**Projet de programmation**

**CargoFlow**

Martet Antoine

[antoine.martet@eduvaud.ch](mailto:antoine.martet@eduvaud.ch)

Clot Ian

[ianalain.clot@eduvaud.ch](mailto:ianalain.clot@eduvaud.ch)



SI-CA2a

30 décembre 2023

**Table des matières**

[1. Introduction 3](#_Toc156081914)

[1.1. Cadre, description et motivation 3](#_Toc156081915)

[1.2. Organisation 3](#_Toc156081916)

[1.3. Objectifs 3](#_Toc156081917)

[1.4. Planification 4](#_Toc156081918)

[2. Analyse 4](#_Toc156081919)

[2.1. Use cases et scénarios 4](#_Toc156081920)

[2.2. Maquettes 5](#_Toc156081921)

[2.3. MCD 5](#_Toc156081922)

[3. Implémentation 5](#_Toc156081923)

[3.1. Choix techniques 5](#_Toc156081924)

[3.1.1. Base de données 5](#_Toc156081925)

[3.1.2. Documentation 5](#_Toc156081926)

[3.1.3. Développement 5](#_Toc156081927)

[3.2. Conventions 5](#_Toc156081928)

[3.2.1. Commentaires 5](#_Toc156081929)

[3.2.2. Conventions de nommage 6](#_Toc156081930)

[3.3. MLD 6](#_Toc156081931)

[3.4. Organisation et fonctionnement du programme 6](#_Toc156081932)

[3.4.1. Diagramme de classes 6](#_Toc156081933)

[3.4.2. DBConnection 6](#_Toc156081934)

[3.4.3. Les Windows Forms 7](#_Toc156081935)

[3.4.1. Les classes métier 10](#_Toc156081936)

[4. Tests 10](#_Toc156081937)

[4.1. Tests effectués 10](#_Toc156081938)

[5. Améliorations possibles 11](#_Toc156081939)

[6. Conclusions 11](#_Toc156081940)

[6.1. Conclusion de groupe 11](#_Toc156081941)

[6.2. Conclusions individuelles 11](#_Toc156081942)

[7. Annexes 11](#_Toc156081943)

[7.1. Sources - Bibliographie 11](#_Toc156081944)

[7.2. Table des abréviations 11](#_Toc156081945)

[7.3. Table des illustrations 12](#_Toc156081946)

[7.4. Index 12](#_Toc156081947)

# Introduction

## 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 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 Form. 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.

## Organisation

Ce projet a été divisé en 3 grosses parties. Tout d’abord, nous avons commencé par planifier les différentes étapes de notre projet et nous sommes penchés sur l’établissement des uses cases. Concernant la planification des tâches, nous avons utilisé iceScrum. Pour ce qui en est des uses cases et des scénarios, il nous a semblé judicieux de l’effectuer sur Excel. Nous avons durant cette même période mis en place le repository Github, afin d’archiver notre progression et de travailler facilement sur les mêmes fichiers.

Dans un second temps, nous avons cherché à modéliser la BDD qui serait utilisée dans notre logiciel ainsi que l’apparence du logiciel. Nous avons ainsi un MCD puis un MLD et réalisé des maquettes.

La troisième et dernière étape était l’écriture du code. Cependant, de nombreuses recherches ont dues être effectuées avant l’écriture de C#, car nous avions besoin d’outils que nous n’avions pas vu jusqu’ici, en particuliers le DataGridView, une classe liée aux WindowsForm permettant de créer dynamiquement des tableaux. En parallèle des recherches et de l’écriture du code, nous avons pu transformer le MLD en requête SQL, pour ensuite intégrer notre base de données dans notre logiciel.

Durant ces 3 parties, nous avons documenté chacun de notre côté, les éléments nous semblant pertinents, afin de simplifier l’écriture de la documentation.

// TODO : merge avec planification ?

## 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 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 Détail 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.

## Planification

Nous avons décidé de diviser le temps que nous avions à disposition, soit du 6 novembre au 19 janvier, en 5 sprint. Le détail de nos activités durant cette période se trouve sur [iceScrum](https://icescrum.cpnv.ch/p/CARGOFLOW/#/planning). 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, 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 était déjà effectué.

# 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 ces services pour suivre ces 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 rendu 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.

## 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.

## 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 dans le branche main.

## MCD

Le MCD se trouve sur Github dans le dossier Documentation sous le nom CargoFlow\_MCD\_v3.drawio dans la branche main.

# Implémentation

## Choix techniques

### 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.

### 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 lisibilité, se trouve dans des documents annexes : MCD, MLD, fichiers de requêtes SQL d’insertion de données, maquettes.

### Développement

Notre outil principal pour le développement a été le logiciel Visual Studio. Il nous a permis d’utiliser les Windows Form, de gérer les erreurs de compilation et de build. 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.

## Conventions

### Commentaires

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

### 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 publique 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

## MLD

Le MLD se trouve sur Github dans le dossier Documentation sous le nom CargoFlow\_MLD\_v3.mwb dans la branche main.

// TODO : justifier certains éléments

## Organisation et fonctionnement du programme

Le code de CargoFlow repose sur des Windows Forms et des classes métier personnalisées. 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.

### Diagramme de classes

// TODO : screenshot + explication/contexte de pourquoi pas d’associations

### 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 éventuelles autres paramètres nécessaires.

### Les Windows Forms

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 caché et non pas fermée.

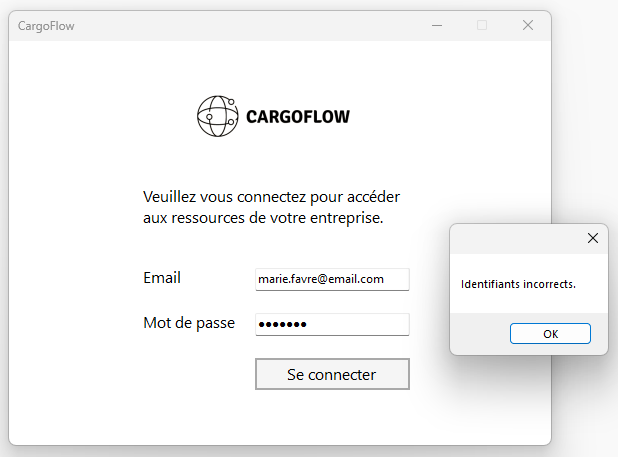


Figure 1 : FrmLogin après insertion de mauvais identifiants

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 différence de fonctionnement du logiciel en fonction du rôle de l’utilisateur connecté.

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.

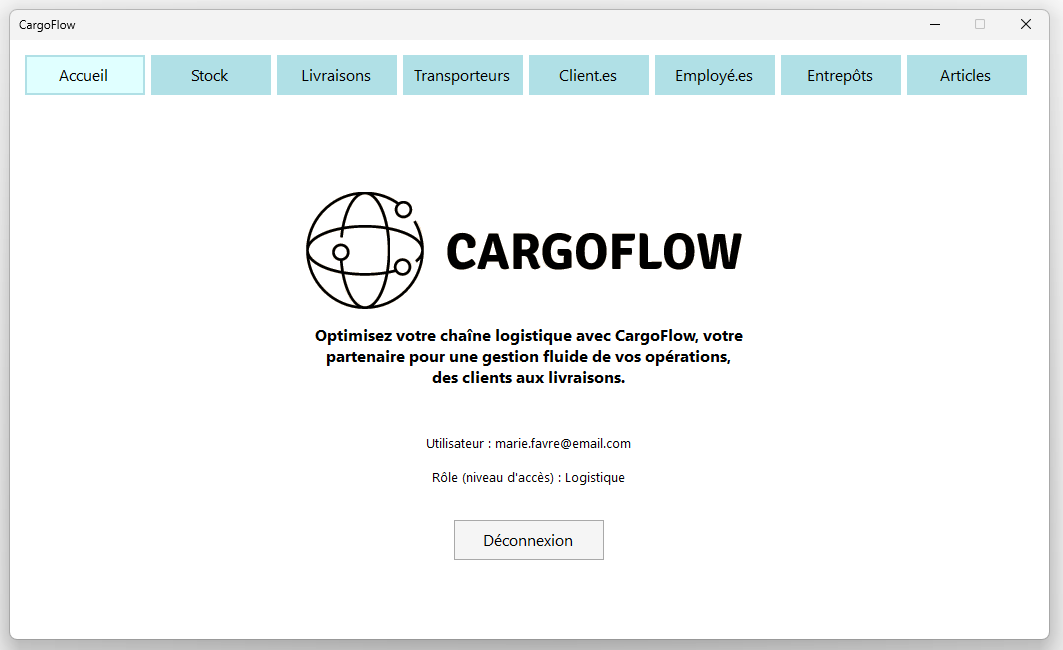


Figure 2 : FrmHome

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 (DataSource) de données 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 inversé. 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.

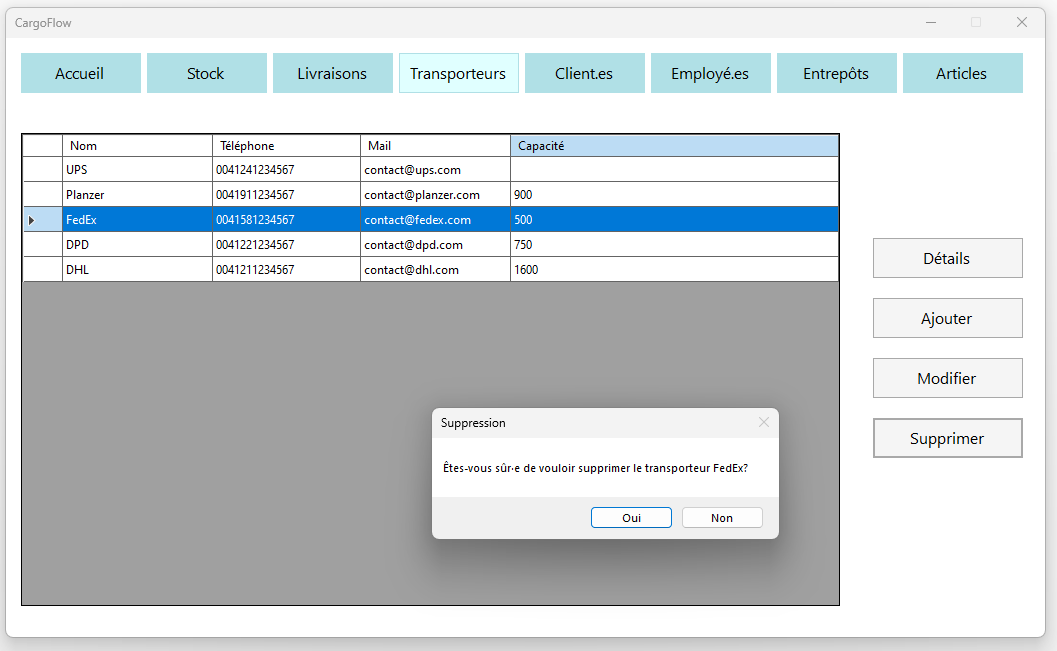


Figure 3 : FrmLists avec les transporteurs triés par ordre alphabétique inverse et un message demandant confirmation avant suppression du transporteur sélectioné

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.

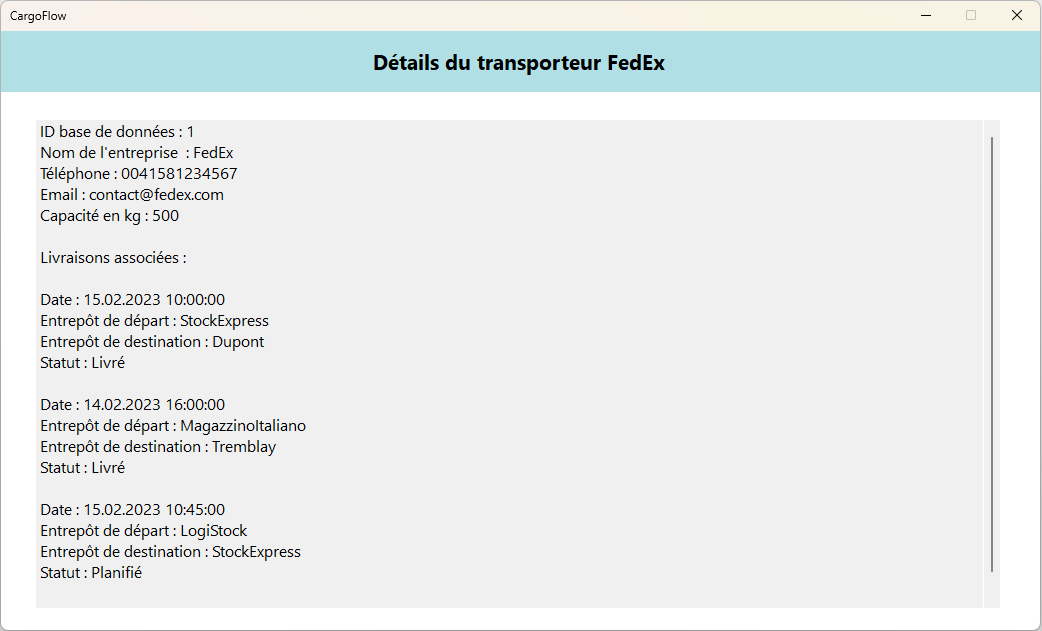


Figure 4 : FrmDetails affichant les détails du transporteur Fedex et les livraisons qui lui sont associées

FrmAddUpdxxx : selon les paramètres reçus, ces Form 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. 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 employee. L’utilisateur peut également créer un nouveau rôle. Ce nouveau rôle apparaîtra alors dans la comboBox lors des ajouts et modifications suivants. Après exécution de la requête, l’utilisateur revient à FrmLists.

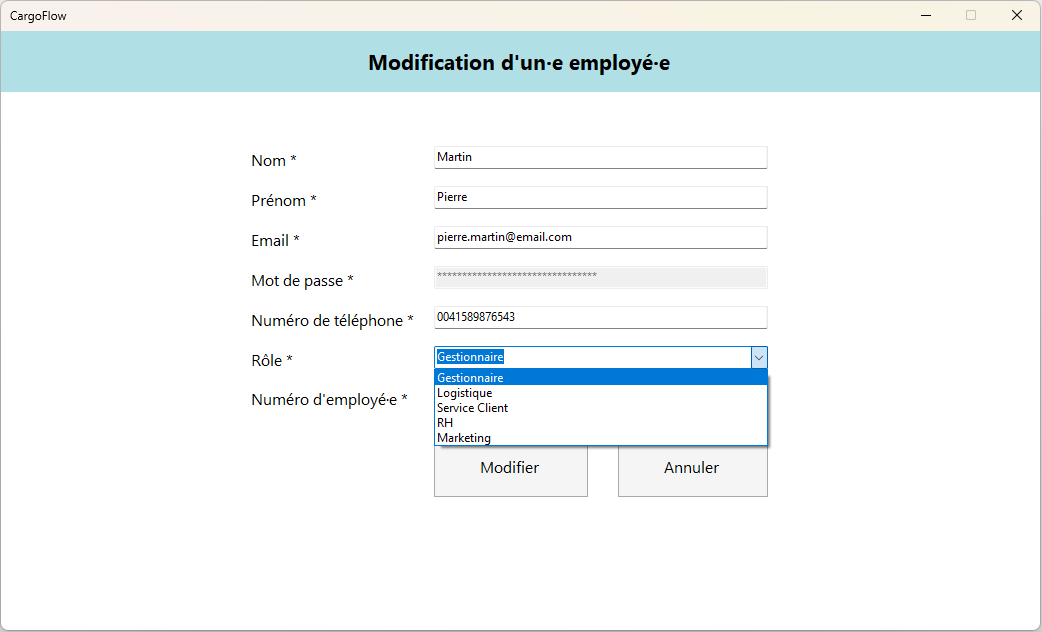


Figure 5 : FrmAddUpdEmployee avec les champs préremplis et une ComboBox listant les rôles

### Les 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, tel qu’une méthode de vérification du bon format d’une adresse mail.

# Tests

## 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.

## 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 par contre 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 ».

# 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 les informations de connexion à la BDD dans le code.
* 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 parent FrmLists qui contiendrait tous les contrôles communs et méthodes communes aux enfants FrmListEmployees, FrmListClients, etc.

# Conclusions

## Conclusion de groupe

Ce projet s’est dans l’ensemble bien déroulé. Nous avons néanmoins été confronté à un certain nombre de problèmes, notamment dans la phase de modélisation de la base de données.

// TODO : compléter / bulshitter

## Conclusions individuelles

Ian :

// TODO

Antoine :

// TODO

# Annexes

## Sources - Bibliographie

// TODO

## 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

## Table des illustrations

// TODO

// TODO : checker style console pour code