BOSIO Alexis Groupe 4104

BETTE Liam PLD Agile

DE CLERCQ Romain 4IF

FLORANT Clément

LE RAL Mathieu

MEKHALFA Taki Eddine

YE Tianjian

**Bilan du PLD :**

Table des matières :

[Planning : 2](#_Toc530118087)

[Planning Prévisionnel de la première itération : 2](#_Toc530118088)

[Planning effectif de la première itération : 3](#_Toc530118089)

[Planning Prévisionnel de la seconde itération : 4](#_Toc530118090)

[Planning effectif de la première itération : 5](#_Toc530118091)

[Choix Architecturaux et Design Pattern : 6](#_Toc530118092)

[Diagramme de classes de la première itération : 8](#_Toc530118093)

[Diagramme de packages : 8](#_Toc530118094)

[Diagrammes de classes : 8](#_Toc530118095)

[Package Calcules : 8](#_Toc530118096)

[Package XMLHelpers : 9](#_Toc530118097)

[Package Metier : 9](#_Toc530118098)

[Diagramme de classes de l’itération finale : 11](#_Toc530118099)

[Diagramme de Packages : 11](#_Toc530118100)

[Package Modèle : 11](#_Toc530118101)

[Package XMLHelpers : 11](#_Toc530118102)

[Package Métier : 11](#_Toc530118103)

[Package Calcules : 11](#_Toc530118104)

[Package Vue : 11](#_Toc530118105)

[Package Contrôleur : 11](#_Toc530118106)

[Comparaison des diagrammes de classes : 12](#_Toc530118107)

[Couverture des tests : 13](#_Toc530118108)

[Bilan technique et humain : 14](#_Toc530118109)

[Glossaire : 15](#_Toc530118110)

# Planning :

## Planning Prévisionnel de la première itération :

Séance 1 :

-Etude du problème

-Organisation de l'équipe

-Définition de différents cas d'utilisation

-Première version du diagramme de cas d'utilisation

-Glossaire

-Diagramme de classes et du diagramme de packages.

-Description textuelle structurée de cas d’utilisation analysés et visés pour la première itération

Séance 2 :

-Fin du diagramme de classes

-Fin de la définition des cas d’utilisation

-Division de l’implémentation.

-Réalisation du début de la lecture des fichiers XML.

Séance 3 :

-Compte-rendu intermédiaire

-Fin de l’implémentation pour les trois binômes

Séance 4 :

-Bilan de la 1ère itération

-Rendu des livrables intermédiaires

## Planning effectif de la première itération :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tâche/Membre Equipe et Temps consacré (\*) | Alexis (ScrumMaster) | Romain  (PO) | Clément | Mathieu | Taki Eddine | Tianjian |
| Séance 1 : 10 Octobre |  |  |  |  |  |  |
| Etude du problème | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Organisation de l’équipe | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Mise en place des outils de travail | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Définition de différents cas d’utilisation | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Réalisation du planning prévisionnel | 1 |  |  |  |  |  |
| Première version du diagramme de cas d’utilisation |  |  |  |  |  | 1.5 |
| Glossaire | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Diagramme de classes et de packages |  |  |  | 0.25 | 1.5 |  |
| Description textuelle structurée de cas d’utilisation analysés et visés pour la première itération |  | 1.5 | 1.5 | 1.5 | 1.5 |  |
|  |  |  |  |  |  |  |
| Séance 2 : 19 Octobre |  |  |  |  |  |  |
| Finalisation du diagramme de classes et de packages |  |  |  |  | 1.5 |  |
| Finalisation des cas d’utilisation | 2 | 2 | 2 | 2 | 2 | 2 |
| Implémentation du Contrôleur |  | 0.5 | 0.5 |  |  |  |
| Diagramme états-transitions |  | 1.5 | 1.5 |  |  |  |
| Implémentation de la Vue | 2 |  |  |  |  | 2 |
| Implémentation du Modèle |  |  |  | 2 | 2 |  |
| Implémentation de la lecture de fichier XML |  |  |  |  | 0.5 |  |
|  |  |  |  |  |  |  |
| Séance 3 : 23 Octobre |  |  |  |  |  |  |
| Implémentation du Contrôleur |  | 4 | 4 |  |  |  |
| Implémentation de la Vue | 4 |  |  |  | 1.5 | 4 |
| Implémentation du Modèle |  |  |  | 2 | 2 |  |
| Début du compte-rendu intermédiaire |  |  |  | 2 | 0.5 |  |
|  |  |  |  |  |  |  |
| Bilan | 12 | 12 | 12 | 12.25 | 15.5 | 12 |

## Planning Prévisionnel de la seconde itération :

Séance 1 :

-Poursuite de l’implémentation : ajout, suppression et modification

Séance 2 :

-Poursuite de l’implémentation et mise en place des undo/redo

Séance 3 :

-Gestion des erreurs

-Fin de l’implémentation pour la vue

Séance 4 :

-Bilan de la 2ème itération

-Rendu du livrable final

-Présentation orale

## Planning effectif de la première itération :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tâche/Membre Equipe et Temps consacré | Alexis (ScrumMaster) | Romain  (PO) | Clément | Mathieu | Taki Eddine | Tianjian | Liam |
| Séance 1 : 06 Novembre |  |  |  |  |  |  |  |
| Implémentation Package Modèle |  |  |  | 4 | 4 |  |  |
| Implémentation Package Vue |  | 4 |  |  |  | 4 | 4 |
| Implémentation Package Contrôleur |  |  | 4 | 4 |  |  |  |
|  |  |  |  |  |  |  |  |
| Séance 2 : 09 Novembre |  |  |  |  |  |  |  |
| Implémentation Package Modèle |  |  |  |  |  |  |  |
| Implémentation Package Vue | 4 |  |  | 4 | 4 | 4 | 4 |
| Implémentation Package Contrôleur |  | 3.5 | 3.5 |  |  |  |  |
| Couverture des tests |  |  |  |  |  | 4 |  |
| Gestions des erreurs |  | 0.5 | 0.5 |  |  |  |  |
| Séance 2 : 13 Novembre |  |  |  |  |  |  |  |
| Implémentation Package Vue | 4 |  |  | 4 | 4 |  | 4 |
| Implémentation Package Contrôleur |  | 2 | 2 |  |  |  |  |
| Gestion des erreurs |  | 2 | 2 |  |  |  |  |
| Couverture des tests |  |  |  |  |  | 3 |  |
| Rapport sur les tests |  |  |  |  |  | 1 |  |
| Séance 4 : 16 Novembre |  |  |  |  |  |  |  |
| Bilan technique et humain |  |  |  |  |  |  |  |
| Organisation de la présentation |  |  |  |  |  |  |  |
| Finalisation du code |  |  |  |  |  |  |  |
| Gestion des erreurs |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Total |  |  |  |  |  |  |  |

# Choix Architecturaux et Design Pattern :

L’application développée avait pour contrainte de suivre l’architecture Modèle Vue Contrôleur (MVC). Nous avons donc opté pour la structure qui nous semblait la plus modulaire possible en séparant les fonctionnalités dans des packages et des classes différents pour garantir une bonne indépendance et une bonne réutilisabilité du code.

Concernant le modèle, nous avons choisi de séparer le traitement des données des données elles-mêmes. Nous avons ainsi un package Model.Metier qui comprend le stockage des données liées aux objets métier comme les livraisons ou le plan. Les traitements de ces données seront effectués par le package Model.Calculs. Malgré une certaine interdépendance créée entre ces deux packages, le code des classes des objets métier s’en trouve très allégé et bien plus lisible. De la même manière, nous avons le package Model.XMLHelpers qui se charge de toutes les opérations liées aux fichiers XML et construit les objets métiers grâce aux informations prises dans ces fichiers. La dernière partie du package Model représente l’interface avec le contrôleur. Le contrôleur possède une instance de la classe Planification qui lui sert à obtenir les informations concernant le modèle et à le modifier si nécessaire. La classe Planification encapsule donc via des méthodes toutes les modifications que le contrôleur a besoin de faire sur le modèle pour que le contrôleur n’ait pas à se soucier de comment faire ces modifications.

Pour simplifier le travail du contrôleur, nous avons choisi d’utiliser le design pattern Etat. Cela nous permet de savoir exactement où nous sommes à n’importe quel moment de l’exécution du programme, les boutons qu’il convient d’activer ainsi que les messages qu’il faut envoyer à l’utilisateur. Le code est ainsi facilement maintenable et très évolutif. Concernant les fonctionnalités Undo/Redo, nous utilisons le design pattern Commande et nous stockons les informations nécessaires dans une liste de commandes à chaque modification pour pouvoir restaurer l’état précédent une modification. Notre liste de commande contient donc des sauvegardes des états précédents et non uniquement les actions effectuées à chaque modification (ce qui aurait nécessité de réaliser l’opération inverse pour restaurer l’état précédent). L’utilisation du design pattern Etat facilite également l’implémentation de cette fonctionnalité.

Dans le cadre de la réalisation de la vue liée à notre modèle MVC, nous avons choisi de l’implémenter en plusieurs éléments, tous contenus dans l’unique package vue.

Pour partir de l’utilisateur, nous avons tout d’abord la classe « InterfaceGUI » qui se charge de la création de la fenêtre et du menu avec ses boutons. L’import de la classe du contrôleur permet ensuite de réagir aux actions de l’utilisateur sur ces boutons et de lancer les appels de méthodes correspondantes. Cette classe se charge aussi de gérer la désactivation des boutons grâce à des méthodes que le contrôleur se chargera ensuite d’utiliser. La saisie des différentes informations comme le nombre de livreurs et la durée d’une livraison est aussi mise en place au sein de cette classe qui se chargera de transmettre ces informations au contrôleur.

Ensuite, le reste de l’affichage est divisé entre deux classes héritant toutes deux de « Vue », qui leur permet de manipuler les « Observer » avec une méthode « update » et organise leur structure commune pour contenir une liste de tournées, un objet planification pour l’appel des éléments du modèle, un plan et une demande de livraisons. De ces deux classes, la première, « VueTextuelle » se charge de remplir et de permettre à l’utilisateur d’interagir avec le tableau sur le côté droit de l’écran alors que la seconde, « VueGraphique », comporte l’affichage concret du plan avec la demande de livraisons, les tournées et les modifications de celles-ci. Le dessin des tournées à proprement parlé se fait via les structures de données que nous avons choisies en utilisant des équivalences pour chaque élément.

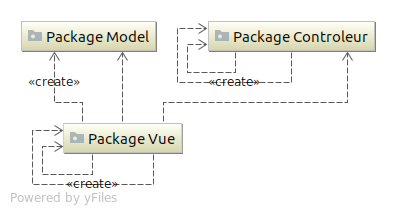
Les rues seront manipulées par une classe « LineModifiee » qui se chargera d’affecter à une « Line » classique une couleur et un nom. Cette classe permettra également à l’utilisateur de sélectionner une rue via une mécanisme d’évènements ainsi que de changer ou d’obtenir le nom de la rue voulue.

Pour la représentation des livraisons sur l’affichage, nous avons utilisé la classe « CercleLivraison » qui nous permet, avec des attributs tels que l’heure de passage, la livraison associée et l’ordre de passage, de gérer cet aspect de l’application. En effet, des méthodes nous permettent de déterminer l’ordre de passage de cette livraison dans sa tournée et d’obtenir la livraison associée tandis que d’autres nous permettent de fixer la couleur et l’heure de passage correspondant à ce point du plan.

Et enfin, pour les nœuds du graphe et donc plus concrètement les intersections entre les rues, nous avons utilisés la classe « CercleIntersection » qui permet de joindre l’identifiant du nœud au cercle en question. En plus d’une méthode pour obtenir cet identifiant, cette classe dispose aussi d’une méthode pour changer l’affichage d’un nœud quand il est sélectionné au niveau de sa taille et de sa couleur.

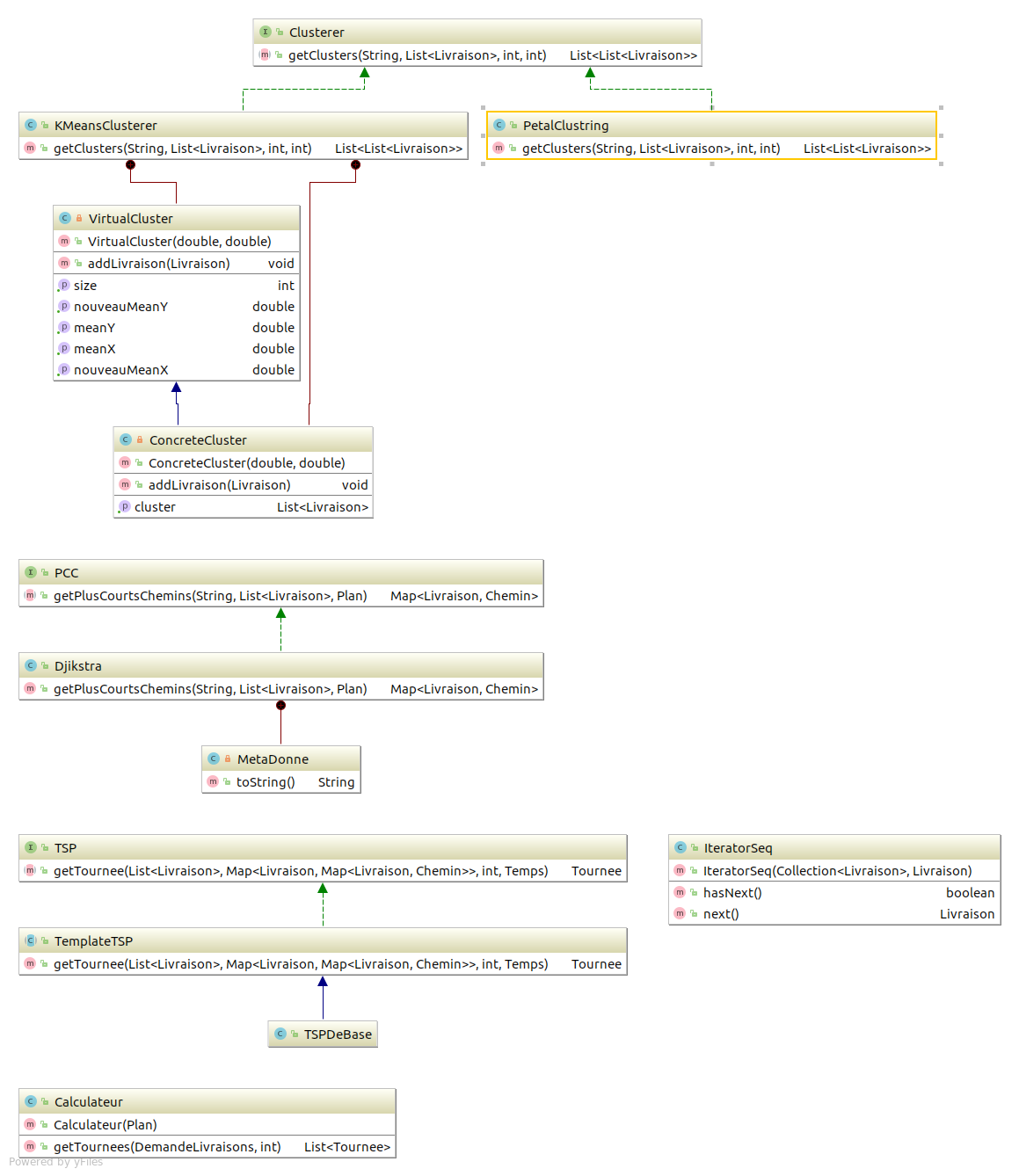
# Diagramme de classes de la première itération :

## Diagramme de packages :

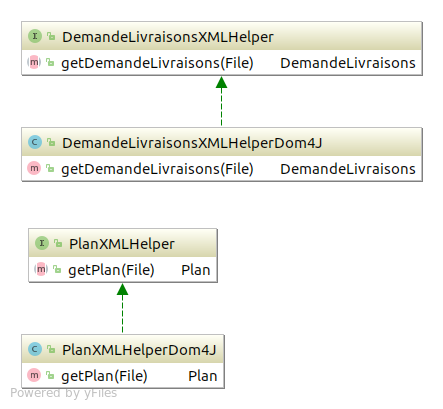


## Diagrammes de classes :

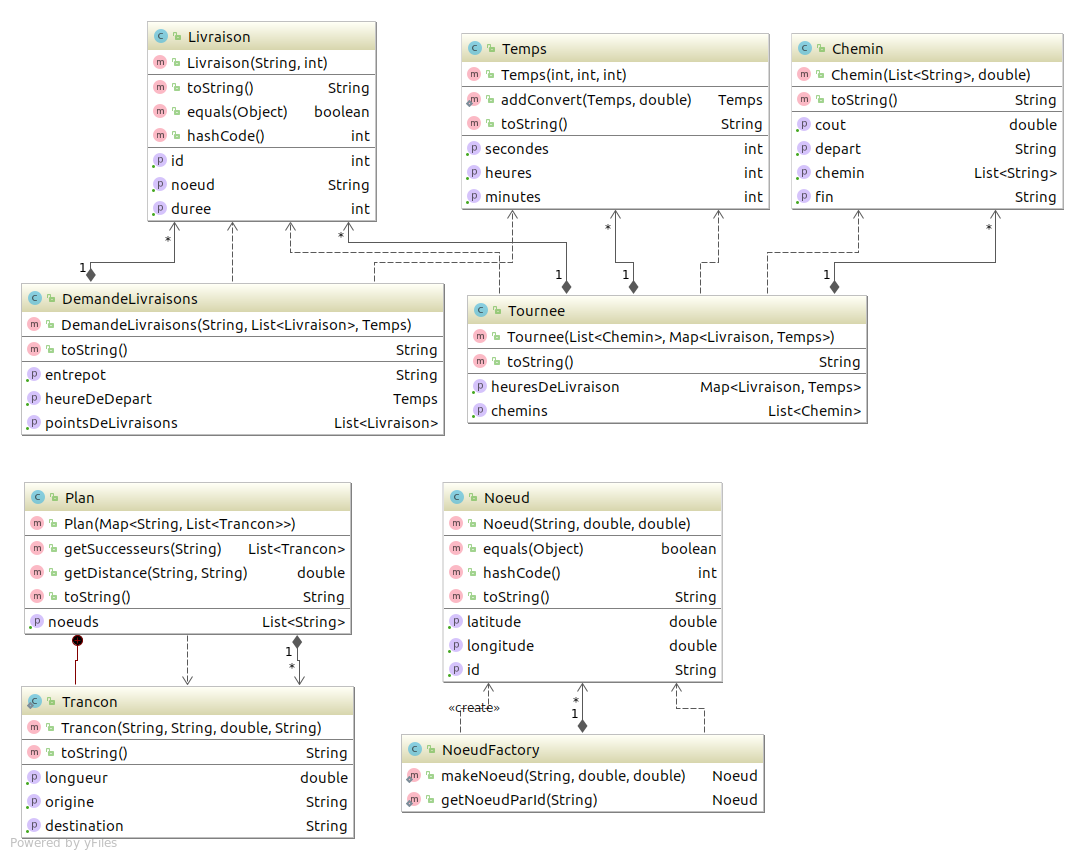
### Package Calcules :



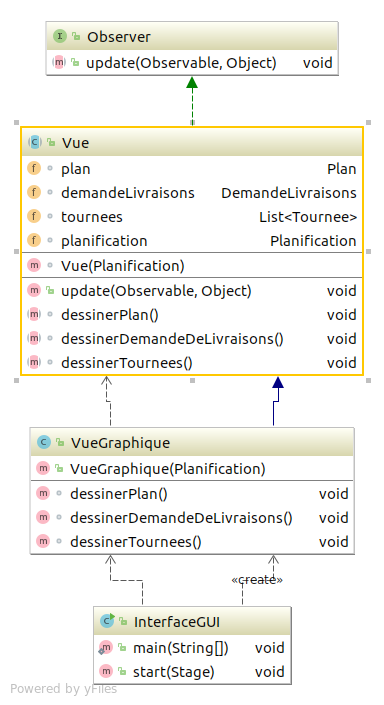
### Package XMLHelpers :



### Package Metier :



Package Vue :



# Diagramme de classes de l’itération finale :

## Diagramme de Packages :

## Package Modèle :

### Package XMLHelpers :

### Package Métier :

### Package Calcules :

## Package Vue :

## Package Contrôleur :

# Comparaison des diagrammes de classes :

Au cours de ce projet, nous avons dû modifier notre organisation avec le progrès de celui-ci car nous avons, afin de permettre un meilleur fonctionnement de plusieurs éléments. En effet,

# Couverture des tests :

# Bilan technique et humain :

Au cours de ce projet, nous avons pu développer de nombreuses compétences techniques afin de parvenir à réaliser les fonctionnalités demandées par le cahier des charges mais pas seulement. En effet, nous avons également dû mettre en place une organisation efficace afin de tirer plein parti de la totalité de l’hexanome tout en gardant une cohérence la plus importante possible.

En ce qui concerne les aspects techniques, nous avons pu mettre en pratique les différents aspects des « design pattern » vus en cours magistral mais également de divers aspects de POO (Programmation Orientée Objet) afin de permettre un fonctionnement plus optimal de notre application. Nous pouvons notamment citer l’usage d’héritage, de polymorphisme et de diverses structures de données telles que des listes, structures de type « map » ou bien des tableaux. De plus, nous avons cherché à maximiser la cohérence tout en permettant un découplage minimal des classes réalisées.

Pour la partie plus organisationnelle de ce projet, nous avons dû mettre en place des méthodes de coordinations afin de maximiser l’efficacité d’un groupe de 7 personne, en passant par des logiciels comme « Trello » pour la conception, « Git » pour le partage et le versionnage du code. Nous avons également réalisé des bilans de groupe de manière régulière afin de déterminer les taches à effectuer, de prioriser celles-ci et de résoudre en commun des problèmes au niveau du fonctionnement de l’application. Ainsi, nous avons pu réaliser de manière concrète l’évolution d’un planning dans le cadre d’un projet, car de nombreuses modifications que ce soit au niveau de la conception ou de l’organisation ont été nécessaire afin d’obtenir un code plus optimal.

Pour finir, nous pouvons constater que ce projet nous aura beaucoup apporté tant par sa charge de travail plus conséquente que les projets de 3IF, que la taille du groupe de travail permettant une progression plus rapide et une réalisation plus complète, ou encore le format avec deux séances par semaines permettant de véritablement mettre en place une organisation commune. Ce PLD nous a donc offert une vision beaucoup plus concrète de la réalisation de projet informatique en entreprise, bien plus proche des travaux réalisés lors de nos stages respectifs que de ceux mis en œuvre durant la 3IF.

Cependant, nous nous sommes heurté à certaines difficultés au long du projet comme la taille inhabituelle de l’équipe qui pouvait mener à certaines confusions dans l’organisation et le partage de la charge de travail, à des incohérences au niveau du code ou encore à des erreurs de coordinations.

# Glossaire :

Nœud : représentation d'une intersection, elle possède un ID et des coordonnées.

Tronçon : représentation d'une portion de route, identifiée par un point de départ et d'arrivée (correspondant à des ID de nœuds), le nom de la route, et la distance.

Chemin : ensemble ordonné de tronçons et de nœuds reliant un nœud de départ et d'arrivée, il possède une distance associée.

Point de livraison : nœud correspondant à une destination intermédiaire, associée à une durée et une heure de livraison.

Demande de livraison : ensemble non ordonné de points de livraison.

Tournée : ensemble ordonné de livraisons, reliés par des tronçons, avec un livreur, une heure de départ et un point de départ et d'arrivée fixes et identiques.

Livreur : salarié de l'entreprise qui réalise une tournée.

Plan : ensemble de nœuds et de tronçons.