Faculté Sciences et Techniques de l'Université de Tours
Parc Grandmont, Av. Monge
37200 Tours, FRANCE

Licence Informatique
$2^{ème}$ année

Projet Transversal - Sujet n°1

Conception et visualisation de tournées agricoles en circuits courts

Etudiants

Alex Pineau
alex.pineau@etu.univ-tours.fr
Lou Potard
lou.potard@etu.univ-tours.fr
Théophane Sam
theophane.sam@etu.univ-tours.fr
Mattéo Langlois
matteo.langlois@etu.univ-tours.fr

Responsables projet

Pierre Desport
pierre.desport@univ-tours.fr
Thierry Brouard
thierry.brouard@univ-tours.fr

Table des matières

Introduction

Choix du sujet

Nous avons choisi le projet n°1 **"Conception et visualisation de tournées agricoles en circuits courts"** car le thème nous a semblé intéressant et important en raison de ses enjeux : **Ecologiques** avec le circuit court permettant d'éviter le déplacement des matières premières sur tout le globe, **Economiques** car un circuit court implique souvent une meilleure réumunération des producteurs, et **Sociaux** car les livraisons n'impliquent pas un nombre important d'intermédiaires.

Les autres sujets étaient :

"Caractérisation de lettrines" , le but du sujet était de réaliser un logiciel permettant de recenser des lettrines afin que l'utilisateur puisse les trier et faire des analyses dessus. Nous n'avons pas choisi ce projet car il nous a semblé que le projet consistait principalement à réaliser des filtres, ce qui ne nous intéressait que peu.

"Saisie des examens" , le but de ce sujet était de créer un logiciel pour le personnel administratif de l'université afin que les examens puissent être entrés facilement et rapidement ainsi que de vérifier qu'il n'y ait pas d'incompatibilités, les emplois du temps auraient été géré par un solveur car il s'agissait d'un algorithme trop compliqué à développer. Ce sujet n'a pas été retenu par notre groupe car il nous a semblé trop complexe.

Explication du sujet

Le projet vise à la réalisation d'un logiciel permettant à un producteur de visualiser ses commandes et ses tournées de livraison.

Le logiciel devra gérer la connexion des producteurs et des administrateurs. Ces derniers seront chargés de la gestion des producteurs et de leurs clients. Quant aux producteurs, ils pourront aussi gérer leurs clients ainsi que leurs commandes.

Le logiciel vérifiera si la tournée est valide en fonction de certains critères, notamment les horaires et le poids des livraisons à transporter. Ces vérifications auront lieu lors de la création ou modification d'une tournée par le producteur.

1 Premier jalon

Le but du premier jalon est de construire les bases du logiciel, et ce grâce à plusieurs diagrammes utilisant la notation UML, une maquette pour se donner une idée des aspects fonctionnels du logiciel, ainsi qu'un schéma de la base de données (MCD). Cela permettra la réalisation des diagrammes de séquence, d'état et d'objets conduisant une réalisation rapide du logiciel et des tests afin d'avoir une meilleure stabilité et portabilité.

1.1 Réalisations

1.1.1 Choix des outils

Nous avons commencé par réfléchir aux outils que nous souhaitions utiliser pour la gestion de projet. Pour la gestion des versions et le partage du code source, nous avons décidé d'utiliser Git et GitHub[7] car nous avions tous déjà un compte. Pour la gestion de projet, nous avons choisi de passer par une méthode d'Agile : Scrum[11]. Pour facilement la mettre en place, nous avons choisi d'utiliser Youtrack[6], un outil de Jetbrains[5].

Youtrack nous a permis de créer des tâches puis de les assigner à un membre du groupe pour la répartition des tâches, mais aussi de les placer dans un Kanban pour nous aider à mieux évaluer ce qu'il reste à faire et l'évolution du projet. Youtrack nous a généré un diagramme de Gantt à partir de ces tâches après lui avoir fourni leurs durées.

Pour la réalisation des diagrammes UML, nous avons choisi d'utiliser Draw.io[3] ainsi que PlantUML[10] car nous connaissions déjà Draw.io et nous avons voulu découvrir un autre logiciel spécialement conçu pour réaliser ce genre de diagrammes. Par la suite, nous nous sommes dirigés vers StarUML[8] car nous l'avons utilisé en cours, et pouvoir générer de code Java directement depuis le diagramme de classes est un bonus pour notre productivité.

Afin de réaliser les schémas de la base de données, nous avons utilisé Looping[13], un logiciel libre de droit permettant de créer des MCD et des MLD ainsi que de générer le SQL pour la création de la base de données à partir du MCD.

1.1.2 Scénarii et diagramme de cas d'utilisation

En raison du nombre important de scénarii, nous avons choisi de les placer en annexes, tout en laissant 2 scénarii (ci-dessous) à titre d'exemple afin de les expliquer. Le diagramme des cas d'utilisation, étant lié aux scénarii, ne sera pas inclus dans cette section (cf. en annexe34).

Les actions présentées dans les scénarii suivants pourront parfois être réalisés par le producteur ou par l'administrateur, ce rôle sera précisé si l'action ne peut être faite que par l'un des deux ou si le scénario diffère. Dans le cas où les deux auront le même scénario, nous préciserons juste qu'il s'agit d'un utilisateur.

Acteurs

Acteur principal : Utilisateur (Producteur ou Administrateur)

Préconditions :

— Le logiciel est lancé
— Le producteur a un compte
— Le producteur doit livrer des commandes

Scénario : Connexion du producteur

Scénario nominal

1. Le producteur entre son identifiant (courriel ou le numéro SIRET de son entreprise)
2. Le producteur entre son mot de passe
3. Le producteur valide et clique sur "Se connecter"
4. Le programme valide les champs
5. Le programme vérifie que l'identifiant et mot de passe renseignés ont une correspondance dans la base de données
6. Le programme redirige le producteur sur son tableau de bord, signalant le succès de la connexion

Scénarii alternatifs

1. `Le producteur entre un identifiant et un mot de passe qui n'ont aucune correspondance dans la base de données`
   a. Le logiciel renvoie une erreur "identifiant ou mot de passe incorrect"
   b. Le scénario nominal reprend au point 1
2. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 1

FIGURE 1 – Scénario de connexion du producteur

Le producteur et l'administrateur auront des fenêtres différentes pour se connecter au logiciel. Nous les avons donc séparés en deux scénarii différents (cf. en annexe14 pour le scénario de connexion de l'administrateur). Le producteur peut donc se connecter avec son adresse email ou le numéro SIRET de son entreprise, et avec un mot de passe. Lors de la saisie, le logiciel vérifie que les champs requis sont bien remplis avec les types de valeur attendues.

Lors de l'élaboration du logiciel, nous utiliserons probablement des expressions régulières pour valider l'entrée de ces données. Ensuite, le logiciel se chargera de regarder dans la base de données afin de trouver l'identifiant et de vérifier que le mot de passe correspond bien. Si c'est le cas, alors la connexion est validée et l'utilisateur sera redirigé vers le tableau de bord. Dans le cas contraire, un message d'erreur sera affiché et il restera sur la page de connexion.

Scénario : Ajout de commandes

Scénario nominal

1. Le producteur entre les informations de la commande (choix du client, horaire minimale et maximale de livraison, poids de la commande)
2. Le producteur valide et envoie le formulaire
3. Le programme valide les champs du formulaire
4. Le programme enregistre les informations renseignées dans la base de données
5. Le programme informe l'utilisateur que les informations ont bien été enregistrées

Scénarii alternatifs

1. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `Le producteur saisit une valeur invalide`
   a. Le logiciel renvoie une erreur indiquant le champ dont la valeur est invalide et ce qu'il est attendu
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule l'ajout`
   a. Le logiciel enlève le formulaire de l'affichage
   b. Le logiciel redirige le producteur vers le tableau de bord

FIGURE 2 – Scénario d'ajout de commandes

Afin de visualiser les tournées, le producteur doit entrer les commandes qui composeront les tournées. Pour cela, il choisit le client dans la liste des clients, précise le nom de la commande, le poids total de la commande ainsi que les horaires pendant lesquels la commande devra être livrée. Il valide ensuite son formulaire.

Le logiciel vérifie les valeurs envoyées et les valide si leur type est bon. Dans ce cas, les données seront enregistrées dans la base de données. Dans le cas contraire, un message d'erreur sera envoyé en fonction du problème. Si l'ajout est validé, l'utilisateur sera redirigé vers son tableau de bord.

1.1.3 Schéma de la base de données

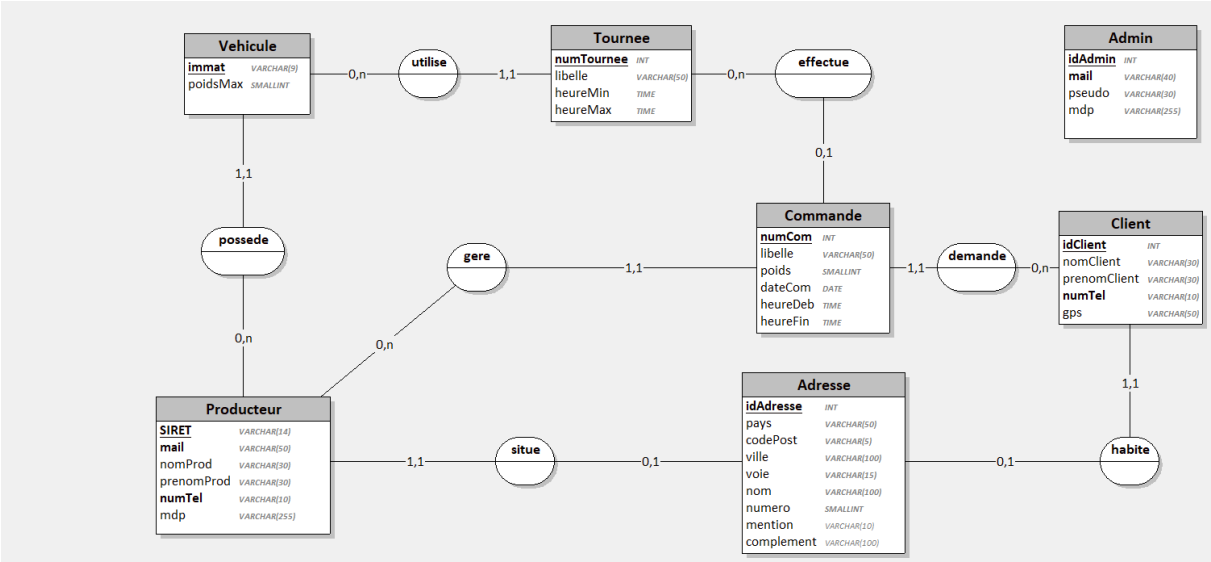

FIGURE 3 – Modèle conceptuel de données pour le projet

Lors de la conception de la base de données, nous avons commencé par réfléchir aux entités à représenter sur le MCD, et ensuite aux associations qui les relient. Les prochaines lignes permettent de comprendre les relations entre les lignes grâce aux cardinalités. Notamment, on peut lire que :

- Un Producteur peut avoir **zéro** à **n** Véhicules
- Un Producteur peut avoir **zéro** à **n** Commandes
- Un Producteur a **une** et **une** seule Adresse

- Un Client fait **zéro** à **n** Commandes
- Un Client a **une** et **une** seule Adresse

- Une Commande est liée à **un** et**un** seul Client
- Une Commande est liée à **un** et **un** seul Producteur
- Une Commande est liée de **zéro** à **une** Tournée

- Une Tournée a **zéro** à **n** Commandes
- Une Tournée utilise **un** et **un** seul Véhicule

Nous avons choisi de créer une table "Adresse" en raison de la complexité et du nombre

d'informations d'une adresse.

La table "Admin" n'a pas d'association avec les autres tables car elle n'est techniquement pas liée aux commandes ou aux producteurs. Les administrateurs peuvent uniquement gérer les clients et les comptes producteurs.

Le MCD a été réalisé pour correspondre aux exigences décrites dans les scénarii, notamment l'unicité des identifiants qui serviront pour la connexion. Le MLD pourra être trouvé en annexe32.

Voici quelques requêtes qui nous seront utiles plus tard lors du développement du projet.

Récupération du poids total d'une tournée

```
SELECT SUM(C.poids) FROM Tournee
INNER JOIN Effectue USING (numTournee)
INNER JOIN Commande C USING (numCom)
GROUP BY numTournee
HAVING numTournee = x;
```

Récupération de l'adresse des clients d'une tournée

```
SELECT CONCAT(A.Ville, " ", A.numero, " ",
   A.voie, " ", A.nom, " (", A.mention,
   A.complement, ")") AS "Adresse"
FROM Tournee
INNER JOIN Effectue USING (numTournee)
INNER JOIN Commande USING (numCom)
INNER JOIN Client USING (idClient)
INNER JOIN Adresse A USING (idAdresse)
GROUP BY numTournee
HAVING numTournee = x;
```

Vérification des horaires d'une tournée

```
SELECT CASE WHEN EXISTS (
   SELECT T.heureMin, T.heureMax FROM Tournee T
   WHERE T.heureMin > (
      SELECT MIN(heureDeb) FROM Commande
      GROUP BY numTournee
      HAVING numTournee = x
   )
   AND T.heureMax < (
      SELECT MAX(heureFun) FROM Commande
      GROUP BY numTournee
      HAVING numTournee = x
   )
   WHERE numTournee = x
```

```
THEN CAST(1 AS BIT)
ELSE CAST(0 AS BIT) END
```

1.1.4 Diagramme de classes

Le diagramme de classes sera découpé afin de le rendre plus lisible, mais sera trouvable en un seul morceau en annexe33.

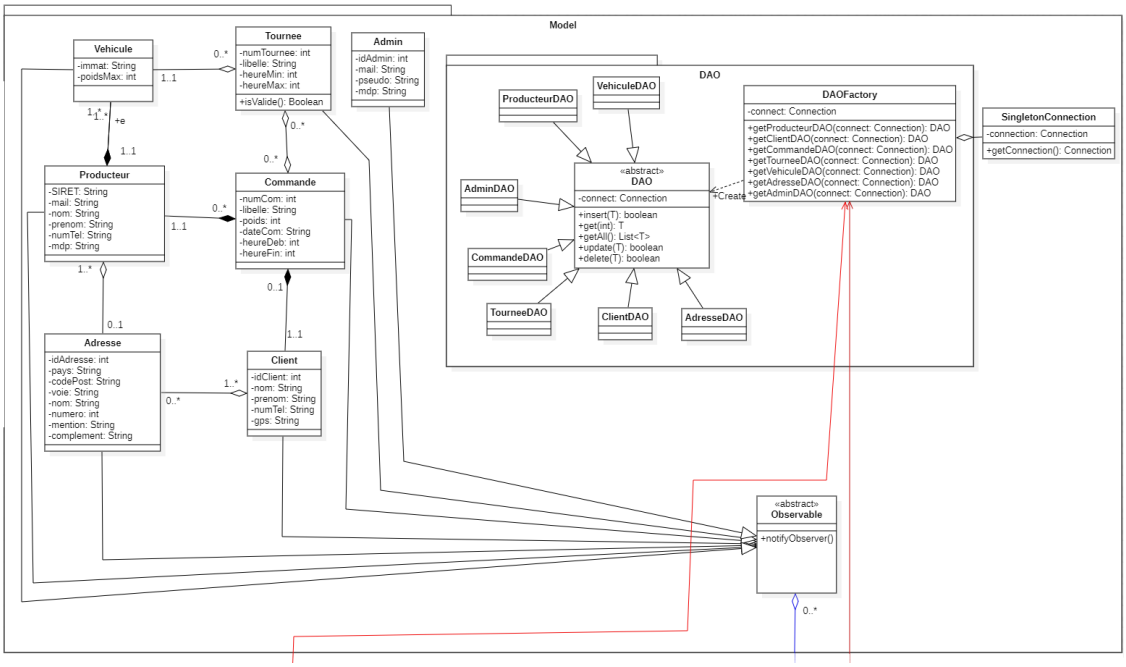

FIGURE 4 – Première partie du diagramme de classe (Modèles)

Nous avons décidé de mettre les classes qui représentent les entités dans la base de données dans le package "Model" afin d'offrir une séparation logique par rapport au reste du programme. Ces classes pourront potentiellement être des observateurs (design pattern "Observer") pour savoir si les vues qu'elles observent changent. Nous avons aussi créé un package "DAO" pour les classes implémentant l'interface "DAO" qui fait usage du design pattern DAO (**D**ata **A**ccess **O**bject) pour permettre l'interaction avec la base de données.

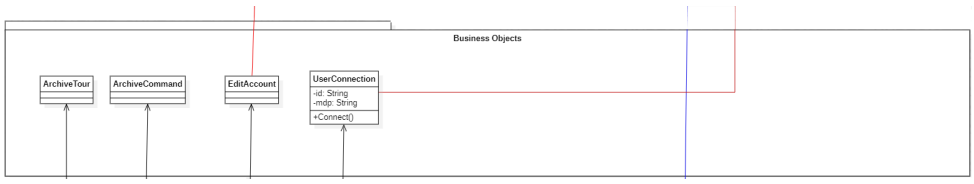

FIGURE 5 – Deuxième partie du diagramme de classe (Objets métiers)

Concernant les objets métiers, nous les avons placés dans un package "Business Objects", encore fois pour avoir une séparation logique. Les classes placées ici contiendront la logique pour modifier les modèles et refléter ces changements dans la base de données. Elles sont destinées à être pilotées par les contrôleurs.

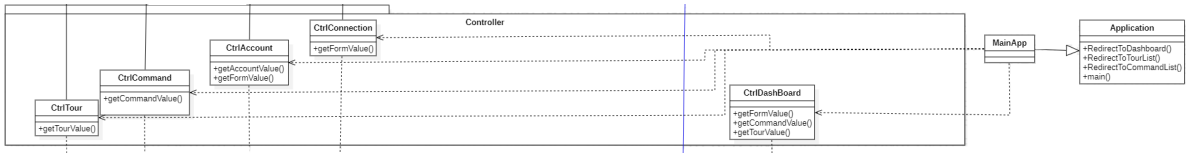

FIGURE 6 – Troisième partie du diagramme de classe (Contrôleurs)

Enfin, pour faire le lien entre le modèle et les vues, il y aura des contrôleurs (situés dans le package "Controller"). Ces classes seront chargées de récupérer les entrées utilisateur afin de les stocker dans les base de données à travers les objets métiers, et d'envoyer aux vues ce qu'elles ont besoin pour leur affichage.

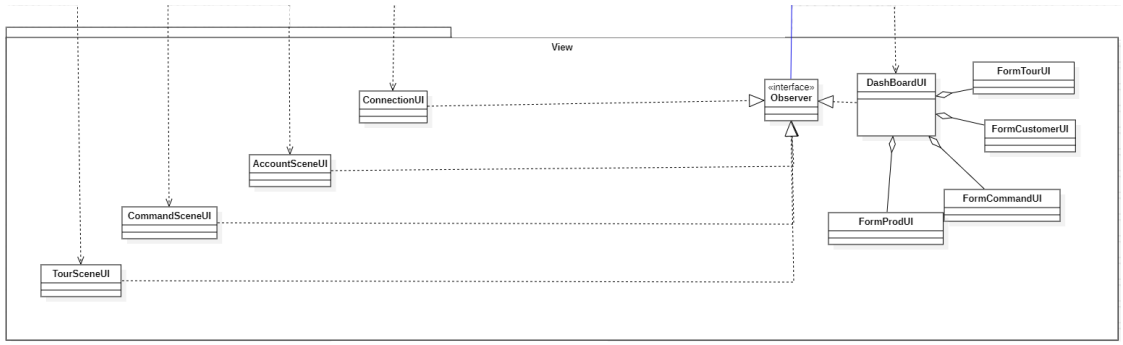

FIGURE 7 – Dernière partie du diagramme de classe (Vues)

L'interface graphique du logiciel sera découpée en plusieurs vues, chacune d'entre-elles sera reliée à un contrôleur pour les entrées utilisateur. Notamment elles implémentent l'interface "Observer" pour permettre de notifier à ses observateurs de ses changements.

Les classes, leurs méthodes et leurs attributs sont sujet à changement. Nous n'avons pas encore mis tous les éléments qui serviront ensuite au bon fonctionnement du logiciel car nous ne savons pas encore tout ce dont nous aurons besoin.

1.1.5 Maquette

Description de la maquette

Lors de la réalisation de la maquette, nous avons commencé par le tableau de bord du logiciel (cf. figure 8), celle permettant de visualiser les commandes et les tournées. Nous avons donc choisi de séparer l'interface en plusieurs composants. Tout en haut de la fenêtre, une barre de menu permet d'accéder aux commandes, aux tournées, au compte utilisateur, et aux clients. En dessous de cette barre, nous avons décidé de mettre la liste des commandes ainsi qu'un calendrier. Et enfin, sur la barre latérale gauche, nous avons mis la liste des tournées. Il y a également une intégration d'OpenStreetMap[9] pour visualiser les tournées sur une carte.

L'interface est donc centrée sur la carte car la visualisation de la tournée est la partie la plus importante du logiciel. Le but de cette interface est d'avoir rapidement à portée de vue toutes les informations sur les prochaines tournées.

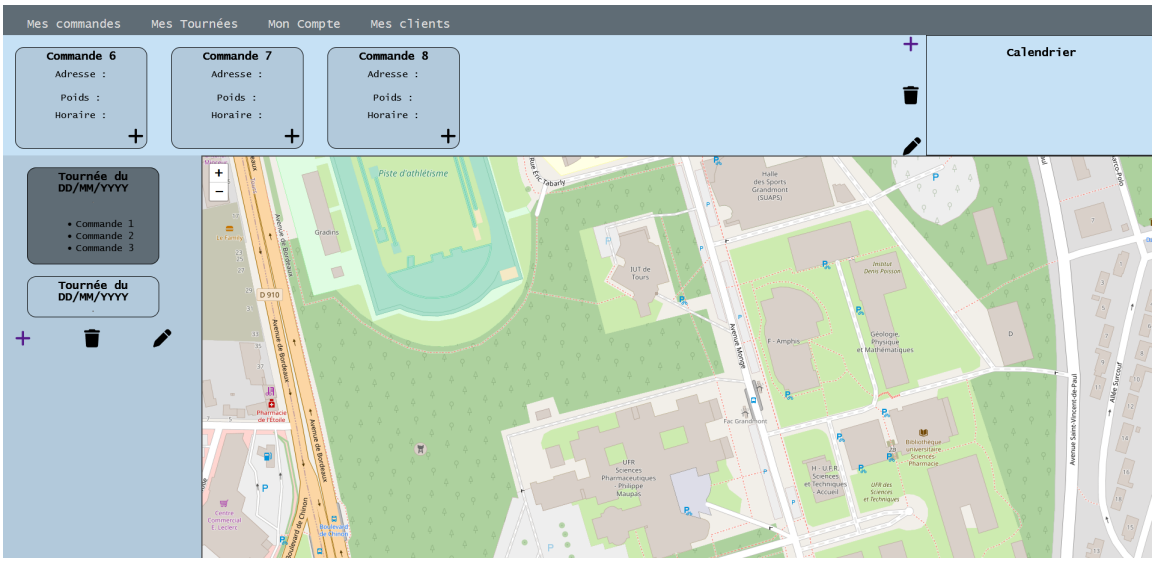

FIGURE 8 – Maquette du tableau de bord du logiciel

Nous avons ensuite réalisé la page de connexion (cf. en annexe51) et réutilisé le style de cette page pour les formulaires d'ajout de tournées (cf. en annexe54) et de commandes (cf. en annexe53).

Explication de l'utilisation du logiciel d'après la maquette

Le producteur ajoute les commandes grâce au bouton "+" situé à droite, puis lorsqu'il a suffisamment de commandes, il peut créer une tournée et la sélectionner ensuite pour y ajouter les commandes de son choix. Lorsque la commande est sélectionnée, elle sera affichée avec l'intégration d'OpenStreetMap. Le calendrier permettra de voir toutes les tournées prévues dans le mois par exemple en mettant en surbrillance les jours concernés.

Les commandes et les tournées dont les dates sont passées peuvent être trouvées respectivement dans les onglets "Mes commandes" et "Mes tournées", ce qui permettra de garder un historique.

1.2 Difficultés rencontrées

L'utilisation d'un outil très complet comme Youtrack nous a ralenti au départ en raison du nombre important d'onglets et de fonctionnalités que le service propose. Puis au bout de quelques jours d'exploration et d'expérimentation, nous avons réussi à le personnaliser afin de n'avoir que ce qui nous importaient.

Après plusieurs semaines d'utilisation, nous nous sommes rendu compte que nous utilisions peu ce logiciel et que pour la répartition des tâches nous passions principalement par une messagerie instantanée. Nous avons donc décidé de n'utiliser plus que la partie Kanban du logiciel afin de lister et répartir les tâches.

En outre, nous avons peu utilisé les diagrammes de Gantt et n'avons pas réalisé de diagramme de suivi. Comme nous ne pouvons pas vraiment diviser les tâches des prochains jalons, nous avons préféré nous passer du diagramme de Gantt et juste afficher les tâches à faire dans le Kanban sans estimer leurs durées.

Lors de l'élaboration des scénarii, nous nous sommes posés plusieurs questions, mais une nous a fait réfléchir plus longtemps. En raison du nombre très important de cas d'utilisation et de l'importance de réaliser tous les scénarii, nous avons voulu regrouper les scénarii similaires et les généraliser afin de raccourcir cette partie pouvant être répétitive. Cependant, en cas de modification des souhaits et attentes du client, il aurait été plus compliqué de modifier les parties du scénario touchées par ces modifications, nous avons donc choisi de séparer les scénarii.

Lors de la conception de la base de données, nous nous sommes heurtés à plusieurs problèmes. Nous nous sommes questionnés pour savoir s'il fallait inclure des clients de pays étrangers, dans le cas où l'utilisateur est un producteur frontalier par exemple. Ce qui aurait donc changé le format de plusieurs informations client comme le numéro de téléphone et le code postal. La présence des adresses nous a fait réfléchir sur la manière de stocker des informations peu présentes comme BIS et TER.

Nous avons aussi réfléchi à la possiblité pour un producteur d'avoir plusieurs locaux, dans ce cas le numéro SIRET aurait été stocké dans une autre table entre "Producteur"

et "Adresse" et cela nous aurait amené à gérer des chaînes de caractères afin de retrouver le numéro SIREN qui aurait pu donc être l'identifiant de connexion.

Le dernier diagramme que nous avons réalisé lors de ce premier jalon a été le diagramme de classe, car c'est celui pour lequel nous devions le plus nous projeter dans le futur. Insérer les classes permettant de lier la base de données au logiciel n'a pas forcément était le plus compliqué car nous avions vu comment le faire lors des cours de "Programmation orientée Objet Avancée". En revanche, comme nous avons peu manipulé les interfaces graphiques avec JavaFX, les parties liées à ces interfaces sont plus susceptibles de changer que le reste afin de mieux faire correspondre nos besoins.

L'une des autres difficultés a été de trouver comment compléter ce diagramme de classe. Il a fallu prévoir les méthodes et attributs dont on aura besoin lors de la réalisation du logiciel.

Bilan du premier jalon

Le projet est encore au premier stade, mais cette fin de jalon nous permet de faire un bilan temporaire de nos avancées, nos difficultés, ainsi qu'apercevoir comment réaliser la suite du logiciel. Cette conclusion sera donc completée par la suite lors des prochains jalons.

Le projet nous a montré l'importance du génie logiciel lors de la réalisation de projet de taille importante. Nous avons aussi pu remarquer que malgré une échéance paraissant assez lointaine le temps pour réaliser les livrables était assez court et que beaucoup de modifications et de corrections de dernière minute pouvaient arriver.

La réalisation de tous ces éléments nous permettra dans les prochaines semaines de concevoir le logiciel. Lors de la création du logiciel, nous réaliserons aussi les modules de tests qui serviront à vérifier le bon fonctionnement de chaque partie du logiciel.

L'interface du logiciel évoluera afin de la rendre la plus pratique possible. La réalisation de la maquette étant assez différente de la création de l'interface du logiciel avec du code, l'interface changera sur certains points. Ces changements seront principalement liés aux limitations des outils utilisés.

Toutes ces modifications et artéfacts pourront permettre de réaliser le logiciel en lui même lors du deuxième et du troisième jalon.

2 Deuxième jalon

Le deuxième jalon consiste à débuter la réalisation du logiciel après sa conception. Pour cela nous avons choisi de suivre la méthode vue en cours, c'est-à-dire de commencer par générer les classes et méthodes depuis le diagramme UML, que nous avons ensuite documenter. Suite à cela, nous avons créé les classes de tests et documenté les tests pour ensuite les réaliser. Pour finir nous avons commencé à développer certaines méthodes.

Ce jalon nous a fait réfléchir sur les requêtes vers la base de données que nous allons devoir réaliser. Ainsi que de les traduire en algèbre relationnel et calcul relationnel.

Nous avons aussi pu construire l'interface graphique en utilisant SceneBuilder afin de ne pas à avoir à écrire à la main l'interface et donc de gagner en vitesse.

2.1 Réalisations

2.1.1 Refonte du diagramme de classe

La première tâche à laquelle nous nous sommes attelés a été de refaire le diagramme de classe en divisant au mieux ses différentes parties afin de le rendre plus lisible et de plus juste.

Nous avons commencé par réaliser un diagramme permettant de visualiser les packages principaux du projet, comme ci-dessous.

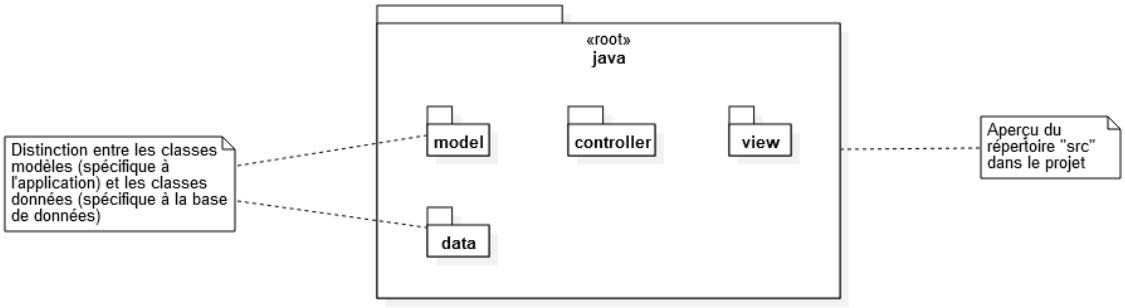

FIGURE 9 – Diagramme de classe des différents packages

Nous avons ensuite fait les classes des différents packages (modèle, vue, contrôleur, et nous avons choisi de séparer le modèle des DAO que nous avons mis dans le package "data").

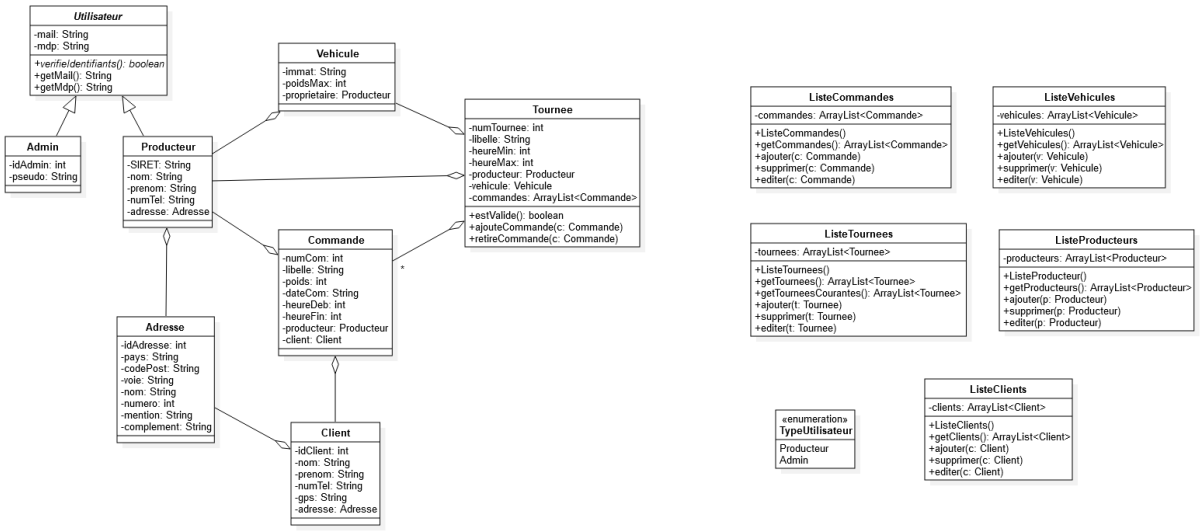

FIGURE 10 – Diagramme de classes de la partie "modèle"

Ce diagramme de classes montre les différentes classes de la partie modèle du logiciel ainsi que les relations entre les différentes classes. Sur la partie de gauche, nous pouvons voir un schéma proche du MCD car il s'agit des relations entre les classes permettant d'enregistrer temporairement les données avant de les envoyer dans la base de données grâce au DAO. A droite, on peut voir d'autres classes qui permettront ensuite de manipuler plus facilement une liste d'objets notamment les commandes, les tournées ou les clients. Ces classes serviront principalement utilisées lors de l'affichage des vues avec toutes les commandes ou tournées par exemple.

L'énumération "TypeUtilisateur" permet de faire la distinction sur le type d'utilisateur. Etant donné que les tables administrateur et producteur sont différentes, il faut pouvoir déterminer le type d'utilisateur pour adapter le comportement du programme. Utiliser une énumération nous permet d'éviter à devoir mettre des nombres magiques nous-même, et à la place, cela nous permet d'utiliser des noms de variables, ce qui est plus explicite.

Sur le schéma nous n'avons pas affiché les getters et les setters afin d'en améliorer la lisibilité. La génération du code à partir du diagramme de classes les incluent déjà sans qu'on ait à les préciser.

Nous avons aussi divisé les différentes parties liées aux fonctionnalités du logiciel pour avoir une meilleure idée des relations entre les différents packages. Comme ci-dessous le diagramme de classe lié aux commandes (ajout, modification, affichage, suppression).

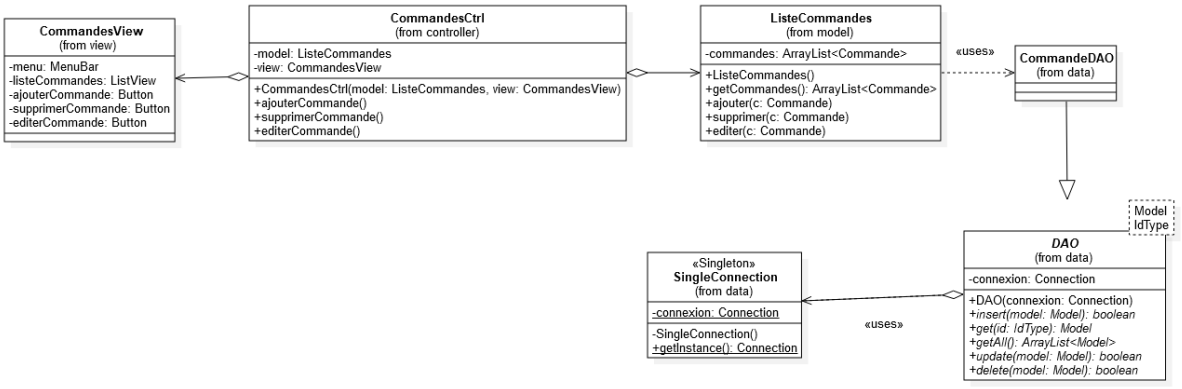

FIGURE 11 – Diagramme de classes de la partie "Commande"

Ce diagramme montre toutes les classes qui seront utiles pour la visualisation de commandes. Tout à gauche, nous avons la vue qui correspond à la page de la liste de commandes. Ensuite il y a le contrôleur, en position de médiateur, qui permet de gérer cette vue. Ce contrôleur est lié au modèle "ListeCommandes" qui contient la liste de toutes les commandes enregistrées dans la base de données. Cette classe peut être amener à faire appel au DAO des commandes afin d'interagir avec la base de données. Pour que l'interaction entre le DAO et la base ait lieu, il faut aussi faire appel au singleton "SingleConnection" qui est affiché dans ce diagramme. Ce singleton sert évidemment à obtenir une instance unique de la connexion vers la base de données.

2.1.2 Diagrammes de séquences

Un diagramme de séquence est un outil de modélisation permettant d'illustrer les interactions entre différents éléments d'un système. Il est souvent utilisé pour visualiser et analyser les interactions entre les objets dans un système en utilisant des boîtes rectangulaires pour représenter les objets et des flèches pour montrer comment ils interagissent. Un diagramme de séquence peut être utilisé pour montrer la chronologie des événements dans un système, en mettant en évidence les dépendances et les relations entre les différents éléments. En résumé, un diagramme de séquence est un outil utile pour comprendre les interactions complexes dans un système et pour identifier les points d'amélioration potentiels.

Afin de bien comprendre certaines parties du logiciel, nous avons choisi de réaliser quelques diagrammes de séquences. Nous nous sommes en particulier intéressés au diagramme concernant la connexion administrateur, ce qui nous a permis de mieux comprendre chaque partie active du logiciel et de réfléchir à comment nous souhaitions le

faire fonctionner.

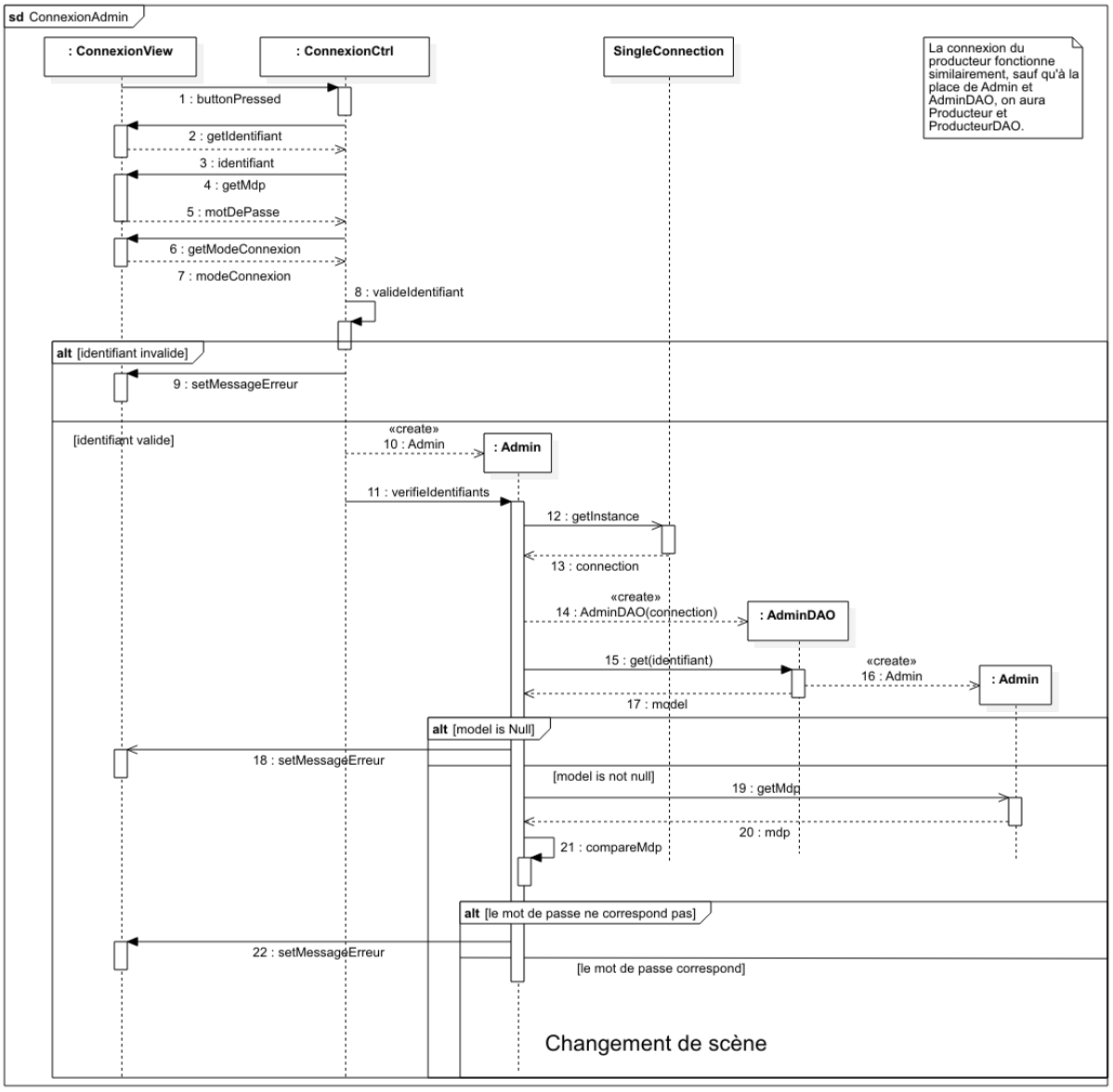

FIGURE 12 – Diagramme de séquence de la connexion administrateur

Le diagramme de séquence commence par l'appui de l'administrateur sur le bouton "se connecter". Le contrôleur récupère ensuite les données entrées et vérifie qu'elles sont bien toutes présentes et sous le bon format. Dans le cas contraire, il envoie une message d'erreur sur la page de connexion. Le contrôleur crée ensuite une instance de la classe Admin avec les informations saisies et lui demande de vérifier les identifiants. Pour cela il crée (ou récupère) une instance de SingleConnection afin d'avoir accès à la base de données. Il peut ensuite invoquer AdminDAO pour qu'il lui retourne une autre instance d'Admin avec comme identifiant celui entré par l'administrateur et comme mot de passe

celui trouvé dans la base de données pour cet identifiant. Si AdminDAO retourne null à la place d'une instance de Admin, un message d'erreur sera affiché. Dans le cas où l'utilisateur existe bien il vérifie que les deux mots de passe correspondent bien. S'il y a correspondance, l'utilisateur sera redirigé vers son tableau de bord, signalant le succès de l'authenfication. Dans le cas contraire, un message d'erreur sera affiché sur la page de connexion.

Nous n'avons pas fait tous les diagrammes de séquence car cela nous aurait demandé beaucoup trop de temps pour en tirer des bénéfices significatifs et l'impact sur le développement du logiciel aurait été faible.

2.1.3 Code

Génération automatique avec Rebel

Après la réalisation de ces diagrammes, nous avons pu générer les classes qui serviront pour le logiciel. Pour cela nous avons utilisé Rebel[2], disponible sur le registre d'extensions de StarUML.

La génération automatique du code nous a permis de gagner un temps non négligeable dans la réalisation du logiciel.

Nous avons ensuite supprimé les annotations laissées par Rebel dans le code afin d'en améliorer la lisibilité. Ces annotations auraient été pratiques dans le cas où on aurait été amené à modifier le diagramme de classe pour améliorer l'application ou la compléter, mais nous avons préféré les effacer. Si nous avons besoin de compléter notre application, nous ajouterons à la main les modifications que nous jugeons nécessaires et utiles dans les classes et méthodes.

Convention de codage et Checkstyle

Pour garder une cohérence dans l'écriture du code nous avons choisi d'utiliser une convention de codage déjà préparée pour gagner du temps. Plus précisément, nous avons opter pour les conventions de codage de Google, qui sont décrites dans le "Google Java Style Guide". Pour les appliquer de manière consistente, nous avons décidé d'utiliser Checkstyle[12] une extension dont de nombreux environnements de développement intégrés (IDE) disposent, notamment IntelliJ et Eclipse. Nous avons configuré Checkstyle avec "Google Checks"[4].

Les conventions de codage et de nommage en programmation ont pour but de faciliter la compréhension et la maintenance du code. En utilisant des conventions de codage et

de nommage standardisées, les développeurs peuvent facilement lire, relire et comprendre le code des autres, ce qui permet de travailler de manière plus efficace en équipe. De plus, en suivant des conventions de nommage claires et précises, il est plus facile de repérer les erreurs et les bugs dans le code, ce qui réduit les temps de débogage et améliore la qualité du logiciel. En résumé, l'utilisation de conventions de codage et de nommage est un élément crucial pour obtenir une meilleure qualité et efficacité de développement logiciel.

Ces conventions de codage ou de nommage consistent principalement à la syntaxe de nommage des classes ou méthodes selon plusieurs règles, la taille des indentations, etc.

Réalisation de la documentation avec JavaDoc

Après avoir généré le code, nous avons choisi de rédiger la documentation JavaDoc des méthodes et des attributs. Nous avons donc commencé par nous partager les différents packages afin que tout le monde puisse en écrire une partie. Nous avons ensuite commenté chaque méthode et attribut puis échangé nos packages afin de les relire pour en améliorer les descriptions.

Par la suite, nous avons généré les classes de tests puis commenté chaque test afin de définir précisément ce qu'il est censé tester.

2.1.4 Ecriture des tests

L'écriture des tests a été assez rapide grâce à l'écriture de la documentation faite avant. Nous avons choisi de ne pas prioriser le nombre de tests écrits mais de prioriser la qualité des tests. C'est à dire avoir des tests complets sur les classes principales, et pas des tests sur toutes les classes. Par exemple, on peut ignorer sans problème les tests des getters et setters qui ne contiennent pas de logique métier.

Nous avons choisi de ne pas tester les Getter et les Setter car ces méthodes sont assez simple et ont peu de chance d'avoir des bugs. Cela nous permettra de gagner du temps.

2.1.5 Requêtes

Voici quelques requêtes qui nous serviront lors de l'utilisation du logiciel.

Récupération du poids total de la tournée n°x

SQL

```
SELECT SUM(C.poids) FROM Tournee
INNER JOIN Commande C USING (numTournee)
GROUP BY numTournee
```

```
HAVING numTournee = x;
```

Algèbre relationnelle

$\pi_{\text{SUM(Commande.poids)}}$

$(\sigma_{\text{numTournee} = \text{x}}$

(Tournee $\bowtie_{\text{numTournee}}$ Commande)

)

Calcul relationnel de tuples

$R = \{\text{SUM(Commande.poids)} \mid$

$\exists$ Tournee$^{(\text{numTournee,libelle,heureMin,heureMax,immat})}$

$\exists$ Commande$^{(\text{numCom,libelle,poids,dateCom,heureDeb,heureFin,SIRET,idClient,numTournee})} \wedge$

Tournee.numTournee = Commande.numTournee $\wedge$

Tournee.numTournee $= x\}$

Calcul relationnel de domaine

$R = \{\text{SUM(p)} \mid$

$\exists\ t_0, t_1, t_2, t_3, t_4$

$\exists\ c_0, c_1, p, c_3, c_4, c_5, c_6, c_7$

Tournee(numTournee : $t_0, t_1, t_2, t_3, t_4) \wedge$

Commande(c_0, c_1, poids : $p, c_3, c_4, c_5, c_6, c_7$, numTournee : t_0) $\wedge$

$t_0 = x\}$

Récupération de l'adresse des clients de la tournée n°x

SQL

```
SELECT CONCAT(A.Ville, " ", A.numero, " ",
   A.voie, " ", A.nom, " (", A.mention,
   A.complement, ")") AS "Adresse"
FROM Tournee
INNER JOIN Commande USING (numTournee)
INNER JOIN Client USING (idClient)
INNER JOIN Adresse A USING (idAdresse)
GROUP BY numTournee
HAVING numTournee = x;
```

Algèbre relationnelle

$\pi_{\text{CONCAT(A.Ville, " ", A.numero, " ", A.voie, " ", A.nom, " (", A.mention, A.complement, ")")}}$

$(\sigma_{\text{numTournee = x}}$

(Tournee $\bowtie_{\text{numTournee}}$ Commande $\bowtie_{\text{idClient}}$Client $\bowtie_{\text{idAdresse}}$ Adresse A)

)

Calcul relationnel de tuples

$R = \{$CONCAT(A.Ville, " ", A.numero, " ", A.voie, " ", A.nom, " (", A.mention, A.complement, ")") |

$\exists$ Tournee$^{(\text{numTournee,libelle,heureMin,heureMax,immat})}$

$\exists$ Commande$^{(\text{numCom,libelle,poids,dateCom,heureDeb,heureFin,SIRET,idClient,numTournee})}$

$\exists$ Client$^{(\text{idClient,nomClient,prenomClient,numTel,gps,idAdresse})}$

$\exists$ Adresse$^{(\text{idAdresse,pays,codePost,ville,voie,nom,numero,mention,complement})}$

Tournee.numTournee = Commande.numTournee $\wedge$

Commande.idClient = Commande.idClient$\wedge$

Client.idAdresse = Adresse.idAdresse $\wedge$

Tournee.numTournee = $x\}$

Calcul relationnel de domaine

$R = \{$CONCAT$(d_3, \text{ " "}, d_5, \text{ " "}, d_3, \text{ " "}, d_4, \text{ " ("}, d_7, d_8, \text{ ")"}) |$

$\exists\ a_0, a_1, a_2, a_3, a_4$

$\exists\ b_0, b_1, p, b_3, b_4, b_5, b_6, b_7$

$\exists\ c_1, c_2, c_3, c_4, c_5$

$\exists\ d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8$

Tournee(numTournee : a_0, a_1, a_2, a_3, a_4)$\wedge$

Commande($b_0, b_1, p, b_3, b_4, b_5, b_6$, idClient : b_7, numTournee : a_0) $\wedge$

Client(idClient : b_7, c_1, c_2, c_3, c_4, idAdresse : c_5) $\wedge$

Adresse(idAdresse : c_5, pays : d_1, codePost : d_2, ville : d_3,

voie : d_4, nom : d_5, numero : d_6, mention : d_7, complement : d_8) $\wedge$

$a_0 = x\}$

2.2 Difficultés rencontrées

Diagramme de classe

Les problèmes rencontrés lors de la refonte du diagramme de classe ont été similaires aux difficultées rencontrées lors du premier jalon. Cependant, la réalisation de certains diagrammes de séquence nous a permis de mieux appréhender le fonctionnement du logiciel et donc de voir les méthodes ou attributs qui nous manquait.

Diagramme de séquence

La réalisation des diagrammes de séquence a été compliquée sur certains points car nous avons dû comprendre un peu plus en profondeur certaines parties du logiciel. Nous avons dû revoir le fonctionnement du design pattern DAO ainsi que des objets de la partie modèle et de leursliens avec le reste du programmme. Cependant, après avoir réfléchi suffisamment pour comprendre sur un seul diagramme ces problèmatiques, nous avons pu avancer bien plus rapidement sur les autres diagrammes car il y a beaucoup de similarités entre les différentes parties du logiciel.

Ecriture des tests

Nous n'avons pas rencontré de problème lors de l'écriture de la documentation générale. Cependant lors de l'écriture de la documentation des tests, nous avons eu un peu plus de mal car nous avons voulu être suffisamment exhaustif. L'écriture des tests étant quelque chose de très importants, nous n'avons pas voulu en oublier, ce qui est compliqué. Nous avons donc fait le choix d'en mettre le plus possible. Dans le cas où nous remarquons qu'il manque un test, nous l'ajouterons afin de rendre le logiciel plus sûr.

Bilan du deuxième jalon

Le projet n'est évidemment pas encore fini, cette fin de jalon nous permet de faire un bilan rapide avant de s'attaquer à la dernière partie du projet, la fin de la réalisation des tests et la réalisation du code. A la fin du projet, une conclusion globale sera réalisée afin de rendre compte de l'entièreté du projet. Ce bilan sera donc uniquement sur le deuxième jalon.

A la fin de ce jalon, nous avons senti une réelle avancée dans le projet. Le fait de commencer à voir et écrire du code nous a permis de nous dire que le projet était bien lancé. Toutes les réflexions précédentes nous ont permis d'écrire les commentaires et les premières lignes de code bien plus rapidement qu'habituellement.

La suite du projet sera donc principalement de l'écriture de code et l'exécution de tests afin d'avoir un logiciel fonctionnel. La réalisation de l'interface graphique fera aussi partie des tâches à accomplir. Pour le moment, nous avons commencé à la créer, mais nous la continuerons lors de la création du logiciel afin de mieux savoir de quoi nous aurons besoin et de faire l'interface au fur et à mesure des implémentations de fonctionnalités.

3 Troisième jalon

Le troisième jalon consiste en la réalisation du logiciel. C'est à dire, la fin de l'écriture du code, l'exécution des tests, la création de la base de données et l'ajout de données. Cette partie permet de remarquer tout ce qui a été oublié lors de la conception via le génie logiciel.

3.1 Réalisations

3.1.1 Ecriture des tests

Lors de ce jalon nous avons continué de rédiger une partie des tests afin d'avoir une couverture de test assez élevée. Cependant, nous avons dû choisir de ne pas tous les réaliser afin d'avoir suffisamment de temps pour réaliser l'application en tant que telle. Nous nous retrouvons donc au final avec une couverture des tests d'environ 60%.

3.1.2 Ecriture du code

La rédaction du code a été l'une des principales étapes de ce jalon. Pour nous organiser, nous avons tenu des réunions régulières afin de déterminer qui devait travailler sur quoi. Au départ, nous avons choisi de répartir les tâches par packages, c'est-à-dire que chaque personne s'occupait d'un certain type de code (vues, contrôleurs, etc.). Cependant, nous avons rapidement constaté que cette approche ralentissait considérablement notre travail et entravait notre motivation, nous avons donc décidé de changer notre stratégie et de coder par fonctionnalités afin de mieux suivre notre avancement. Par exemple, nous avons commencé par mettre en place la connexion utilisateur.

En parallèle de l'écriture du code, nous avons créé les interfaces graphiques de l'application à l'aide de SceneBuilder afin de pouvoir les afficher et de tester l'application. Nous avons également installé la base de données sur un serveur dédié pour éviter de devoir configurer un serveur sur chaque machine. Cela nous a permis de mener les tests et de vérifier que les interfaces fonctionnaient correctement.

Enfin, nous avons implémenté OpenStreetMap pour pouvoir visualiser les clients de la tournée sur une carte. Cette tâche s'est avérée assez complexe en raison de certains problèmes de compatibilité entre les bibliothèques.

Lors des derniers jours avant la fin du jalon, nous avons remarqué que nous n'avions pas implémenté l'ordre des commandes car nous pensions qu'un solveur devait le faire et donc que nous devions juste créer un logiciel auquel un solveur pourrait être implémenté

afin de gérer cette ordre des commandes. Cependant, après relecture du sujet, nous avons vu que le producteur proposerait lui même ses trajets, il a donc fallu implémenter l'ordre des commandes assez rapidement.

3.1.3 Ajout de données

L'ajout de données dans la base de données a été l'une des étapes les plus longues, en dépit de la relative facilité de la tâche en elle-même. Cela a été principalement dû au fait que nous avons dû trouver comment générer ces données de manière efficace.

Au départ, nous avons choisi d'utiliser la Base d'Adresses Nationales Ouvertes (IGN) pour générer les adresses. Cependant, nous avons rapidement réalisé que cela prenait beaucoup de temps et ne produisait pas de résultats très aléatoires. Nous avons donc opté pour une stratégie différente : générer des coordonnées GPS en France, puis vérifier si l'adresse correspondante existait dans la Base d'Adresses Nationales Ouvertes[1] et récupérer les informations associées. Cette approche nous a permis de gagner du temps et d'obtenir des résultats plus aléatoires.

Pour générer les noms, nous avons d'abord essayé d'utiliser chatGPT (un modèle de langage créé par OpenAI qui permet de produire du texte à la demande). Cependant, nous avons vite constaté que cette méthode était très lente et peu aléatoire. Nous avons donc opté pour la bibliothèque Java Faker, qui nous a permis de générer facilement tous les types de données nécessaires. Une fois ces données générées, nous les avons envoyées dans la base de données afin de pouvoir les utiliser dans l'application.

3.1.4 Exécution des tests

L'exécution des tests a été plutôt problèmatique, car lors du codage, nous n'avons pas touché aux tests afin d'avoir un logiciel fonctionnel, cependant, certaines méthodes prévues au départ ont été supprimées, d'autres ont été ajoutées et certains ont changées d'objectif. Cela a entraîné des erreurs dans de nombreux tests. Nous avons donc choisi d'en remettre quelques uns à jour afin d'avoir une couverture de test suffisante sur certains des packages.

La couverture des tests unitaires est donc de : 77 méthodes / 133 méthodes, donc environ 58%, ce qui est bas.

3.2 Difficultés rencontrées

Pendant ce dernier jalon, nous avons rencontré deux principales difficultés : organiser notre temps de travail afin de terminer le logiciel dans les délais impartis et trouver la motivation nécessaire pour continuer à avancer. La première difficulté a été résolue lorsque nous avons pu reprendre le travail à plusieurs, après les fêtes, et que nous avons vu l'application prendre forme. La motivation est revenue grâce aux réunions journalières que nous avons tenues pendant la dernière semaine, afin de pouvoir nous entraider et nous organiser.

Pendant la phase de codage, nous avons également rencontré plusieurs difficultés. Nous avons dû effectuer plusieurs gros nettoyages dans le code pour éviter de mélanger les vues et les contrôleurs. Nous avons aussi eu du mal avec certains formulaires, notamment en ce qui concerne la récupération et la vérification des données, en raison de problèmes que nous n'avions pas anticipés lors de la conception.

L'implémentation de l'ordre des commandes dans les tournées a été sujet de plusieurs difficultés à cause d'un manque de réflexion à ce propos. Cela nous a donc demandé une refonte de beaucoup de choses dans le logiciel et la base de données afin d'avoir quelque chose de fonctionnel assez rapidement.

La génération de données a également posé plusieurs problèmes, en particulier pour la génération d'adresses. Cela s'explique par le grand nombre de différences pouvant exister entre deux adresses (par exemple, certaines n'ont pas de numéro, d'autres ont un numéro avec une mention, comme "bis" ou "ter"). La génération de commandes et de tournées a également été complexe, car elles impliquaient la plupart des autres classes, ce qui signifiait qu'une erreur dans une autre classe entraînait une erreur dans ces classes. Nous avons donc dû nous assurer que toutes les autres classes fonctionnaient correctement avant de pouvoir tester ces classes.

Comme la génération d'adresses est aléatoire, il arrive que les clients soient très éloignés des producteurs, ce qui va à l'encontre du concept de circuit court. Nous avons donc décidé de prendre le temps de résoudre ce problème afin d'avoir des données plus cohérentes et qui respectent mieux le principe du circuit court.

3.3 Améliorations possibles pour le futur

Bien que notre logiciel soit terminé et fonctionnel, nous avons identifié une liste de points à améliorer pour rendre son utilisation plus agréable et complète.

Par exemple, l'ajout d'un solveur qui proposerait au producteur un ordre des comamdnes pour sa tournée lui éviterait de devoir réfléchir longuement pour élaborer sa tournée. Cependant, étant donné que la réalisation d'un solveur est complexe et prend du temps, nous avons décidé de ne pas poursuivre cette piste.

Un autre exemple d'amélioration concerne la gestion de la confidentialité. Nous pourrions par exemple lier les clients aux producteurs de manière à ce que, lorsqu'un producteur sélectionne un client dans la base de données, il ne voie pas tous les autres clients des autres producteurs. De même, actuellement, un admin voit tous les producteurs, alors qu'il devrait seulement voir ceux qu'il est censé gérer. Cela éviterait le problème de confidentialité actuel.

Il serait possible d'améliorer certaines données, comme les informations sur les véhicules. Actuellement, un producteur doit connaître les plaques d'immatriculation de ses véhicules pour savoir lequel il souhaite utiliser pour sa tournée. En ajoutant le modèle et la marque du véhicule dans la base de données, nous pourrions résoudre ce problème.

Nous pourrions également envisager des améliorations plus profondes, comme la gestion des stocks du producteur. Cela passerait par une centralisation des outils, ce qui permettrait d'éviter de devoir utiliser plusieurs logiciels différents.

L'une des modifications à faire en priorité serait d'optimiser le logiciel, le temps de lancement est actuellement trop long pour que son utilisation soit agréable, pareil pour le temps de déconnexion.

Une amélioration utile serait d'avoir une page de détails des tournées afin de voir pourquoi une commande n'est pas valide et voir la liste des commandes sous un autre format afin d'améliorer l'affichage. Il y a plusieurs autres petites améliorations d'interfaces qui pourraient être faites mais que nous n'avons pas eu le temps d'implémenter.

Conclusion

Ce projet a été intéressant à réaliser, notamment grâce à l'approche basée sur le génie logiciel, qui était nouvelle pour nous. Le sujet étant actuel et important, cela nous a motivés pour réussir le projet.

Toutefois, la dernière semaine avant la remise a été plutôt compliquée, car nous avons dû coder une grande partie du logiciel. L'approche basée sur le génie logiciel nous a aidés à avancer rapidement, mais nous avons rencontré des concepts de programmation que nous n'avions pas encore eu l'occasion de mettre en pratique à cette échelle, ce qui a ralenti le développement afin de mieux comprendre ces concepts.

L'approche basée sur le génie logiciel nous a permis de comprendre à quel point il est important de réfléchir au produit que nous voulons concevoir avant sa réalisation, afin d'éviter les changements massifs qui nécessitent de modifier des parties déjà terminées du logiciel et qui peuvent entraîner des problèmes sur les parties fonctionnelles. Cette expérience nous a permis de découvrir certaines bonnes et mauvaises pratiques et de nous améliorer grâce au travail.

Bien que le logiciel soit terminé et remis, il pourrait être intéressant de l'améliorer afin de le rendre plus utilisable et complet. Nous avons donc réfléchi aux éventuelles améliorations qui pourraient être apportées, comme indiqué précédemment. Nous pensons qu'il est important de prévoir ce qui pourra être fait ultérieurement, notamment avant la conception, afin que le logiciel puisse être amélioré et complété tout en restant durable.

Bibliographie

[1] Gouvernement Français , OpenStreetMap. *Base d'Adresses Nationale Ouverte - BANO*. https://bano.openstreetmap.fr/data/.

[2] Archetype Software Engineering. *Rebel for StarUML : A powerful and time-saving code generator*. https://www.archetypesoftware.com/.

[3] Alder Gaudenz et Benson David. *diagrams.net*. https://app.diagrams.net/.

[4] Google. *Google Java Style Guide*. https://google.github.io/styleguide/javaguide.html.

[5] Jetbrains. *Essential tools for software developers and teams*. https://www.jetbrains.com/.

[6] Jetbrains. *Youtrack : Project Management*. https://www.jetbrains.com/youtrack/.

[7] Microsoft. *Github*. https://github.com/.

[8] Ltd. MKLabs Co. *StarUML*. https://staruml.io/.

[9] Fondation OpenStreetMap. *OpenStreetMap*. https://www.openstreetmap.org.

[10] Arnaud Roques. *Plant UML*. https://plantuml.com/.

[11] Scrum.org. *What is Scrum ?* https://www.scrum.org/resources/what-is-scrum/.

[12] Jamie Shiell. *CheckStyle-IDEA*. https://plugins.jetbrains.com/plugin/1065-checkstyle-idea.

[13] Université de Toulouse III. *Looping - Modélisation Conceptuelle de données*. https://www.looping-mcd.fr/.

Annexes

Scénarii

Acteurs

Acteur principal : Utilisateur (Producteur ou Administrateur)

Préconditions :

— Le logiciel est lancé

— Le producteur a un compte

— L'administrateur a un compte

— Le producteur doit livrer des commandes

Scénarii de connections

Scénario : Connexion du producteur

Scénario nominal

1. Le producteur entre son identifiant (courriel ou le numéro SIRET de son entreprise)
2. Le producteur entre son mot de passe
3. Le producteur valide et clique sur "Se connecter"
4. Le programme valide les champs
5. Le programme vérifie que l'identifiant et mot de passe renseignés ont une correspondance dans la base de données
6. Le programme redirige le producteur sur son tableau de bord, signalant le succès de la connexion

Scénarii alternatifs

1. `Le producteur entre un identifiant et un mot de passe qui n'ont aucune correspondance dans la base de données`
   a. Le logiciel renvoie une erreur "identifiant ou mot de passe incorrect"
   b. Le scénario nominal reprend au point 1
2. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1

FIGURE 13 – Scénario de connexion du producteur

Scénario : Connection de l'administrateur

Scénario nominal

1. L'administrateur entre son pseudonyme ou son courriel
2. L'administrateur entre son mot de passe
3. L'administrateur valide
4. Le programme valide les champs
5. Le programme vérifie que l'identifiant et mot de passe renseignés ont une correspondance dans sa base de données
6. Le programme redirige l'administrateur sur son tableau de bord, signalant le succès de la connexion

Scénarii alternatifs

1. `L'administrateur entre un identifiant et un mot de passe qui n'ont aucune correspondance dans la base de données`
   a. Le logiciel renvoie une erreur "identifiant ou mot de passe incorrect"
   b. Le scénario nominal reprend au point 1
2. `L'administrateur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1

FIGURE 14 – Scénario de la connection de l'administrateur

Scénarii d'ajout de données

Scénario : Ajout de commandes

Scénario nominal

1. Le producteur entre les informations de la commande (choix du client, horaire minimale et maximale de livraison, poids de la commande)
2. Le producteur valide et envoie le formulaire
3. Le programme valide les champs du formulaire
4. Le programme enregistre les informations renseignées dans sa base de données
5. Le programme informe l'utilisateur que les informations ont bien été enregistrées

Scénarii alternatifs

1. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `Le producteur saisit une valeur invalide`
   a. Le logiciel renvoie une erreur qui indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule sa saisie en cliquant sur le bouton "Annuler" sur le formulaire`
   a. Le logiciel enlève le formulaire de l'affichage
   b. Le logiciel redirige le producteur vers le tableau de bord

FIGURE 15 – Scénario d'ajout de commandes

Scénario : Ajout de tournées

Scénario nominal

1. Le producteur entre les informations de la tournée
2. Le producteur ajoute les commandes qui seront livrées lors de cette tournée
3. Le producteur valide le formulaire
4. Le programme valide le formulaire
5. Le programme enregistre les informations renseignées dans sa base de données
6. Le programme informe l'utilisateur que les informations ont été enregistrées

Scénarii alternatifs

1. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `Le producteur entre une valeur invalide`
   a. Le logiciel renvoie une erreur qui indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
3. `La tournée n'est pas valide à cause du poids des commandes et/ou des horaires`
   a. Le logiciel renvoie une erreur
   b. Le scénario nominal reprend au point 2
4. `Le producteur annule sa saisie en cliquant sur le bouton "Annuler" sur le formulaire`
   a. Le logiciel enlève le formulaire de l'affichage
   b. Le logiciel redirige le producteur vers le tableau de bord

FIGURE 16 – Scénario d'ajout de tournées

Scénario : Ajout de client

Scénario nominal

1. L'utilisateur entre les informations clients
2. L'utilisateur valide le formulaire
3. Le programme valide les champs du formulaire
4. Le programme enregistre les informations client dans sa base de données
5. Le programme informe l'utilisateur que les informations ont été enregistrées

Scénarii alternatifs

1. `L'utilisateur n'entre pas une des informations requises`
   a. Le logiciel renvoie une erreur "les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `L'utilisateur entre une valeur invalide`
   a. Le logiciel renvoie une erreur qui indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule sa saisie en cliquant sur le bouton "Annuler" sur le formulaire`
   a. Le logiciel enlève le formulaire de l'affichage
   b. Le logiciel redirige l'utilisateur vers le tableau de bord

FIGURE 17 – Scénario de l'ajout de client

Scénario : Ajout d'un producteur

Scénario nominal

Préconditions L'administrateur est connecté

1. L'administrateur entre le SIRET de l'entreprise du producteur, son mail, son nom et prénom ainsi que son numéro de téléphone
2. L'administrateur entre un mot de passe pour le producteur
3. L'administrateur valide
4. Le programme valide les champs
5. Le programme ajoute le producteur dans la base de données
6. Le programme redirige l'administrateur sur son tableau de bord, signalant le succès de l'ajout

Scénarii alternatifs

1. `L'administrateur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `L'administrateur annule sa saisie en cliquant sur le bouton "Annuler" sur le formulaire`
   a. Le logiciel enlève le formulaire de l'affichage

FIGURE 18 – Scénario d'ajout de producteurs

Scénario : Ajout d'un véhicule

Scénario nominal

1. Le producteur entre le numéro d'immatriculation du véhicule et le poids maximal de produits qu'il peut transporter
2. Lle producteur valide le formulaire
3. Le programme valide les champs du formulaire
4. Le programme ajoute le véhicule dans la base de données
5. Le programme informe l'utilisateur que les informations ont été enregistrées

Scénarii alternatifs

1. `Le producteur entre une valeur invalide dans le formulaire`
   a. Le logiciel renvoie une erreur et indique le ou les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
2. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule sa saisie en cliquant sur le bouton "Annuler" sur le formulaire`
   a. Le logiciel enlève le formulaire de l'affichage

FIGURE 19 – Scénario d'ajout de véhicules

Scénarii de modifications de données

Scénario : Modification d'une commande

Scénario nominal

1. Le logiciel affiche le même formulaire que l'ajout d'une commande mais les champs sont remplies avec les informations actuelles
2. Le producteur modifie une ou plusieurs information(s) qu'il souhaite changer
3. Le producteur valide et envoie le formulaire
4. Le programme valide les champs du formulaire
5. Le programme enregistre les informations renseignées dans sa base de données
6. Le programme informe l'utilisateur que les informations ont bien été enregistrées

Scénarii alternatifs

1. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `Le producteur saisit une valeur invalide`
   a. Le logiciel renvoie une erreur et indique le ou les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule la modification de la commande`
   a. Le logiciel retire le formulaire de l'interface

FIGURE 20 – Scénario de modification d'une commande

Scénario : Modification d'une tournée

Scénario nominal

1. Le logiciel affiche le même formulaire que l'ajout d'une tournée mais les champs sont remplies avec les informations actuelles
2. Le producteur modifie une ou des information(s) qu'il souhaite changer
3. Le producteur ajoute ou enlève des commandes qui seront livrées lors de cette tournée
4. Le producteur valide le formulaire
5. Le programme valide le formulaire
6. Le programme enregistre les informations renseignées dans sa base de données
7. Le programme informe l'utilisateur que les informations ont été enregistrées

Scénarii alternatifs

1. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivnet être renseignés"
   b. Le scénario nominal reprend au point 1
2. `Le producteur saisit une valeur invalide`
   a. Le logiciel renvoie une erreur et indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
3. `La tournée n'est pas valide à cause du poids des commandes et/ou des horaires`
   a. Le logiciel renvoie une erreur
   b. Le scénario nominal reprend au point 2
4. `Le producteur annule la modification de la tournée`
   a. Le logiciel retire le formulaire de l'interface

FIGURE 21 – Scénario de modification d'une tournée

Scénario : Modification de client

Scénario nominal

1. Le logiciel affiche le même formulaire que l'ajout de client mais les champs sont remplies avec les informations actuelles
2. L'utilisateur modifie une ou plusieurs information(s) qu'il souhaite changer
3. L'utilisateur valide le formulaire
4. Le programme valide les champs du formulaire
5. Le programme enregistre les informations client dans sa base de données
6. Le programme informe l'utilisateur que les informations ont été enregistrées

Scénarii alternatifs

1. `L'utilisateur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 1
2. `L'utilisateur entre une valide invalide`
   a. Le logiciel renvoie une erreur et indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule la modification sur le client`
   a. Le logiciel retire le formulaire de l'interface

FIGURE 22 – Scénario de la modification d'un client

Scénario : Modification d'un producteur par un administrateur

Scénario nominal

Préconditions L'administrateur est connecté

1. Le logiciel affiche le même formulaire que l'ajout d'un producteur mais les champs sont remplies avec les informations actuelles
2. L'administrateur modifie une ou plusieurs information(s) qu'il souhaite changer
3. L'administrateur valide
4. Le programme demande à l'administrateur son mot de passe par sécurité
5. L'administrateur entre son mot de passe afin que les changements soient effectifs
6. Le programme vérifie que le mot de passe est correcte
7. Le programme valide les champs du formulaire
8. Le programme modifie le producteur dans la base de données
9. Le programme redirige l'administrateur sur son tableau de bord, signalant le succès de la modification

Scénarii alternatifs

1. `L'administrateur saisit un mot de passe incorrect lors de la validation du formulaire`
   a. Le logiciel renvoie une erreur "mot de passe incorrect"
   b. Le scénario nominal reprend au point 4
2. `L'administrateur saisit une valeur invalide`
   a. Le logiciel renvoie une erreur et indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 2
3. `L'administrateur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 2
4. `L'administrateur annule la modification sur le producteur`
   a. Le logiciel retire le formulaire de l'interface

FIGURE 23 – Scénario de modification d'un producteur par un administrateur

Scénario : Modification d'un producteur par lui-même

Scénario nominal

Préconditions Le producteur est connecté

1. Le logiciel affiche le même formulaire que l'ajout d'un producteur mais les champs sont remplies avec les informations actuelles
2. Le producteur modifie une ou plusieurs information(s) qu'il souhaite changer
3. Le producteur valide
4. Le producteur entre son mot de passe afin que les changements soient effectifs
5. Le programme valide les champs
6. Le programme ajoute le producteur dans la base de données
7. Le programme redirige l'administrateur sur son tableau de bord, signalant le succès de la modification

Scénarii alternatifs

1. `Le producteur saisit une valeur invalide`
   a. Le logiciel renvoie une erreur et indique les champs dont la valeur est invalide
   b. Le scénario nominal reprend au point 1
2. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs requis doivent être renseignés"
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule la modification de ses informations`
   a. Le logiciel retire le formulaire de l'interface

FIGURE 24 – Scénario de modification d'un producteur par lui-même

Scénario : Modification d'un véhicule

Scénario nominal

1. Le logiciel affiche le même formulaire que l'ajout d'un véhicule mais les champs sont remplies avec les informations actuelles
2. Le producteur modifie une ou plusieurs information(s) qu'il souhaite changer
3. Lle producteur valide le formulaire
4. Le programme valide les champs du formulaire
5. Le programme modifie le véhicule dans la base de données
6. Le programme informe l'utilisateur que les informations ont été enregistrées

Scénarii alternatifs

1. `Le producteur entre une valeur invalide dans le formulaire`
   a. Le logiciel renvoie une erreur et indique le ou les champs dont la valeur est invaldie
   b. Le scénario nominal reprend au point 1
2. `Le producteur ne saisit pas tous les champs requis`
   a. Le logiciel renvoie une erreur "tous les champs doivent être renseignés"
   b. Le scénario nominal reprend au point 1
3. `Le producteur annule sa saisie en cliquant sur le bouton "Annuler" sur le formulaire`
   a. Le logiciel enlève le formulaire de l'affichage
4. `Le producteur annule la modification sur le véhicule`
   a. Le logiciel retire le formulaire de l'interface

FIGURE 25 – Scénario de modification d'un véhicule

Scénarii de suppressions de données

Scénario : Supressions de commandes

Scénario nominal

1. Le producteur sélectionne une commande à supprimer
2. Le programme demande confirmation sur la suppression
3. Le producteur valide son choix
4. Le programme enlève les données sur la commande dans sa base de données
5. Le programme informe l'utilisateur que les informations ont bien été supprimées

Scénarii alternatifs

1. `Le producteur annule la confirmation en cliquant sur le bouton "Annuler"`
   a. Le logiciel enlève la confirmation de l'affichage
   b. Le logiciel redirige le producteur vers le tableau de bord

FIGURE 26 – Scénario de suppressions de commandes

Scénario : Supressions de tournées

Scénario nominal

1. Le producteur sélectionne une tournée à supprimer
2. Le programme demande confirmation sur la suppression
3. Le producteur valide son choix
4. Le programme enlève les données sur la tournée dans sa base de données
5. Le programme informe l'utilisateur que les informations ont bien été supprimées

Scénarii alternatifs

1. `Le producteur annule la confirmation en cliquant sur le bouton "Annuler"`
   a. Le logiciel enlève la confirmation de l'affichage
   b. Le logiciel redirige le producteur vers le tableau de bord

FIGURE 27 – Scénario de suppressions de tournées

Scénario : Supressions de clients

Scénario nominal

1. L'utilisateur sélectionne un client à supprimer
2. Le programme demande confirmation sur la suppression
3. L'utilisateur valide son choix
4. Le programme enlève les données sur la commande dans sa base de données
5. Le programme informe l'utilisateur que les informations ont bien été supprimées

Scénarii alternatifs

1. `L'utilisateur annule la confirmation en cliquant sur le bouton "Annuler"`
   a. Le logiciel enlève la confirmation de l'affichage
   b. Le logiciel redirige l'utilisateur vers le tableau de bord

FIGURE 28 – Scénario de suppressions de clients

Scénario : Supression de producteurs

Scénario nominal

Préconditions L'administrateur est connecté

1. L'administrateur sélectionne une commande à supprimer
2. Le programme demande confirmation sur la suppression
3. L'administrateur valide son choix
4. L'administrateur entre son mot de passe afin que la suppresion soit effectuée
5. Le programme valide les champs
6. Le programme enlève les données sur la commande dans sa base de données
7. Le programme informe l'utilisateur que les informations ont bien été supprimées

Scénarii alternatifs

1. `L'administrateur annule la confirmation en cliquant sur le bouton "Annuler"`
   a. Le logiciel enlève la confirmation de l'affichage
   b. Le logiciel redirige l'administrateur vers le tableau de bord
2. `L'administrateur n'entre pas le bon mot de passe pour confirmer la suppression`
   a. Le logiciel renvoie une erreur "Le mot de passe n'est pas correct"
   b. Le scénario nominal reprend au point 1

FIGURE 29 – Scénario de suppressions de producteurs

Scénario : Supression de véhicules

Scénario nominal

1. Le producteur sélectionne un véhicule à supprimer
2. Le programme demande confirmation sur la suppression
3. Le producteur valide son choix
4. Le programme enlève les données sur le véhicule dans sa base de données
5. Le programme informe l'utilisateur que les informations ont bien été supprimées

Scénarii alternatifs

1. `Le producteur annule la confirmation en cliquant sur le bouton "Annuler"`
   a. Le logiciel enlève la confirmation de l'affichage
   b. Le logiciel redirige le producteur vers le tableau de bord

FIGURE 30 – Scénario de suppressions de véhicules

Diagrammes

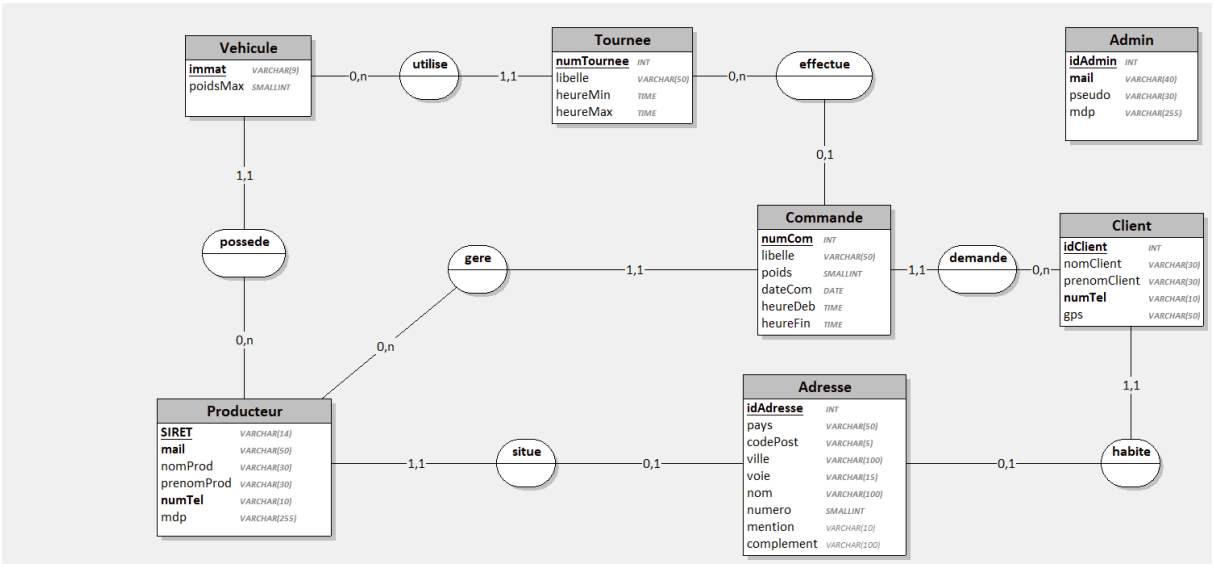

FIGURE 31 – Modèle conceptuel de données pour le projet

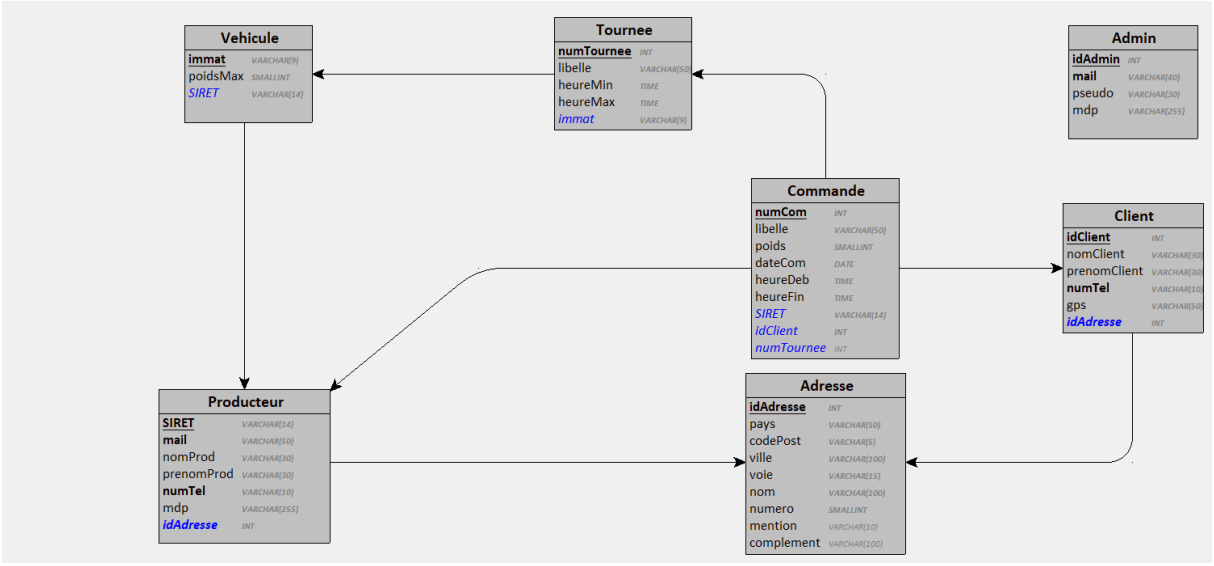

FIGURE 32 – Modèle logique de données pour le projet

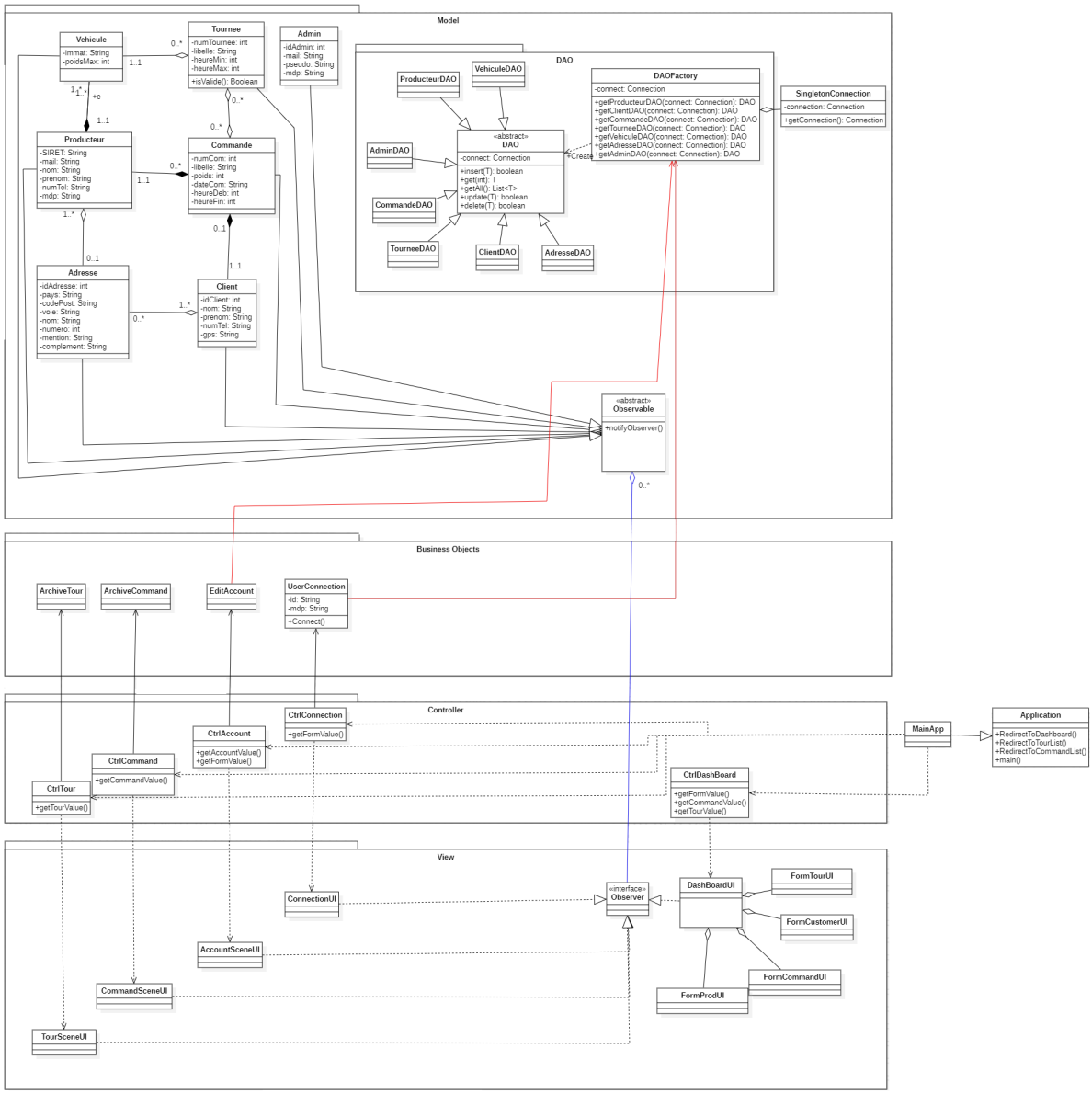

FIGURE 33 – Diagramme de classe

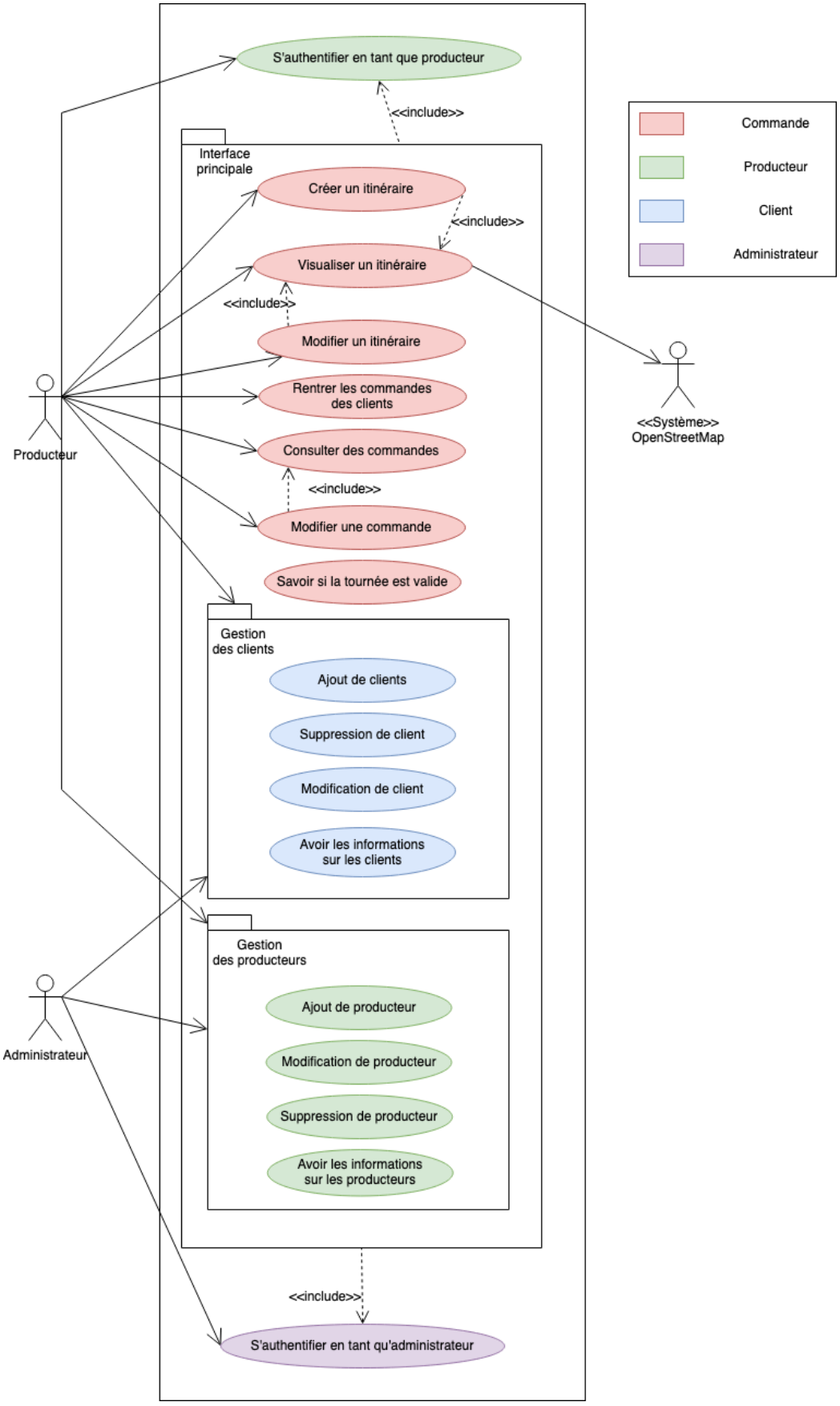

FIGURE 34 – Diagramme de cas d'utilisations

Diagramme du deuxième jalon

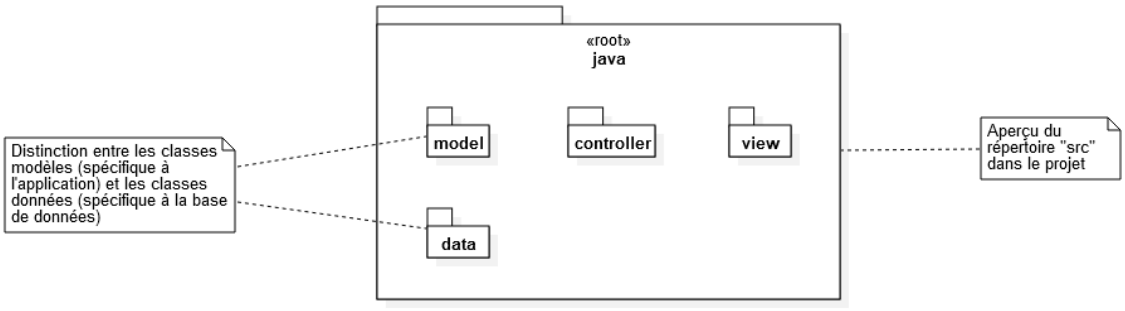

FIGURE 35 – Diagramme de classe global

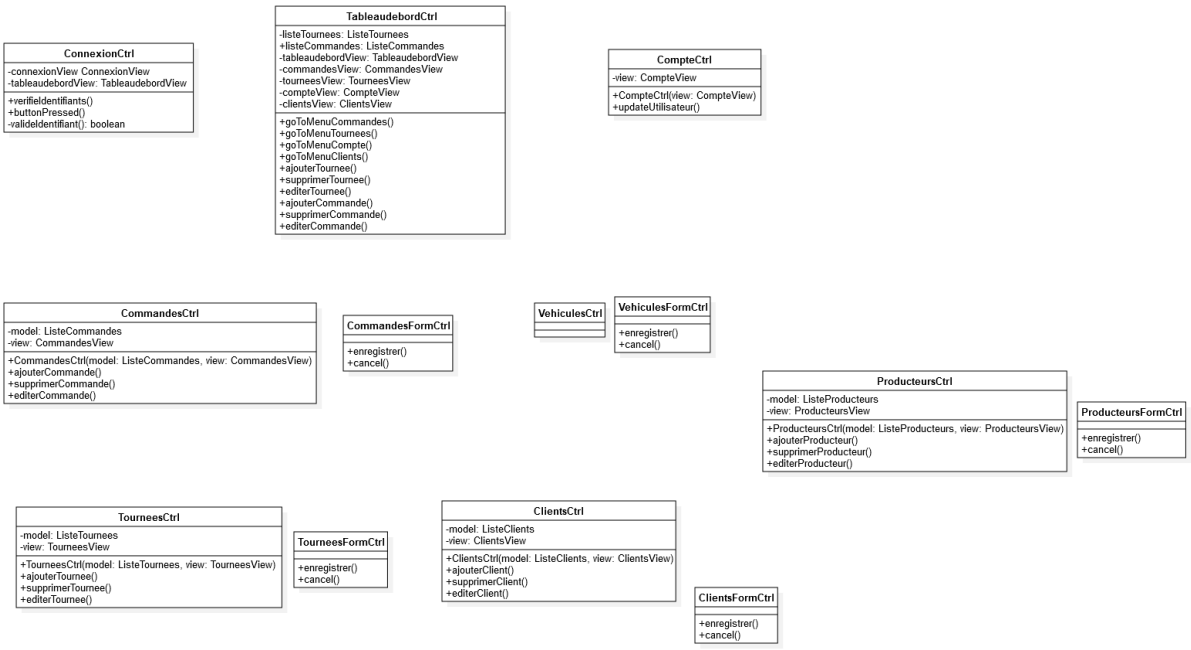

FIGURE 36 – Diagramme de classe des contrôleurs

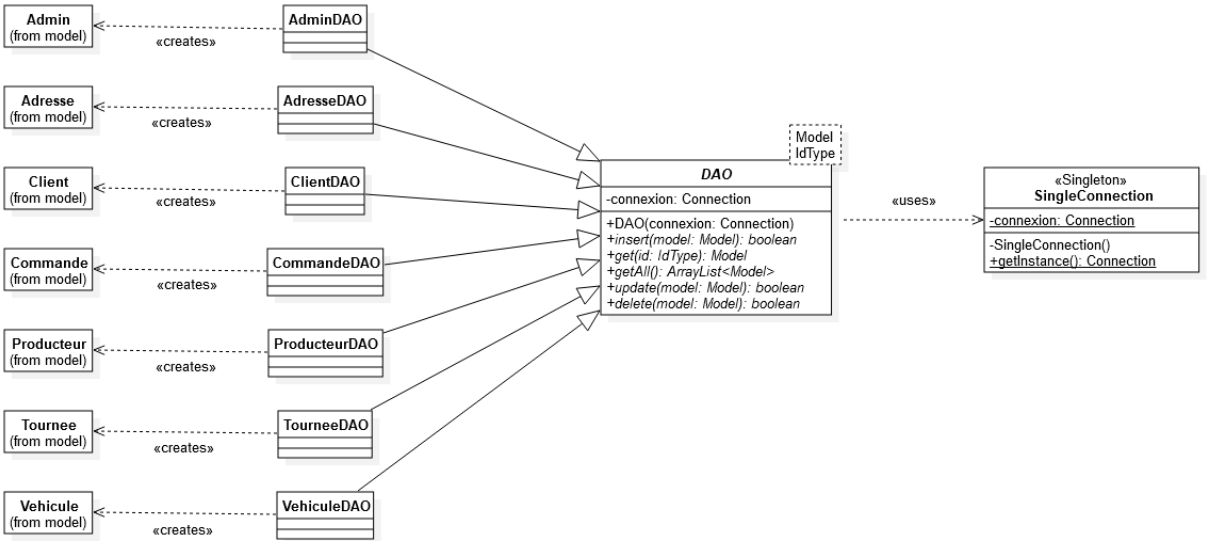

FIGURE 37 – Diagramme de classe des données

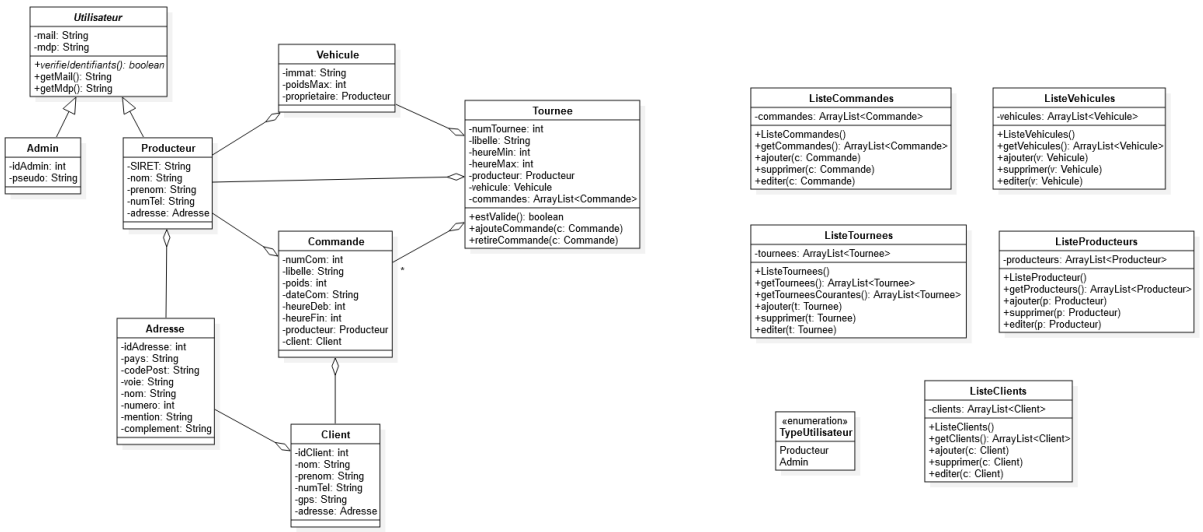

FIGURE 38 – Diagramme de classe des modèles

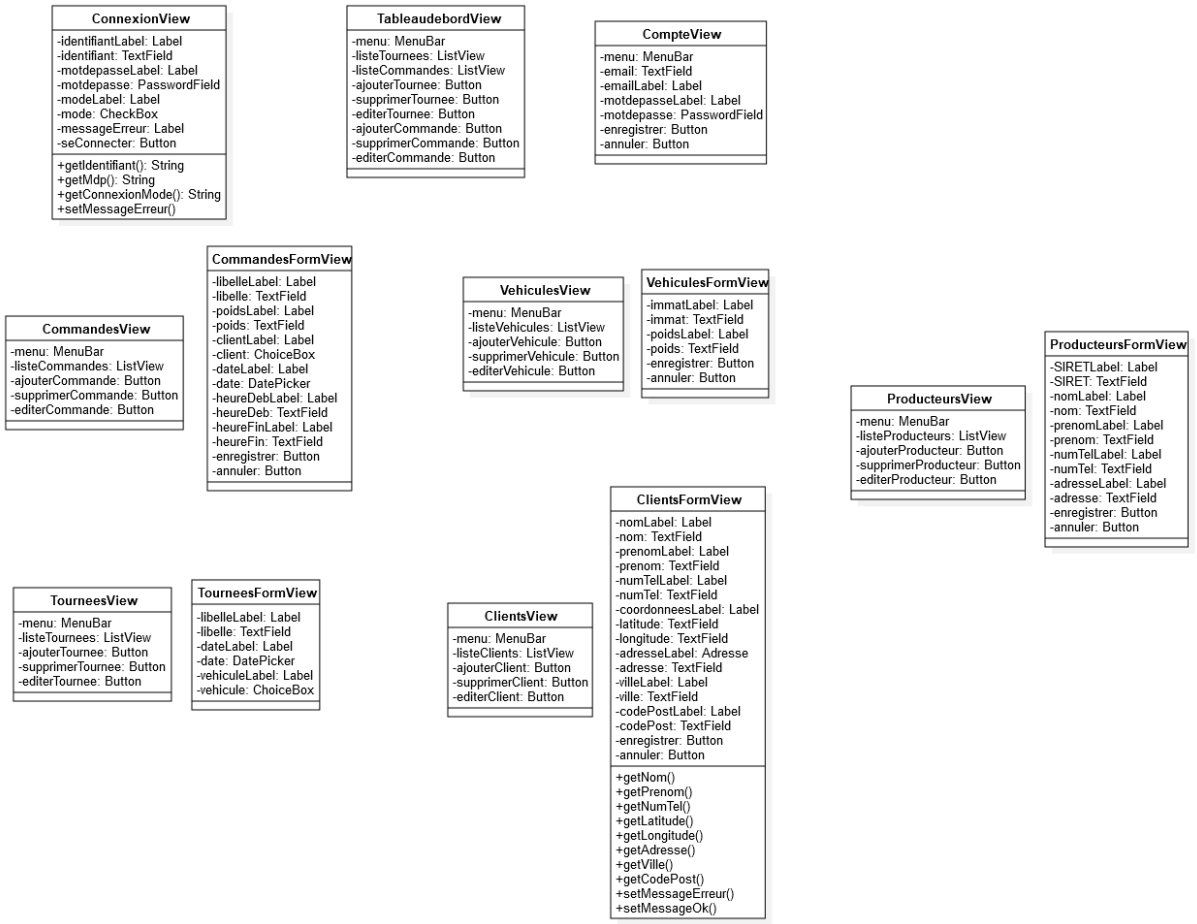

FIGURE 39 – Diagramme de classe des vues

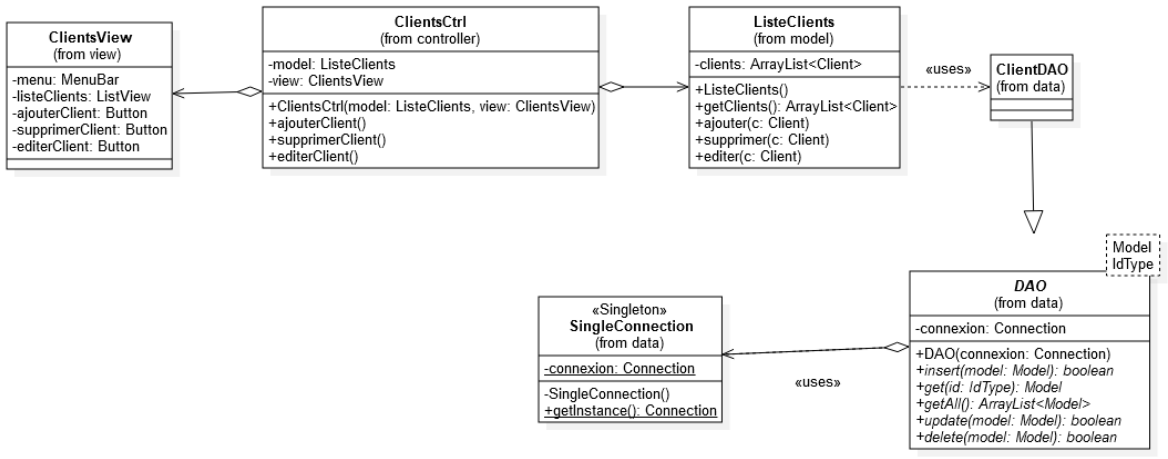

FIGURE 40 – Diagramme de classe de la partie Client

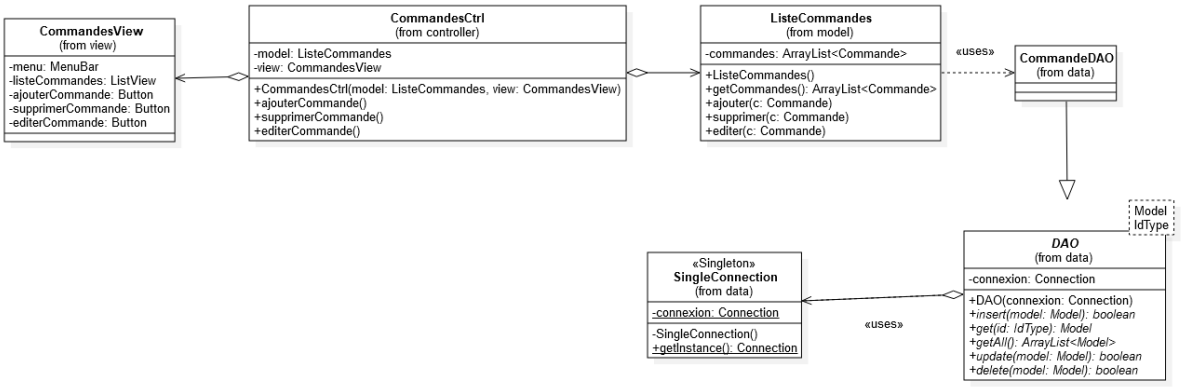

FIGURE 41 – Diagramme de classe de la partie Commande

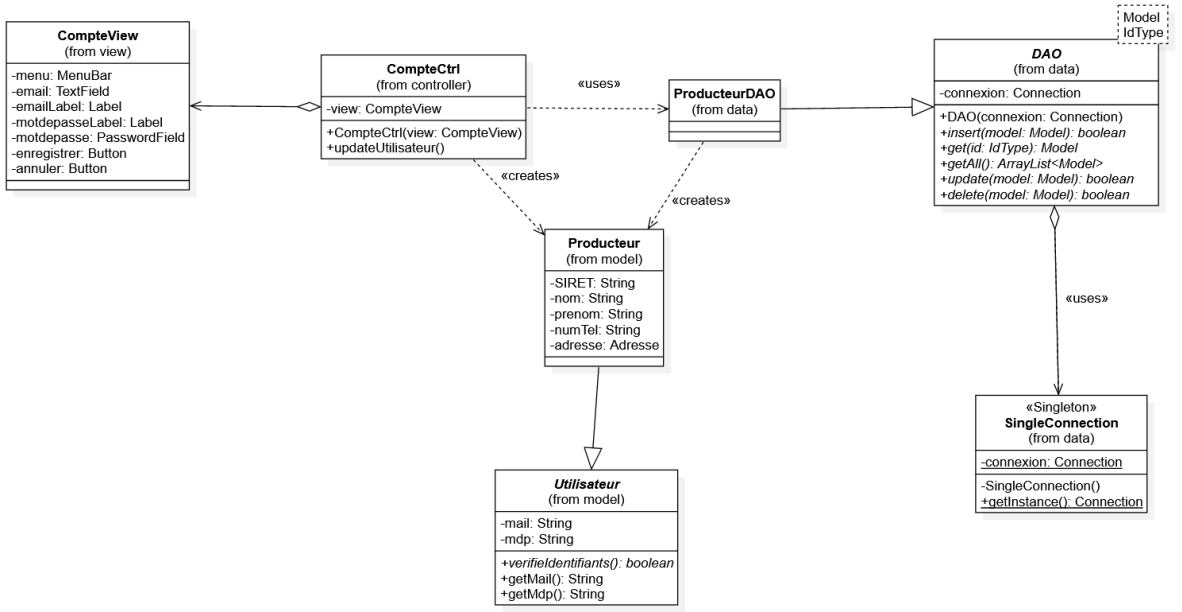

FIGURE 42 – Diagramme de classe de la partie Compte Producteur

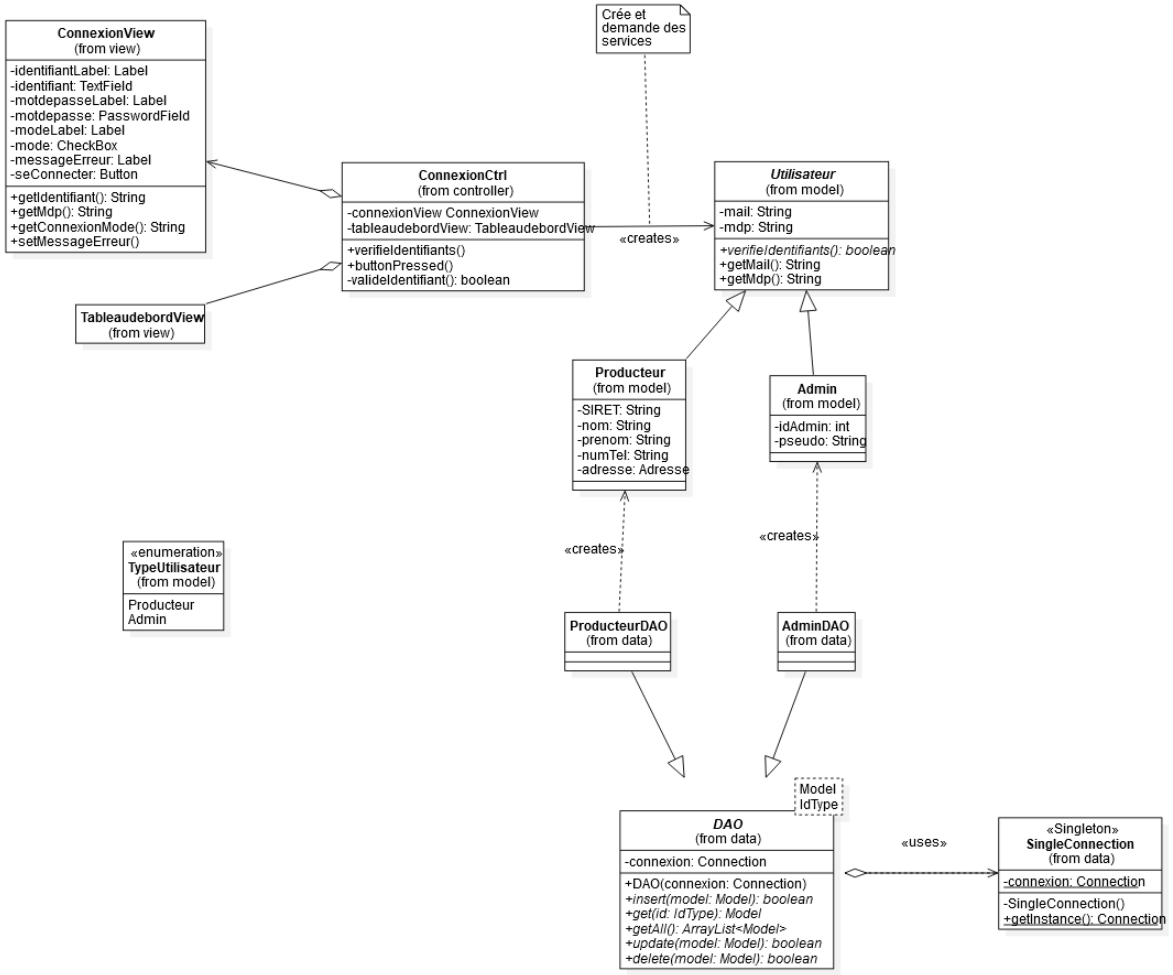

FIGURE 43 – Diagramme de classe de la partie Connexion

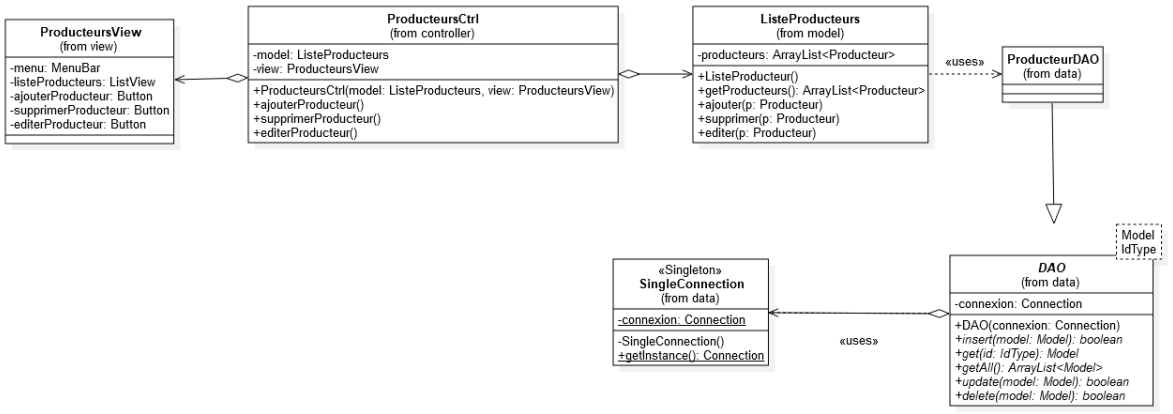

FIGURE 44 – Diagramme de classe de la partie producteur

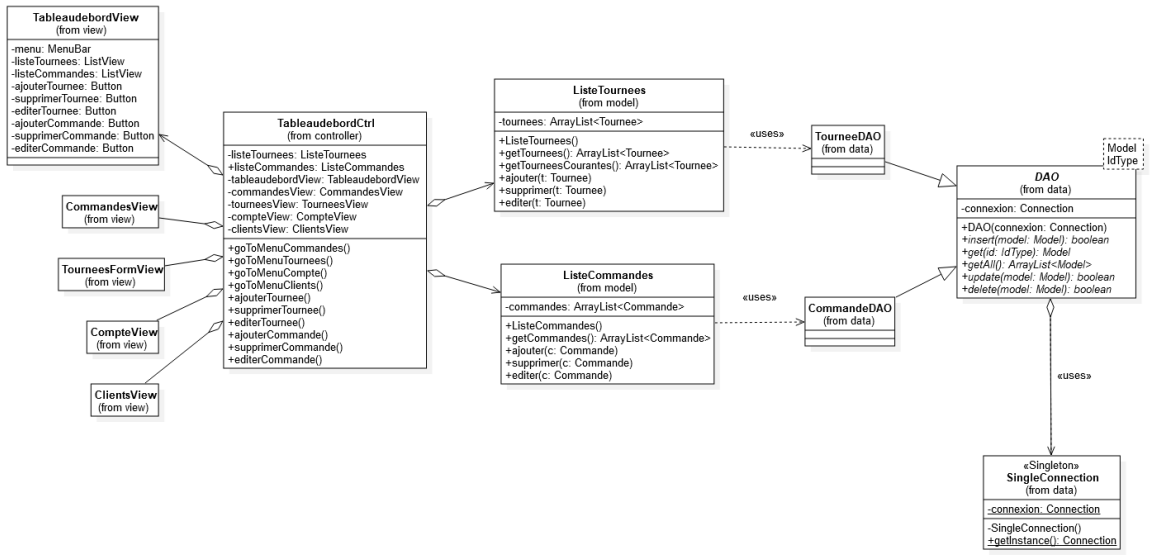

FIGURE 45 – Diagramme de classe du tableau de bord

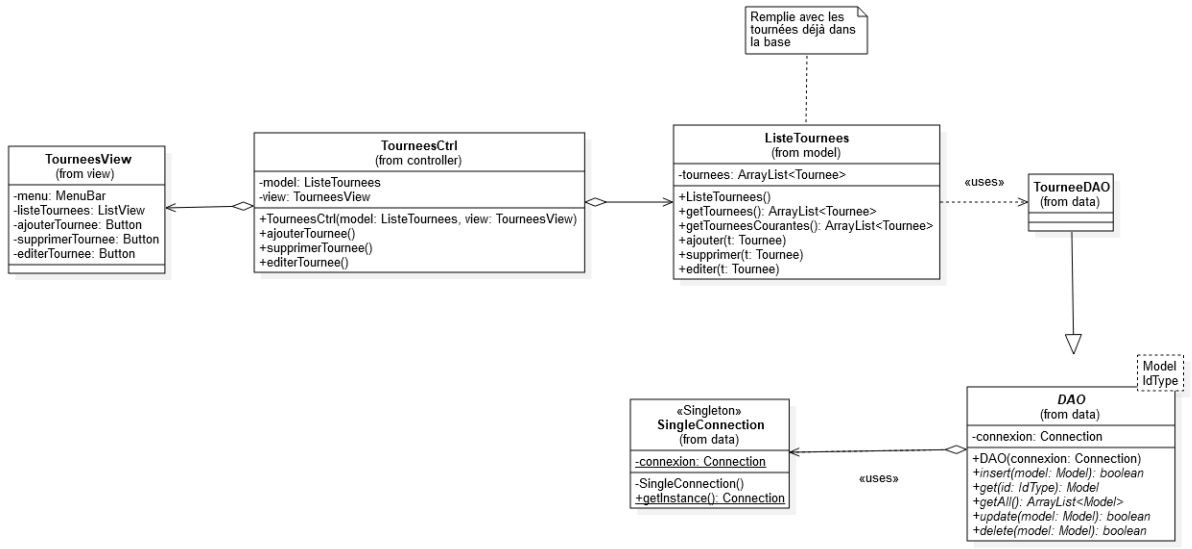

FIGURE 46 – Diagramme de classe des Tournees

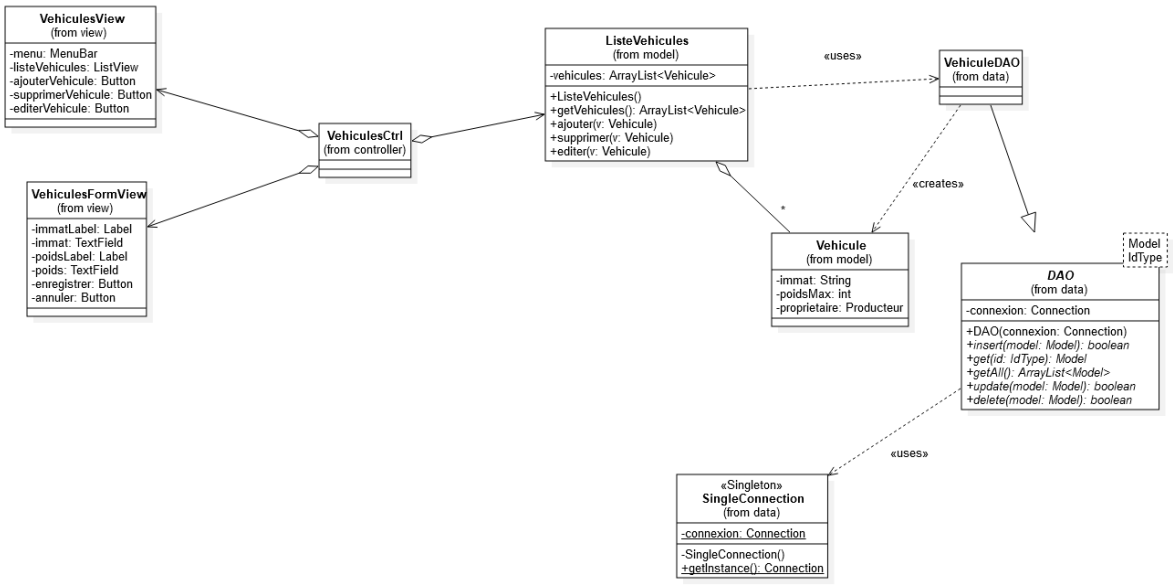

FIGURE 47 – Diagramme de classe des Véhicules

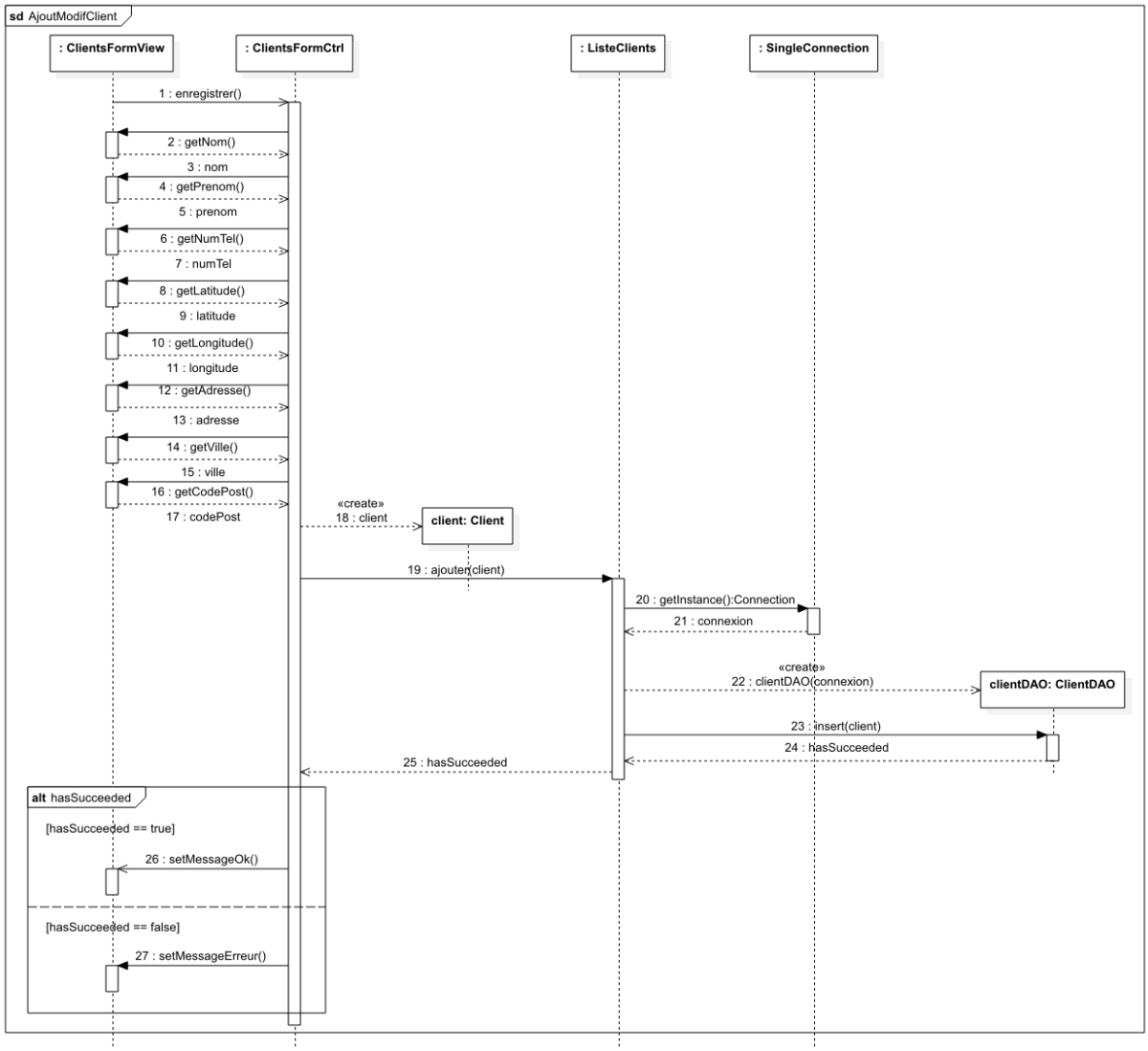

FIGURE 48 – Diagramme de séquence de gestion de clients

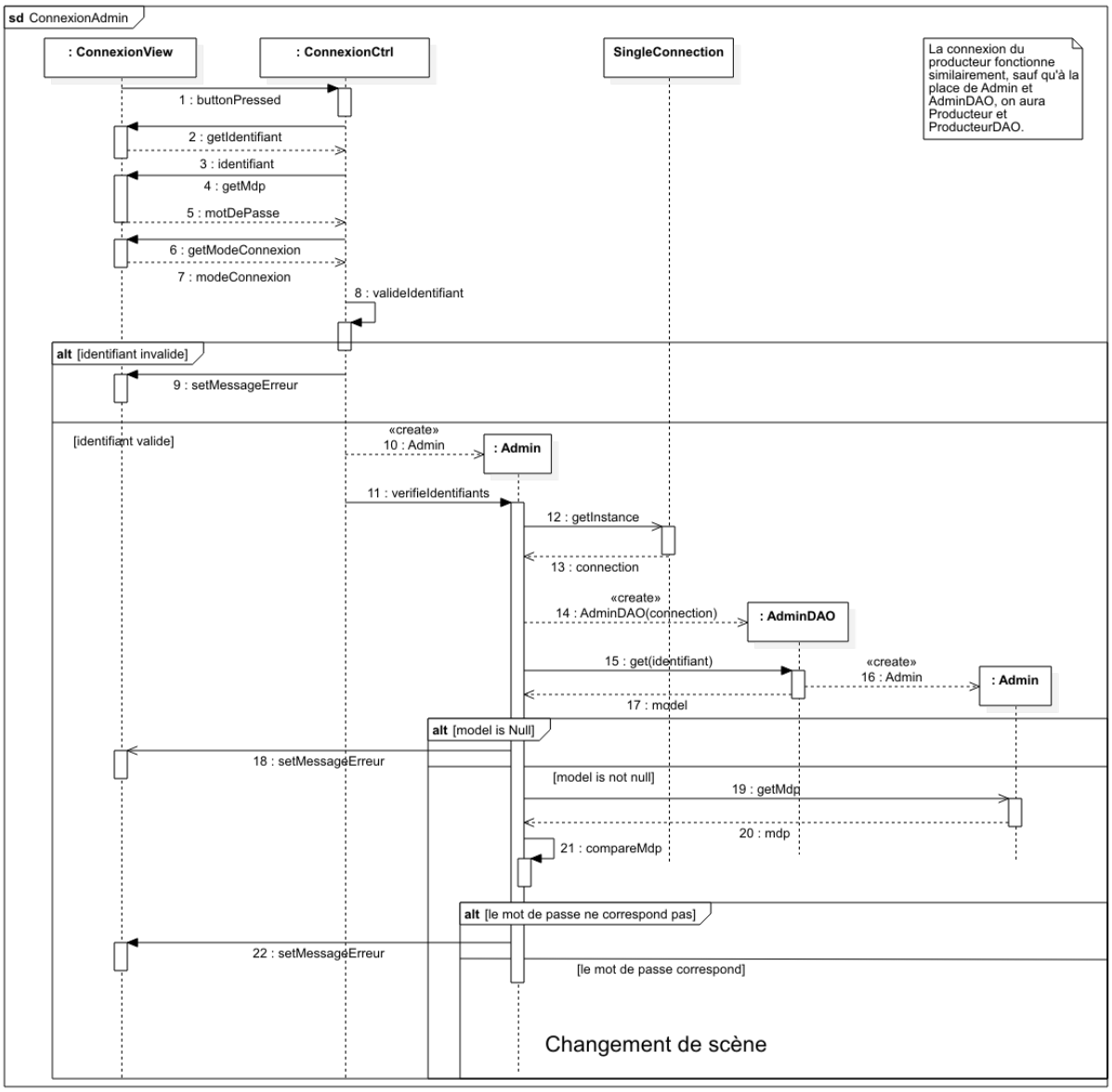

FIGURE 49 – Diagramme de séquence de la connexion d'un administrateur

Diagrammes de classe générés par IntelliJ

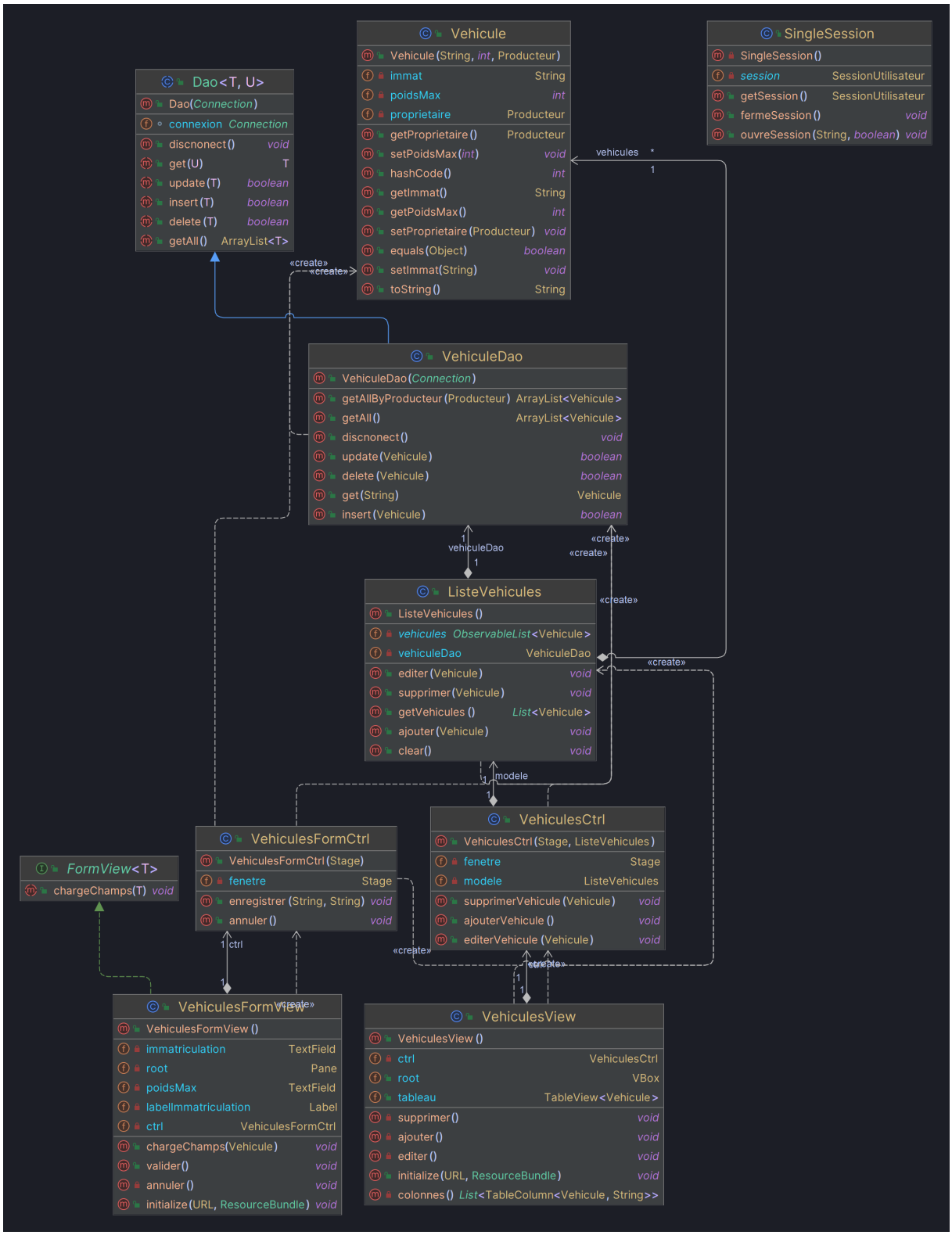

FIGURE 50 – Diagrammes de classes liés au véhicules

Maquettes

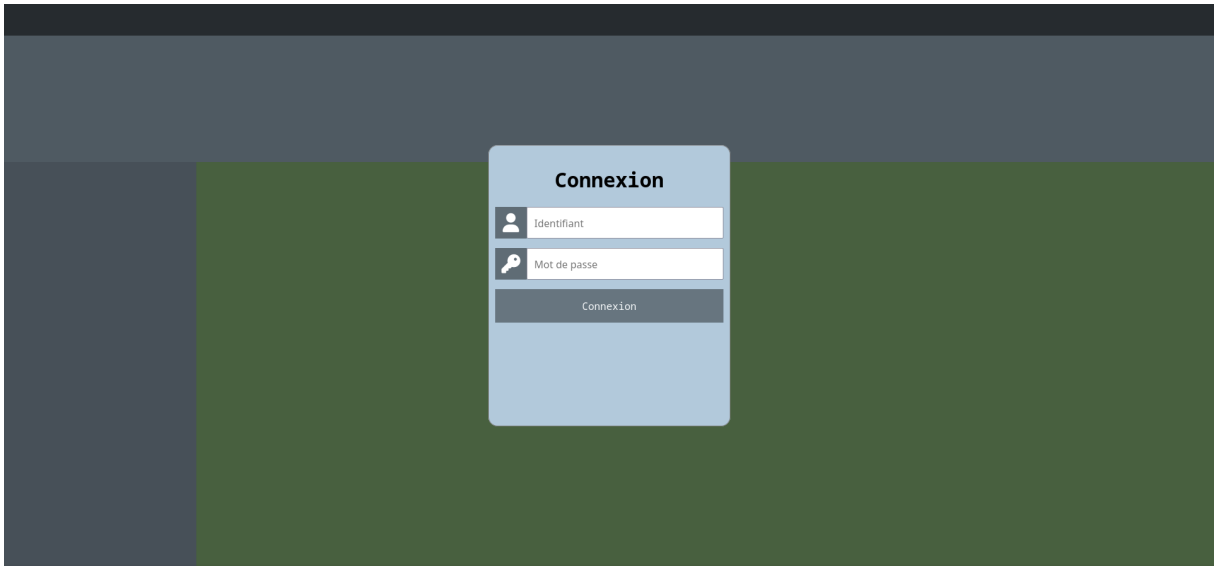

FIGURE 51 – Page de connexion de la première maquette

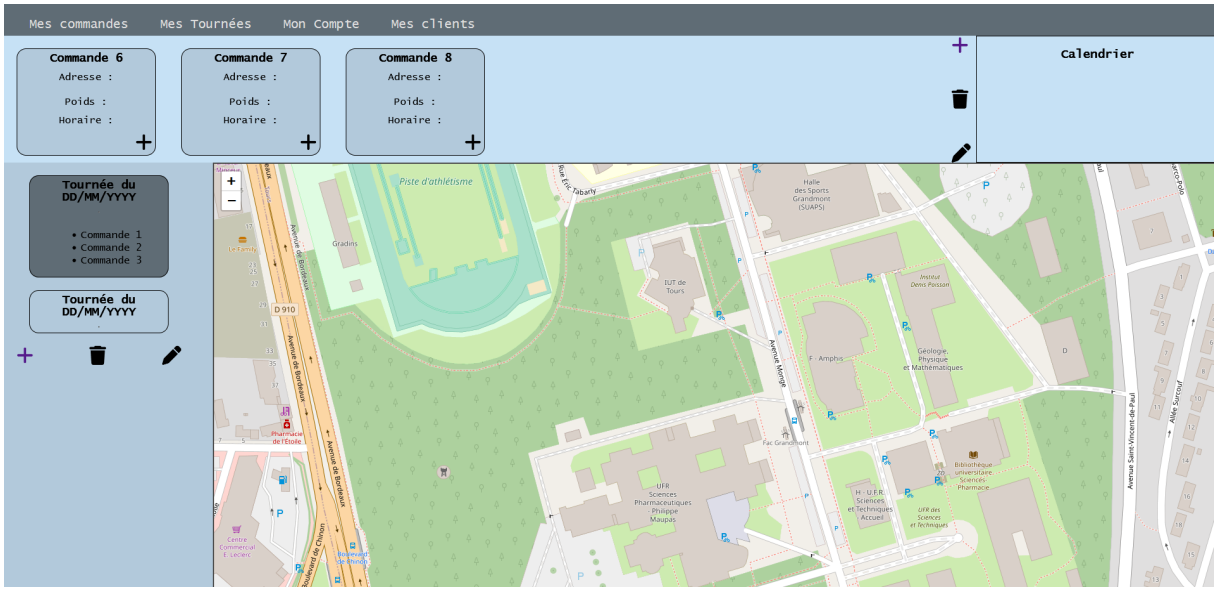

FIGURE 52 – Page principale de la première maquette

FIGURE 53 – Formulaire d'ajout de commande de la première maquette

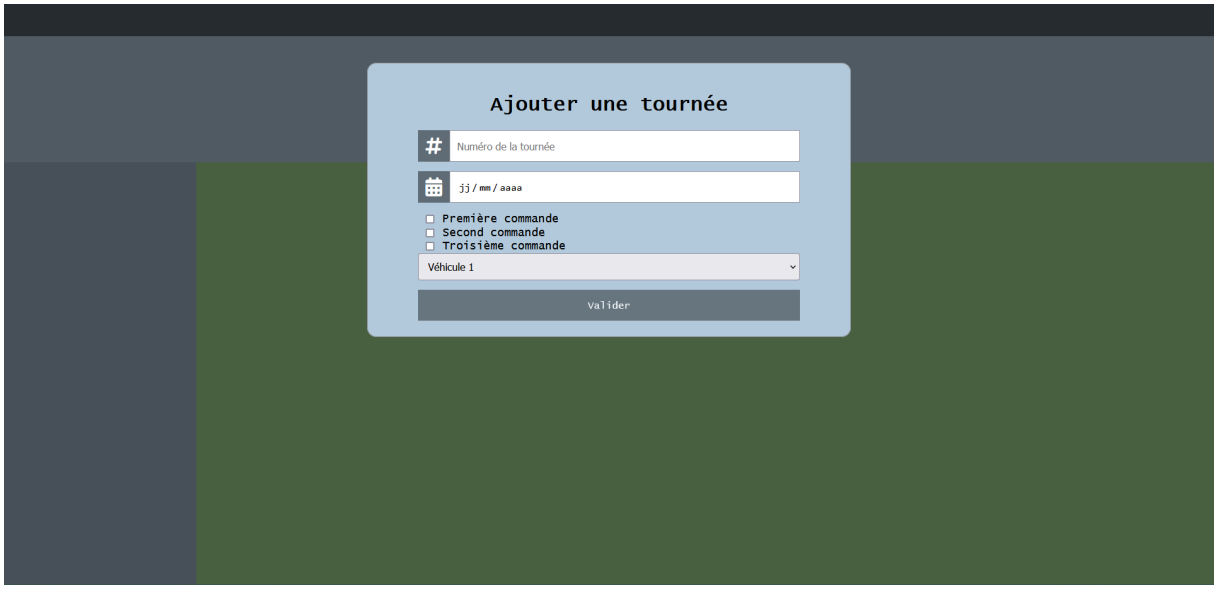

FIGURE 54 – Formulaire d'ajout de tournée de la première maquette