

M. Rebai Maher



ÉCOLE
D'INGÉNIEURS
PARIS-LA DÉFENSE

PROJET INFORMATIQUE

GESTION D'UN
PARC DE LOCATION

GALI MAIKEL

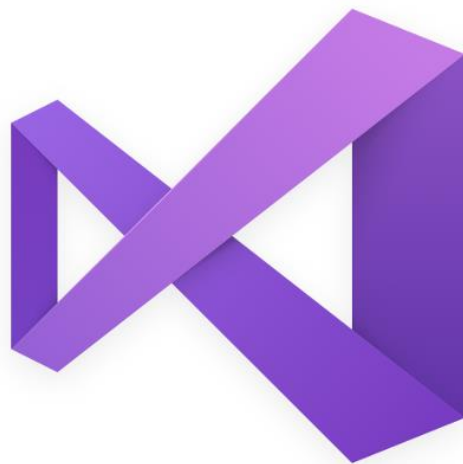
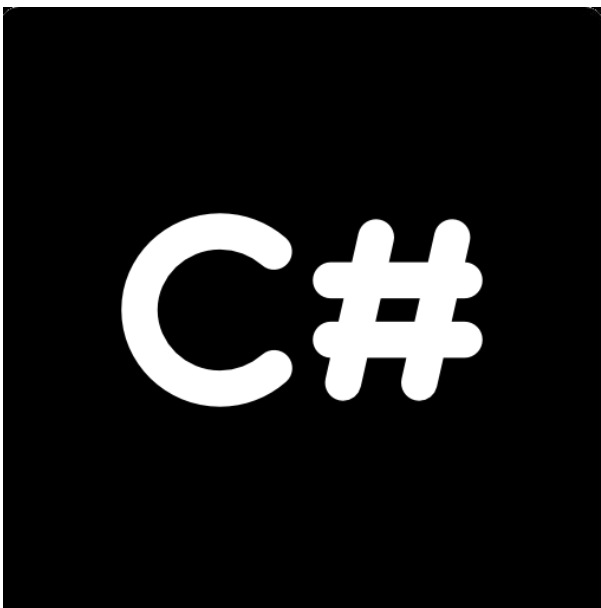
DUFORT DONATIEN



ECE PARIS
ÉCOLE D'INGÉNIEURS

GESTION D'UN PARC DE LOCATION

Présentation du projet :	2
UML (diagramme de classe) :	3
Explication de l'UML :	4
Nos méthodes :	6
Méthode de sécurité :	6
Explication technique :	6
L'interface « IAttributsFichier » :	7
Explication technique :	7
Méthode « CalculCoutLocation » :	8
Explication technique :	8
Méthode delegate « public delegate void supprimeLocationFichier(uint id, string nomFichier); » : ..	9
Explication technique :	9
Autre :	10
Pour finir :	10

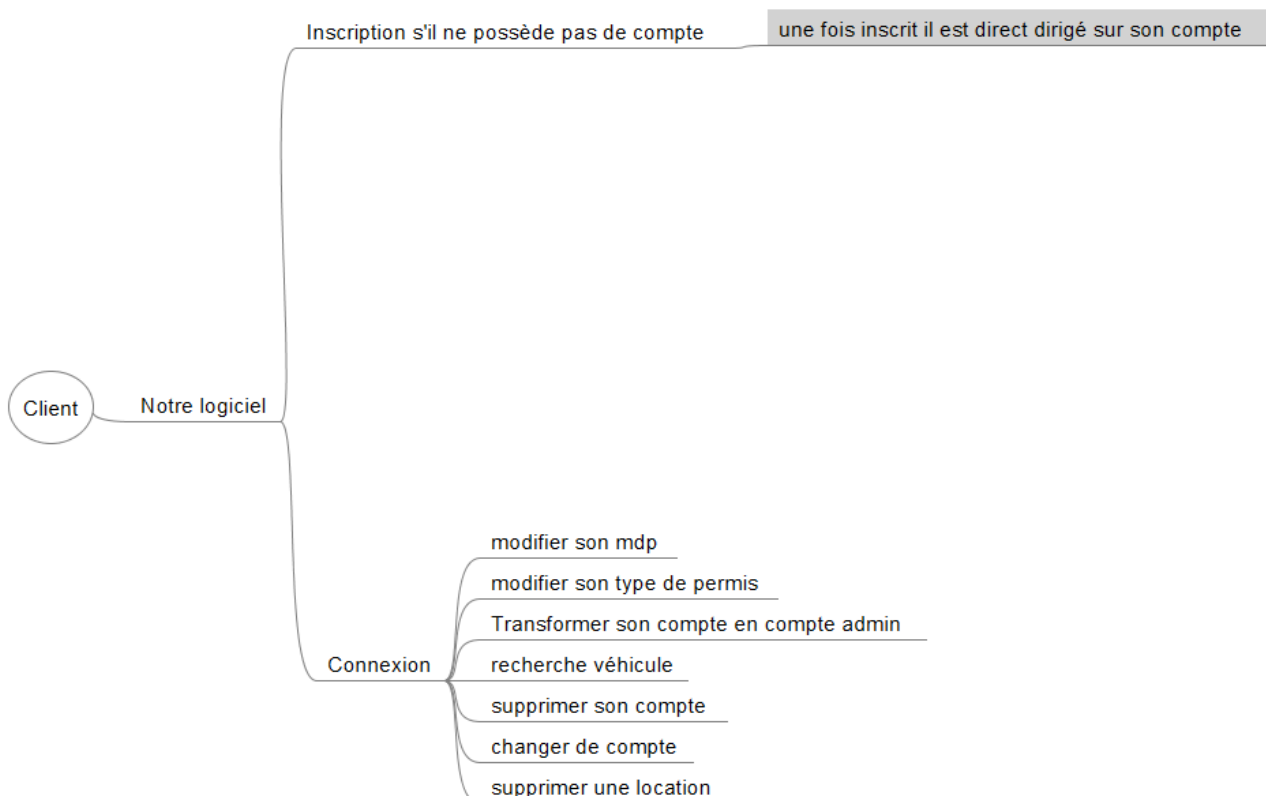


Présentation du projet :

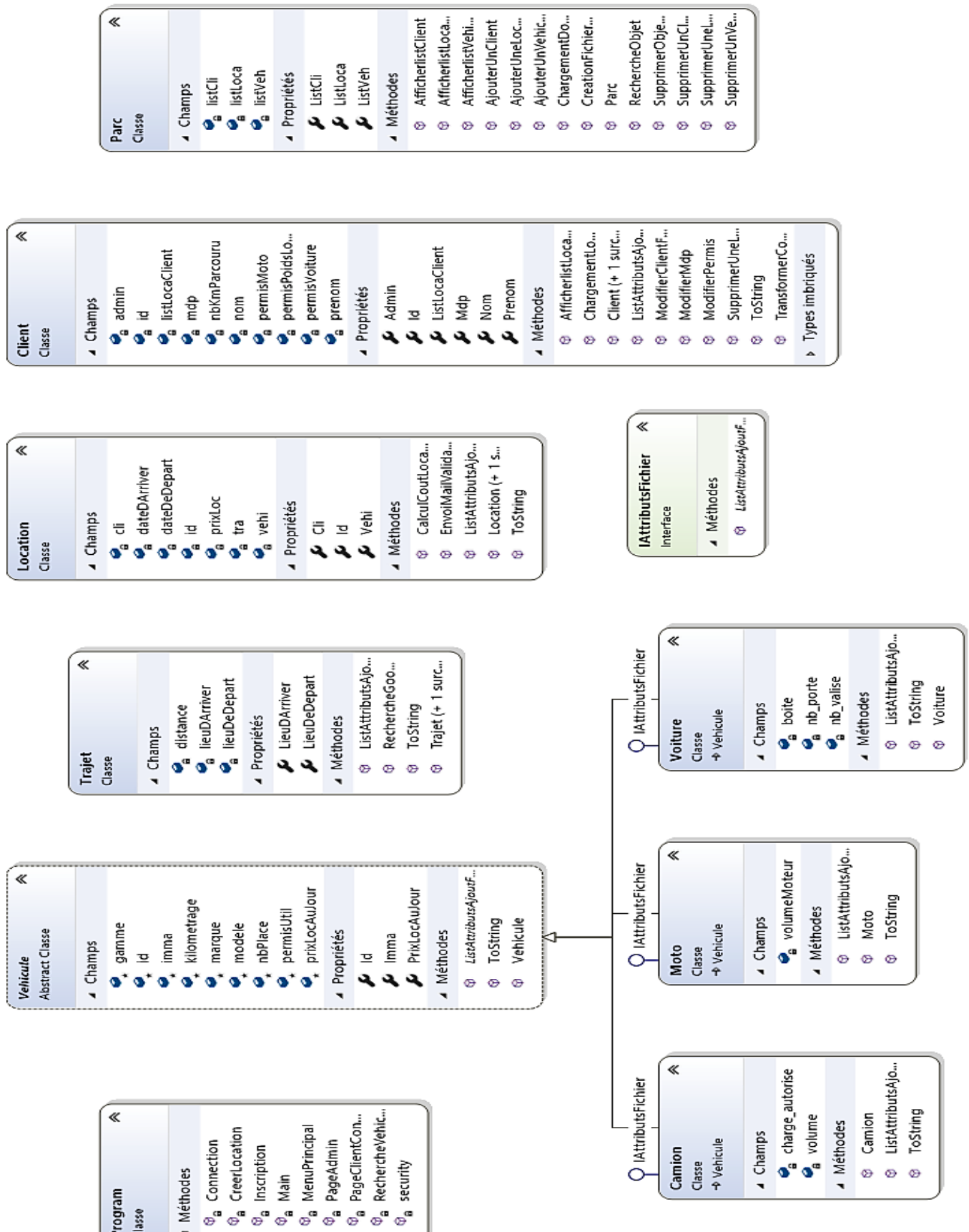
Notre projet consiste à créer un programme en langage C# pour une agence de location de véhicules.

Dans cette agence, nous pouvons louer des voitures, des motos et même des camions. Chaque véhicule à ses propres caractéristique, exemple la voiture à pour caractéristiques la marque, la gamme, le modèle, le nombre de kilométrage, le nombre de place, le nombre de portes...etc

On peut même définir si cette voiture possède une boîte manuelle ou automatique. Voici un schéma qui explique le fonctionnement de notre logiciel qui permet de gérer la location en automatique.



UML (diagramme de classe) :



Explicaton de l'UML :

Nous avons 9 classes (en comptant la classe program), une classe mère abstract « Vehicule » et 3 classes filles, une interface.

- Classe mère : c'est une classe général qui donne naissance à des classe filles, ces classe filles doivent obligatoirement reprendre toutes les méthodes et tous les attributs de cette calsse mère.
- Classe fille : c'est une classe qui hérite d'une classe mère, elle hérite toutes les méthodes et tout les attribus.

Pour mieux optimiser le code, nous avons utiliser une interféface « IAttributsFichier », une interféface va nous permettre d'utiliser des programmes dans certaines classes filles qu'on va choisir.

Nous avons utilisé un délégate, pour éviter d'écrire plusieurs fois le même code et donc charger intuiement le programme le delegate va nous permittre d'écrire un prgramme et l'appeler qu'on en aura besoin de l'utiliser à condition d'utiliser la même signature.

On a une calsse « Parc », c'est dans cette classe que nous allons gérer la liste de nos véhicules, clients et location.

On a une classe client, c'est à travers de cette classe que nous allons pouvoirs créer des objets clients.

On a une calsse location, pour pour faire une simulation, un devis ou même valider une location, nous avons opter pour la méthode de locatio. Dans une location nous avons la date d'aller, de départ, un identifiant, un client, un trajet et un véhicule. Notre méthode classe se définit de la façon suivante, nous avons mit des prix de location par jour dans chaque type de véhicule. Nous avons ensuite un programme qui va nous permettre de faire la différnece entre la date d'aller et la date de départ et nous envoi ensuite le prix de la location...

Nous avons opter pour cette technique de calcule car ça nous a permis de réduire le nombre de classe, le nombre de ligne de code, donc cela nous éviter de charger le programme inutilement.

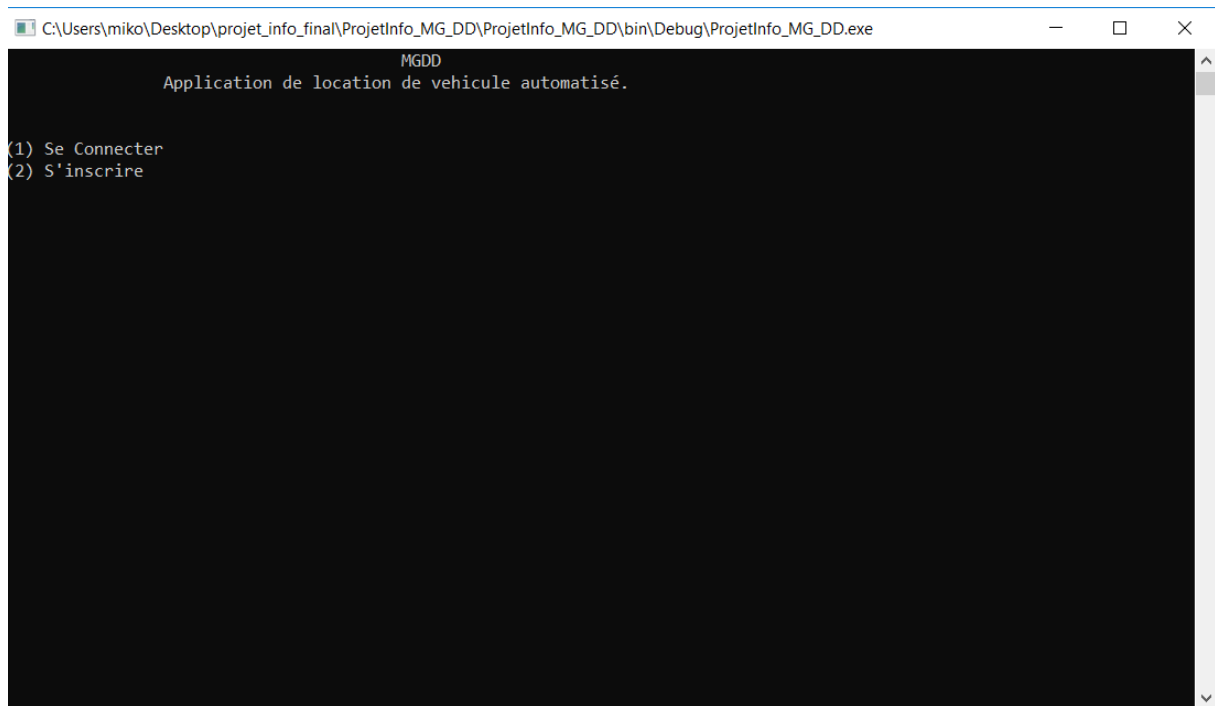
La classe programme, c'est la calsse général de base, c'est de la que tout est dirigé, dès que le client démarre le programme, il est confronté à un choix, soit il s'inscrit si il n'a pas de copte soit il se connecte. Nous mit des sécurité de choix pour pas que le client choisisse un choix qui n'existe pas.

Une fois connecté, plusieurs possiblilité s'offre à lui, soit il demande de cahnger de mot de passe s'il en a envi, car on estime que le chaque personne possédant un compte sur une plateforme X doit impérativement changer son mot de passe de temps en temps pour mieux sécuriser son compte.

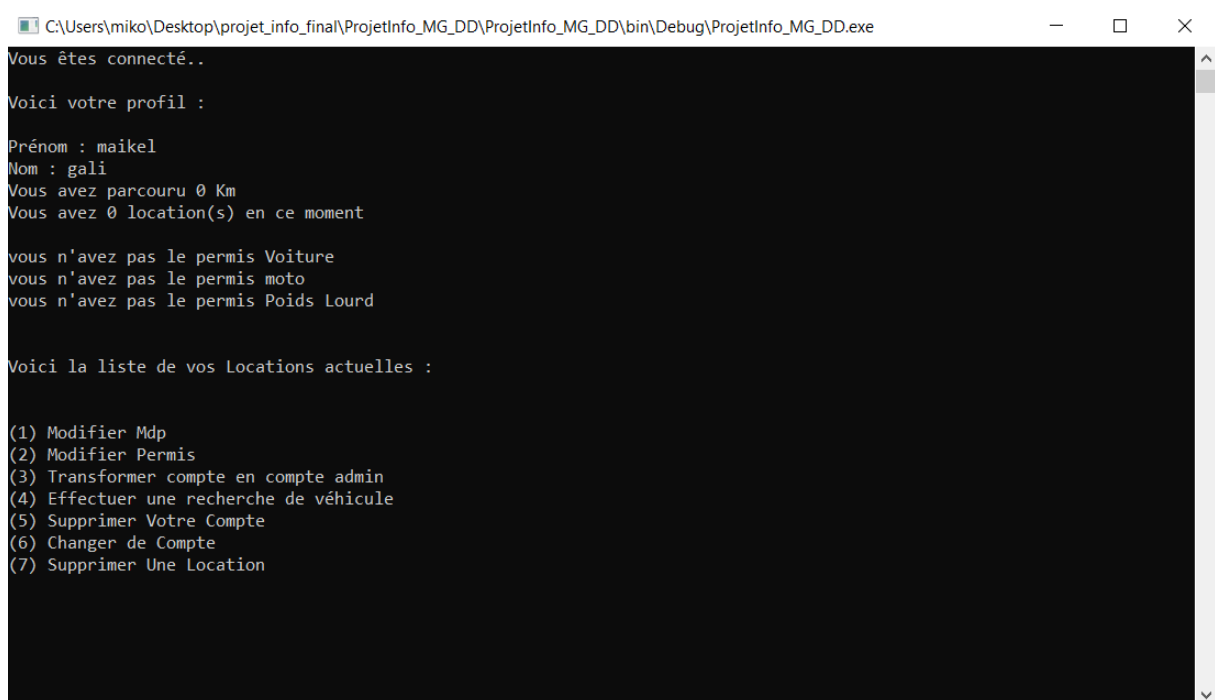
Soit si il vient de s'acquérir un nouveau permis, il a la possibilité de l'enrgistrer sur son compte pour pouvoir louer un véhicule.

Soit il transforme son compte en compte admin à condition d'être autorisé par l'admin déjà définit par défaut.

Il a aussi la possibilité de surfer sur nos collections de voitures. Il peut supprimer ou cahnger de compte, aussi supprimer ou ajouter une location.



```
C:\Users\miko\Desktop\projet_info_final\ProjetInfo_MG_DD\ProjetInfo_MG_DD\bin\Debug\ProjetInfo_MG_DD.exe
MGDD
Application de location de vehicule automatisé.
(1) Se Connecter
(2) S'inscrire
```



```
C:\Users\miko\Desktop\projet_info_final\ProjetInfo_MG_DD\ProjetInfo_MG_DD\bin\Debug\ProjetInfo_MG_DD.exe
Vous êtes connecté..
Voici votre profil :
Prénom : maikel
Nom : gali
Vous avez parcouru 0 Km
Vous avez 0 location(s) en ce moment
vous n'avez pas le permis Voiture
vous n'avez pas le permis moto
vous n'avez pas le permis Poids Lourd
Voici la liste de vos Locations actuelles :
(1) Modifier Mdp
(2) Modifier Permis
(3) Transformer compte en compte admin
(4) Effectuer une recherche de véhicule
(5) Supprimer Votre Compte
(6) Changer de Compte
(7) Supprimer Une Location
```

Nos méthodes :

Méthode de sécurité :

```
8 références
static int security(int nbPosibiliteChoix) // Méthode de sécurité de saisie
{
    string saisie = "";
    bool redemande = false;
    int rep = 0;
    do
    {
        if (redemande)
        {
            Console.WriteLine("Saisie Incorrect, veuillez resaisir votre réponse..");
        }
        redemande = true;
        saisie = Console.ReadLine();
    } while (!int.TryParse(saisie, out rep) || rep < 0 || rep > nbPosibiliteChoix);
    return rep;
}
3 références
```

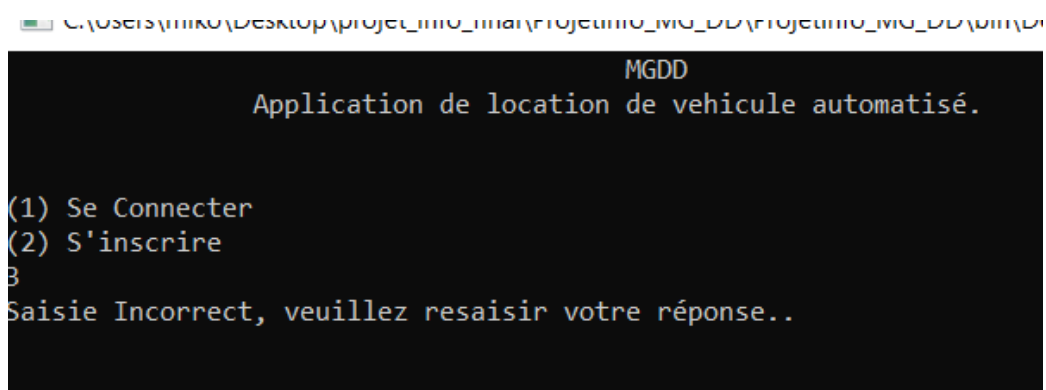
Nous utilisons cette méthode à plusieurs reprises dans notre programme pour sécuriser le saisie de l'utilisateur.

Cette méthode est très importante car elle assure le bon fonctionnement du programme. Son fonctionnement est le suivant, la méthode reçoit un nombre entier, ce nombre est le nombre de choix possible dans une fonction, il n'y a rien de mieux qu'un exemple qui illustre cette explication.

Nous prenons l'exemple de la page d'accueil de notre programme, sur cette page nous proposons 2 choix, soit « s'inscrire » ou « se connecter », la méthode « security » sert ici à vérifier si le l'utilisateur à respecter ou non le nombre de choix qui lui sont proposé. Si le client choisit entre 1 et 2 la fonction vérifie et autorise donc l'utilisateur à passer à l'étape suivante sinon un message d'erreur apparait au client en lui indiquant « saisie incorrecte, veuillez ressaisir votre réponse.. » et l'utilisateur aura donc un autre essai pour corriger son saisie.

Explication technique :

La méthode « security » reçoit un nombre entier un « int », et renvoi un « int », la boucle « while » est activée tant que le nombre de saisie par l'utilisateur est plus grand au choix proposé, ou si on reçoit un « string » au lieu d'un « int » nous envoyons le même message d'erreur pour permettre à l'utilisateur de corriger son choix.



```
MGDD
Application de location de vehicule automatisé.

(1) Se Connecter
(2) S'inscrire
3
Saisie Incorrect, veuillez resaisir votre réponse..
```

L'interface « IAttributsFichier » :

Rappel de la définition d'une interface : c'est une classe dans laquelle on ne retrouve que des méthodes abstraites, elles sont nécessairement publiques et abstraites.

```
3 références
interface IAttributsFichier // Interface accueillant la méthode de retour de la chaîne de caractères à enregistrer dans les fichiers
{
    6 références
    string ListAttributsAjoutFichier();
}
```

Nous avons utilisé cette classe pour les utiliser dans certaines classes filles, contrairement aux méthodes qui se trouvent dans une classe mère qui sont obligées d'être utilisées dans toutes les classes filles, avec la classe interface nous avons la possibilité de définir ou est-ce qu'on veut utiliser ses méthodes, ici nous avons qu'une seule méthode qui va nous permettre de mettre à jour un fichier modifié.

Explication technique :

```
3 références
public string ListAttributsAjoutFichier() // Retourne la chaîne de caractères à enregistrer dans les fichiers
{
    string str = id + ";" + prenom + ";" + nom + ";" + mdp + ";" + nbKmParcours + ";" + permisVoiture + ";" + permisMoto + ";" + permisPoidsLourd + ";" + admin;
    return str;
}
```

Nous utilisons cette méthode qui renvoie une chaîne de caractères après la modification d'un client, d'un véhicule ou autre...

Les nouvelles valeurs des attributs sont enregistrées dans cette chaîne et renvoyées directement vers le fichier pour être enregistrées.

Méthode « CalculCoutLocation » :

Pour permettre à notre programme d'aller plus vite et être plus efficace, nous avons décidé de mettre en place un système de calcul très puissant.

Nous avons décidé de calculer le coût par rapport au nombre de jour de location à l'aide des prix déjà définis pour chaque véhicule, notre programme détermine la différence entre le jour de la location et le jour où le client rend le véhicule et nous donne le prix exact.

```
public void CalculCoutLocation() // Calcul le temps de location avec les dates et estime le prix de location total
{
    string[] d = dateDeDepart.Split('/');
    string[] a = dateDArriver.Split('/');
    double nbJour = (Convert.ToInt32(a[2]) - Convert.ToInt32(d[2])) * 365 + (Convert.ToInt32(a[1]) - Convert.ToInt32(d[1])) * 30 + Convert.ToInt32(a[0]) % 2 + Convert.ToInt32(a[0]) - Convert.ToInt32(d[0]);
    prixLoc = vehi.PrixLocAuJour * nbJour;
    Console.WriteLine("Vous voulez louer pour : " + nbJour + "jour(s), cela vous coutera " + prixLoc + "€");
    Console.ReadKey();
}
```

Explication technique :

Nous avons défini deux tableaux « string » dans lesquelles nous allons séparer les jours, mois et années de la date départ et arrivé récupérer sous cette forme : JJ/MM/AAAA

Nous séparons chaque partie de la date (à l'aide de la fonction « split('/') » dans une case du tableau, le jour dans la 1ere case, le mois dans la 2eme et l'année dans la dernière case.

Nous effectuons ensuite notre calcul sur chaque case et le programme nous renvoie donc le prix de la location sous forme d'un « double ».

Une chaîne de caractère accompagne le prix de la location est affichée pour le client qui demande le prix de la location.

```
1 référence
public Location(Vehicule vehi, Client cli, Trajet tra) // Constructeur 2
{
    this.cli = cli;
    this.vehic = vehi;
    this.tra = tra;
    Console.WriteLine("Date de départ : \t\t(jj/mm/aaaa)");
    dateDeDepart = Console.ReadLine();
    Console.WriteLine("Date d'arriver : \t\t(jj/mm/aaaa)");
    dateDArriver = Console.ReadLine();
    CalculCoutLocation();
    EnvoiMailValidationLoc();
}
```

Nous avons utilisé ce constructeur pour afficher les informations de la location tel que les informations du véhicule du client et du trajet et on affiche aussi le coût de la location.

Nous avons fait une simulation d'envoi de confirmation par mail pour chaque location validée par le client.

Méthode delegate « `public delegate void supprimeLocationFichier(uint id, string nomFichier);` » :

Rappel de la définition du « delegate » : c'est une variable spéciale qui pointe vers une méthode, le nom de cette variable est une signature d'une méthode. Ce qui veut dire qu'on stock une méthode dans une variable qu'on peut utiliser selon le besoin.

```
public delegate void supprimeLocationFichier(uint id, string nomFichier);  
  
public void SupprimerUneLocation(uint idVeh, supprimeLocationFichier supprLoc)  
{  
    listLocaClient.RemoveAt(Convert.ToInt32(idVeh));  
    supprLoc(idVeh, "FichierVehicule.CSV");  
    Parc parc = new Parc();  
}
```

La méthode :

```
if(cli.ListLocaClient.Count != 0)  
{  
    Console.WriteLine("Entrez l'identifiant de la location à supprimer :");  
    int id = security(cli.ListLocaClient.Count);  
    Console.WriteLine("\nChargement..");  
    cli.SupprimerUneLocation(Convert.ToInt32(id), parc.SupprimerObjetFichier);  
    parc.SupprimerUneLocation(cli.ListLocaClient[id]);  
}
```

Nous avons constaté que nous avons eu plusieurs fois utiliser la même méthode pour supprimer un objet, nous avons donc créé un delegate qui va nous permettre de supprimer ces objets.

Explication technique :

Nous « delegate » va recevoir un « unit » (un entier positif ou nul) qui va représenter l'identifiant de notre objet, ainsi que le nom du fichier dans lequel l'objet est stocké. Une fois ces informations reçues, la méthode se met en marche et effectue le travail pour laquelle elle a été conçu qui est la suppression d'un objet.

Autre :

Nous avons beaucoup d'autre méthode tel que la sécurité la sécurité des nombres entier, si l'utilisateur tape une chaîne de caractère au lieu d'un entier, la sécurité est déclenchée pour l'avertir et lui demander de recommencer.

Nous avons aussi une sécurité au cas où les fichiers d'informations s'efface le programme détecte tout seul si y a besoin ou non de recréer les fichiers inexistantes.

On a aussi une méthode pour les chargements de données dans les listes et les fichiers.

On a lié les listes au fichier pour qu'à chaque modification effectuée dans les liste ou les fichiers l'autre est aussi modifié.

Nous avons fait en sorte qu'un compte client peut se transformer en compte admin.

Nous avons appris beaucoup de chose à travers ce projet, nous avons appris qu'il faut être patient, organisé, prendre des notes et planifier, faire des recherches, réfléchir à comment organiser son travail et le rendre le plus efficace possible malgré qu'on n'ait pas réussi à faire la partie « GOOGLE MAPS », mais nous avons réussi à travailler en groupe, partager nos idées, proposer des idées efficaces et mieux structurées pour avoir un travail de qualité.

Pour finir :

Nous remercions toute l'équipe ECE et surtout M. REBAI MAHER de nous avoir supporter pendant cette période :D, de nous avoir appris pas que la programmation en langage C# mais bien plus, merci pour votre présence, Bonne continuation.