BERTHEL Gabriel - LAGASSE Adrian - MSIAH Romain - REMY Léo

Conception et visualisation de tournées agricoles en circuits courts.

L2 Informatique

Groupe 2B

Cadrage du projet

Présentation

Après constitution du groupe, le choix du projet fut une étape assez brève. En effet, nous avons tous choisi la création d'un logiciel de gestion et validation de tournée agricole en circuit court (Sujet 1).

Ce choix fut motivé par les perspectives de visualisation des trajets sur OpenStreetMap. C'est une idée particulièrement attrayante que nous aimerions grandement mettre en œuvre.

Objectif

Il s'agira donc de concevoir un logiciel permettant à des Producteurs de planifier et vérifier la faisabilité de tournées agricoles. Une fois les véhicules et commandes insérés, cela représente un gain de temps majeur. En effet, si invalidée, il suffirait d'en recréer une en utilisant les informations déjà en base.

Deux acteurs ont été identifiés : les Producteurs qui utilisent le logiciel au quotidien, et les Administrateurs qui sont les seuls à pouvoir entrer Administrateurs, Producteurs et clients en base de données.

Il faudra de ce fait créer deux interfaces pour ces derniers.

Normes et conventions

Comme outils, nous utiliserons Java, MySQL et JavaFX ainsi que JDBC. Nous choisirons de coder en camelCase et les commentaires seront sous format JavaDoc. Nous utiliserons aussi la bibliothèque JUnit pour effectuer les tests unitaires. Git sera notre système de gestion des versions, et la base de code sera disponible en permanence sur GitHub.

Première réunion - Sprint 0.

Ainsi, nous avons rapidement convenu d'une première réunion prévue pour reprendre les demandes du client et clarifier les quelques zones d'ombre qui planent sur le projet.
Pendant cette première réunion, nous avons réalisé les pitchs (Annexe 1) et la carte d'impact (Annexe 2).

Reprise de contact avec le client

Il a fallu recontacter notre client dans les plus brefs délais. Ce à quoi il fut convenu que :

- L'onglet de connexion Administrateur peut être le même que celui des Producteurs.

- Il y aura un Administrateur de base qui se chargera d'ajouter les premiers Administrateurs

- Les Administrateurs peuvent ajouter d'autres Administrateurs.

- L'Administrateur fixe le mot de passe initial des profils qu'il ajoute.

- Les Administrateurs et Producteurs peuvent changer de mot de passe.

- Si un créneau n'est pas libre, l'ajout d'un client à la tournée renverra un message d'erreur

Première version du MCD

Cette réponse nous a permis de finaliser le MCD (Annexe 3) qui avait été esquissé dès le premier rendez-vous, rendez-vous au cours duquel nous avons décidé que le mot de passe sera directement stocké dans la table "Producteur".

Plus tard, par souci de simplification, il sera aussi décidé que chaque Producteur est identifié par un identifiant numérique unique, et qu'il possède dans sa table à la fois coordonnées GPS et adresse, pour faciliter les calculs avec OpenStreetMap et garantir la précision de l'affichage de son adresse.

Suite à cela, nous nous sommes concertés pour convenir d'un canal de communication, et d'un système de gestion des versions.

Définition de canaux de communication

Ainsi, nous avons créé un serveur discord pour discuter de façon organisée du projet. Plusieurs canaux thématiques ont été créés ce qui permet de s'y retrouver aisément.
Lien : https://discord.gg/9NXBPBFGTD (n'expire jamais)

Les différents salons seront d'autant plus utiles lorsque nous décomposerons nos scénarios en user-stories et que nous remplirons le backlog. Dès lors, il nous sera possible de travailler de manière tout à fait asynchrone. En effet, nous pourrons à tout moment partager nos interrogations sans pour autant mélanger les informations.

Choix d'un gestionnaire de versions

De plus, nous avons opté pour la paire Git & GitHub comme système de gestion des versions. Il serait ainsi envisageable de systématiser les tests unitaires à chaque pull request. S'il est vrai que tout dépend de la qualité des tests, cela permet néanmoins de garder un certain contrôle sur la base de code déjà en production.
Lien : https://github.com/phoboswolf/Gestionnaire-Distribution-Agricoles

Il sera d'ailleurs nécessaire qu'il y ait au moins un relecteur par pull-request, et que chaque demande de changement soit résolue. Cela devrait permettre de minimiser le risque d'erreurs à chaque mise à jour de l'application.

Il faudra aussi penser à écrire les tests tout au long du développement, et si possible, soumettre les fonctions critiques à des tests structurels.

Enfin, le choix du format Javadoc permettra la génération automatique de la documentation sur le dépôt en ligne.

Esquisse d'une carte d'impact

La réunion ayant été productive, nous avons pu établir une carte d'impact (Annexe 2). Cela nous a permis d'avoir une vision globale de l'application tout en préparant le terrain pour l'écriture des cas d'utilisation. Lors de cette première réunion, nous avons établi une carte d'impact qui nous a permis d'avoir une vision globale de l'application, et qui avait pour objectif de préparer l'écriture des cas d'utilisation.

Deuxième Réunion : Écriture de scénario (Annexe 4)

Lors de la deuxième réunion, nous avons fait face à notre première difficulté. En effet, la notion de scénario d'utilisation était assez ambiguë. Il a donc fallu essayer de lever au mieux ce doute.

Après une longue session de recherche, nous avons fait le choix de rester niveau système et avons décidé que l'analyse se fera dans les diagrammes de séquence.

Suite à cela, nous avons écrit un scénario nominal ainsi que son scénario alternatif. Le contexte est le logiciel, l'acteur primaire est le Producteur et l'acteur secondaire est l'Administrateur.

Dans un premier temps, nous avons supposé que le Producteur possède déjà un véhicule, insère les commandes dans les bons créneaux, et enfin crée sa tournée. C'est le scénario nominal.

Ensuite, nous avons anticipé les différents cas de figures provoquant des erreurs, c'est le scénario alternatif.

Cette réunion fut assez longue, cependant nous n'avons pu faire qu'un seul scénario, car il a fallu enquêter sur le sujet.

Si nous avions un modèle et quelques exemples faits en cours, nous avons été confus par la masse d'information contradictoire présente en ligne.

Il a fallu retrouver les travaux dirigés pour les analyser en profondeur.

C'est sans doute la réunion la plus difficile que nous ayons dû faire. En effet, si cette étape était primordiale, nous n'avons pas pu avancer autant que nous le désirions, d'autant plus qu'il restait encore à faire les diagrammes de séquence, le diagramme de classe et la maquette.

Troisième Réunion : Use-Cases

Diagramme de cas d'utilisation

Lors de la troisième réunion et après concertation, nous avons choisi de nous concentrer sur les diagrammes de cas d'utilisation (Annexe 5).

En effet, il a été jugé qu'il est préférable de les établir avant de faire des diagrammes de séquences.

Ainsi, trois diagrammes ont été établis. Un diagramme expliquant le fonctionnement global de l'application, un `diagramme résumant la` gestion des comptes et un diagramme expliquant la gestion des tournées.

Nous sommes restés niveau système dans l'idée que ces derniers seront complétés par des diagrammes de séquences eux plus spécifiques.

Lors de cette réunion, il a aussi été établi quelques diagrammes de séquences, néanmoins ces derniers seront refaits lors de la prochaine.

Planification des réunions à venir

Globalement, c'est une réunion qui s'est déroulée sans accroc. Il y avait initialement de l'appréhension au vu des difficultés rencontrée lors de la précédente, mais nous avons su rebondir et avons pu avancer comme nous le souhaitions. En effet, le projet semblait enfin prendre forme !

De ce fait, avant de conclure, nous avons débattu sur la meilleure façon d'aborder la maquette. Deux approches :

- Convergente : Nous créons chacun une maquette puis mettons en commun nos schémas.
- Divergente : L'un des membres réalise une maquette puis le groupe se concerte sur les changements éventuels.

Le retour d'expérience vis à vis de nos projets précédents et les contraintes de temps nous ont poussé à préférer l'approche divergente. Le projet se précisant, cela nous a permis de segmenter et répartir les tâches à venir.

Quatrième Réunion : Diagrammes de séquences et maquette

Lors de la quatrième réunion, ont été commencés parallèlement diagrammes de séquences, maquette et diagramme de classes. L'objectif étant de finaliser la maquette avant la cinquième pour ensuite compléter le diagramme de classe.

Diagrammes de séquences (Annexe 6)

Les cas d'utilisation étant niveau système, les diagrammes de séquences permettent de mieux détailler certaines fonctionnalités.

Ainsi, en plus d'un diagramme résumant le scénario nominal, nous nous sommes mis d'accord sur la création de diagrammes représentant :

- L’ajout d’un Véhicule par le Producteur

- L’ajout d’un Producteur en base par l’Administrateur

- La connexion d’un Producteur

- La création et validation d’une Tournée par le Producteur

- La visualisation d’une Tournée valide

Nous avons décidé de décrire aussi l’interaction avec les contrôleurs de scène, même si nous n’avions pas encore appris à utiliser JavaFx. Cela permet entre autres d’avoir une vision approximative de la structure du projet ainsi qu’une idée des contrôleurs de scènes à implémenter. En effet, il en faudrait un par page.

Cela nous a aussi permis de voir comment ces contrôleurs interagissent avec le modèle et quelles requêtes pourraient être nécessaires pour remplir les vues.

Ces différents diagrammes nous ont aussi permis de soulever des contraintes importantes. En effet, lors d’un ajout de Véhicule, d’une Inscription ou d’une Connexion, il est important que les formats soient respectés.

Diagramme de classes (Annexe 7)

Comme nous programmons en objet, nous avons trouvé judicieux de représenter chaque table comme un objet (POJO). Chaque agrégation sera représentée par une liste d’objets. Il faudra donc implanter une méthode add et remove pour ces dernières.

Pour charger ces objets, nous passerons par un objet-service propre à chaque entité (DAO), et avec les méthodes SELECT, UPDATE, CREATE, DELETE, FIND et FINDALL et potentiellement des méthodes plus particulières.

On pourrait penser faire une classe abstraite générique, et surcharger les méthodes au besoin.

Ces derniers interrogeront la base de données puis instancieront les entités, ce qui permet de toujours manipuler des objets et donc de

profiter du principe d'encapsulation. Ou pourrait imaginer des méthodes de formatage sur les entités pour transformer les données dans un format particulier.

A chaque vue sera associé un contrôleur avec plusieurs actions, eux se chargeront de gérer les flux de données vers et depuis le modèle. Il n'y a pas de logique autre que d'assurer la forme et cohérence des données. C'est beaucoup de classes mais cela permet de traiter chaque action indépendamment.

Si jamais des actions requièrent des traitements spécifiques, on les regroupe. Ainsi, l'on pourrait imaginer une classe validateur de format (CIRET, email, code postal).

De plus, nous choisirons d'implémenter la connexion à la base de données comme un Singleton que nous passerons au constructeur de chaque DAO. Il suffira de changer de connecteur pour changer de base.

Enfin, il faudra penser à créer une classe permettant de manipuler l'API OpenStreetMap, et un validateur de Tournée.

Cinquième et sixième Réunion

Au cours de la cinquième et sixième réunion, le groupe a continué de travailler sur la maquette.
Sa réalisation s'est grandement appuyée sur les diagrammes de cas d'utilisation qui ont permis de déterminer pour chaque acteur les actions qu'il pouvait réaliser, et par la même occasion les vues et méthodes auxquelles il pourrait avoir accès.
La maquette a également permis de mettre à jour la base de données, notamment via l'ajout de libellés pour les Commandes et Tournées, qui permettent une meilleure lisibilité pour le Producteur qu'un identifiant qu'il ne comprend pas ou une date.
Réaliser la maquette nous a également permis d'établir un certain nombre de contraintes auxquelles nous devrons répondre lors de l'écriture des méthodes.
Ces contraintes sont :
- On ne peut pas modifier ou supprimer un Client concerné par une Tournée.
- La modification d'une Commande ne doit pas entrer en contradiction avec une Tournée.
- La modification de l'adresse d'un Producteur ne doit être possible que s'il

n’a pas de Tournée en cours.
- Lors de l’ouverture d’une pop-up, celle-ci doit être fermée avant de pouvoir interagir de nouveau avec l’interface l’ayant appelée.
- Lorsque l’on clique sur l’un des boutons “Modifier” ou “Supprimer”, un élément doit être préalablement sélectionné.
- Lors de la fermeture d’une pop-up, on recharge les informations de la base, la valeur du champ recherche et ses résultats, et l’élément sélectionné par l’utilisateur s’il y en avait un et qu’il n’a pas été supprimé
- Lors de la recherche, on n’utilise pas de bouton “Valider”. La recherche se fait à mesure de la saisie grâce à un KeyListener (ou équivalent JavaFX)
- Il est impossible de supprimer une Commande concernée par une Tournée (passée ou non)
- Les Commandes et Tournées passées sont grisées et affichées en bas de la liste pour des raisons ergonomiques.
- Il doit y avoir au minimum une Commande dans une Tournée
- L’heure de départ de la tournée est calculée automatiquement en fonction de l’heure de la première Commande concernée, et l’heure de fin est estimée avec OpenStreetMap
- Lors de la suppression d’un Véhicule, celui-ci ne doit être concerné par aucune Tournée en cours
- On a aussi rappelé à l’occasion que les mots de passe devront être hashés
- Une question a émergé concernant les battements entre chaque Commande lors d’une Tournée, battement représentant le temps de déchargement. Faut-il décider d’une durée fixe ou l’adapter au poids de la Commande ? C’est une question que l’on posera lors du compte rendu de ce Jalon à notre client.

Après une revue, la maquette a été mise au propre (Annexe 8).

En parallèle, le diagramme de classes s’est précisé. Nous avons aussi vérifié que l’on pourra utiliser StarUML pour générer une partie du code, ce qui représenterait un gain de temps considérable.

Rien de particulier à dire sur ces deux réunions puisque nous avons tous avancé sur nos différentes tâches.

Le futur du projet

Le projet étant désormais structuré, la prochaine tâche devra être le remplissage du backlog, backlog qui d'ailleurs sera hébergé directement sur GitHub. Cela évite de multiplier les environnements.

Il faudra aussi penser à se former sur l'API OpenStreetMap que nous n'avons jamais utilisé auparavant. En effet, nous sommes conscients que la difficulté réside principalement dans l'utilisation de cette dernière.

Si le cœur de l'application, qui consiste essentiellement en des formulaires, est un gestionnaire de Tournées, l'aspect visualisation et affichage pourrait demander un certain temps d'apprentissage et de réflexion.

Ainsi, l'adoption d'une architecture MVC semble tout à fait pertinente. L'on pourra dans un premier temps développer le cœur de l'application, puis y greffer le validateur de tournée, ainsi que le visualiseur.
Le modèle MVC permettra aussi de montrer des versions semi-fonctionnelles du logiciel au client afin de recueillir son avis et préciser ses besoins.

Cette architecture permet de travailler sur les différentes composantes en parallèle. En théorie, et grâce à la classe validatrice de données, chaque action ne devrait pas faire plus que quelques lignes. On pourra donc greffer les vues au modèle assez aisément.

De ce fait, nous devrions avoir une marge de manœuvre nécessaire pour développer les fonctionnalités citées plus tôt. Et dans le pire des cas, le logiciel sera tout de même fonctionnel.

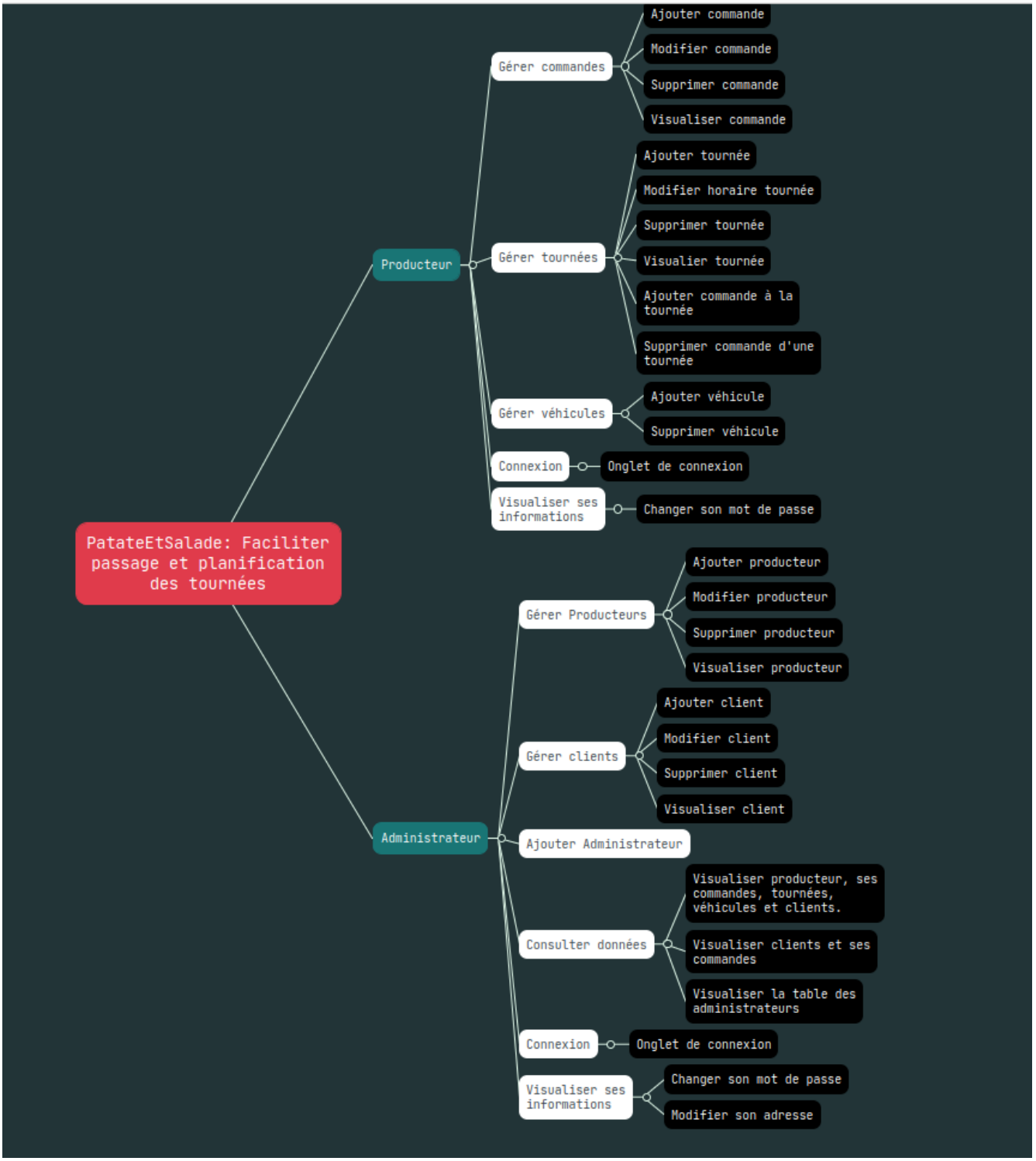
Enfin, il faudra penser à écrire les tests tout au long du développement. Il pourrait donc être intéressant d'associer des tests cases à chaque user story.

Annexe 1 : Pitchs

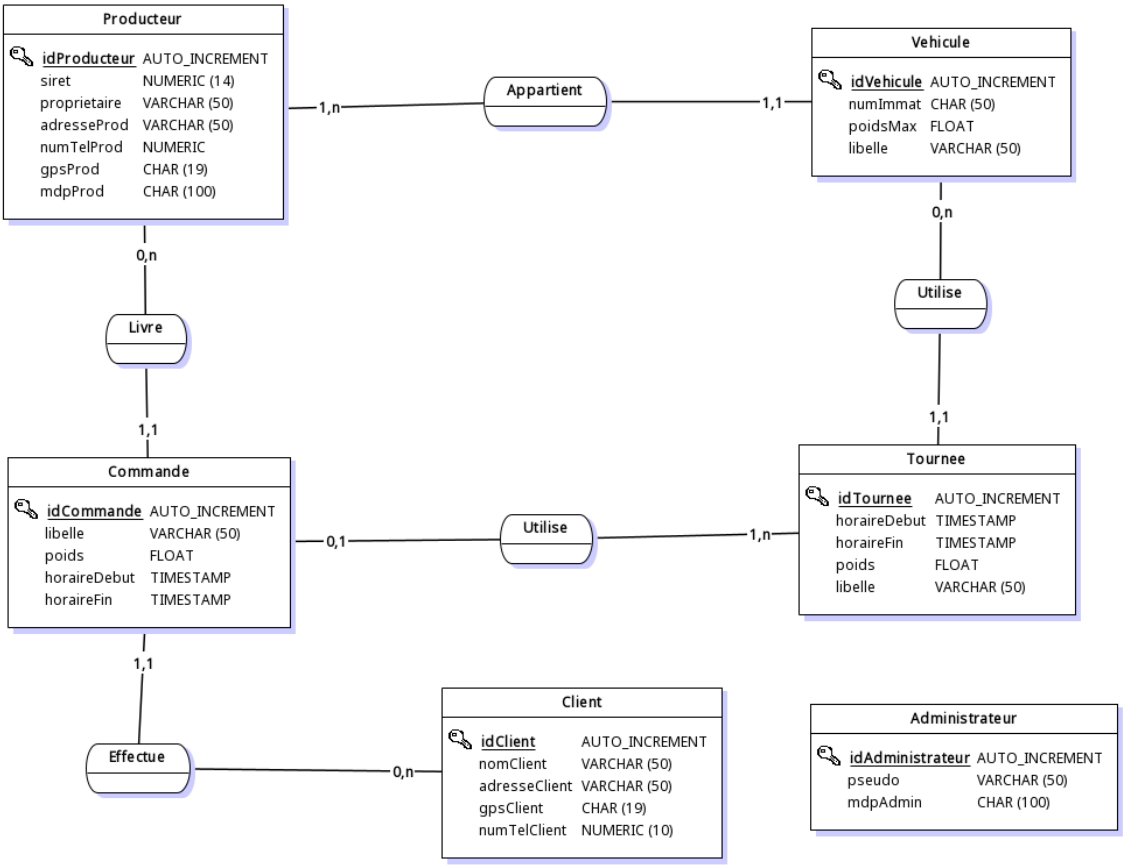
POUR les producteurs
QUI SOUHAITENT répertorier leurs clients et leurs véhicules ; ajouter, modifier et supprimer des commandes ; ajouter, modifier et supprimer des tournées
NOTRE PRODUIT EST un outil graphique
QUI permet la gestion de commandes et de tournées en fonction du poids et des horaires prédéfinis

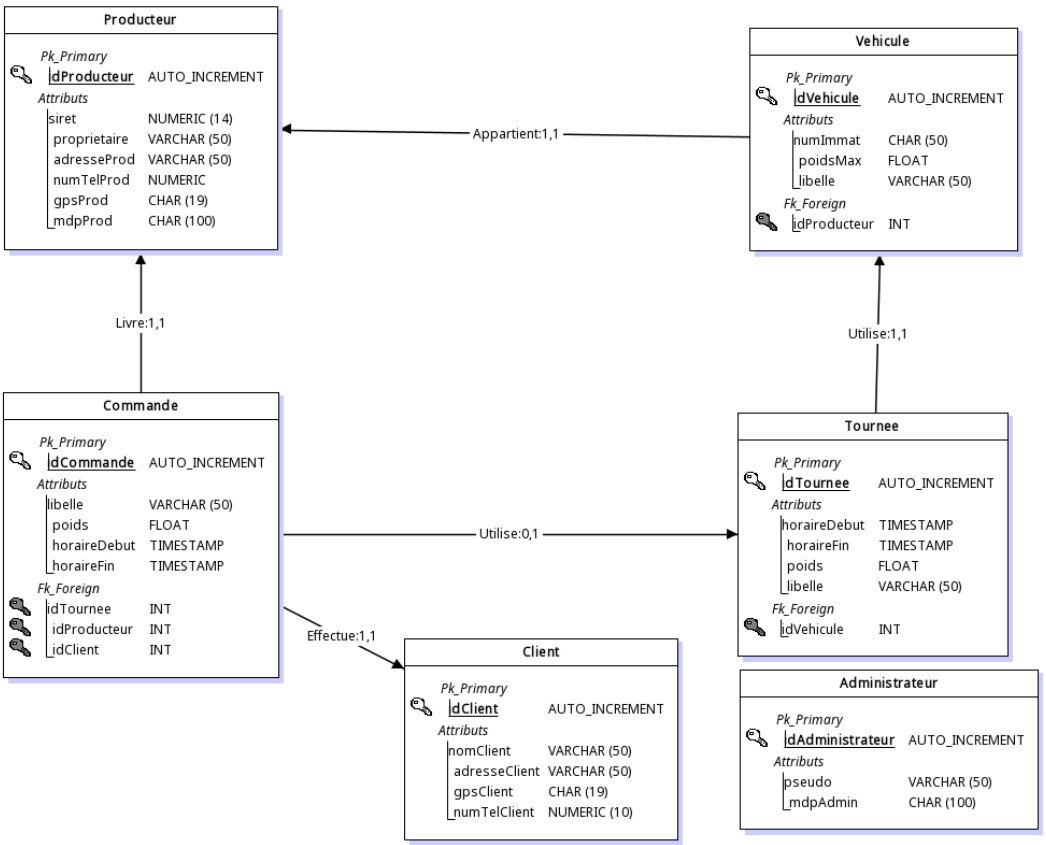
POUR les administrateurs
QUI SOUHAITENT ajouter, modifier, supprimer des producteurs et consulter l'ensemble des données
NOTRE OUTIL EST un outil graphique
QUI PERMET la gestion des producteurs et l'affichage de toutes les données du logiciel

Annexe 2 : Impact Map



Annexe 3 : MCD & MLD





BERTHEL Gabriel - LAGASSE Adrian - MSIAH Romain - REMY Léo

Annexe 4 : Scénarios d'utilisation

Acteur 1er : Producteur
Acteur 2nd : Administrateur
Contexte : Logiciel
Niveau : Système
Stakeholders & Interests :

- Administrateur : assurer le bon fonctionnement du système.
- Producteur : pouvoir utiliser le logiciel de gestion de tournée

Préconditions : Un admin doit avoir un compte et être connecté
Résultat minimum : l'administrateur enregistre un utilisateur
Résultat maximum : le producteur planifie une tournée
Déclencheur : demande informelle de l'ajout d'un producteur

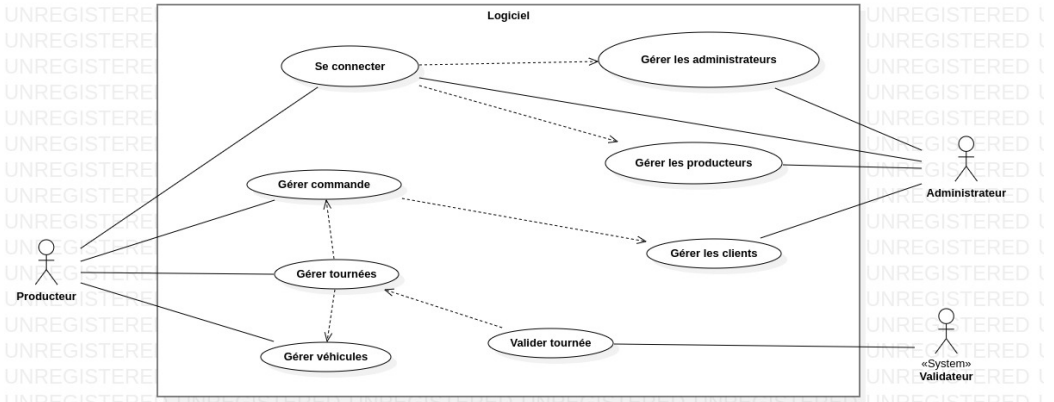
Scénario nominal :

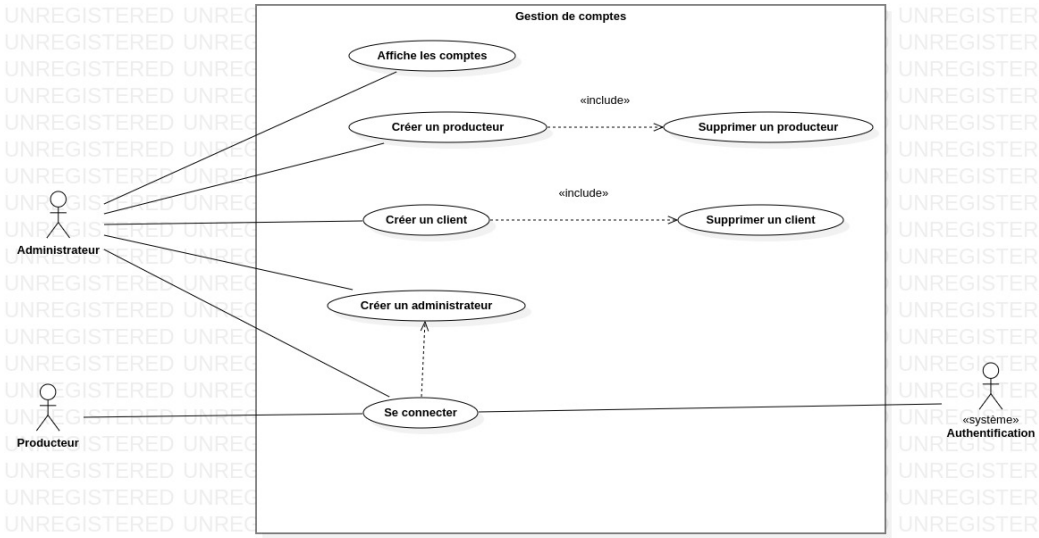
1. Administrateur rentre les informations du producteur
2. Création nouveau producteur dans la BDD
3. Producteur entre ses informations de connexion
4. Connexion producteur
5. Producteur entre les informations des commandes
6. Enregistrement des commandes
7. Producteur prépare une tournée
8. Enregistrement de la tournée

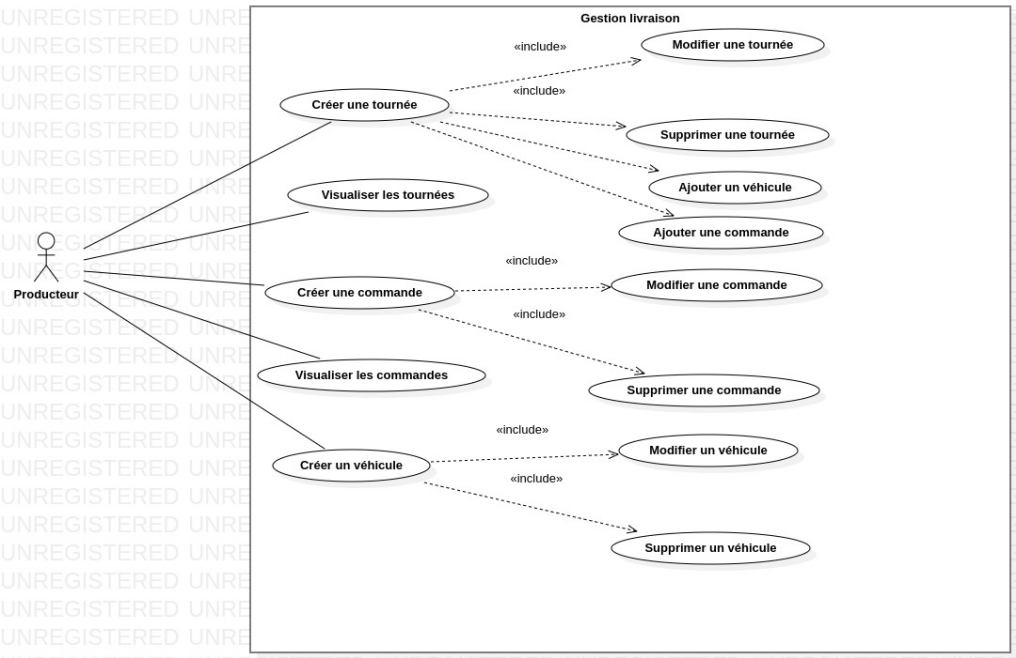
Scénario alternatif :

1a. des informations sont manquantes ou n'ont pas le bon format
    1a1. Logiciel signale l'erreur
    1a2. L'administrateur rentre les bonnes informations
2a. L'utilisateur existe déjà (SIRET ou pseudo administrateur)
    2a1. Le logiciel signale l'erreur
4a. Les informations sont erronées
    4a1. Le logiciel signale l'erreur
    4a2. Le user rentre les bonnes données
6a Mauvais format / informations manquantes
    6a1. Le logiciel signale l'erreur
    6a2. Le producteur rentre les bonnes dates
7a. Le créneau n'est pas bon
    7a1. Erreur
    7a2. Change l'horaire de la tournées
7b. Le véhicule n'est pas disponible
    7b1. Erreur
    7b2. Le producteur sélectionne un autre véhicule
    7b2a. Aucun véhicule libre
    7b2b. Erreur

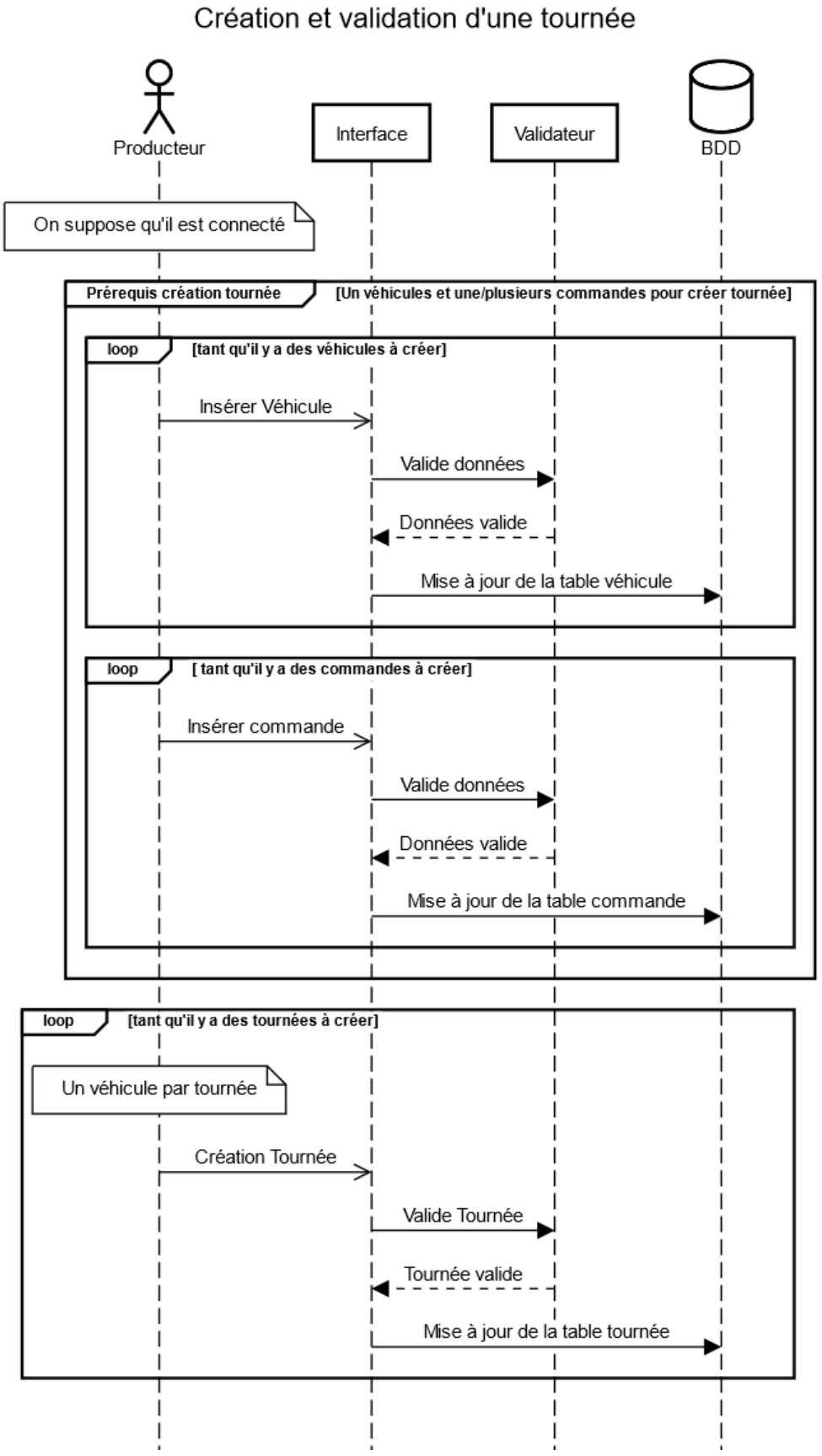
Annexe 5 : Diagrammes de cas d'utilisation

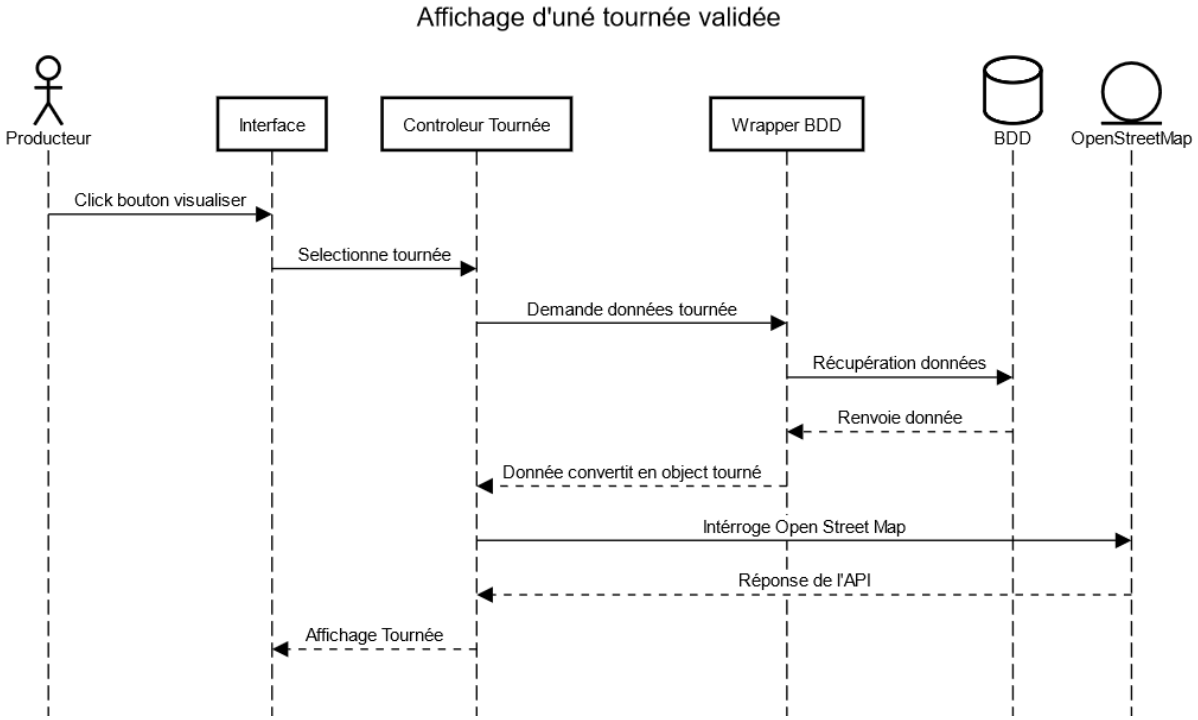


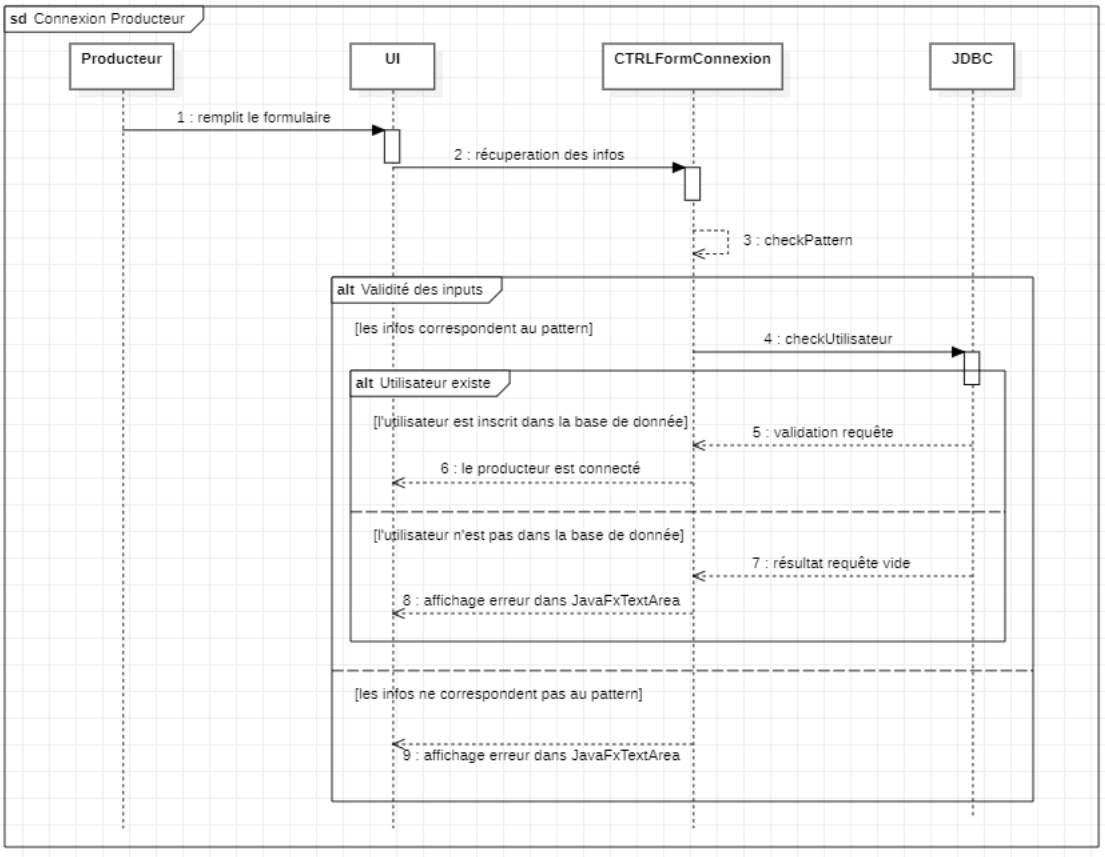


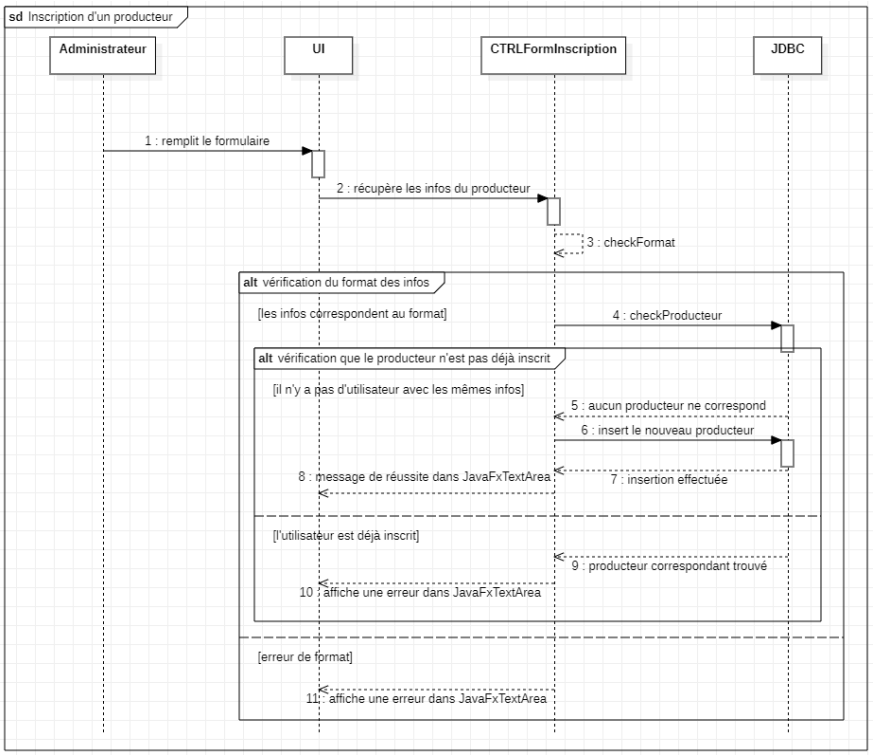


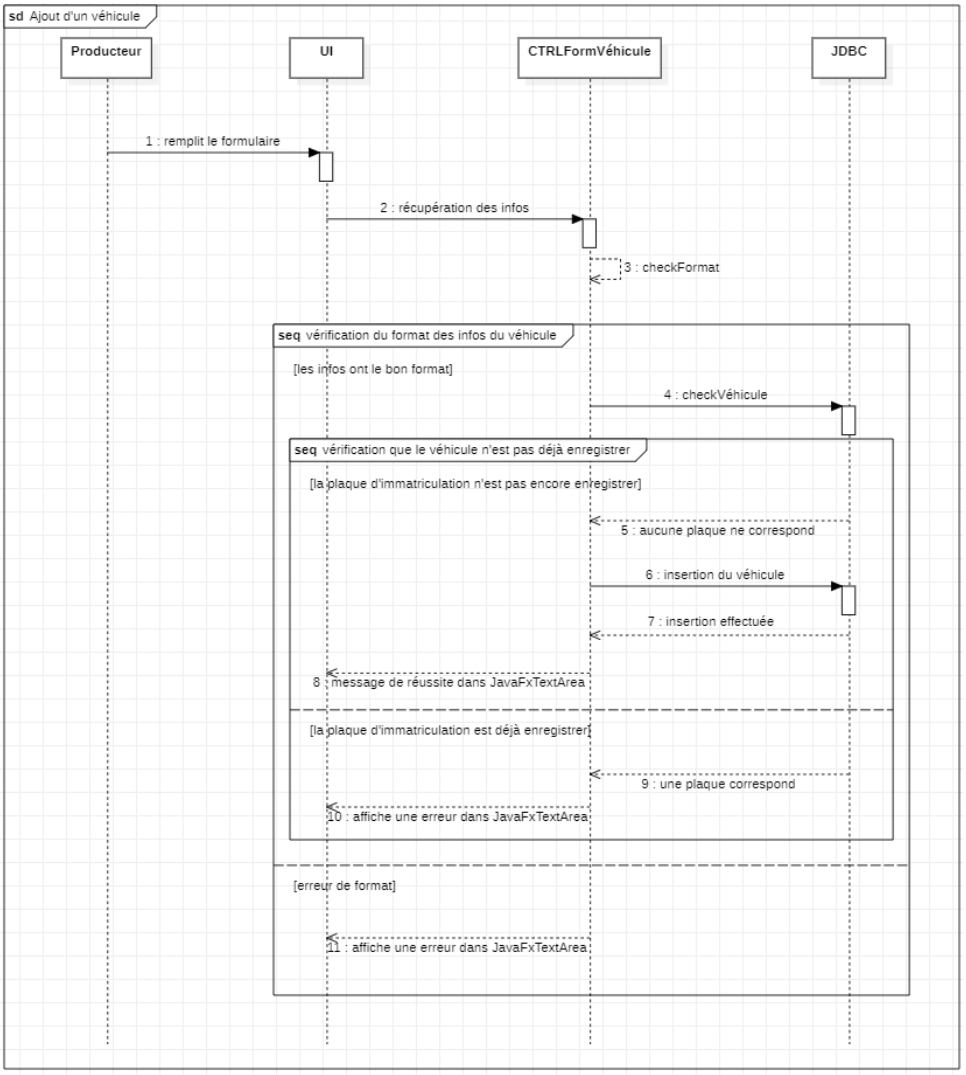
Annexe 6 : Diagrammes de séquence



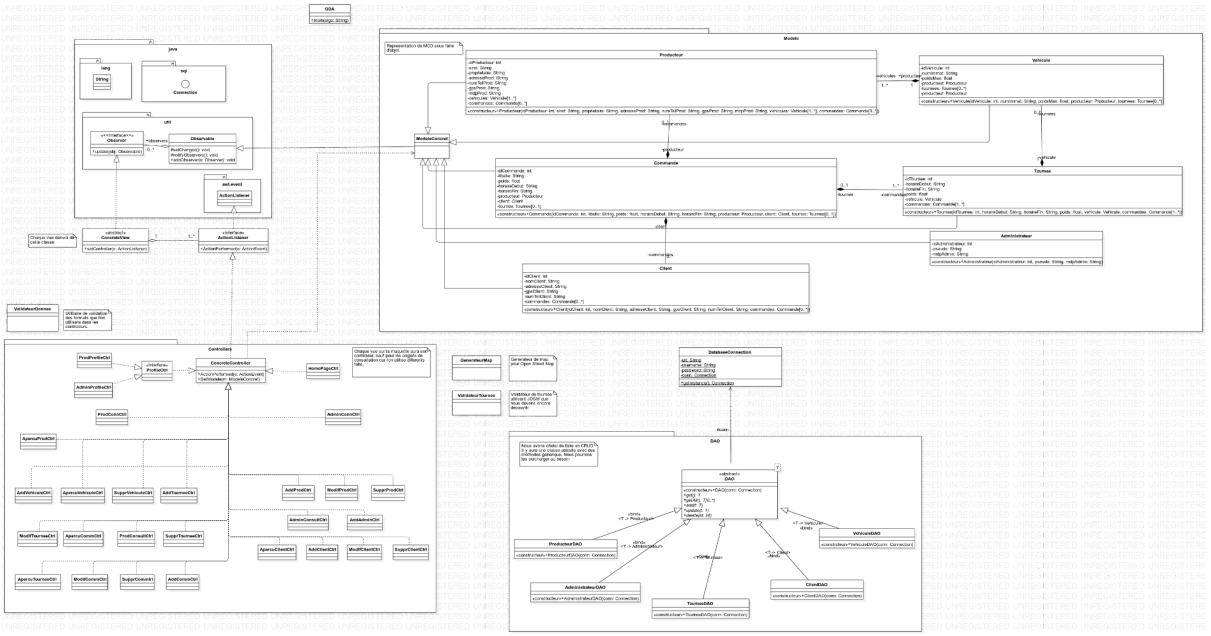

Affichage d'uné tournée validée
Producteur
Interface
Controleur Tournée
Wrapper BDD
BDD
OpenStreetMap
Click bouton visualiser
Selectionne tournée
Demande données tournée
Récupération données
Renvoie donnée
Donnée convertit en object tourné
Intérroge Open Street Map
Réponse de l'API
Affichage Tournée


sd Connexion Producteur
Producteur
UI
CTRLFormConnexion
JDBC
1 : remplit le formulaire
2 : récuperation des infos
3 : checkPattern
alt Validité des inputs
[les infos correspondent au pattern]
4 : checkUtilisateur
alt Utilisateur existe
[l'utilisateur est inscrit dans la base de donnée]
5 : validation requête
6 : le producteur est connecté
[l'utilisateur n'est pas dans la base de donnée]
7 : résultat requête vide
8 : affichage erreur dans JavaFxTextArea
[les infos ne correspondent pas au pattern]
9 : affichage erreur dans JavaFxTextArea


sd Inscription d'un producteur
Administrateur
UI
CTRLFormInscription
JDBC
1 : remplit le formulaire
2 : récupère les infos du producteur
3 : checkFormat
alt vérification du format des infos
[les infos correspondent au format]
4 : checkProducteur
alt vérification que le producteur n'est pas déjà inscrit
[il n'y a pas d'utilisateur avec les mêmes infos]
5 : aucun producteur ne correspond
6 : insert le nouveau producteur
7 : insertion effectuée
8 : message de réussite dans JavaFxTextArea
[l'utilisateur est déjà inscrit]
9 : producteur correspondant trouvé
10 : affiche une erreur dans JavaFxTextArea
[erreur de format]
11 : affiche une erreur dans JavaFxTextArea


sd Ajout d'un véhicule
Producteur
UI
CTRLFormVéhicule
JDBC
1 : remplit le formulaire
2 : récupération des infos
3 : checkFormat
seq vérification du format des infos du véhicule
[les infos ont le bon format]
4 : checkVéhicule
seq vérification que le véhicule n'est pas déjà enregistrer
[la plaque d'immatriculation n'est pas encore enregistrer]
5 : aucune plaque ne correspond
6 : insertion du véhicule
7 : insertion effectuée
8 : message de réussite dans JavaFxTextArea
[la plaque d'immatriculation est déjà enregistrer]
9 : une plaque correspond
10 : affiche une erreur dans JavaFxTextArea
[erreur de format]
11 : affiche une erreur dans JavaFxTextArea

Annexe 7 : Diagramme de classes



Annexe 8 : Maquette