

# Optimod'Lyon

## Compte Rendu de l'Itération 2

-H4302-

Lynn Ghandour

William Occelli

Joseph Simonin

Pierre Faure--Giovagnoli

Etienne Chauvet

Mathilde Moureau

Charlotte Delfosse

# Sommaire

<b>Choix Architecturaux</b>	<b>2</b>
Architecture Générale (REST)	2
Vue	4
Structure	4
Package controleur	4
Package model	4
Package display	5
Package util	5
Package lib	5
Technologies	5
Back End	7
Structure	7
Technologies	7
<b>Design Patterns</b>	<b>8</b>
MVC	8
State	8
Singleton Pattern	8
Template Method	8
<b>Diagrammes de Classes</b>	<b>9</b>
Contrôleur et états	9
Modèle	11
Algorithmes	12
<b>Choix algorithmiques</b>	<b>13</b>
Principe général	13
TSP	13
<b>Tests et Gestion des exceptions</b>	<b>15</b>
Tests Unitaires	15
Exceptions	16
<b>Plannings des tâches</b>	<b>17</b>
Itération 1	17
Itération 2	19
<b>Bilan Technique et Humain</b>	<b>21</b>
Organisation de l'équipe	21
Outils de codage	21
Conclusion et ressentis	21
<b>Annexes - Fonctionnalités</b>	<b>23</b>
Chargement de la Map et des points de livraison	23
Choix du nombre de livreur	23
Calcul des itinéraires	23
Détails des itinéraires	24
Ajout d'un point	27
Suppression d'un point	29

# Choix Architecturaux

## Architecture Générale (REST)

Pour ce projet, nous avons décidé de créer une application web afin d'avoir un maximum de flexibilité pour la création de l'interface. Nous avons choisi de mettre en place une architecture de type REST. Les responsabilités sont donc partagées entre un client et un serveur, qui communiquent via une API afin d'appeler (côté client) ou répondre (côté serveur) à une requête HTTP permettant l'accès à des ressources. Ces ressources sont identifiées de manière unique grâce à des URI. Le format de réponse est normalisé : l'utilisation du JSON, réputé pour sa légèreté, comme format d'envoi et de retour pour tous nos services permet une rapidité accrue de l'interface sur les navigateurs et est très agréable à manipuler (il a été conçu par Javascript). Le client correspond à la partie Front End de l'application, il s'agit de la vue ou encore de l'interface. Le serveur correspond quant à lui à la partie Back End de l'application. Ce style d'architecture nous a permis de simplifier l'implémentation des composants de l'application : la séparation vue/services réduit la complexité du système et le peu de services développés dans le cadre de cette application, fait qu'il n'est pas forcément nécessaire de les rendre indépendants dans le Back End.

Cependant nous n'avons pas exactement créé une API REST, car nous avons choisi de stocker l'état de l'application non seulement dans la vue mais aussi dans la partie Back End de l'application. Nous verrons que cet état n'est pas exactement le même côté client et côté serveur.

## Vue

### Structure

#### Package controleur

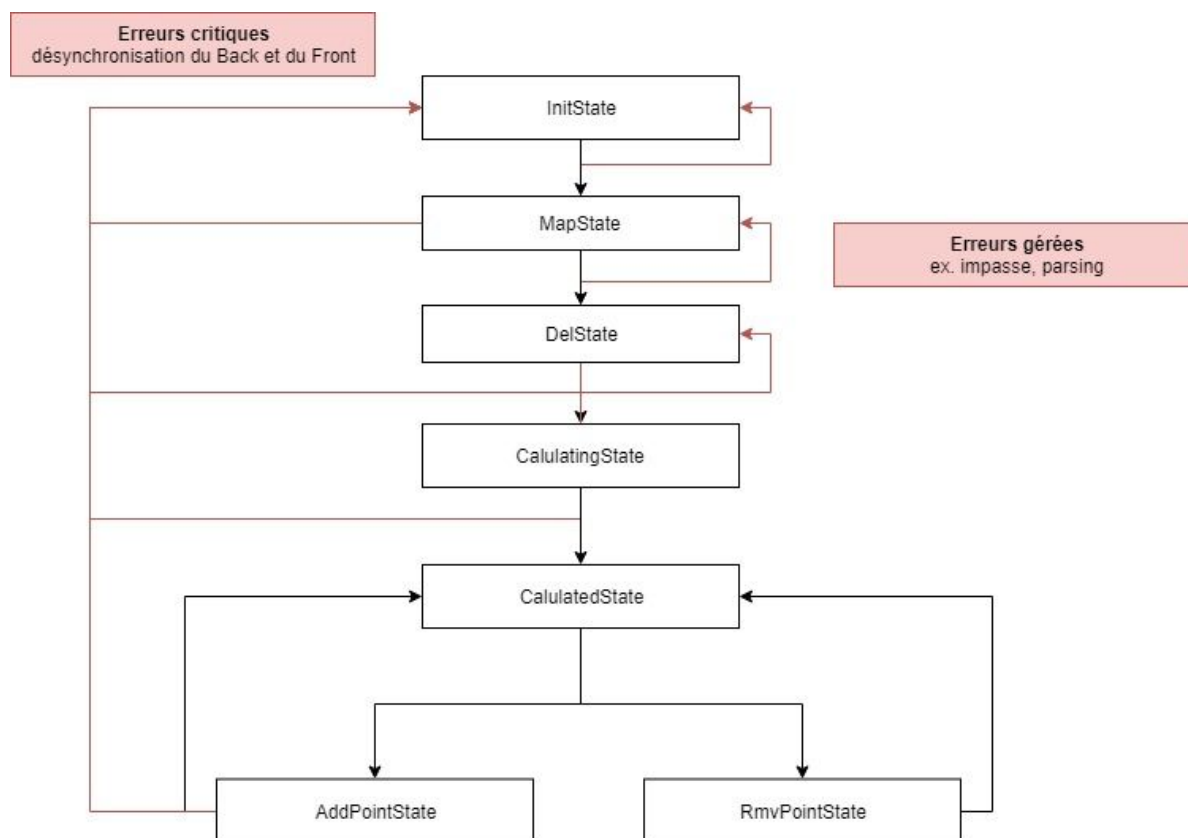
La vue est gérée par un **contrôleur à états**. Les fonctions du **contrôleur** sont :

- Gérer les changements d'état suite à l'appui sur les boutons de l'interface
- Gérer le modèle de la vue, notamment en appelant les fonction associées.

Les fonctions des **états** sont :

- Activer/Désactiver les boutons de l'interface propre à l'état
- Afficher/Cacher les parties de l'interface (ex. timeline) en fonction de l'état
- Gérer les entrées souris et clavier -> renvoie les infos au contrôleur pour traitement

Voici le diagramme de changement d'états :



#### Package model

Le modèle de la vue est une version réduite du modèle du back-end stockée en local sur le navigateur (ceci permettant une grande réactivité de l'interface) contenant uniquement les

données utiles pour l'affichage. De plus, les données sont stockées sous un format optimisant la complexité de l'algorithme d'affichage permettant ainsi l'ajout des fonction de zoom et de panning sans compromettre l'expérience utilisateur.

Le but de cette organisation est de mettre les bons formats de données aux bons endroits. En effet, les besoins de l'algorithme de Dijkstra ne sont pas les même que ceux de l'affichage. Ainsi, chaque module se sert dans le modèle et convertit les données au format le plus optimal pour ses besoins.

Chaque élément du modèle **implémente** la méthode *load* pour charger ses données depuis le backend ainsi que la méthode *display* pour s'afficher.

### Package display

Ce package contient un service d'affichage. Nous avons conçu le service de manière à le rendre générique et modulaire. Il contient les infos du canvas d'affichage et la fonction de normalisation des coordonnées pour toute l'application. Il est capable de tracer des cercles, des lignes, des carrés ainsi que de gérer un tampon de données (buffer) pour un affichage plus rapide.

Ce service est utilisé dans tout le modèle car chaque objet sait s'afficher. Comment rendre une unique instance du service accessible partout tout en s'assurant de l'encapsulation des données ? Nous avons utilisé le design pattern **Dependency Injection**. Il y a de nombreuses manières de mettre en place ce mécanisme. Nous avons pour notre part décidé de passer le service par pointeur et de le rendre attribut des classes qui en ont besoin.

### Package util

Ce package contient des fonctions générales utilisées partout dans l'application. On y trouve :

- Des fonctions de gestion du temps
- Des fonctions d'affichage d'infos pour l'utilisateur

### Package lib

Ici sont stockées les librairies externes non accessible en CDN. Il n'y en a à l'heure actuelle qu'une seule, une bibliothèque bootstrap de gestion de slider.

## Technologies

### Stack

- *HTML5*
- *CSS3 (Bootstrap4)*
- *Javascript : (jQuery)*

Pour la vue, nous avons décidé d'utiliser un site html/css/js statique hébergé par un serveur HTTP. Une application WEB permet une grande modularité et une compatibilité accrue avec presque tous les supports (Windows, Mac, Linux, Android, ios...).

Le combo html/css permet d'obtenir très rapidement des interfaces ergonomiques et nous l'avons de plus combiné à Bootstrap 4 afin d'accélérer l'implémentation de l'interface. De son côté, le

Javascript est un langage en plein essor permettant une grande liberté de codage et des performances très élevées. Il est également très léger (à l'heure actuelle, la vue pèse 350ko). Sa grande communauté permet un support rapide et une grande réactivité de résolution des problèmes.

## Back End

### Structure

Le Back End se découpe en 5 grandes parties :

1. Les points terminaux (ou endpoints) qui permettent de répondre aux requêtes HTTP
2. Un contrôleur dont les méthodes sont appelées par les points terminaux et qui orchestre des états
3. Les états, qui permettent d'autoriser ou non l'appel de certaines méthodes selon l'état de l'application
4. Le modèle, qui contient nos objets métiers et qui sont manipulés par les états.
5. Les algorithmes, qui sont manipulés par les états.

### Technologies

#### Stack

- *Java - SpringBoot (Maven, Tomcat)*
- *JUnit*

Afin de simplifier la mise en place de l'architecture back-end, nous avons décidé d'utiliser l'environnement Spring Boot qui fournit parmi de nombreuses fonctionnalités un serveur web (Tomcat) embarqué. Il a également l'avantage de faciliter la mise en place des tests unitaires.

# Design Patterns

## MVC

Nous avons utilisé le design pattern Modèle/Vue/Contrôleur qui a l'avantage d'offrir une séparation claire des responsabilités, un couplage faible et une forte cohésion (bien qu'elle ajoute de la complexité car il en résulte un nombre plus conséquent de classes).

Dans ce pattern la vue fournit une interface graphique. Le contrôleur est celui du Back End, qui répond aux requêtes de la vue afin de traiter les actions de l'utilisateur. Le modèle quant à lui contient les données affichées, il s'agit de l'univers dans lequel s'inscrit l'application (plan, livreurs, livraisons...). Il est complètement indépendant des autres modules : il ne se sert ni de la vue ni du contrôleur. Le contrôleur sert d'intermédiaire entre la vue et le modèle, et la vue, contrairement au patron MVC classique, ne lit pas le modèle : elle interagit uniquement avec le contrôleur via des points terminaux.

## State

Le patron état (state) propose deux classes principales :

- la classe État, qui définit l'abstraction des comportements du patron
- la classe Contexte, ici Contrôleur (Back End), interface entre les états et le reste de l'application.

## Singleton Pattern

Afin de rendre la classe CityMap (plan de la ville) accessible partout et de n'en gérer qu'une seule instance, nous avons décidé d'en faire un Singleton. Cependant nous nous sommes rendu compte que d'autres éléments devaient posséder les mêmes propriétés (livraisons, entrepôt, livreurs). Comme d'un point de vue métier il ne nous semblait pas judicieux de faire apparaître ces éléments dans CityMap, nous avons créé une classe MapManagement, qui contient une CityMap ainsi qu'une liste de livraisons et de livreur et un entrepôt, et nous lui avons appliqué le patron Singleton.

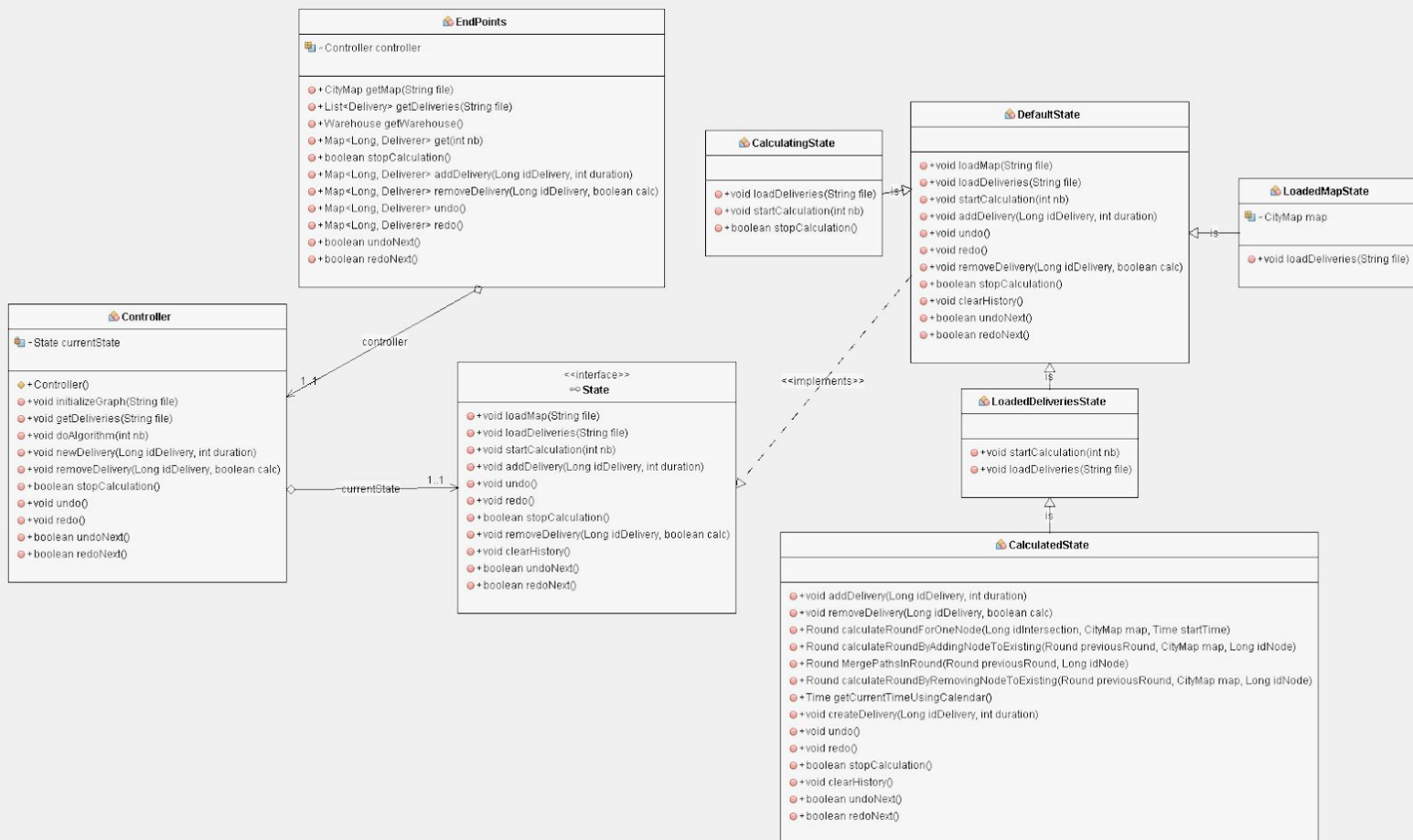
## Template Method

Pour pouvoir implémenter différentes heuristiques pour l'algorithme du TSP, nous avons utiliser la méthode de Template. Nous avons donc dû implémenter une interface pour la classe TSP, une classe Template et ses classes filles. De cette façon nous n'avions à implémenter qu'une fois l'algorithme pour plusieurs TSP différents. Dans les classes fille du Template, seules les méthodes pour calculer la borne supérieur du "branch and bound" et l'itérateur sont implémentées.



# Diagrammes de Classes

## Contrôleur et états



Voici ci-dessus notre diagramme de classes pour le contrôleur du Back end avec les différents états qu'il orchestre. Nous avons également représenté la classe `EndPoints` qui contient nos points terminaux. Par rapport à l'itération 1, la séparation des actions en 3 types de classes (points terminaux, contrôleur, états), permet de mieux comprendre ce que fait chacun et donc d'augmenter la cohérence.

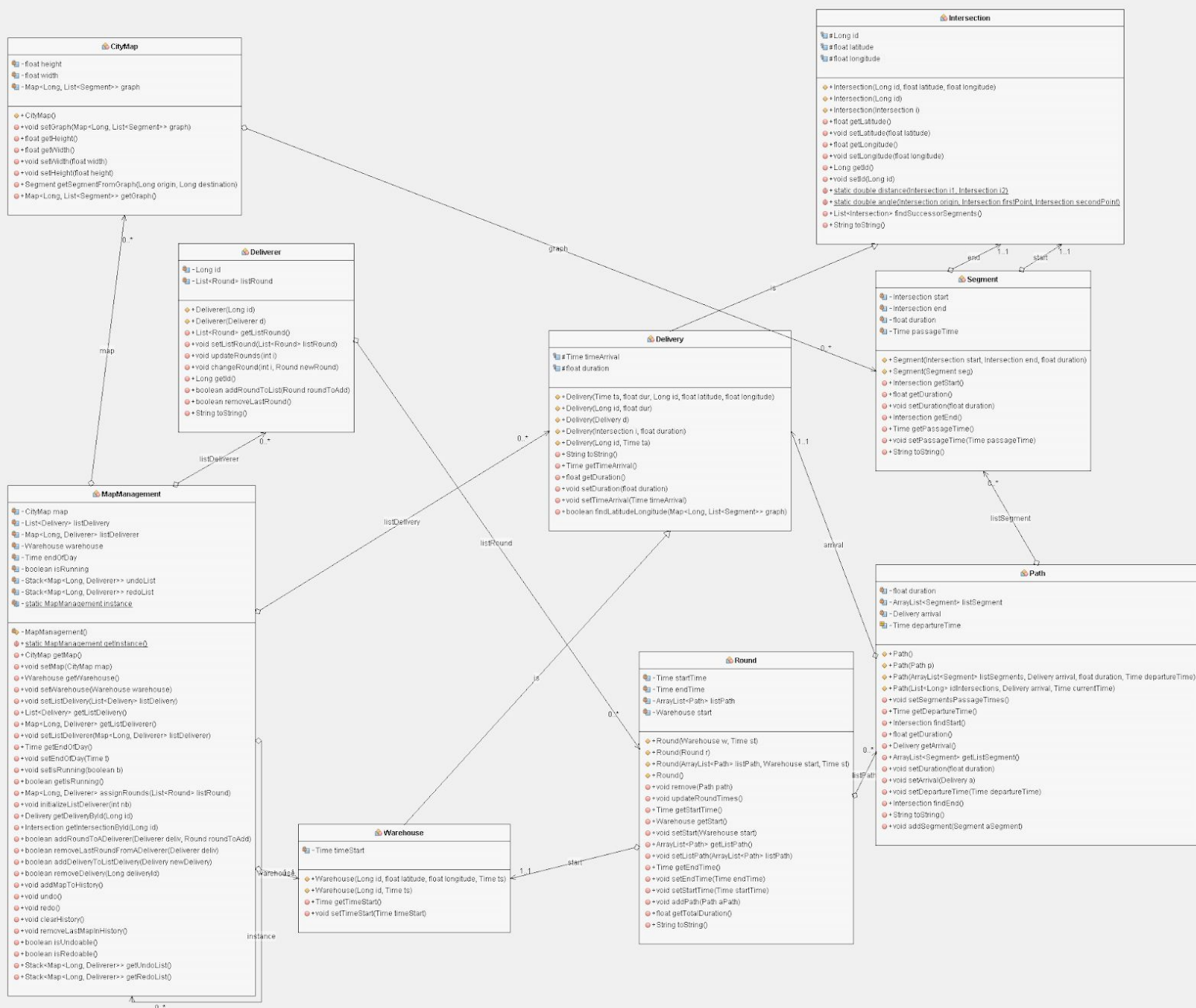
Concrètement, la classe `EndPoint` appelle une méthode du contrôleur dans chaque point terminal. Le contrôleur possède un attribut `currentState` qui lui permet de changer d'état et d'appeler les méthodes que chaque état possède (faute de quoi si l'on essaye d'accéder à une méthode que `currentState` ne possède pas, une exception est lancée).

Les états sont au nombre de 5 :

- DefaultState, l'état par défaut (dans lequel un plan peut être chargé). Tous les autres états héritent de cette classe par défaut.
- LoadedMapState, l'état "plan chargé" (dans lequel nous avons alors le droit de charger des livraisons).
- LoadedDeliveriesState, l'état livraisons chargées (dans lequel nous avons le droit de lancer un calcul de tournées).
- CalculatingState, l'état "calcul en cours" (dans lequel on peut relancer un calcul ou bien l'arrêter).
- CalculatedState, qui hérite de LoadedDeliveriesState et dans lequel on peut ajouter/supprimer des livraisons et faire des retours avant/arrière. Il s'agit donc de l'état le plus avancé.

Les états implémentent l'interface "State" qui définit toutes les méthodes que peuvent posséder un état et qui par défaut lancent une exception.

## Modèle



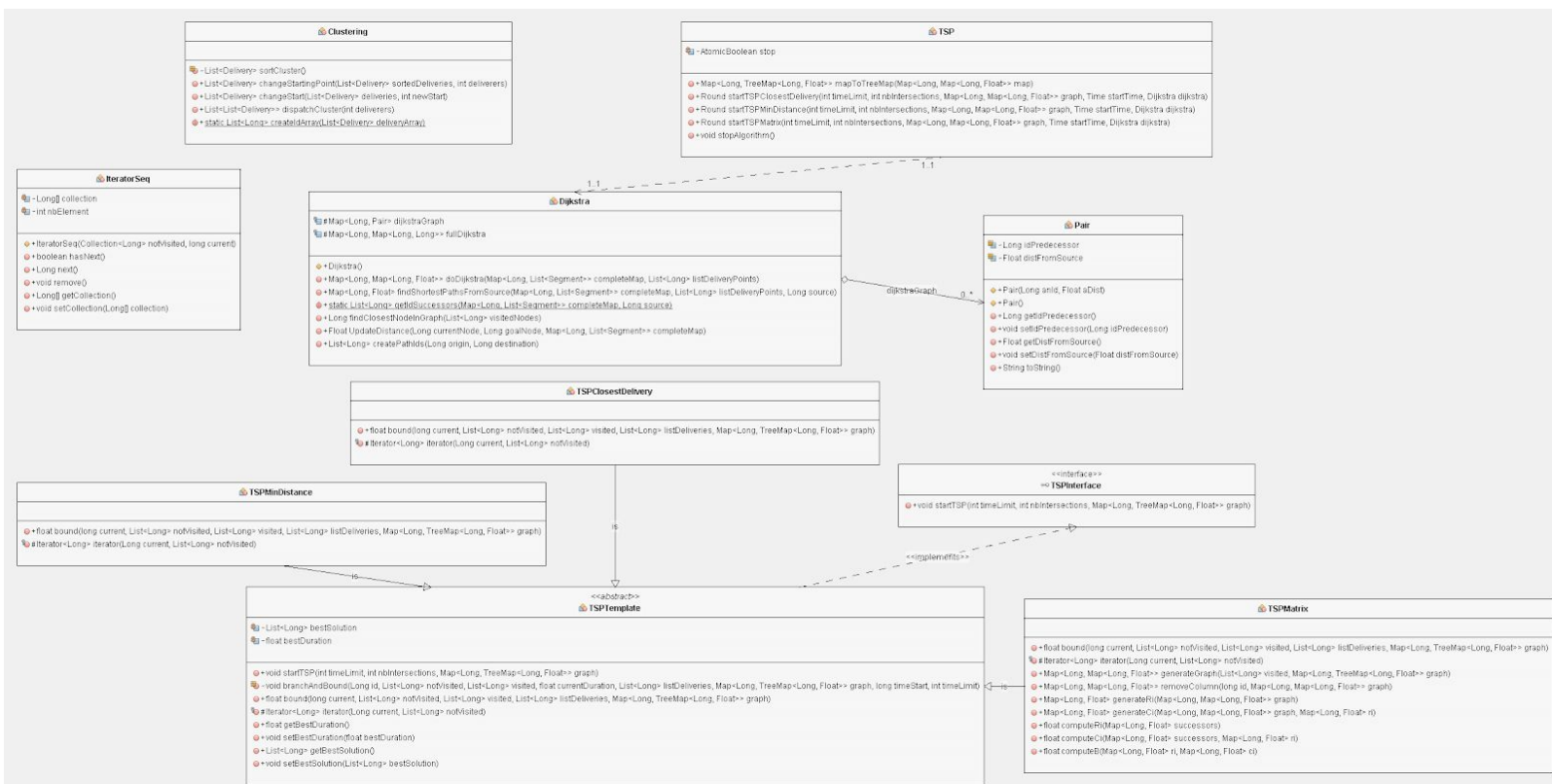
Voici notre diagramme de classe pour le package contenant le modèle. Par rapport à l'itération 1, nous avons ajouté une classe servant d'interface entre le modèle et le reste de l'application : il s'agit de MapManagement, classe utilisant le pattern Singleton. Nous avons fait ce choix car nous

ne voulions pas avoir des listes de livreurs et de livraisons dans la classe CityMap. En effet, nous avons estimé que celle-ci devait contenir essentiellement la représentation sous forme de graphe du plan de la ville.

De plus nous avons choisi d'ajouter une classe Deliverer pour le livreur afin de pouvoir l'identifier et lui attribuer une ou plusieurs tournées plus facilement.

Enfin, pour améliorer notre application et implémenter de nouvelles fonctionnalités (par exemple l'affichage de l'heure d'arrivée des points de livraison) nous avons dû ajouter de nouveaux attributs pour faciliter le calcul, notamment pour la classe Path et Round.

## Algorithmes



Pour les algorithmes, la principale différence avec l'itération 1 est que nous avons implémenté un nouveau TSP plus optimisé que le précédent (cf partie "choix algorithmiques"). Nous avons donc de nouvelles classes associées et nous avons également choisi d'utiliser un template (cf "design patterns")

Pour implémenter notre algorithme, nous avons créé au moins une classe par algorithme à exécuter (Clustering, Dijkstra, TSP). Nous avons également créé une classe Pair adaptée à nos besoins de calcul d'algorithme.

Pour le TSP, nous avons utilisé le modèle Template, c'est pourquoi nous avons une interface, une classe mère et plusieurs classes filles (correspondant aux différentes heuristiques trouvées pour le TSP). Nous avons également une classe TSP appelant les différents TSP possible et qui permet d'obtenir un objet Round.

## Choix algorithmiques

### Principe général

Pour calculer les trajets les plus optimaux pour les livreurs, nous avons procédé en plusieurs étapes.

Tout d'abord, nous avons appliqué un algorithme de Dijkstra pour connaître la plus petite distance entre les différents points de livraison et l'entrepôt. Nous avons ensuite réalisé un Cluster des points de livraisons en se basant sur leur coordonnées polaires par rapport à l'entrepôt. Puis pour chacun des clusters de points de livraison nous avons appliqué l'algorithme du TSP.

Par ailleurs, nous avons choisi pour le calcul des algorithmes, pour gagner en rapidité, de ne pas utiliser des objets mais uniquement les informations nécessaires (identifiants des Intersections et des Points de Livraison, durée des Segments).

### TSP

Pour le TSP, nous avons mis en place la méthode de "branch and bound" et avons essayé trois heuristiques différentes pour calculer la borne.

1. L'heuristique du point le plus proche, qui renvoie la plus petite distance entre le point actuel et ceux qui restent à parcourir
2. L'heuristique des distances minimum, qui renvoie la somme des distances entre le point actuel et ceux qui restent à parcourir
3. L'heuristique basée sur la construction du graphe sous forme de matrice. Pour réaliser cette heuristique nous avons utilisé des éléments d'une thèse trouvée sur internet :
































<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=13&cad=rja&uact=8&ved=2ahUKEwiy1deg5bDfAhVSXhoKHfrqCWAQFjAMegQICBAC&url=https%3A%2F%2Fhrcak.srce.hr%2Ffile%2F236378&usg=AOvVaw3H65VkEVp7BKfJwvkdXgzx>

Après avoir tester les différents TSP pour différents cas de figure nous nous sommes rendu compte que la plus juste et la plus rapide était la première.



# Tests et Gestion des exceptions

## Tests Unitaires

▼ agile		88.6 %	14,847	1,914
▶ src/test/java		99.7 %	10,727	29
▼ src/main/java		68.6 %	4,120	1,885
▼ optimodlyon.agile.algorithmic		93.8 %	2,271	149
▶ Clustering.java		87.6 %	692	98
▶ Dijkstra.java		100.0 %	463	0
▶ TSPMatrix.java		100.0 %	425	0
▶ TSP.java		94.1 %	320	20
▶ TSPTemplate.java		98.5 %	203	3
▶ IteratorSeq.java		98.3 %	58	1
▶ Pair.java		46.0 %	23	27
▶ TSPClosestDelivery.java		100.0 %	50	0
▶ TSPMinDistance.java		100.0 %	37	0
▼ optimodlyon.agile.models		99.3 %	1,559	11
▶ MapManagement.java		97.4 %	415	11
▶ Round.java		100.0 %	259	0
▶ Path.java		100.0 %	243	0
▶ Deliverer.java		100.0 %	191	0
▶ Intersection.java		100.0 %	181	0
▶ Delivery.java		100.0 %	114	0
▶ Segment.java		100.0 %	70	0
▶ CityMap.java		100.0 %	63	0
▶ Warehouse.java		100.0 %	23	0
▶ optimodlyon.agile.states		0.0 %	0	1,013
▶ optimodlyon.agile.xml		0.0 %	0	422
▼ optimodlyon.agile.util		88.3 %	286	38
▶ Time.java		88.3 %	286	38
▶ optimodlyon.agile.endpoints		0.0 %	0	126
▶ optimodlyon.agile.controller		0.0 %	0	85
▶ optimodlyon.agile.exceptions		10.8 %	4	33
▶ optimodlyon.agile		0.0 %	0	8

Pour les tests unitaires, nous avons décidé de tester uniquement le modèle et les algorithmes (ainsi que la classe Time très utile aux algorithmes). Ci-dessus voici la couverture des tests sous Eclipse : l'application est couverte à 68,6% avec la partie algorithmie couverte à 93,8% et la partie modèle à 99,3%. Le pourcentage de couverture global reste faible car nous avons manqué de temps. Notamment, il aurait été judicieux de tester la classe de désérialisation de fichier XML car celle-ci peut rapidement lever des exceptions ou rencontrer des problèmes (auxquels nous avons cependant essayé de répondre au mieux).

## Exceptions personnalisées

Nous avons souhaité envoyer des messages d'erreurs customisé au Front End pour lui indiquer les éventuels problèmes d'accès aux ressources (en plus d'exceptions déjà implémentées par Java). Ces derniers ont un statut HTTP propre. Nous avons 4 exceptions personnalisées :

Nom	Message	Statut HTTP	Exemple de levée
UnprocessableEntityException	"Some files must be loaded first."	422	Lorsqu'un utilisateur cherche à accéder à un service (ex : chargement de livraisons) alors qu'il doit d'abord appeler d'autres services (ex : charger un plan).
UndoRedoException	"You cannot redo or undo."	403	Lorsqu'un utilisateur appuie sur le bouton de retour arrière ou avant alors que cela n'est pas possible (plus d'historique).
FunctionalException	"The request doesn't respect fonctionnal requisites (a round is too long for example)"	406	Lorsqu'un utilisateur demande à effectuer un calcul trop coûteux en mémoire.
DijkstraException	"The user asked for something that is not computable for our algorithm (one way road for example)"	400	Lorsqu'un utilisateur demande à ajouter un point à un endroit du plan inaccessible.



Pour ce qui est de l'exception `UnprocessableEntityException`, nous avons hésité à la mettre en place car si l'on considère que l'interface et le back end sont d'un seul tenant (ce qui est le cas ici), cela n'est pas nécessaire. Cependant si on imagine que plusieurs interfaces peuvent se connecter à notre back end, cela est souhaitable et rend l'application plus robuste.

## Plannings des tâches

### Itération 1

	Tâche	Ressource	Date	Durée (h)
<b>Conception</b>	Définition des cas d'utilisation	Tout le monde	20/11/2018	1
	Définition des fonctionnalités	Tout le monde	20/11/2018	1
	Réflexion sur le modèle	Tout le monde	20/11/2018	1
	Définition des classes principales et de leur rôle	Tout le monde	20/11/2018	2
	Diagramme de classes	Lynn Ghandour	20/11/2018	1
	Diagramme de Séquence	Mathilde Moureau	20/11/2018	1
	Description des cas d'utilisation	Mathilde Moureau	20/11/2018	1
<b>Technique</b>	Choix des technologies et de l'environnement de travail	Tout le monde		-
<b>Organisation</b>	Répartition des tâches	Tout le monde		-
	Revue et rétrospective de l'itération 1	Tout le monde		0,25
<b>Front End</b>	Conception de la vue avec choix des technologies	Pierre Faure--Giovagnoli	20/11/2018	2
	Mise en place du serveur et des librairies	Pierre Faure--Giovagnoli	20/11/2018	1

	Dessin des IHMs	Pierre Faure--Giovagnoli	20/11/2018	0,5
	Premier affichage sommaire de la carte	Pierre Faure--Giovagnoli	23/11/2018	2
	Amélioration de l'affichage + première version de l'affichage de points de livraison tests	Pierre Faure--Giovagnoli	23/11/2018	3
	Affichage de chemins tests	Pierre Faure--Giovagnoli	24/11/2018	0,5
	Implémentation sommaire des fonctions d'ajout et de suppression de point	Pierre Faure--Giovagnoli	24/11/2018	3
	Amélioration de l'architecture + intégration d'un contrôleur de vue à état	Pierre Faure--Giovagnoli	27/11/2018	3
	Connexion de la vue avec le backend	Pierre Faure--Giovagnoli	27/11/2018	3
	Amélioration de performance et d'ergonomie + correction de bug	Pierre Faure--Giovagnoli	30/11/2018	2
<b>Back End</b>	Mise en place du serveur Spring	Charlotte Delfosse		1
	Implémentation d'un déserialiseur XML	Charlotte Delfosse		1
	Implémentation du service de chargement de plan en REST	Charlotte Delfosse		1
	Implémentation du service de chargement de points de livraison en REST	Lynn Ghandour	27/11/2018	1

	Implémentation de la gestion du nombre de livraisons par cluster	Mathilde Moureau	29/11/2018	2
	Implémentation du Clustering	Joseph Simonin, Mathilde Moureau		3
	Implémentation de l'algorithme de Dijkstra	William Occelli	28/11/2018	10
	Test des méthodes	Etienne Chauvet, Mathilde Moureau	24/11/2018	2
	Implémentation d'une version brut force du TSP	Étienne Chauvet		2
	Codage des classes du modèle	Lynn Ghandour	23/11/2018	1
<b>Divers</b>	Rédaction du Compte Rendu	Charlotte Delfosse, Pierre Faure Giovagnoli		2

## Itération 2

	Tâche	Ressource	Durée prévue (h)	Durée effective (h)
<b>Conception</b>	Définition des nouveaux cas d'utilisation	Tout le monde	0,5	1*7
	Définition des fonctionnalités	Tout le monde	1	1*7
	Réflexion sur les classes du modèles à modifier/ajouter	Tout le monde	-	2*7
<b>Technique</b>	Mise en place d'un serveur d'intégration sur GitHub	Charlotte Delfosse	1,5	1,5
<b>Organisation</b>	Planification de l'itération 2	Charlotte Delfosse	2	2
	Revue et rétrospective de l'itération 1	Tout le monde	-	0,25*7
<b>Front End</b>	Codage de la timeline	Pierre Faure--Giovagnoli	-	1
	Implémentation du menu récapitulatif des rounds des livreurs sur le panel de droite	Pierre Faure--Giovagnoli	-	3

	Connexion de la timeline avec le reste de l'application	Pierre Faure--Giovagnoli	-	3
	Refonte architecturale : Dependency Injection	Pierre Faure--Giovagnoli	-	2,5
	Diverses modifications et optimisations	Pierre Faure--Giovagnoli	-	2
<b>Back End</b>	Amélioration du Clustering	Mathilde Moureau	2	3
	Nouveaux TSP	Lynn Ghandour, Joseph Simonin	8	12
	Tests unitaires algorithmie	Mathilde Moureau, William Occelli, Etienne Chauvet	8	15
	Autres tests unitaires	Mathilde Moureau	10	10
	Gestion des états	Charlotte Delfosse	3	2
	Ajout des fonctionnalités undo/redo	Charlotte Delfosse	1	2
	Gestion des cas limites, erreurs, exceptions et codes HTTP renvoyés	Tout le monde	-	4
	Enrichissement du modèle (ex : restructuration avec MapManagement)	Lynn Ghandour	-	2
	Ajout de la fonctionnalité d'arrêt du calcul	Etienne Chauvet	-	2
	Fonctionnalité d'ajout et de suppression de points	Étienne Chauvet, William Occelli	8	10
	Ajout d'une classe Time pour la gestion des durées	Étienne Chauvet, William Occelli	-	5
	Correction de bugs	Tout le monde	-	25
<b>Divers</b>	Rédaction du Compte Rendu	Charlotte Delfosse, Pierre Faure Giovagnoli, Lynn Ghandour	2	3

# Bilan Technique et Humain

## Organisation de l'équipe

La méthode AGILE/SCRUM nous a confronté à un nouveau mode de fonctionnement auquel nous avons dû nous adapter. Nous avons dû définir un SCRUM Master (Charlotte Delfosse) et un Project Owner (Pierre Faure--Giovagnoli).

Il nous a fallu tout d'abord définir nos objectifs et répartir les tâches pour l'itération 1. Afin de produire une estimation sur l'effort de développement de l'application, nous avons utilisé un "planning poker" : à l'aide d'un jeu de carte adapté, nous avons tous misé sur la durée estimée de chaque tâche et après discussion nous avons trouvé un compromis.

A chaque début de réunion, nous avons fait le point pendant 10 à 15 minutes sur l'avancement des tâches et ce qu'il restait à accomplir. Cela nous a permis d'être plus efficaces et de mieux comprendre comment s'intégrait le travail de chacun pour s'avoir à qui s'adresser en cas de problème.

## Outils de codage

### *Spécification techniques*

- *Eclipse (partie Java)*
- *Visual Studio Code (partie Web)*
- *Git (avec serveur d'intégration Jenkins)*

Afin de travailler de manière efficace, nous avons utilisé l'outil de versionning Git. Afin de garantir l'intégrité du projet, nous avons mis en place un système de tests unitaires automatiques qui rejette la mise à jour du dépôt distant en cas d'échec d'un ou plusieurs tests.

## Conclusion et ressentis

L'itération 2 a été plus intense que l'itération 1 en terme de travail contrairement à ce que nous avons imaginé. En effet, nous pensions que comme nous avons déjà une application fonctionnelle basique, que nous avons conçu en partant de rien, le reste des fonctionnalités à implémenter serait plus rapide. Cependant cela n'a pas été le cas, car nous voulions que l'application réponde au mieux à des exigences de qualité en terme non seulement de fonctionnalités mais également de robustesse et de rapidité.

En terme d'organisation, il s'est avéré difficile au début de communiquer entre nous et surtout avec le client, ce qui d'ailleurs nous a été reproché. En effet, nous étions trop concentrés sur la manière dont on allait pouvoir construire l'application et pas assez sur la question de savoir ce que voulait le client et qu'elles étaient les idées qui pouvaient l'intéresser (et qui été réalisables en un temps fini). Il aurait été bénéfique par ailleurs d'adopter le réflexe de prendre des notes pour bien mémoriser à la fois les remarques de chacun et les demandes du client.

En conclusion, ce projet a permis de nous mettre dans des conditions proches de la réalité du monde du travail (client, contraintes, organisation en équipe). Côté technique, il nous a fait monter en compétence sur des outils intéressants que peu connaissaient ou maîtrisaient. Il a été également très appréciable de pouvoir assister à la démonstration des autres équipes car nous avons pu constater des manières différentes d'aborder le problème.

## Annexes - Fonctionnalités

### Chargement de la Map et des points de livraison

- Map corrompue



- Fichier syntaxiquement incorrect
- Données invalides

### Choix du nombre de livreur

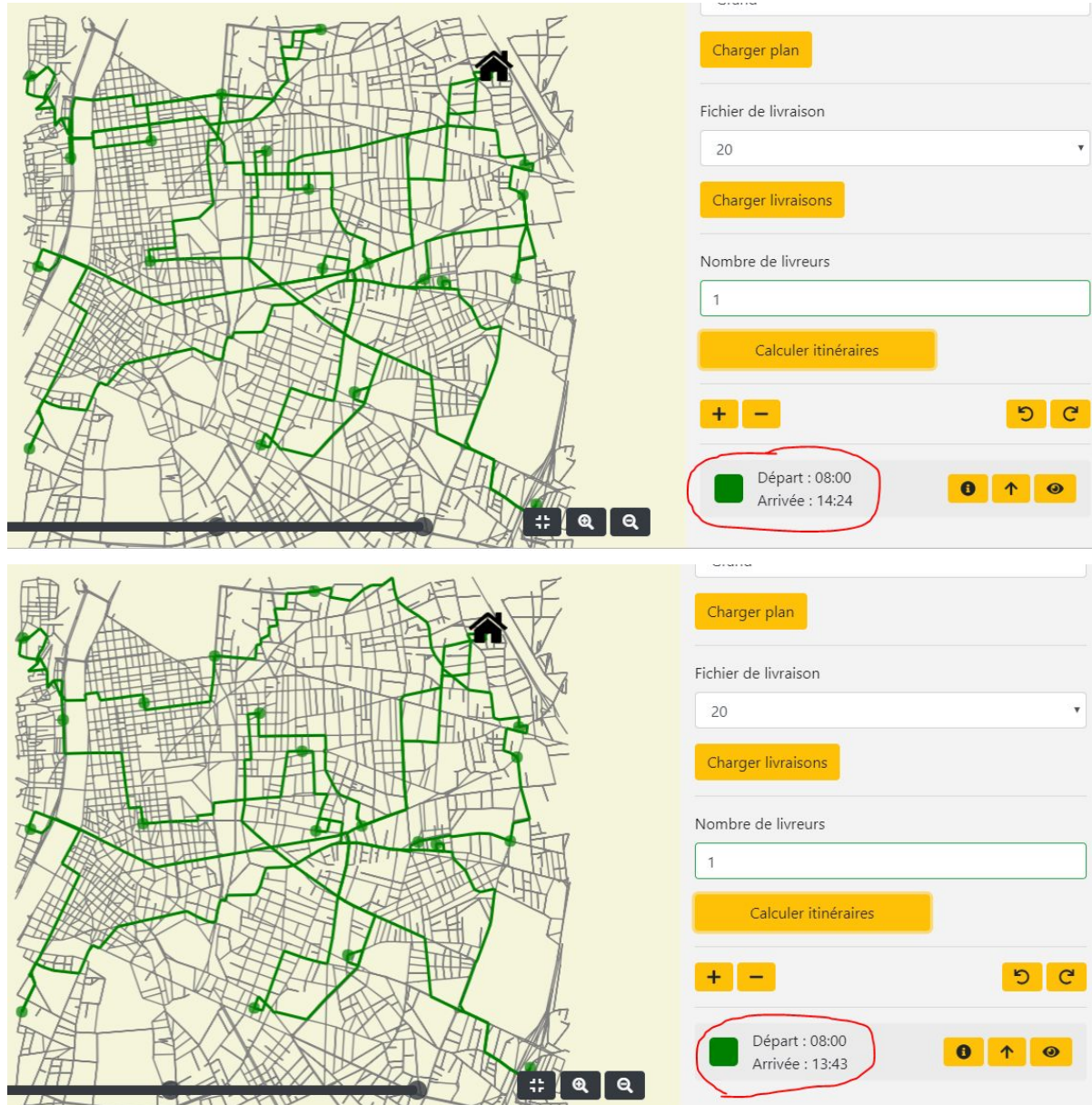
- Grand nombre de livreurs
- Entrée invalide (float, string ou nombre négatif)

A screenshot of a web form titled "Nombre de livreurs". It features a text input field containing the value "2.5". Below the input field, a red error message reads "Veuillez rentrer un nombre valide.". At the bottom of the form is a yellow button labeled "Calculer itinéraires".

### Calcul des itinéraires

- Si moins de livraisons que de livreurs
- Si des points de livraison ont été ajouté précédemment
- Si calcul trop long
- Annulation du calcul

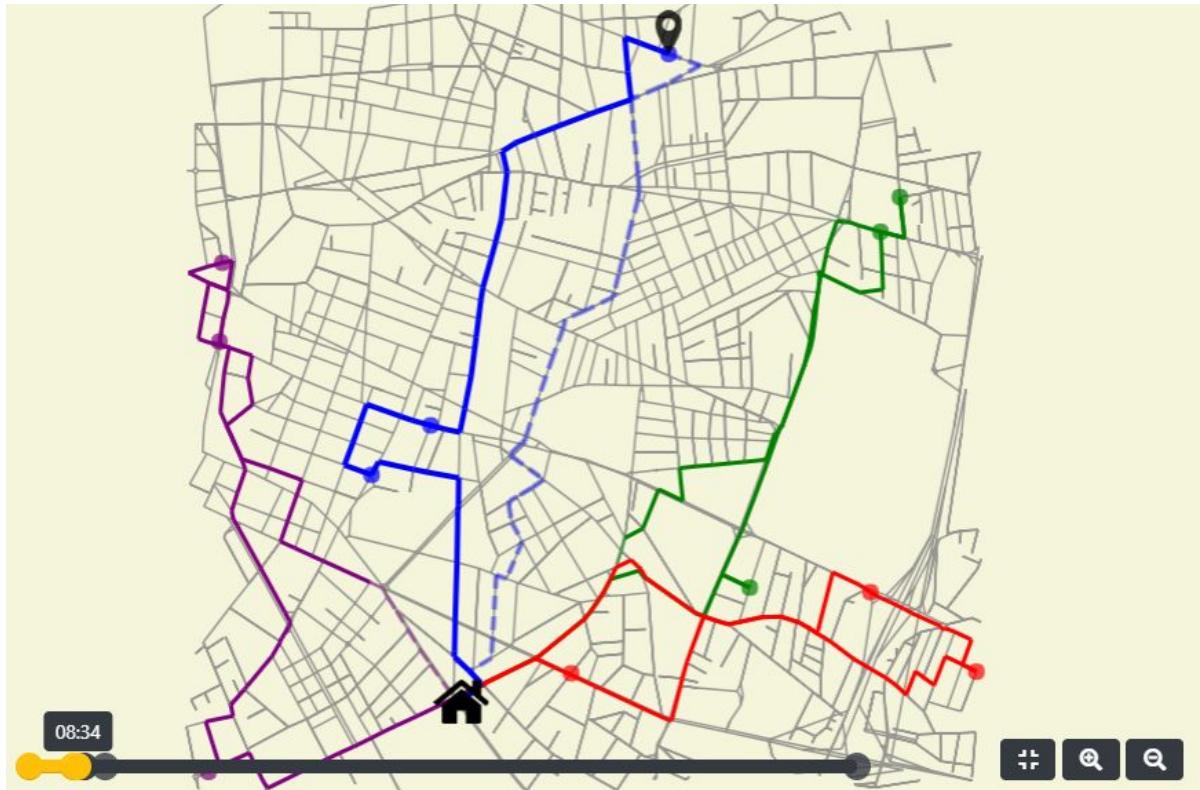
A screenshot of a web form titled "Nombre de livreurs". It features a text input field containing the value "1". Below the input field is a red button labeled "Annuler".



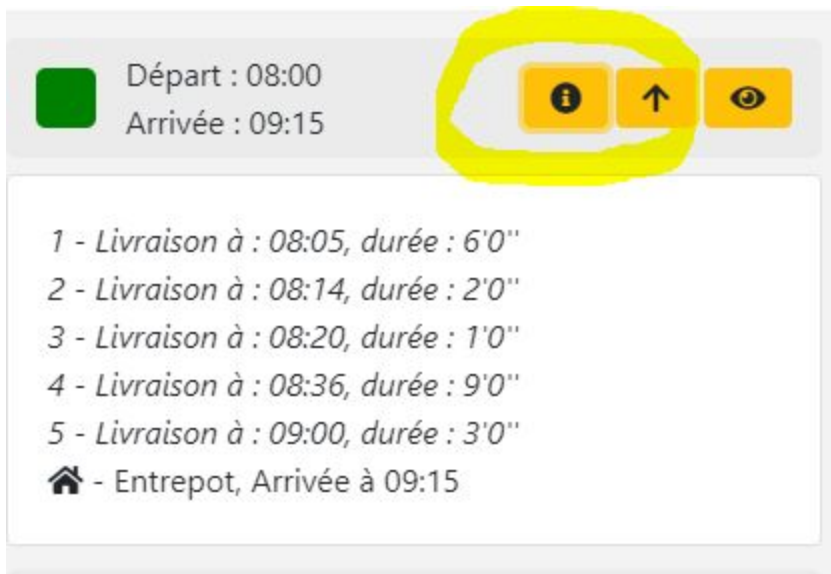
## Détails des itinéraires

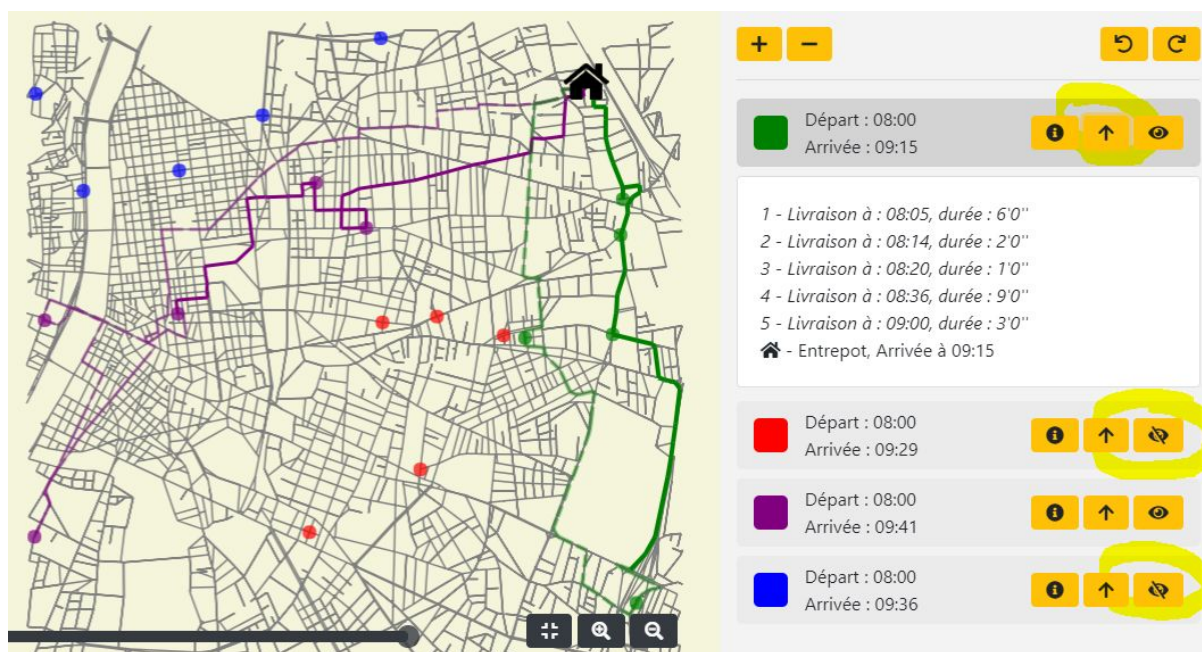
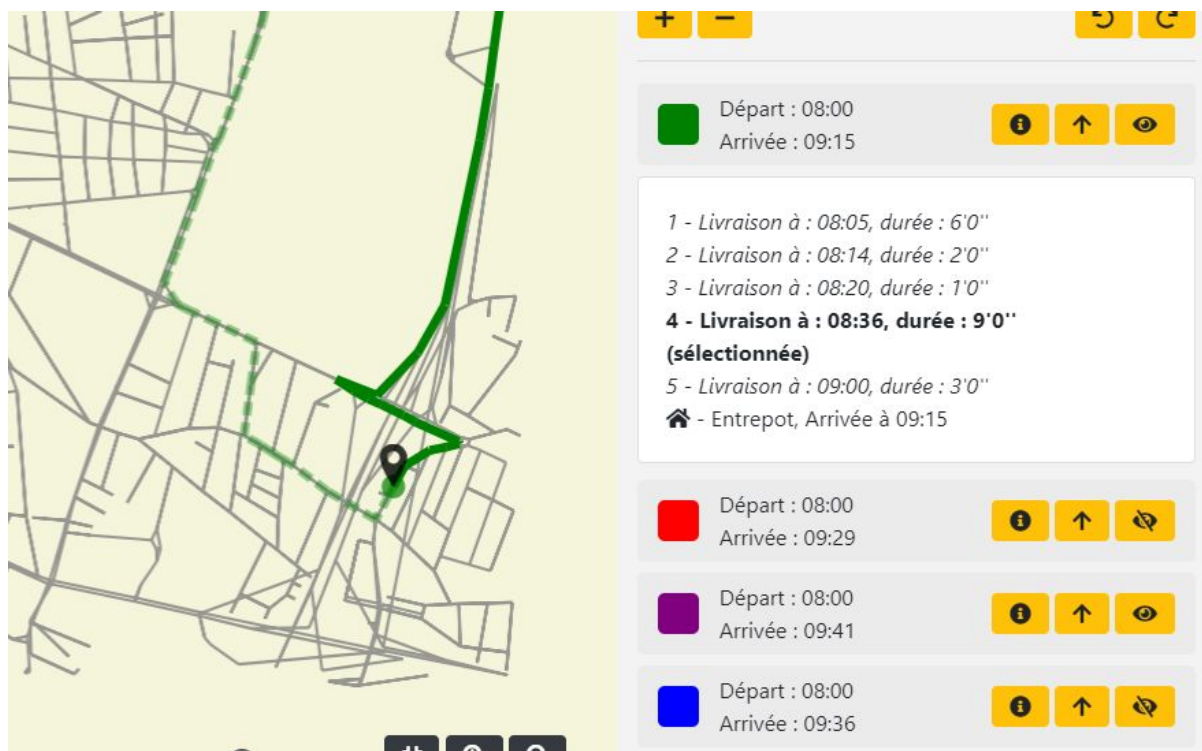
- Clic sur un point





- Vue textuelle



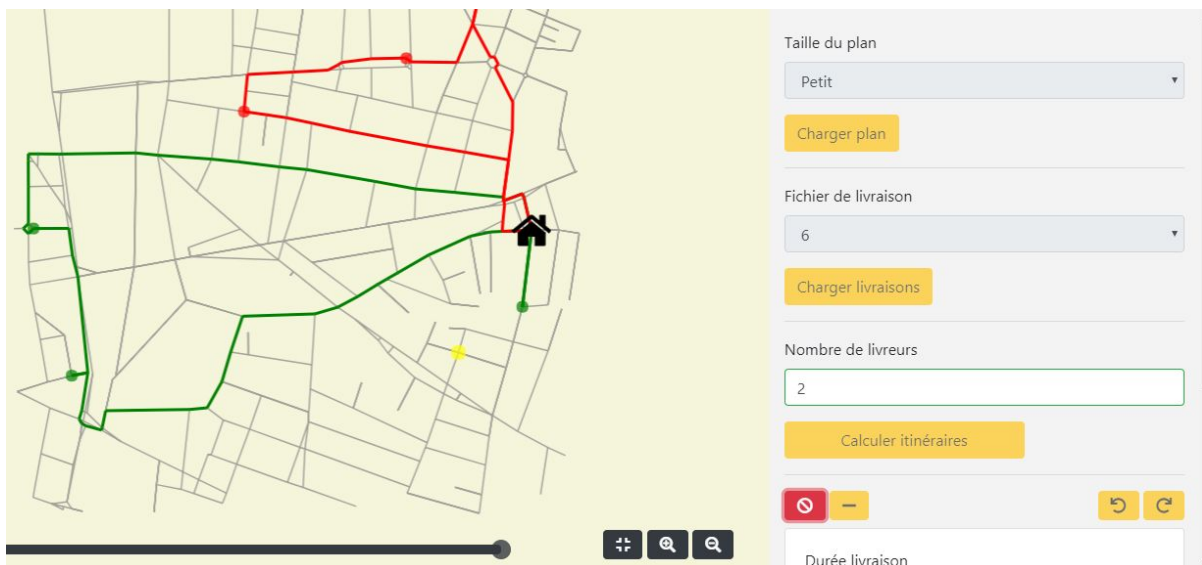


- Slider



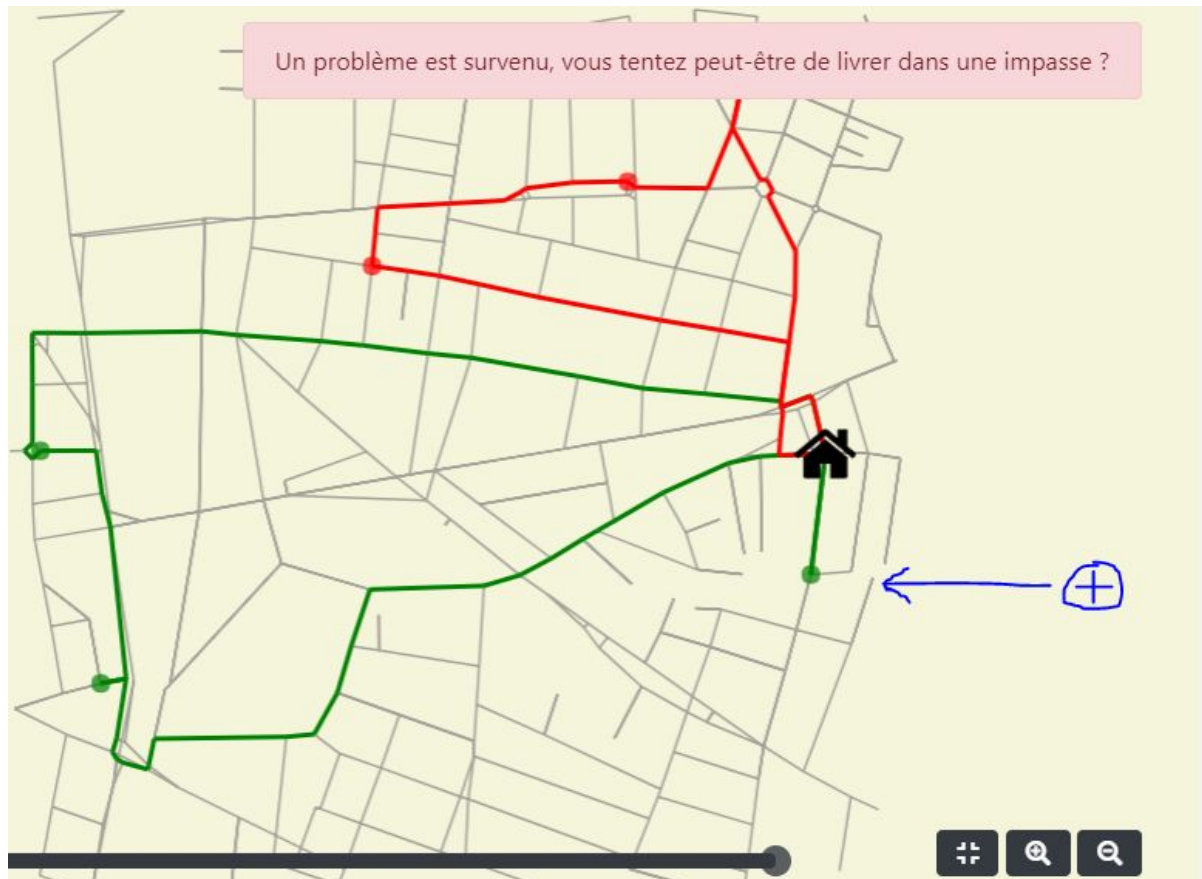
## Ajout d'un point

- 1er ajout



- Ajout supplémentaire : recalcul des trajets si le livreur n'est pas parti
- Ajout dans une impasse





- Ajout faisant dépasser l'heure de fin de journée



## Suppression d'un point

- Choix de garder ou changer trajet



## Undo/redo

