pouvez le voir sur la figure ci-dessous :



Ainsi, nous avons des sous fichiers tels que « Classes » contenant toutes nos classes implémentées, le fichier de ressources contient les images utilisées dans les différents User Control.

Pour l'interface client, la technologie des User Control (abrégé « UC ») a été préféré à l'utilisation de Form car un UC nous permet de désigner des templates que l'on va décider d'afficher dans un Form, en fonction de la partie de la commande dans laquelle le client se trouve. Le Form1.cs sert donc juste de fenêtre pour l'affichage des différents UC.

Le UC étant une classe en C#, il est possible de lui passer des paramètres dans son constructeur, ainsi on décide de créer un objet de type « Order » lorsque le client commence une commande. Ensuite, nous passons ce même objet en paramètre des UC et cela jusqu'au « Cart » final. Cela permet d'agir facilement sur les objets de la commande en cours.

Concernant la base de données, celle-ci est hébergée sur un serveur local grâce à l'application Web phpMyAdmin. Cette application a été choisie car elle permet de générer et gérer simplement des

tables d'une base de données en cas de problème dans l'application de l'interface magasinier. Le langage pour créer la base de données et effectuer les requêtes est le SQL. Nous l'avons choisi car c'est un langage qui avait déjà été utilisé par un membre du groupe.

## 5 BASE DE DONNEES

Notre base de données va permettre de stocker différentes informations concernant l'entreprise. Nous allons stocker les informations sur les commandes, mais également les informations sur le stock, la clientèle et les fournisseurs

Pour ce faire nous avons besoin d'un système de gestion de bases de données relationnelles (SGBDR). Nous avons choisi d'utiliser MySQL avec phpMyAdmin.

L'avantage est qu'il est multi-utilisateur, gratuit et facile d'utilisation. MySQL supporte aussi bien Windows que MAC ou Linux et supporte aussi bien le C# que le python comme langage de programmation.

Pour voir un peu chez la concurrence :

- Oracle, considéré comme un des SGBDR les plus performants.
- Microsoft SQL Server, est intégré au framework .NET et donc dans Visual Studio.
- MySQL, est très répandu dans la programmation web.
- Access, est présent dans la suite Microsoft Office. Et dispose une interface graphique. Mais est mono-utilisateur et limité en fonctionnalité.

La structure de notre système sera comme ceci : Serveur -> Base de données -> Table -> Données (ligne et colonne).

Dans la table, les données sont de deux types : numérique entier « int » et chaîne de caractères « var char ».

Ci-dessous, un exemple de structure d'une colonne :

The screenshot shows the 'Column Definition' form in phpMyAdmin. The column name is 'NomClient'. The type is 'TINYTEXT'. The length is empty. The default value is 'NULL'. The character set is 'latin1\_swedish\_ci'. The 'Null' checkbox is checked. The 'Unsigned' checkbox is checked. The 'Zerofill' checkbox is unchecked. The 'Comment' field is empty. The 'Virtuality' dropdown is set to 'Virtuality'.

### 5.1 PASSER DU FICHIER .CSV AU FICHIER .SQL

Sur Claco, le fichier disponible représentant le stock est au format .csv. Nous allons donc utiliser ce fichier pour stocker le stock dans notre BDD

Mais pour l'utiliser il faut le modifier comme ceci :

- Modifier les noms avec des caractères spéciaux tel que /, - etc.
- Eliminer les espaces.

Avant :

Réf	Code	Dimensions(cm)	hauteur	profondeur	largeur	Couleur	Enstock
-----	------	----------------	---------	------------	---------	---------	---------

Après :

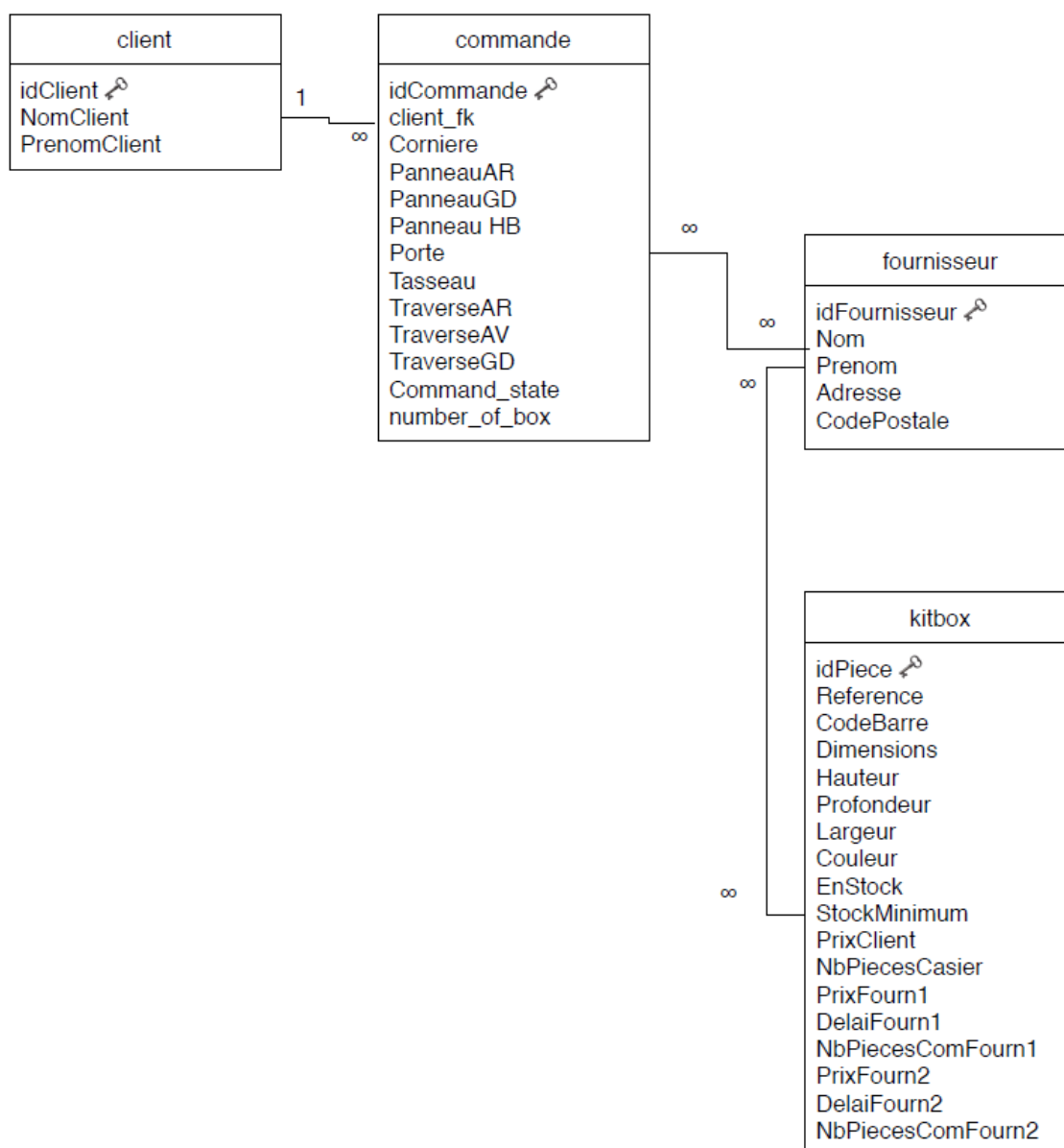
Reference	CodeBarre	Dimensions	Hauteur	Profondeur	Largeur	Couleur	Enstock
-----------	-----------	------------	---------	------------	---------	---------	---------

Après il ne nous reste plus qu'à convertir ce fichier en un fichier SQL, et on peut alors l'importer sur phpMyAdmin.

Ensuite, différentes modifications ont été faites sur cette base de données : ajout des tables « client », « commande » et « fournisseur » en plus de la table « kitbox » (cette dernière représentant le stock).

- Client : contient les informations concernant le client
- Commande : contient les informations concernant le fournisseur
- Kitbox : contient les informations concernant l'armoire.

## 5.2 STRUCTURE DE LA BASE DE DONNEES :





Chaque table a une clé primaire qui l'identifie de manière unique. Ces clés sont idClient, idCommande, idFournisseur et idPiece.

Un client est composé de son nom et de son prénom. Il peut avoir plusieurs commandes. Ces commandes sont reliés au client via la clé « idClient ». Une commande est composée de différentes valeurs, qui sont les codes de chaque pièce, ainsi que l'état de la commande qui peut-être « enregistrée » ou « terminée ». Le nombre de casiers est aussi stocké dans cette table. Connaissant le nombre de pièces requises par casier, on peut ainsi déterminer le nombre total de pièces.

La table « kitbox » représente le stock et est relié à la table « fournisseur » représentant le fournisseur. Dans « kitbox », toutes les pièces sont stockées ainsi que leurs différentes caractéristiques.

### 5.3 COMMANDE SQL

Pour interagir avec les tables, nous avons utilisé des requêtes SQL :

- SELECT : pour sélectionner un ou plusieurs éléments.
- INSERT : pour insérer un ou plusieurs éléments.
- UPDATE : pour mettre à jour un ou plusieurs éléments.
- DELETE : pour supprimer un ou plusieurs éléments.

### 5.4 INTEGRATION DE SQL EN C# (VISUAL STUDIO)

Il est possible d'envoyer des requêtes SQL en C# sous l'environnement Visual Studio afin d'interagir avec la base de données. Pour ce faire il faut commencer par ouvrir une connexion avec celle-ci et puis envoyer la requête et pour finir fermer la connexion.

La ligne de code ci-dessous permet d'utiliser la .DLL MySQLClient :

```
using MySql.Data.MySqlClient;
```

Les lignes de code ci-dessous permettent de créer la fonction de connexion vers la base de données :

```
private MySqlConnection connection;
public frmWM()
{
    this.InitConnexion();
    InitializeComponent();
}
private void InitConnexion()
{
    // Création de la chaîne de connexion
    string connectionString = "SERVER=127.0.0.1; DATABASE=kitbox_database;
    UID=root; PASSWORD=";
    this.connection = new MySqlConnection(connectionString);
}
```

Une requête SQL pour l'Insert ressemble à ça :

```
INSERT INTO Insert (Column) VALUES (Values)
```

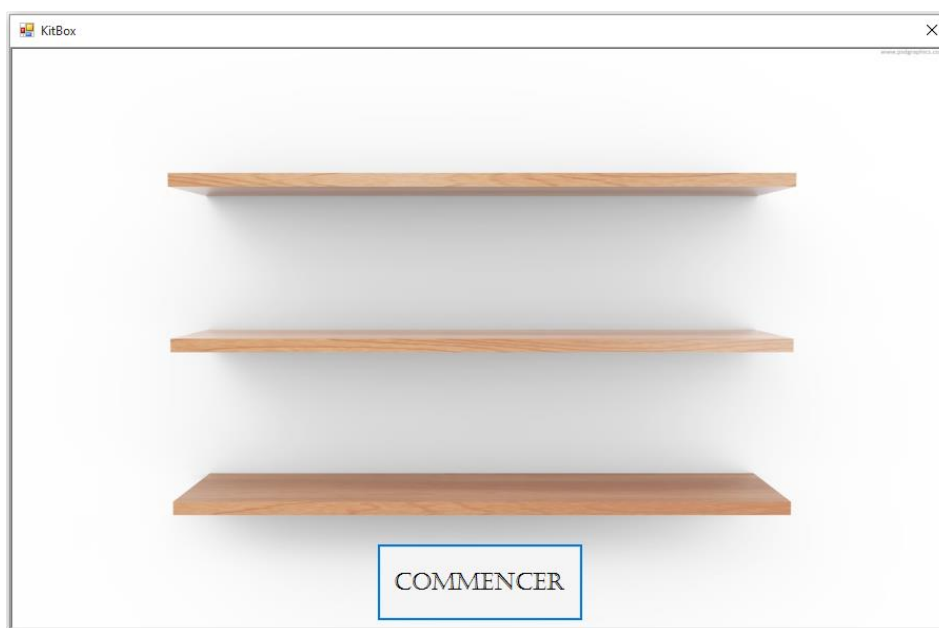
- On remplacera Insert par le nom de la table.
- On remplacera Column par les noms de colonnes souhaité.
- On remplacera Values par la valeur souhaitée

Vu que les différentes fonctions SQL sont utilisés de nombreuses fois dans l'application (aussi bien que pour l'interface magasinier et l'interface V1\_kitbox), toutes ces fonctions ont été rassemblées dans un fichier appelé « functions\_sql.cs ». A chaque fois que l'on a besoin d'utiliser une fonction SQL, il nous suffit d'appeler les fonctions définies préalablement dans ce fichier.

## 6 INTERFACE CLIENT

### 6.1 PAGE D'ACCUEIL

Lorsque la borne est en attente d'un client pour une nouvelle commande, une page d'accueil est affichée. Lorsqu'on souhaite effectuer une commande, il suffit alors de cliquer sur « Commencer ».

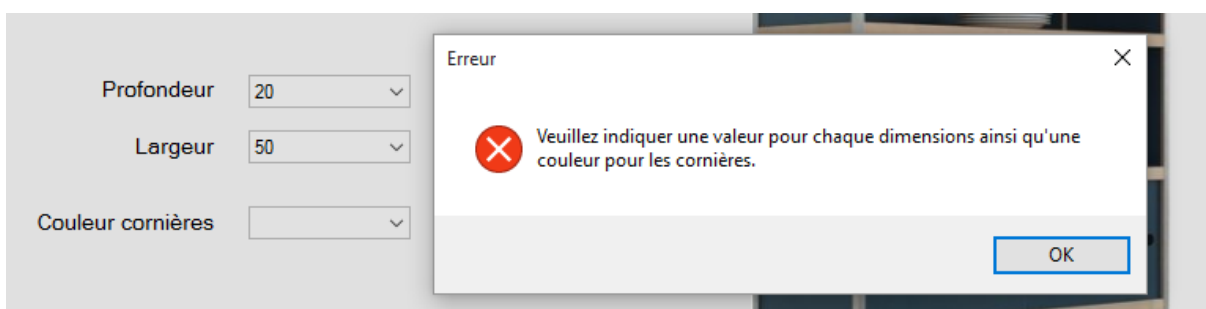


### 6.2 CHOIX DES CARACTERISTIQUES

Ensuite, l'interface demande d'encoder les caractéristiques principales de l'armoire recherchée. Ces choix seront alors fixés pour l'armoire qui va être ajoutée au panier. Le client doit donc bien connaître les dimensions de la surface dans laquelle il compte disposer sa future armoire. La couleur des quatre cornières n'étant pas personnalisables à hauteur de chaque étage de l'armoire, celle-ci doit être choisie une fois pour toute elle aussi.

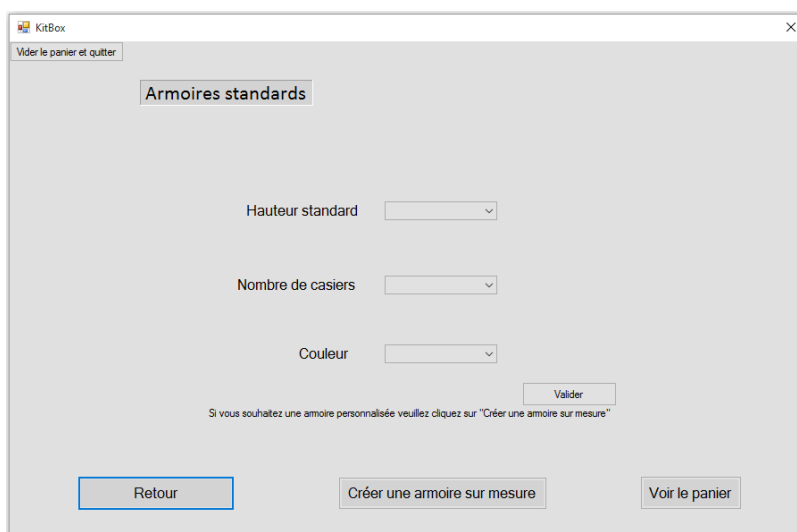


Ces paramètres étant indispensables à la constitution d'une armoire, une page popup apparaîtra si l'on tente de cliquer sur « continuer » alors que tous les paramètres n'ont pas été choisis.



### 6.3 ARMOIRES STANDARDS

Pour les personnes ne cherchant pas une armoire bien spécifique, la page suivante propose de choisir des armoires standards à couleur unie. Une fois la hauteur totale sélectionnée, la liste proposant le nombre d'étage souhaité est adaptée à cette dernière. Une fois la couleur choisie, en cliquant sur « valider » l'armoire est ajoutée au panier. Il est donc rapide de commander plusieurs armoires différentes aux caractéristiques généraux semblables.

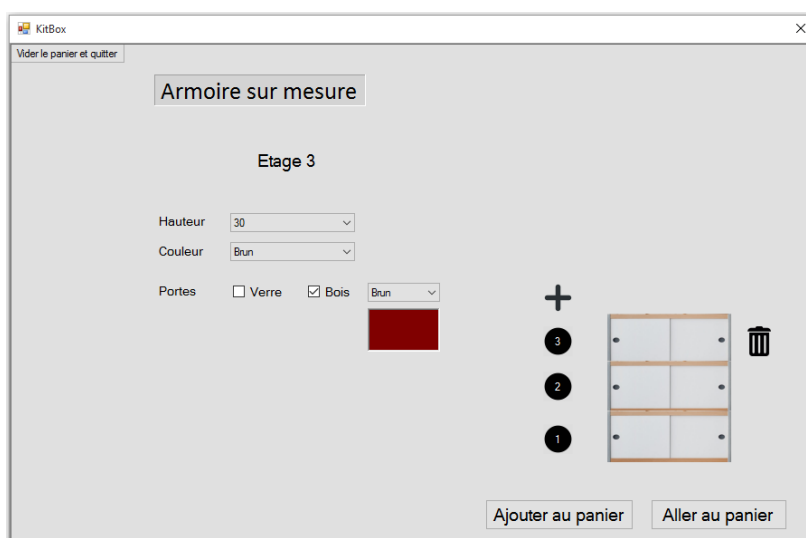


Si le client a des envies particulières quant aux couleurs, portes et hauteurs de chaque étage, il peut cliquer sur « créer une armoire sur mesure ».

## 6.4 ARMOIRE SUR MESURE

Cette page permet au client de commander une armoire totalement sur mesure. En effet, pour chaque étage, il faut choisir une hauteur, la couleur des panneaux ainsi que le type de porte. Si le bois est choisi comme matériau pour les portes de l'étage, une liste apparaît pour en choisir la couleur. Si aucun matériau n'est sélectionné, aucune porte ne sera fournie pour l'étage concerné.

À tout moment, l'utilisateur peut ajouter un étage ou supprimer le dernier en cliquant respectivement sur le + et la poubelle. Tous les étages souhaités peuvent ainsi être ajoutés dès le début en quelques clics sur le bouton +. Mais dans ce cas, il ne faut pas oublier de cliquer sur les numéros de tous les étages afin de choisir leurs caractéristiques un à un. Si toutefois des données sont manquantes pour un ou des étages, une page popup apparaît lorsqu'on clique sur « ajouter au panier » pour indiquer à quel étage des choix sont encore à faire.



## 6.5 PANIER

Avant de valider sa commande, le client a la possibilité de passer en revue toutes les caractéristiques qu'il a choisies pour ses armoires. La première colonne d'affichage permet de connaître les caractéristiques générales de l'armoire sélectionnée. Juste à droite, sont affichés les propriétés de l'étage sélectionné. Une fois tout cela vérifié, le client est amené à entrer ses coordonnées pour enfin valider et payer sa commande, ou encore continuer de remplir le panier.

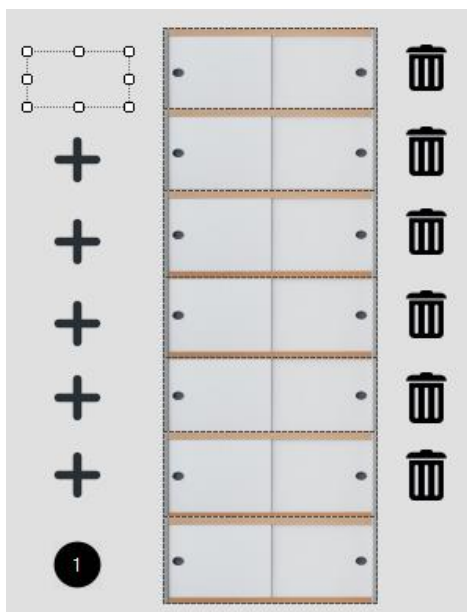
**Votre panier**

Vider le panier et quitter

Vos armoires	Etages de l'armoire	Entrez vos coordonnées :
<p>Armoire 1</p> <p>Prix : 400,704 €</p> <p>Hauteur : 144</p> <p>Largeur : 100</p> <p>Profondeur : 52</p> <p>Corières : Brun</p>	<p>Etage 1</p> <p>Etage 2</p> <p>Etage 3</p> <p>Etage 4</p> <p>Hauteur : 32</p> <p>Couleur : Brun</p> <p>Portes : - Matériau : Bois</p> <p>- Couleur : Brun</p>	<p>Prénom : <input type="text"/></p> <p>Nom : <input type="text"/></p> <p>Prix total : 400,704 €</p> <p>Disponibilité : Dans une semaine</p> <p>Continuer les achats</p> <p>Valider et payer</p>

## 6.6 CODE

Dans l'ensemble, le code de l'interface est relativement simple à comprendre. La partie légèrement plus complexe à cerner est celle de la constitution d'une armoire sur mesure.



En fait, tout est une question de visibilité. Lorsque le client arrive sur le User Control, la majorité des boutons et des panneaux contenant les images des différents étages sont cachés. Seuls le boutons « + » et « 1 » sont visibles avec l'image du premier étage. L'armoire est entourée de deux colonnes de boutons. Chaque colonne correspond à un type d'événement. Un click sur un boutons de gauche appellera donc la fonction « button\_click » alors que ceux de droite appelleront la fonction « btn\_delete\_Click ».

Ainsi, lorsqu'on clique sur le « + » pour ajouter un deuxième étage, la fonction analyse le texte affiché par ce bouton. Puisqu'il est vide lorsque l'image de fond est le symbole « + », le casier 2 et son bouton « delete » passent à l'état visible. Le bouton « delete » de l'étage du dessous, si ce n'est pas le premier étage, est alors caché. L'image de fond du « + » devient un rond noir et le texte affiché prend la

valeur du numéro de l'étage.

Afin de pouvoir continuer à ajouter des casiers, le bouton du dessus devient alors visible avec le « + »

comme image de fond.

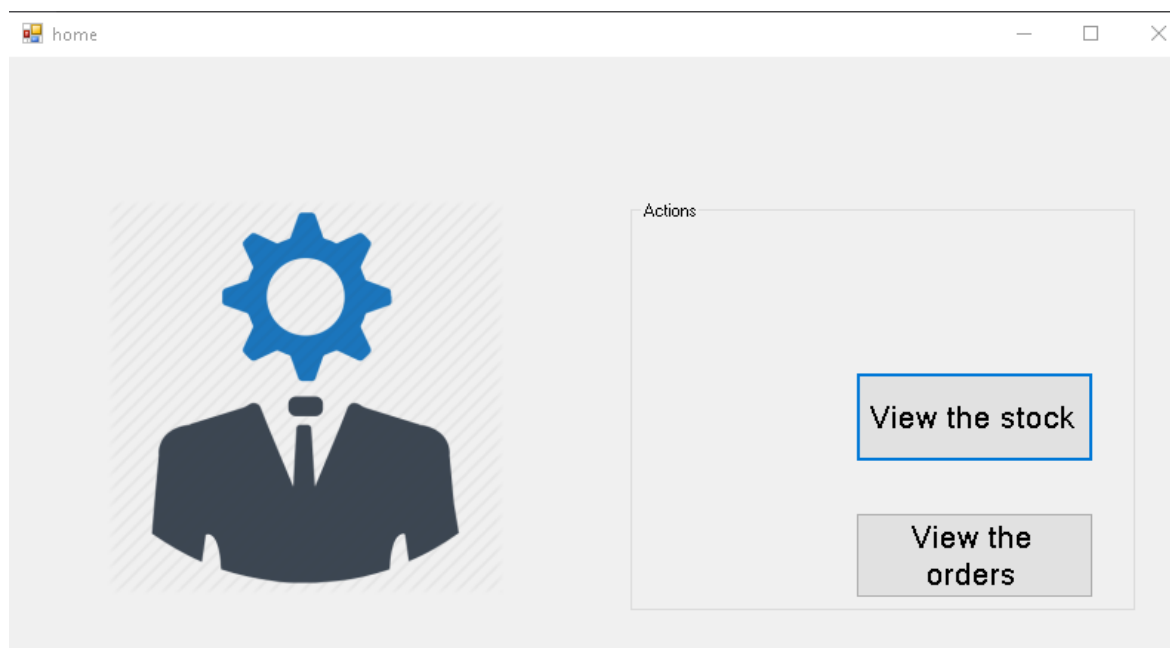
Quand on clique sur le bouton « delete » du dernier étage, c'est l'opération inverse qui est effectuée.

Afin de simplifier l'utilisation des objets de type Cabinet, ceux-ci ne sont créés qu'une fois que le client ajoute son armoire au panier. Pour cela, nous utilisons des listes qui contiennent les propriétés choisies par le client pour chaque étage de casier. C'est pour modifier ces listes qu'une variable `BoxIndex` est utilisée. Un click sur un bouton de gauche affichant un numéro d'étage changera cet index afin que les ComboBox affichent les derniers choix qui ont été fait pour cet étage sélectionné.

Avec la couleur des cornières et les dimensions de la base de l'armoire passées en paramètres au User Control, on possède enfin toutes les données nécessaires à la création d'un objet Cabinet complet.

## 7 INTERFACE MAGASINIER

Le magasinier a accès à une interface qui lui est propre. Il lance l'application « `interface_magasinier` » et tombe sur cette page d'accueil :



A partir de là, il a 2 choix :

- Accéder au stock
- Accéder aux commandes

## 7.1 ACCES AU STOCK

Après avoir cliqué sur « View the stock », on a accès à cette page :

The 'Stock' application window displays a table with the following columns: idPiece, Reference, CodeBarre, Dimensions, Hauteur, Profondeur, Largeur, and C. The table contains 9 rows of data for various 'Comieres' items. Below the table, there are two main control sections: 'Update' and 'Delete'. The 'Update' section has input fields for 'CodeBarre' and 'Enstock', with an 'Update' button. The 'Delete' section has an input field for 'CodeBarre' and a 'Delete' button. To the right of these sections are three buttons: 'Previous', 'Re-load the table', and 'Add a new item'.

idPiece	Reference	CodeBarre	Dimensions	Hauteur	Profondeur	Largeur	C.
1	Comieres	COR100BLDEC	100(h)decoupee	100	0	0	Bl
2	Comieres	COR100RDEC	100(h)decoupee	100	0	0	R
3	Comieres	COR100GLDEC	100(h)decoupee	100	0	0	G
4	Comieres	COR100NRDEC	100(h)decoupee	100	0	0	N
5	Comieres	COR108BL	3x32(h)	108	0	0	Bl
6	Comieres	COR108R	3x32(h)	108	0	0	R
7	Comieres	COR108GL	3x32(h)	108	0	0	G
8	Comieres	COR108NR	3x32(h)	108	0	0	N
9	Comieres	COR112BL	2x52(h)	112	0	0	Bl

Toutes les pièces disponibles dans notre stock sont stockées dans une base de données. Ici, nous importons cette bdd dans un tableau, avec toutes pièces ainsi que leurs caractéristiques. Le magasinier, à partir de cette interface, peut modifier le stock :

- Modifier le nombre d'unités pour une pièce avec la partie « update »
- Supprimer une pièce du stock avec la partie « delete »

Pour ce faire, il doit insérer le code barre de la pièce. Il peut aussi, tout simplement, double-cliquer sur le code barre d'une pièce afin que son code barre soit insérer automatiquement.

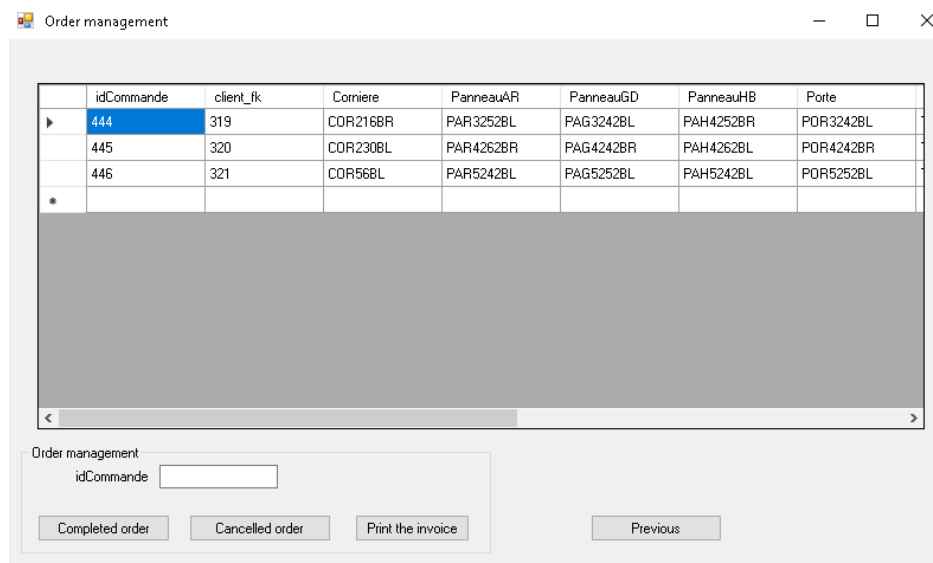
A droite, en cliquant sur « add a new item », on peut ajouter un nouvel élément à la base de données :

The 'New item' application window contains a form titled 'Data of the item'. The form has two columns of input fields. The left column includes: REF, CodeBarre, Dimensions, Hauteur, Profondeur, Largeur, Couleur, and EnStock. The right column includes: StockMinimum, PrixClient, NbPiecesCasier, PrixFourn1, DelaiFourn1, PrixFourn2, and DelaiFourn2. At the bottom right of the form is a blue 'Add to database' button. Below this button is a 'Previous' button.

Toutes les cases doivent être remplies afin d'éviter tout problème. En cliquant sur « Add to database », la pièce est ajoutée à la base de données.

## 7.2 ACCES AUX COMMANDES

A partir de la page « Home », en cliquant sur « View the orders », le magasinier à accès aux commandes.




A partir de cette interface, il peut faire 2 actions après avoir indiqué l'id de la commande dans « idCommande » :

- Terminer une commande après que le client a reçu son armoire, en cliquant sur le bouton « Completed order ». Après cela, le stock de pièces est décrémenté du nombre de pièces composant l'armoire. L'état de la commande sera « Terminée »
- Annuler une commande, en cliquant sur le bouton « Cancelled order ». Après cela, la commande sera effacée de la base de données.
- Imprimer la facture, en cliquant sur le bouton « Print the invoice ». Il faut d'abord qu'il crée un fichier.txt. Ensuite il clique sur « Print the invoice » et sélectionne le fichier .txt préalablement créé.



---

Exemple de facture générée par l'application :

 Facture.txt - Bloc-notes

Fichier Edition Format Affichage ?

KitBox, L'armoire sur mesure.

Client n°: 319

Commande n°: 444

Date et heure: 16/06/2019 22:18:35

Cornière : COR216BR

PanneauHB: PAH4252BR

PanneauGD: PAG3242BL

PanneauAR: PAR3252BL

Porte: POR3242BL

Tasseau: TAS27

TraverseAV: TRF52

TraverseGD: TRG42

TraverseAR: TRR52

Prix: En cours de négociation

A la prochaine

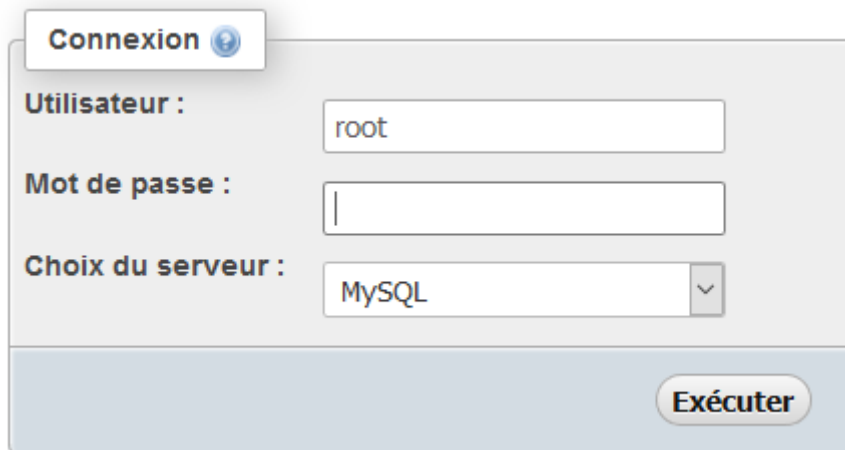
### 7.3 RELATION AVEC LA BASE DE DONNEES/FONCTIONNEMENT

Pour l'interface magasinier, la relation avec la base de données est beaucoup plus « proche » que pour l'interface utilisateur.

En effet, le magasinier a un rôle d'administrateur. De ce fait, il a beaucoup plus de liberté pour agir sur la base de données en ajoutant, supprimant et modifiant des valeurs directement à partir de l'interface.

## 8 MODE D'EMPLOI

1. Démarrer les services de « Wampserver ».
2. Ouvrir le navigateur.
3. Aller sur localhost/phpmyadmin
4. Connecter vous avec root et sans mot de passe.



5. Créer une database nommé kitbox\_database



6. Aller sur l'onglet importer

### Fichier à importer :

Le fichier peut être compressé (gzip, bzip2, zip) ou non.

Le nom du fichier compressé doit se terminer par **[format].[compression]**. Exemple : **.sql.zip**

Parcourir les fichiers :  kitbox\_database.sql (Taille maximale : 128Mio)

Il est également possible de glisser-déposer un fichier sur n'importe quelle page.

Jeu de caractères du fichier :

---

7. Sélectionner le fichier `kitbox_database.sql` disponible dans le dossier du projet.

8. Démarrer l'application :

- Pour l'interface client, aller sur le dossier « V1\_KitBox » et démarrer « V1\_KITBOX.exe - Raccourci ».
- Pour l'interface magasinier, aller sur le dossier « interface\_magasinier » et démarrer « Interface\_magainier.exe - Raccourci ».

## 9 CONCLUSION

---

### 9.1 BILAN DES OBJECTIFS

Dans son ensemble le bilan des objectifs est nuancé, nous sommes en mesure de réaliser toute la commande d'une seule armoire standard et d'enregistrer cette commande dans la base de données afin d'y accéder dans l'interface magasinier pour la manipuler. Mais nous rencontrons des problèmes lorsque nous souhaitons commander plusieurs armoires ou que nous créons une armoire personnalisée. L'interface magasinier quant à elle, permet de rajouter aisément des pièces de son côté et gérer le stock grâce à l'application développée.

Lors de la création d'une armoire personnalisée, quelques problèmes subsistent lors de l'enregistrement de la commande dans la base de données ce qui ne nous permet pas d'assurer que la création d'armoire personnalisé se fasse dans la base de données. Nous créons toutefois l'objet de type « armoire » et remplissons les caractéristiques de ces étages, l'objet est donc accessible et manipulable grâce aux méthodes de sa classe.

Malgré le fait que notre application soit fonctionnelle, la capacité qu'a celle-ci d'interagir avec la base de données est donc limitée, le client n'a par exemple pas l'occasion d'être informé directement lors du passage de sa commande si toutes les pièces sont encore disponibles et doit attendre la fin de son choix pour que le magasinier l'en informe. En effet, gérer l'intelligence du programme communiquant avec la base de données ne fut pas aisé et notre manque de maîtrise ne nous a pas permis de remplir toutes les fonctionnalités que nous voulions implémenter.

L'interface client est relativement simpliste visuellement et nous aurions aimé pouvoir rendre le tout un peu plus beau pour le client en intégrant une personne supplémentaire à l'équipe afin que celle-ci se consacre uniquement à l'aspect « graphique » de l'application.

Etant donné le temps limité pour réaliser l'application dû à la période d'immersion en entreprise des différents collaborateurs du projet, nous avons donc donné, dans la mesure du possible la priorité aux fonctionnalités qui nous semblaient les plus primordiales afin que les commandes et la gestion de stock de la société KitBox soient informatisés. C'est pour cela par exemple, qu'aucun algorithme ne propose des commandes de pièces toutes faites basées sur les ventes des derniers mois.

## 9.2 PISTES D'AMELIORATIONS

Comme mentionnez précédemment, l'aspect visuel pourrait être revu pour rendre l'application la plus agréable pour le client ainsi que pour le magasinier.

Le programme peut également être amélioré notamment dans l'organisation du code et l'optimisation de celui-ci. Ainsi, refactorer le code pourrait simplifier grandement la compréhension du programme et la clarté du code dans le cas où une nouvelle équipe de développement reprendra le projet. L'ajout de davantage commentaires permettrait également d'aller dans ce sens.

Il serait intéressant de pouvoir supprimer les armoires d'une commande une fois que le client se trouve dans le « Cart » où les infos de sa commande sont reprises.

Une des fonctionnalités pourrait offrir la possibilité au client de modifier sa commande une fois que celui-ci se trouve dans le « UC\_Cart ». Ainsi, si le client se rend compte qu'il voulait ajouter un casier supplémentaire, il pourrait retourner dans un « UC\_Custom » pour pouvoir en ajouter un.

Le fait de pouvoir personnaliser entièrement l'armoire à l'aide d'une visualisation 3D pouvant être vue sous différents angles donnerait également plus l'envie aux clients de venir passer une commande d'armoire et augmenterait certainement les ventes de la société. Mais dans un premier temps, la possibilité de supprimer n'importe quel étage de l'armoire pourrait s'avérer très pratique. Puisque, actuellement, l'ajout et la suppression d'étage se fait comme avec une pile.

Un point important concerne le nombre d'itérations et de réunions avec le client de KitBox. Nous devrions organiser plus de réunions avec celui-ci afin de savoir quelles sont les priorités qu'il souhaite en lui faisant tester l'application au fur et à mesure de son développement.

## 9.3 OUTILS UTILISES

Les différents outils utilisés sont :

- **Trello** – qui nous a offert la possibilité de répartir les tâches à réaliser par plusieurs personnes. De cette manière, l'équipe pouvait voir l'avancée du projet et se représenter d'une meilleure façon le travail qu'il reste à accomplir.
- **GitHub** – avec lequel nous avons mis à jour le programme par des commits réguliers. Cette plateforme permet le partage du code afin que chacun puisse participer dessus en tant que « Collaborateur ».
- **Groupe Facebook** – avec lequel on pouvait échanger entre membres pour pouvoir poser toutes nos questions lorsque nous travaillions à distance l'un de l'autre.

## 9.4 COLLABORATION DANS LE GROUPE

La collaboration au sein du groupe ne fut pas toujours simple. Nous avons choisi de répartir la tâche entre l'implémentation des classes et la réalisation de l'interface client par Julien et Yannis et de l'autre côté c'est Rudy et Deniz qui étaient en charge de réaliser l'interface magasinier et la communication générale avec la BDD.

Cependant, le manque de concertation entre nous nous a ralenti et le fait d'avoir développé des parties à part, a pu parfois nous compliquer la tâche lors de la fusion de celles-ci. Nous pensons

notamment à la communication avec la base de données qui ne fut pas évidente pour l'interface client.

Nous avons donc parfois effectué du développement inutile comme pour les fonctions SetCode() (voir diagramme de classe) car nous ne savions pas de quelle manière les requêtes SQL allaient être utilisées.

Il était aussi difficile de comprendre comment chacun de nous se représente la façon dont devrait fonctionner certaines des fonctionnalités du programme avant de commencer à les développer. A cause de cela, nous n'avions donc pas toujours les mêmes attentes et cela pose problème car nous programmions en fonction de celles-ci.

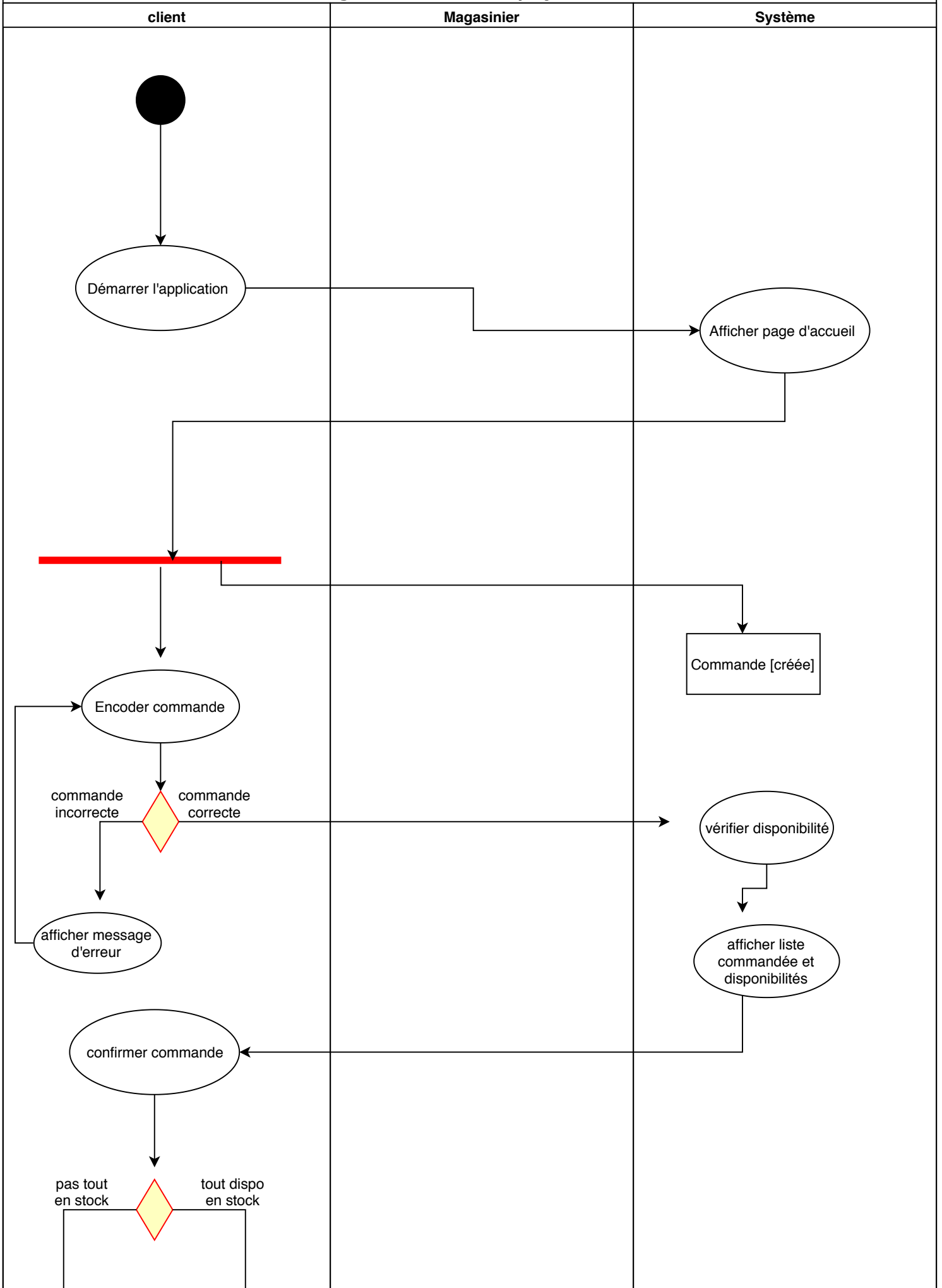
## 10 ANNEXES

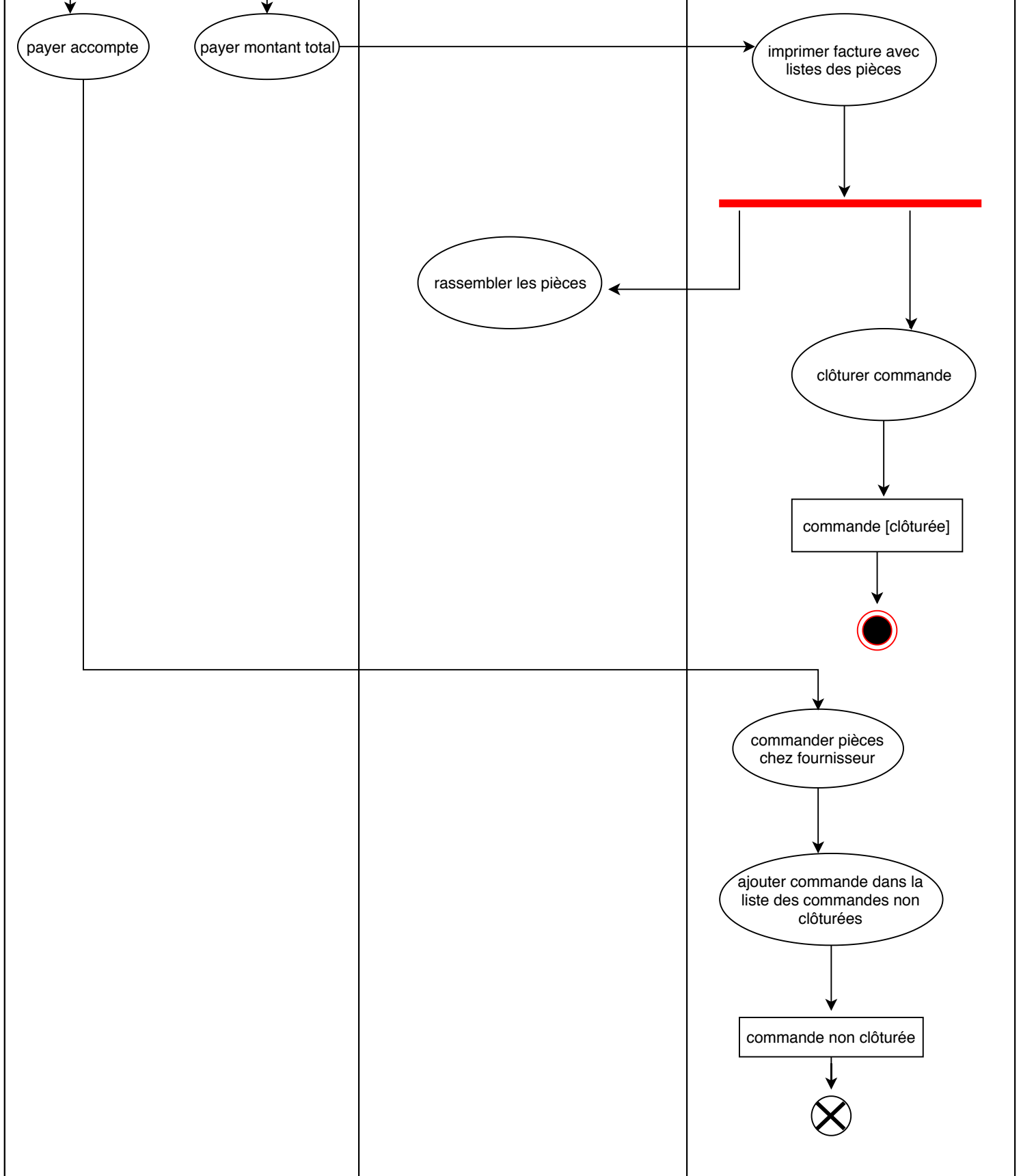
---

### 10.1 ANNEXE 1 : DIAGRAMME D'ACTIVITE

Voir pages suivantes

# Diagramme d'activité du projet KitBox







## 10.2 ANNEXE 2 : DIAGRAMME DES CAS D'UTILISATIONS

Voir page suivante

