CARGOFLOW

Projet de programmation

Martet Antoine
antoine.martet@eduvaud.ch

Clot Ian
ianalain.clot@eduvaud.ch

SI-CA2a

14 janvier 2024

Table des matières

1. Introduction

1.1. Cadre, description et motivation

CargoFlow est un projet de programmation qui a pour objectif de nous apprendre à gérer un projet de développement tout en nous apprenant à gérer un projet de sa planification à sa concrétisation. CargoFlow est un logiciel permettant la gestion de la logistique dans le sens large du terme dans une entreprise de logistique. Il permet de gérer le stock, les client·es, les employé·es, les transporteurs, les entrepôts ainsi que les livraisons.

Ce projet respecte les conditions du module et nous intéresse dans la mesure où il est relativement utile et concret : on pourrait être amené, un jour, à travailler sur un projet similaire en entreprise.

Ce projet est développé en C# et se base sur des Windows Forms. Il inclut également un serveur local MySQL hébergeant la base de données contenant les informations nécessaires à la gestion du stock par le logiciel.

Le logiciel est prévu pour être utilisé localement sur un ordinateur individuel mais rien n'empêcherait d'héberger la base de données sur un serveur en ligne.

Tous les documents relatifs au projet se trouve dans la branche main du repository du projet sur GitHub à cette adresse : https://github.com/AntoineMartet/CargoFlow/tree/main

1.2. Organisation

Les rôles ont globalement été réparties ainsi :

	Ian	Antoine
Use cases	30 %	70 %
MCD	50 %	50 %
MLD	50 %	50 %
Maquettes	100 %	0 %
Fichiers SQL	90 %	10 %
Code C#	10 %	90 %
Tests icescrum	50 %	50 %
Documentation	50 %	50 %

Le projet est suivi et évalué par M. Julien Ithurbide.

1.3. Objectifs

Notre page de login doit interagir avec la base de données, afin d'avoir accès à l'adresse email et au mot de passe des personnes (les employé·e) ayant accès à notre logiciel. Si les informations d'identification sont correctes, la page d'accueil de notre logiciel doit s'afficher.

Sur le haut de la page d'accueil du logiciel, des onglets permettent de choisir quelles sont les informations que l'on souhaite afficher. Lors du chargement de l'onglet sélectionné, le logiciel interagit avec la base de données pour afficher les données nécessaires.

Lorsqu'on se trouve sur un de ces onglets, par exemple le transporteur, la liste des éléments que le client a exigée, s'affiche dans le DataGridView. A droite de cette liste, quatre boutons permettent d'effectuer les opérations CRUD. Le bouton Détails permet d'afficher toutes les caractéristiques du transporteur se trouvant dans la BDD. Le bouton Ajouter permet d'ajouter un nouveau transporteur dans la BDD. Le bouton Modifier permet de modifier un transporteur existant déjà dans la BDD. Le bouton Supprimer permet de supprimer un transporteur de la BDD.

1.4. Planification

Nous avons décidé de diviser le temps que nous avions à disposition, soit du 6 novembre au 19 janvier, en 5 sprints. Le détail de nos activités durant cette période se trouve sur iceScrum. Sur demande de M. Ithurbide, nous avons aussi effectué un diagramme de Gantt, afin de schématiser lors de la première semaine du module, le déroulement approximatif de notre projet sur l'ensemble du trimestre.

Concernant la division du temps à notre disposition, comme dit ci-dessus, nous avons décidé d'effectuer 5 sprints. Aux 4 premiers sprints ont été attribuées deux semaines chacun. Concernant la dernière semaine à disposition, nous avons préféré créer un 5e sprint plutôt que d'allonger le 4e d'une semaine. La raison de cette décision a été que cette dernière semaine était assez différente des autres, dans le sens où elle était prévue essentiellement pour la fin de l'écriture de la documentation et de la préparation de la présentation. L'essentiel de l'implémentation serait déjà effectuée.

2. Analyse

Nous avons été confrontés principalement à deux difficultés :

- Le manque de connaissances sur la logistique dans un environnement de travail réel : à quoi correspondent chaque code barre sur un colis ? Comment est effectué le tracking d'un colis ? Est-ce qu'il existe une base de données partagée entre un transporteur et l'entreprise qui requiert ses services pour suivre ses colis ? Nous avons évidemment cherché des informations sur internet mais nous avons également tenté d'en apprendre plus en faisant jouer nos contacts. Nous avons en effet pu discuter avec le mari d'une amie qui est logisticien mais sa spécialisation ne lui permettait pas de nous aider pour répondre précisément à ces questions. Nous nous sommes aussi rendus à l'Imprimerie de Sainte-Croix avec laquelle un autre ami a des contacts, mais à nouveau ils n'ont pas pu nous aider : ils n'ont pas un inventaire suffisamment grand à gérer pour utiliser des systèmes de gestion des stocks.
- Le juste milieu à trouver entre la complexité réelle du sujet et la faisabilité de l'implémentation du projet à notre niveau.

2.1. Use cases et scénarios

Comme indiqué plus haut, nous avons détaillé les uses cases sur un document Excel. Il se trouve sur Github dans le dossier Documentation sous le nom CargoFlow_Gantt_UseCases.xlsx.

2.2. Maquettes

Nous avons conçu nos maquettes avec Balsamiq. Chaque onglet de notre logiciel a été créé sur une nouvelle fenêtre, afin de bien distinguer les différences entre les pages malgré leurs similitudes. Le fichier de maquettes se trouve dans le dossier Documentation sous le nom maquette_cargoFlow.bmpr.

2.3. MCD

Le MCD se trouve sur Github dans le dossier Documentation sous le nom CargoFlow_MCD_v3.drawio.

3. Implémentation

3.1. Choix techniques

3.1.1. Base de données

Pour faire fonctionner notre BDD, nous avons utilisé deux logiciels. Le logiciel jouant le rôle de serveur est MySQL server et celui jouant le rôle du client est HeidiSQL. Concernant les données, elles sont pour la plupart modifiables dans CargoFlow mais nous avons également jugé pertinent de créer manuellement un ensemble d'enregistrements afin de mieux travailler par la suite et pour pouvoir présenter le logiciel au client. Il a fallu s'assurer que les données aient une certaine cohérence, tant au niveau des champs qu'au niveau des clés étrangères. La BDD est hébergée localement sur l'ordinateur de la personne utilisant le logiciel.

3.1.2. Documentation

Tout au long du module, nous avons pris des notes sur les éléments devant figurer dans la documentation de notre projet. Nous avons par la suite effectué une mise en commun de nos notes, afin de les inclure dans cette documentation. Ce document représente le document principal de notre documentation. Il fait référence à d'autres documents qui, pour des questions de praticité et de lisibilité, se trouve dans des documents annexes : maquettes, Gantt, MCD, MLD, fichier de requêtes SQL, uses cases.

3.1.3. Développement

Notre outil de développement a été le logiciel Visual Studio. Il nous a permis d'utiliser les Windows Forms, de gérer les erreurs de compilation et de build, ainsi que, de manière plus générale, d'implémenter le code. En bref, il a été le centre de notre projet. De plus, nous avons utilisé git et Github pour se transmettre et stocker les différentes versions des fichiers.

3.2. Conventions

3.2.1. Commentaires

Le code contient des commentaires. Ceux-ci sont écrits au-dessus de la ligne ou des lignes de code concernées.

3.2.2. Conventions de nommage

Les conventions sont les mêmes que celles utilisées habituellement en C# au CPNV :

- Classes : PascalCase
- Objets : camelCase
- Attributs privés d'une classe : camelCase
- Propriétés d'une classe : CamelCase
- Méthodes publiques d'une classe : PascalCase
- Méthodes privées d'une classe : PascalCase

Contrôles des WinForms :

- Form : frm
- Button : btn
- FlowLayoutPanel : flp
- Label : lbl
- TextBox : txt
- DataGridView : dgv
- ComboBox : cmb

3.3. MLD

Le MLD se trouve sur Github dans le dossier Documentation sous le nom CargoFlow_MLD_v3.mwb. Voici la justification de certains éléments de ce MLD :

- Les attributs barcode dans les tables articles et deliveries sont de type BIGINT car les code-barres standard EAN ont 13 chiffres, ce qui va au-delà de la valeur maximale pour un int (2,147,483,647).
- Les attributs streetNb et postalCode pour les tables warehouses et clients sont de type VARCHAR car il pourrait y avoir des numéros de rue comme « 12b » et car certains pays ont des codes postaux incluant des lettres (Royaume-Uni, Pays-Bas…).
- Il y a deux liaisons entre les tables deliveries et warehouses car la table deliveries peut avoir deux Foreign Key référençant la table warehouses : warehouse_origin_id (obligatoire) et warehouse_destination_id (optionnel).
- L'attribut phoneNumber dans la table carriers est de type VARCHAR car il est possible que des utilisateurs souhaitent rentrer ces numéros sous des formes pas seulement numériques. Exemple : +41771235476, 024/123 45 67, 1-800-555-5555, etc.
- De nombreuses Foreign Key peuvent être NULL afin de pouvoir supprimer les enregistrements des tables liées.

3.4. Organisation et fonctionnement du programme

Le code de CargoFlow repose sur des Windows Forms (situées dans un projet CargoFlowForms) et des classes métier personnalisées (dans un projet CargoFlowMgmt). Nous présenterons ici brièvement la classe DBConnection qui interagit avec la BDD, puis les Windows Forms, puis les classes métier autres que DBConnection.

Le code du programme est commenté. Vous trouverez également la documentation générée par Doxygen sur le repository GitHub. Pour accéder à la documentation, il faut ouvrir le fichier index.html pour chaque projet de la solution (CargoFlowForms et CargoFlowMgmt). Pour des raisons que nous ne parvenons pas à identifier, la documentation générée par Doxygen pour le projet CargoFlowForms est très parcellaire ; nous avons décidé de la laisser sur le repo GithHub mais vous suggérons de regarder directement le code pour avoir accès à toute la documentation.

Enfin, nous n'avons pas eu le temps d'implémenter toutes les fonctionnalités prévues dans les uses cases. Le CRUD est fonctionnel pour les onglets Transporteurs, Client.es et Employé.es mais pas pour les onglets Stock, Livraisons, Entrepôts et Articles. La fonction de tri n'a été implémentée que sur la colonne Nom pour les Transporteurs. Elle sert de proof-of-concept pour les implémentations sur les autres colonnes et tables. La fonction de filtre n'a pas été implémentée du tout par manque de temps.

3.4.1. Diagramme de classes

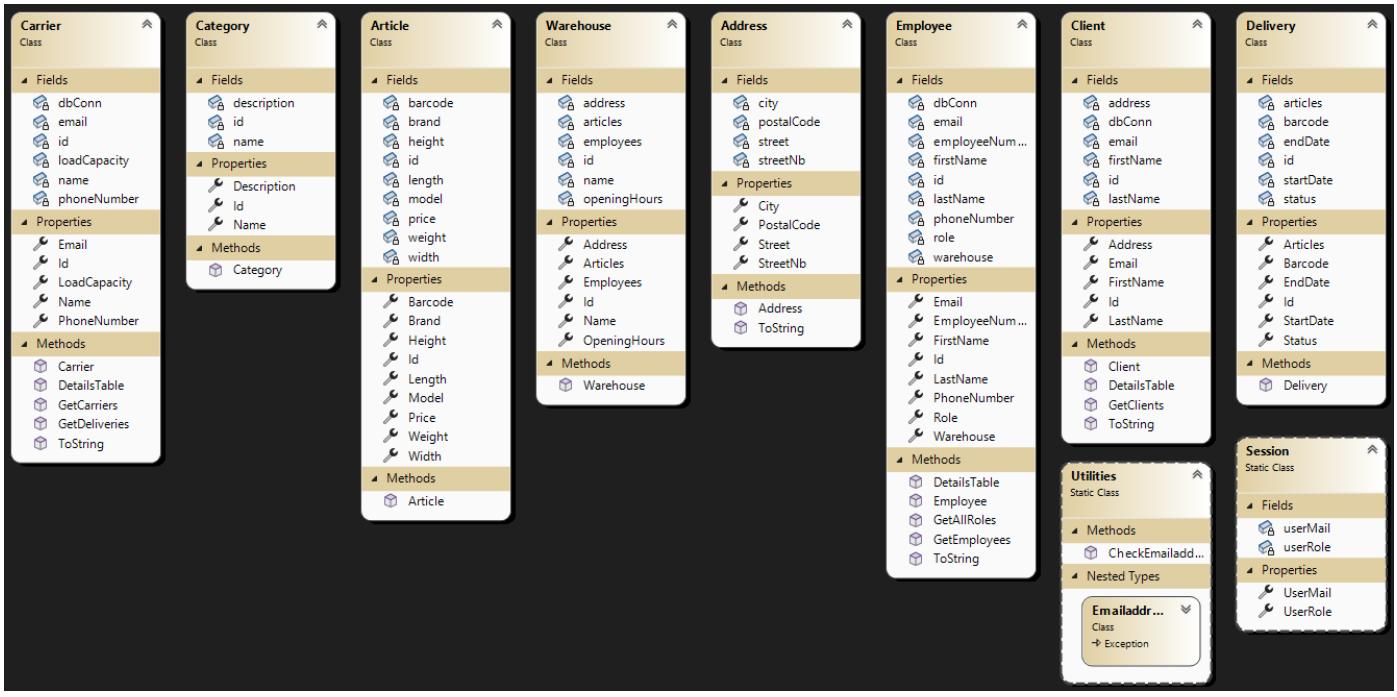

Figure 1 : Diagramme de classes

Il n'y a pas de relations d'associations car après réflexions, nous avons conclu que, étant donné que les modifications que nous apportions à nos données se faisaient via des requêtes SQL, les associations entre les tables n'était plus utiles (les associations se font via des INNER JOIN dans les requêtes SQL). Techniquement, nous pourrions récupérer et manipuler toutes

les informations de la base de données sans avoir à créer des instances de nos classes. On utiliserait alors des classes statiques et des méthodes statiques pour interagir avec DBConnection et la BDD.

Mais étant donné que nous n'avions pas encore ces connaissances au moment de la phase de conception du projet, nous nous sommes retrouvés avec des classes dynamiques. Nous les instancions puis en créons des listes que nous donnons comme source à la DataGridView (voir section 3.4.3).

3.4.2. DBConnection

La classe `DBConnection` est dédiée à l'établissement et à la fermeture de connexions avec la base de données ainsi qu'à l'exécution de différents types de requêtes. Nous avons essayé, dans la mesure du possible, de créer des méthodes d'exécution des requêtes les plus « généralistes » possibles, par exemple `GetAllRecords()` plutôt que `GetAllEmployees()`. Ces méthodes reçoivent en paramètres les requêtes SQL à effectuer et les éventuels autres paramètres nécessaires.

3.4.3. Windows Forms

3.4.3.1. FrmLogin

Le Form apparaissant en premier à l'exécution du programme est celui du Login. Il permet à l'utilisateur de se connecter à son compte s'il rentre correctement ses identifiants.

Les mots de passe des utilisateurs sont stockés dans la BDD dans une forme hashée (via l'algorithme MD5). Le mot de passe entré durant le login est donc hashé aussi et le hash est comparé à celui de la BDD. Ce Form est particulier dans la mesure où son instance est associée au lancement de l'application via la ligne `Application.Run(frmLogin)`. Ce faisant, le fermer met fin à l'exécution du programme. Pour cette raison, après que des identifiants corrects soient entrés et avant que le Form de l'accueil ne soit affiché, l'instance de `FrmLogin` est simplement cachée et non pas fermée.

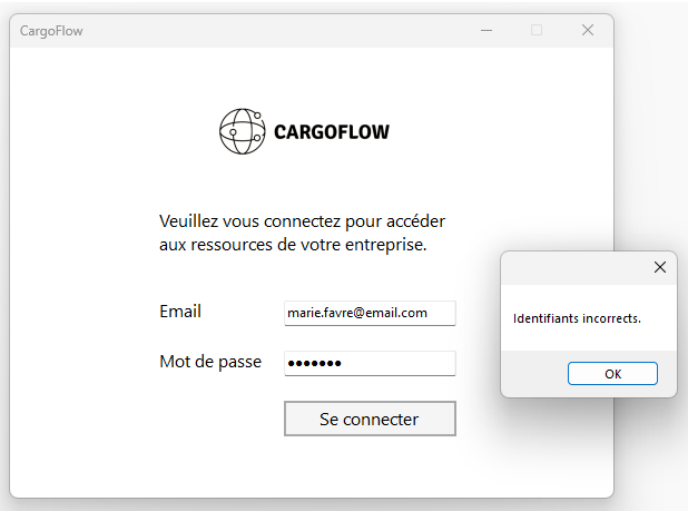

Figure 2 : FrmLogin après insertion de mauvais identifiants

Après un login réussi, l'adresse mail et le rôle de l'utilisateur sont stockés dans des variables de session (des attributs statiques de la classe statique `Session`). Le rôle est une information importante car il permet de savoir à quelles fonctionnalités l'utilisateur a accès. Nous n'avons malheureusement pas eu le temps de mettre en place des différences de fonctionnement du logiciel en fonction du rôle de l'utilisateur connecté.

3.4.3.2. FrmHome

Après une connexion avec les bons identifiants, l'utilisateur arrive sur le Form d'accueil. Son adresse mail et son rôle sont affichés. Depuis cette page, il peut accéder à tous les onglets « productifs » de l'application qu'il est autorisé de consulter et qui permettent de visualiser le contenu de la BDD et de le modifier. Cliquer sur un de ces onglets ferme l'instance de `FrmHome` et ouvre une instance de `FrmLists`. Depuis `FrmHome`, il est également possible pour l'utilisateur de se déconnecter. Les variables de session sont alors réinitialisées et l'instance de `FrmLogin`, qui était simplement cachée jusque-là, réapparaît.

Figure 3 : FrmHome

3.4.3.3. FrmLists

Ce Form permet de passer d'un onglet productif à l'autre (Clients, Employés, Transporteurs…) sans changer de Form. C'est le formulaire principal du programme, celui à partir duquel on peut réaliser les opérations CRUD classiques : Affichage, Ajout, Modification, Suppression.

L'affichage des données d'une entité de la BDD se fait à l'aide d'un contrôle nommé `DataGridView`. `DataGridView` permet d'afficher un set de données sous la forme d'un tableau et de gérer son état et son comportement via des propriétés et méthodes déjà implémentées. Après sa création, il faut lui assigner une source de données (`DataSource`) à afficher. Cette source peut être de différents types : `DataTable`, `DataSet`, `IList`, `IBindingList`… Nous avons choisi de lui donner comme source des listes d'objets, par exemple `List<Employee>`. Ce choix découle assez naturellement de la manière dont nous récupérons les données de la BDD. En effet, nous avons mis en place un certain nombre de classes représentant la majorité des entités présentes dans notre BDD et nous créons des listes d'objets avec les résultats de nos requêtes SQL. Ce passage des données brutes vers des classes permet de mieux respecter les principes de la programmation orientée objet et permet d'implémenter plus facilement des méthodes spécifiques à chacune de ces classes.

En plus de la `DataGridView`, `FrmLists` présente aussi 4 boutons : Détails, Ajouter, Modifier, Supprimer. Cliquer sur un de ses boutons déclenche un évènement lié à l'objet sélectionné dans la DataGridView :

- Cliquer sur Détails ouvre le Form `FrmDetails` qui affiche des informations supplémentaires (s'il y en a) liées à l'objet sélectionné.
- Cliquer sur Ajouter ouvre le Form `FrmAddUpdxxx` correspondant à l'objet xxx sélectionné avec les champs vides.
- Cliquer sur Modifier ouvre le Form `FrmAddUpdxxx` correspondant à l'objet xxx sélectionné avec les champs préremplis.
- Cliquer sur Supprimer demande une confirmation à l'utilisateur puis supprime ou non l'objet sélectionné. La suppression se fait d'abord dans la BDD, puis le contenu de la BDD est récupéré à nouveau, puis la `DataSource` de la `DataGridView` et mise à jour avec ces nouvelles données de la BDD.

Enfin, pour l'onglet Transporteurs de `FrmLists`, cliquer sur le header de la colonne Nom permet de trier les transporteurs par leur nom, alternativement dans l'ordre alphabétique et inversement. Nous n'avons malheureusement pas réussi à implémenter cette fonctionnalité par le biais des propriétés et méthodes de `DataGridView`. Nous avons à la place récupéré le nom de la colonne dont on a cliqué le header puis exécuté une requête SQL affichant les résultats triés par nom.

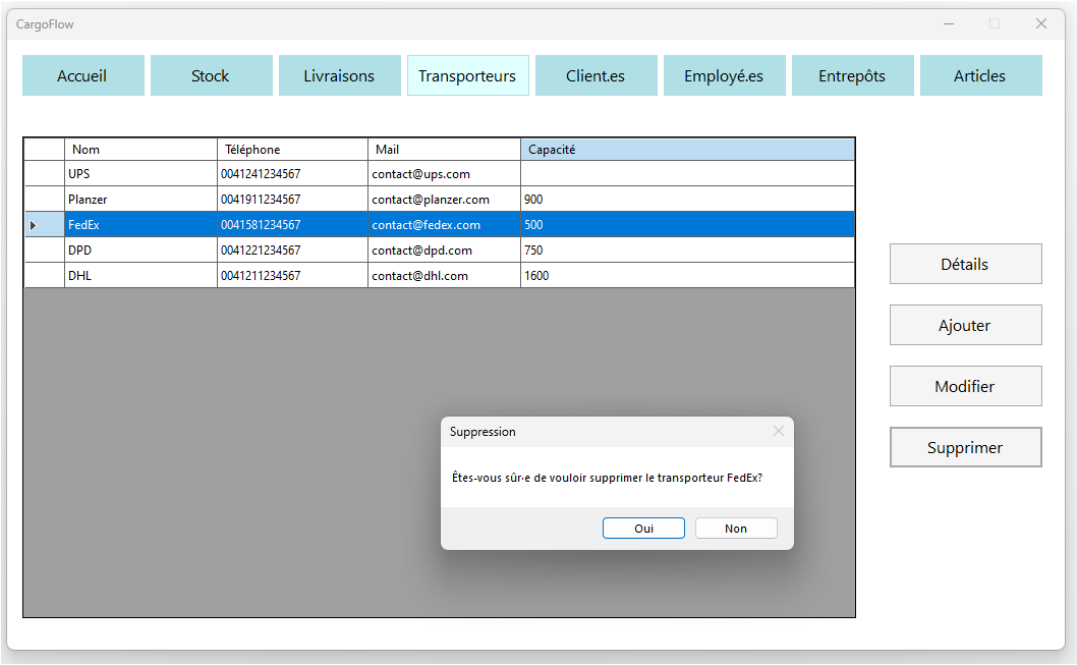

Figure 4 : FrmLists avec les transporteurs triés par ordre alphabétique inverse et un message demandant confirmation avant suppression du transporteur sélectioné

3.4.3.4. FrmDetails

Ce Form reçoit en paramètre le titre et les informations détaillées à afficher. Il s'ouvre en parallèle de `FrmLists` qui est toujours affiché. Les informations détaillées sont tirées de la BDD via une méthode statique de la classe de l'objet concerné.

Figure 5 : FrmDetails affichant les détails du transporteur Fedex et les livraisons qui lui sont associées

3.4.3.5. FrmAddUpdxxx

Selon les paramètres reçus, ces Forms affichent un formulaire vide à remplir ou un formulaire prérempli à modifier. Cliquer sur le bouton Ajouter/Modifier en bas de ce formulaire déclenche une vérification que les champs obligatoires soient bien remplis, puis une série de vérification du bon format des valeurs entrées par l'utilisateur, puis le cas échéant prépare les données pour une requête SQL. Parmi les vérifications faites, la validité du format de l'adresse mail s'effectue via une tentative d'instanciation de la classe `MailAddress` : si l'instanciation lève une exception, c'est que le format n'est pas valide.

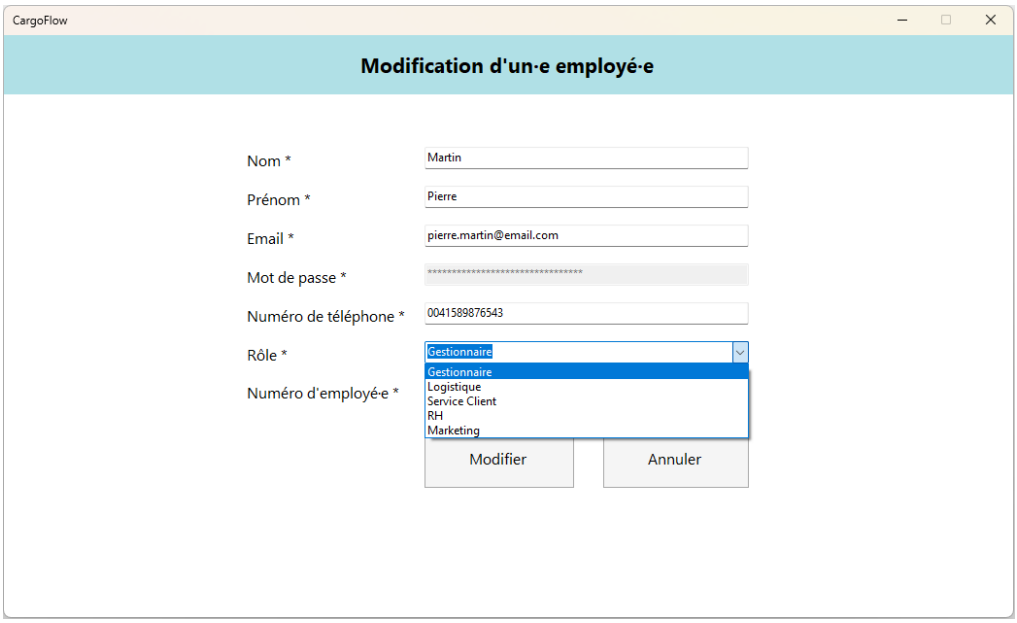

Figure 6 : FrmAddUpdEmployee avec les champs préremplis et une ComboBox listant les rôles

Dans le cas de `FrmAddUpdEmployee`, le champ « Rôle » n'est pas une TextBox mais une ComboBox afin de limiter les erreurs de saisie. Cette ComboBox est remplie grâce aux données reçues via une requête SQL qui récupère tous les rôles existants dans la table employees. L'utilisateur peut également créer un nouveau rôle en écrivant dans la comboBox comme s'il s'agissait d'une TextBox. Ce nouveau rôle apparaîtra alors dans la ComboBox lors des ajouts et modifications suivants. Après exécution de la requête, l'utilisateur revient à FrmLists.

3.4.1. Classes métier

En plus de `DBConnection`, d'autres classes ont été implémentées permettant de stocker et manipuler les données extraites de la BDD. Nous avons également créé une classe statique `Session` pour stocker les variables de session et une classe Utilities pour accéder à des méthodes pratiques depuis l'ensemble du logiciel, telle qu'une méthode de vérification du bon format d'une adresse mail.

4. Tests

4.1. Tests effectués

De nombreux tests ont été décrits dans les story sur icescrum. À chaque fin de sprint, les tests sont réalisés et passés en statut Passé ou Echoué.

4.2. Erreurs restantes

Voici les tests qui ne sont pas passés et les autres bugs que nous avons repérés en dehors des phases de tests :

- Son d'erreur Windows lorsque l'utilisateur tape sur Enter dans un des champs de `FrmLogin`. Ce son n'est cependant pas émis lorsqu'on appuie sur le bouton « Se connecter » (ou que l'on appuie sur Enter lorsque ce bouton est sélectionné).
- La fonction pour valider le bon format des adresses email ne fonctionne pas parfaitement. Malgré l'utilisation d'une méthode d'une des librairies C# de base, elle ne semble exiger que le format « texte@texte » au lieu de « texte@texte.texte » qui aurait dû être exigé.
- L'unique moyen de fermer le logiciel est de fermer le formulaire de login. Fermer les autres formulaires ferment les formulaires en question mais le programme reste ouvert en arrière-plan.

5. Améliorations possibles

- Finir les onglets productifs restants.
- Modifier les mots de passe d'un employé en tant qu'admin.
- Ajouter un menu pour changer son mot de passe pour les utilisateurs qui ne sont pas admin.
- Ne pas écrire en dur dans le code les informations de connexion à la BDD.
- Rendre statique la classe `DBConnection`.

- Faire les ouvertures et fermetures de connexion à la BDD dans `DBConnection` et non pas en dehors.
- Affiner la gestion des exceptions avec des try catch plus localisés et pertinents.
- Réussir à utiliser les fonctions de tri internes à la `DataGridView` pour s'épargner les nombreuses requêtes SQL.
- Créer des relations d'héritage pour les formulaires, avec par exemple une classe parente FrmLists qui contiendrait tous les contrôles communs et méthodes communes aux enfants `FrmListEmployees`, `FrmListClients`, etc.
- Le fait d'avoir mis la plupart des Foreign Key en NULL possible est une mauvaise idée car on perd les informations des enregistrements liés en cas de suppression. Une meilleure conception aurait été d'ajouter un champ booléen isActive dans les tables liées. Ainsi quand l'employé appuie sur le bouton Supprimer, l'enregistrement est conservé mais le champ isActive passe à false et le DGV affiche simplement les éléments actifs.

6. Conclusions

6.1. Conclusion de groupe

Compte tenu du thème de notre projet, nous avons eu une grande peine à trouver un équilibre entre la réalité de la logistique et la faisabilité en matière de code. Nous avons tout de même réussi à en trouver un, même si un peu trop ambitieux. Nous avons pris le temps de nous documenter suffisamment sur le monde de la logistique, tout en implémentant notre logiciel au mieux. Lié à ceci, la conception du MCD/MLD n'a pas été une mince affaire. Nous avons fait un certain nombre de brouillons avant d'arriver aux versions actuelles. Il y a des éléments qui restent incertains ou sur lesquels nous ne sommes pas convaincus mais le temps étant ce qu'il est, il a fallu à un moment donné se fixer sur une version plus ou moins définitive.

Concernant les fonctions de notre programme, l'essentiel des fonctions ont été implémentées. Nous sommes particulièrement contents du résultat malgré la possibilité de toujours faire mieux. Nous avons néanmoins remarqué qu'il aurait été possible d'améliorer le code de manière générale en le simplifiant. Cela aurait permis de diminuer la quantité de répétition et donc de rendre le code plus lisible et plus facilement maintenable. Si nous avions plus de temps à disposition, ne reprendrions ces parties, afin de les améliorer.

Si le temps nous l'avait permis, nous aurions aussi implémenté la gestion des droits et donc des rôles, afin que chaque personne ayant le même rôle, ait la même interface. Ceci aurait eu comme objectif que la personne gérant la logistique ait une interface correspondant à ses besoins, les RH une interface quelque peu différente correspondant à leurs besoins, etc. Pour l'instant, notre logiciel s'adresse à l'administrateur de l'entreprise sans prendre en compte les différents rôles contenus dans une entreprise.

Rétrospectivement, ce projet s'est très bien déroulé. Nous avons pris énormément de plaisir à créer de toute pièce un logiciel potentiellement utile. La collaboration entre nous s'est passé à merveille, nous avons pu se répartir les tâches tout au long du travail, afin d'optimiser le temps que nous avions à disposition. Nous avons su surmonter les obstacles qui se sont dressé devant nous tout au long du projet, en cherchant sur internet par exemple, en en discutant

entre nous lorsqu'il en était possible ou en demandant de l'aide à M. Ithurbide ou M. Benzonana, ou parfois aussi à certains de nos collègues.

6.2. Conclusions individuelles

Ian :

Ce projet a été une excellente opportunité de se frotter à un projet d'une assez grande ampleur. Par le choix du sujet que nous avons fait, j'ai trouvé le projet très réaliste et concret tant au niveau des réflexions à son sujet que lors de sa conception. Par sa teneur, je l'ai aussi trouvé très motivant.

J'ai pris beaucoup de plaisir à l'effectuer, malgré que la programmation ne soit pas mon sujet favori. J'ai néanmoins acquis de nouvelles compétences très intéressantes et de l'expérience qui, tous deux, me seront certainement utiles dans d'autres projets futurs. Je tiens d'ailleurs à remercier Antoine pour la détermination et la ténacité qu'il a manifesté tout au long du projet, face aux nombreux concepts que nous n'avions pas encore explorés en cours, mais qu'il a su implémenter dans le code.

Je suis très satisfait du travail que nous avons fait, que ça soit dans la conception du projet au niveau du code, dans la conception de la BDD ou de la maquette. Dans l'ensemble, son déroulement s'est très bien passé, nous avons réussi à être rigoureux tout au long du projet en nous tenant au diagramme de Gantt à quelques exceptions près. Bien entendu, de nombreuses améliorations sont possibles, mais le temps à disposition étant limité, je trouve que le travail que nous avons fourni est vraiment bon. Malheureusement, nous n'avons pas eu le temps de finir le projet, mais dès sa présentation à M. Ithurbide lors de la première semaine, il nous a mis en garde que le projet était très ambitieux et qu'il fallait être prêt à une telle éventualité.

Antoine :

Le projet s'est dans l'ensemble bien déroulé. J'ai pu pratiquer des aspects de la conception et de la programmation que je connaissais déjà (Windows Forms, classes, exceptions…) et surtout j'ai pu acquérir quelques connaissances et compétences supplémentaires (interactions en C# avec une BDD sur un serveur, bases de la DataGridView, hashage en C#, classes statiques).

La phase de réflexion sur la logistique « dans le monde réelle » et la modélisation de la BDD a été intéressante mais fastidieuse, nous aurions peut-être dû plus rapidement accepter de ne pas tout savoir et limiter la portée de notre projet. Cela dit, cela nous a permis de partir sur de bonnes bases en ce qui concerne la structure de la base de données. Nous n'y avons apporté que des changements mineurs plus tard dans le projet.

En revanche, en ce qui concerne la phase de réflexion sur l'organisation du code C#, je pense que j'aurais dû me renseigner davantage sur le fonctionnement de la DataGridView qui est un élément central du projet. Mes premières recherches m'avaient amené à donner comme DataSource à ce DataGridView une liste d'objets car j'avais vu de tels exemples et car je suis à l'aise avec le concept de liste d'objets. Cependant, ce choix m'a posé de nombreux soucis lorsqu'il a fallu trier et filtrer les données par la suite. D'autres formats semblent plus adaptés

pour cela. J'ai même vu vers la fin du projet qu'il était possible de donner comme DataSource à la DGV directement une BDD et qu'il existe un paquet NuGet permettant d'implémenter rapidement des fonctions de tri et de filtre (cf. dernière source). Malgré cela, je reste assez satisfait du code que nous avons produit dans la mesure où il est fonctionnel et stable. Théoriquement, avec plus de temps, je pense que nous pourrions remplir le cahier des charges établi au début du projet.

Quant à la collaboration avec Ian, elle s'est bien déroulée, nous avons su nous partager les tâches équitablement et n'avons pas rencontré de problèmes particuliers, ni en termes de communication, ni en termes techniques (pas de pertes de fichiers, pas de conflits de versions de fichiers à résoudre sur GitHub…).

7. Annexes

7.1. Sources - Bibliographie

Conventions / abréviations utilisées pour les contrôles des Windows Form :
https://gist.github.com/andyyou/3052671

MySqlConnection : Établir une connexion en C# à un serveur MySQL :
https://dev.mysql.com/doc/connector-net/en/connector-net-tutorials-connection.html

MySqlCommand : Interagir avec la BDD une fois la connexion établie :
https://dev.mysql.com/doc/connector-net/en/connector-net-tutorials-sql-command.html

DbDataReader : Lire les données reçues après une requêtes SQL :
https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/retrieving-data-using-a-datareader

Documentation sur la classe Dictionary<TKey,TValue>, utilisés dans notre code pour passer les valeurs des paramètres SQL :
https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-8.0&redirectedfrom=MSDN

Documentation sur la DataGridView :
https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.datagridview?view=windowsdesktop-8.0

Hasher des valeurs en C# pour stocker les mots de passe hashés dans la BDD
https://learn.microsoft.com/en-us/troubleshoot/developer/visualstudio/csharp/language-compilers/compute-hash-values

Donner directement une BDD comme source à la DGV :
https://www.youtube.com/watch?v=RaZeeHRRCGA

7.2. Table des abréviations

- BDD : Base de données
- MCD : Modèle conceptuel de données
- MLD : Modèle logique de données
- WinForms : Windows Froms
- DB : Database
- CRUD : Create, read, update, delete

7.3. Table des illustrations

7.4. MCD

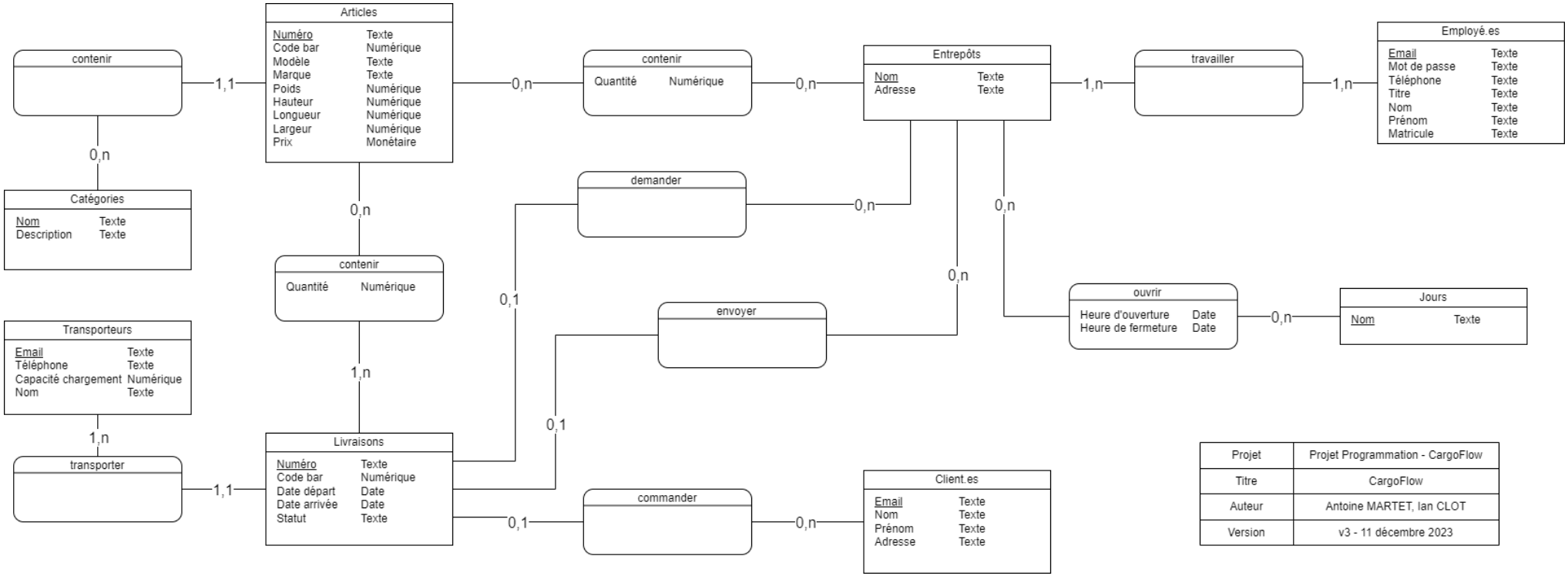

Figure 7 : Modèle Conceptuel de Données de CargoFlow

7.5. MLD

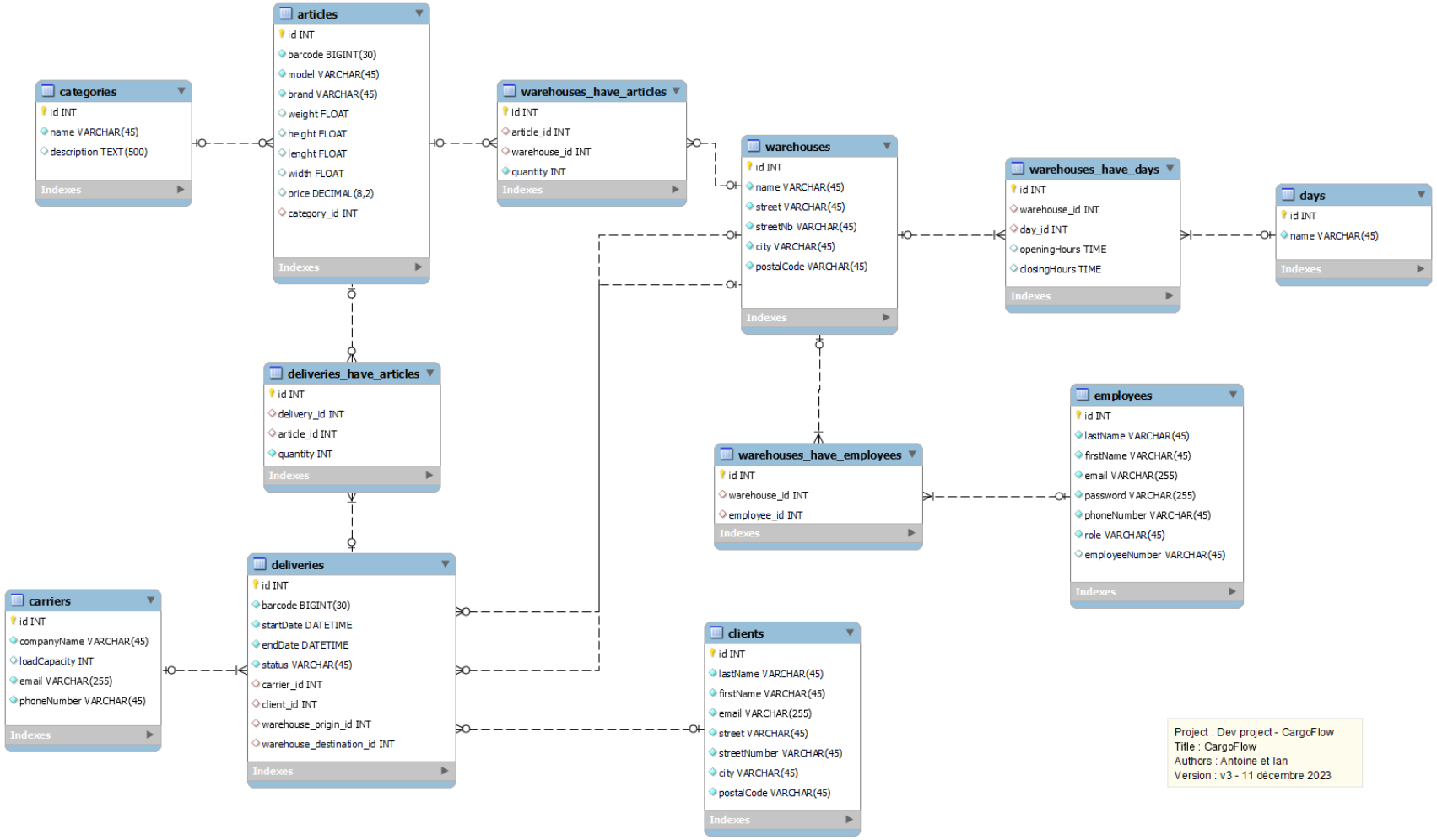

Figure 8 : Modèle Logique de Données de CargoFlow