

**Nom de l’entreprise : Valkorion.h**

**Nom du projet : Nebula**

**Adresse : 204 Boulevard Godard**

**Tel : 0606060669**

**Email : contact@valkorion-h.fr**

|  |
| --- |
|  |

DOCUMENTATION TECHNIQUE

Table des matières

[I. BASE DE DONNÉES : 3](#_Toc121247528)

[MCD & MLD : 3](#_Toc121247529)

[II. COMMUNICATION CODE & BDD : 4](#_Toc121247530)

[Data Context : 4](#_Toc121247531)

[Contrôleur : 5](#_Toc121247532)

[III. LE CODE DU LOGICIEL: 5](#_Toc121247533)

[Arborescence : 5](#_Toc121247534)

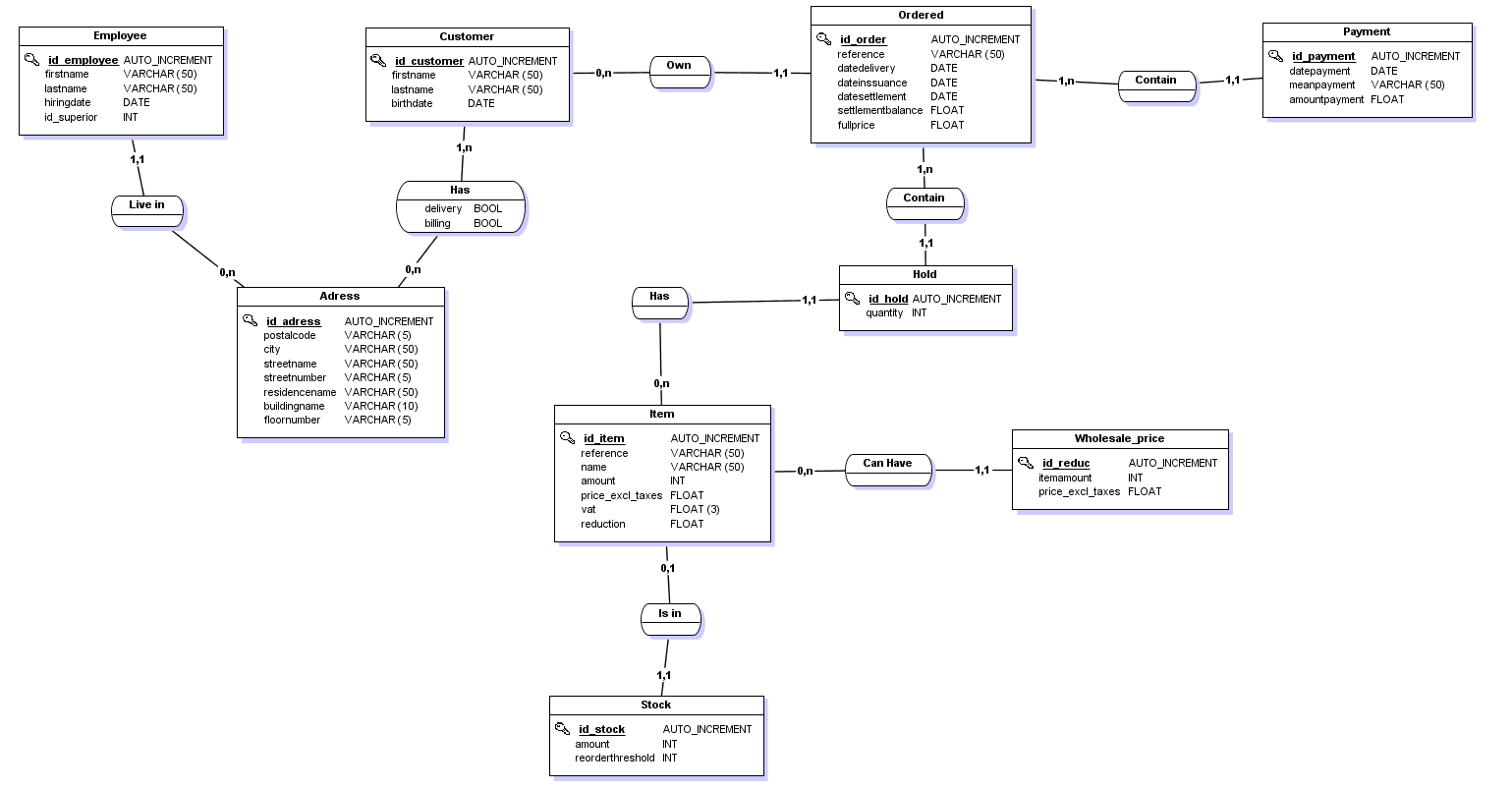
[Formulaire : 6](#_Toc121247535)

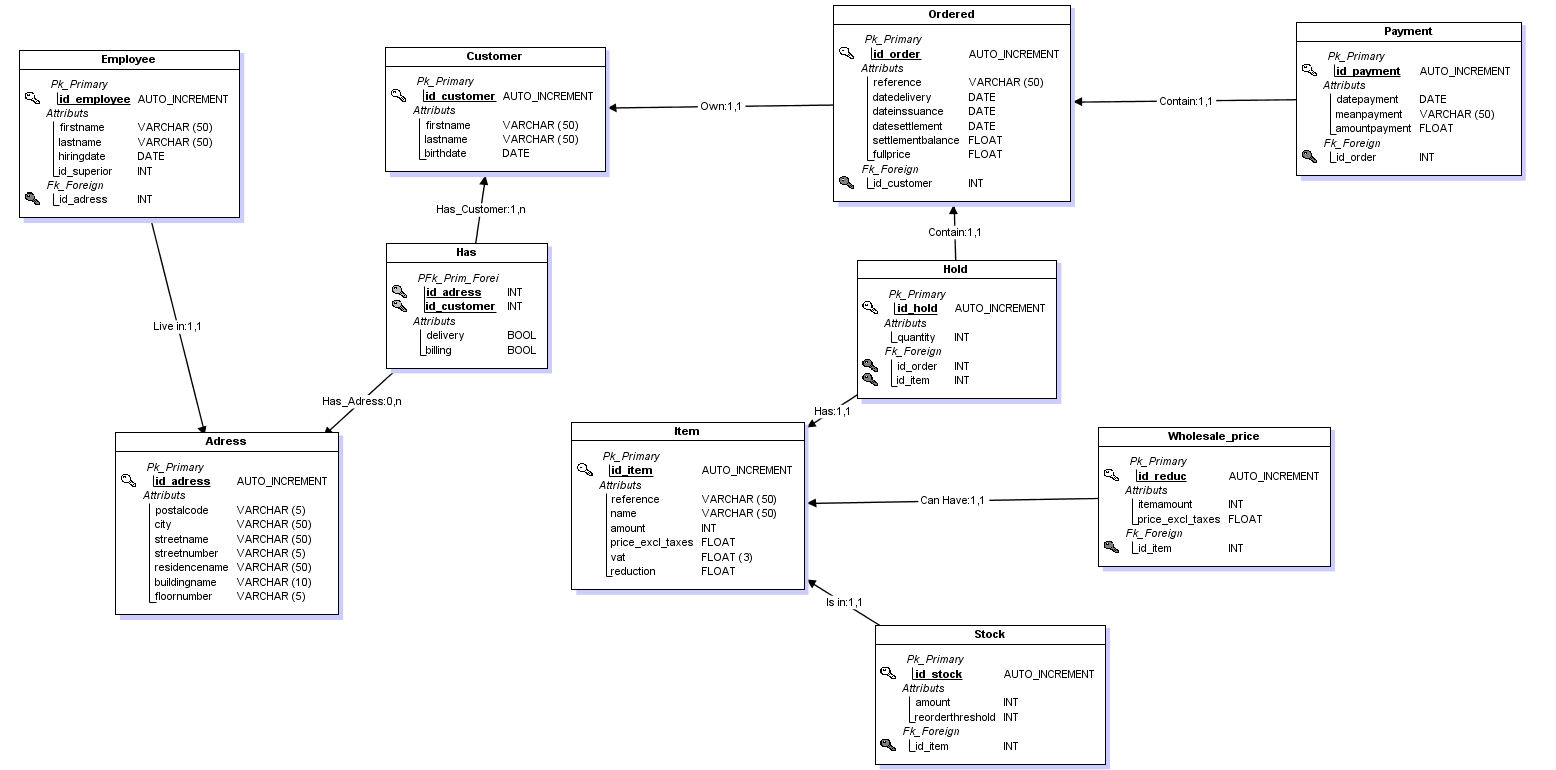
[Modèle : 7](#_Toc121247536)

# BASE DE DONNÉES :

La base de données que nous utilisons est une base de données SQL Server. Pour les besoins de démonstration nous hébergerons la base de donnée en locale. Pour l’utilisation réel il sera cependant tout à fait possible (même fortement recommandé d’utiliser une base de données hébergé sur un serveur).

## MCD & MLD :

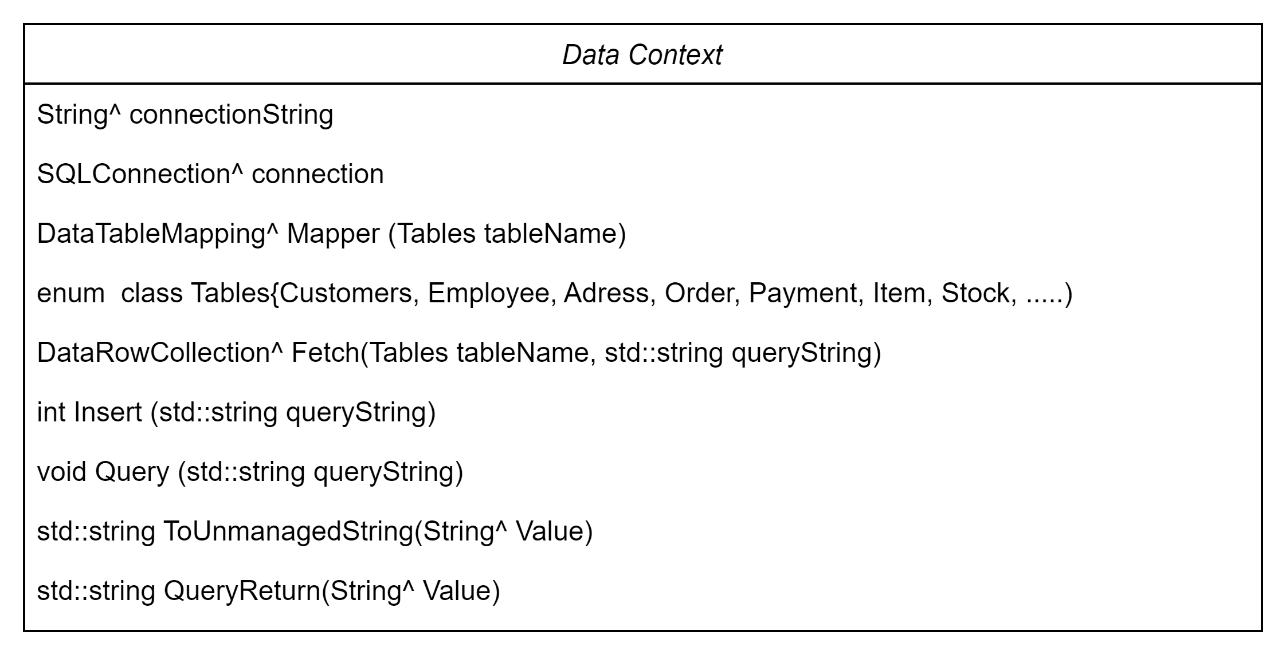




Nous avons créé notre base de données en respectant la forme normale.  
Nous stockons dans la table « Hold » la quantité d’un article commandé afin d’éviter l’apparition de plusieurs lignes.  
Il sera simple de créer de nouvelles colonnes dans une table, il faudra juste faire attention aux redondances.

# COMMUNICATION CODE & BDD :

## Data Context :



Voici le diagramme de classe de « Data Context ». Cette classe permet la liaison directe avec notre BDD.  
Elle contient la « connectionString » qui sert à définir à quelle BDD le logiciel se connectera, et les différentes requêtes classiques (« Fetch », « Insert », « Query », « QueryReturn ») qui seront appelées dans nos différents contrôleurs nommés « Service ». Cette classe contient également Mapper permettant de réorganiser les données obtenues via les différentes requêtes, ainsi qu’une méthode « ToUnmanagedString » permettant de convertir les string managés en string non managés. Cette méthode est effectivement utile car notre programme est entièrement managé afin que la gestion de la mémoire soit optimale.

## Contrôleur :



Voici le diagramme de classe de notre contrôleur d’adresse. Celui-ci permet d’effectuer des requêtes en rapport avec la table « Adress ».  
Ce contrôleur contient les méthodes classiques. Certains contrôleurs peuvent en contenir plus selon le besoin, il sera donc simple pour vous de créer des méthodes. En effet, chaque contrôleur appelle des méthodes de la classe « DataContext » et, avec les méthodes déjà présentes dans « DataContext », il vous sera facilement possible de créer de nouvelles requêtes dans votre contrôleur.

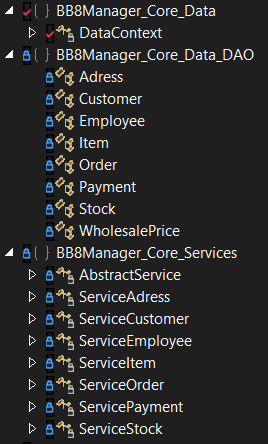
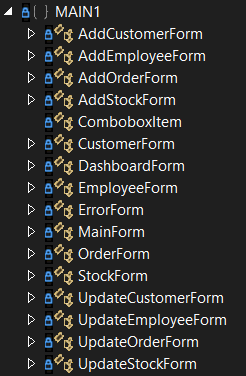
# LE CODE DU LOGICIEL:

Le code est un code C++, utilisant pour la partie GUI, CLR/CLI car ce sont les frameworks les plus utilisés pour faire une interface graphique avec C++. Il faudra également avoir une licence du framework Bunifu à jour car nous avons choisis d’utiliser ce framework pour gérer tous les affichages des grilles de données (DATAGRIDVIEW).

Note :

Nous avons dans notre cas utilisé l’environnement de développement Visual Studio dans sa version 2022). Et la version 18 de SQL Server.

## Arborescence :

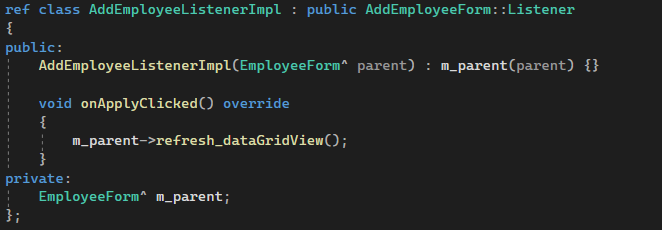
« BB8Manager\_Core\_Data\_DAO » contient nos différents modèles.  
« BB8Manager\_Core\_Services » contient nos différents contrôleurs.  
« MAIN1 » contient nos différents formulaires.

## Formulaire :

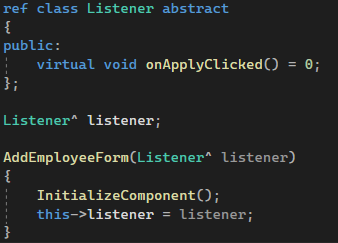
Il existe 4 types de formulaire :

* Le form principal « MainForm » qui contient le menu déroulant ainsi que l’emplacement des différents panels à afficher.
* Le form panel (par exemple « EmployeeForm ») qui vient se fixer sur le main dans l’emplacement adéquat
* Le form d’ajout/edition qui s’ouvre lorsque l’utilisateur souhaitera, comme son nom l‘indique, ajouter ou éditer.
* Le form d’erreur qui s’ouvre lorsque l’utilisateur effectue mal une action. Un message d’erreur est joint sur ce dernier.

Concernant les formulaires d’ajout et d’édition nommé « AddXXXForm » et « UpdateXXXForm », où XXX correspond au nom du modèle (par exemple « AddCustomerForm »), il était nécessaire de créer une classe listener afin de permettre la communication entre le Form principal et le Form d’ajout ou de modification.   
Par exemple notre « EmployeeForm » contient :



Et notre « AddEmployeeForm » contient :



Cela nous permet donc d’appeler une méthode, en l’occurrence « refresh\_dataGridView » qui va nous permettre d’actualiser les données de la datagridview lorsqu’un utilisateur sera ajouté.

## Modèle :

Une image contenant texte

Description générée automatiquement

Un modèle est constitué d’attribut correspondant aux colonnes de la table auquel il réfère dans la BDD, cela comprend aussi bien les primary key que les foreign key. Notre classe contient également un constructeur surchargé ainsi que des Setter et Getter pour chaque attribut.  
Si vous souhaitez ajouter des tables par la suite, il faudra donc penser à créer un modèle correspondant.