

ChronoPostIF - H4301

I. Sommaire

I. Sommaire	1
II. Outils utilisés	2
III. Choix architecturaux et design patterns utilisés	2
III.1. Présentation de l'architecture	3
III.2. Détails sur nos choix de conception	4
III.3. Designs patterns utilisés	4
III.4. Détails sur le TSP	4
IV. Diagramme de classes/packages global	8
V. Diagramme de classes/packages complet	8
VI. Rapport sur la couverture des tests	9
VII. Captures d'écran de l'application	9
VIII. Fonctionnalités non implémentées et améliorations possibles	11
VIII.1. Fonctionnalités non implémentées	11
VIII.1.a. Consulter les moments importants d'une tournée	11
VIII.1.b. Modifier le nombre de livreurs : ajouter/supprimer un livreur	11
VIII.1.c. Ajouter une livraison à un programme de Pickup&Delivery	11
VIII.1.d. Supprimer une livraison d'un programme de Pickup&Delivery ou d'une tournée	11
VIII.1.e. Charger des tournées à partir d'un fichier	11
VIII.1.f. Changer l'ordre de passage aux points	12
VIII.2. Autres améliorations possibles	12
VIII.2.a. Exportation des tournées	12
VIII.2.b. Comparer nos résultats de TSP avec d'autres groupes pour voir si on a un bon équilibre entre optimalité et temps de calcul	12
IX. Bilan personnel et humain	13
IX.1. Bilan technique	13
IX.2. Bilan organisationnel et humain	13
IX.3. Retour sur les itérations	14
IX.4. Charges par personne	14
IX.5. Planning prévisionnel et définitif	15

II. Outils utilisés

Type d'outil	Nom de l'outil
IDE	VSCode
API & Tests	PostMan

III. Choix architecturaux et design patterns utilisés

III.1. Présentation de l'architecture

Notre projet est une application Web qui se base sur une architecture MVC à 5 couches :

- Vue
- Contrôleur
- Services
- Modèle
- Données

Nous utilisons {REACT + Next.js} pour pouvoir communiquer avec le Backend développé en Java. {REACT + Next.js} se comporte aussi comme contrôleur, requérant le Backend et changeant le contenu de la VUE. Nous avons également, du côté du Backend, un contrôleur (en réalité il y en a plusieurs) qui permet de faire le lien entre le contrôleur FRONT et le Backend.

Vue

REACT assimile le rôle de Vue avec une structure modulaire permettant d'ajouter des modules et fonctionnalités facilement et rapidement. La vue communique avec le contrôleur Backend qui lui renvoie ce dont elle a besoin.

Contrôleur

Le contrôleur Backend mappe les requêtes et interroge les services associés pour réaliser la logique métier. Il renvoie ensuite les résultats des services au contrôleur FRONT pour l'affichage. Il existe en réalité plusieurs contrôleurs, un par classe de service.

Services

Le contrôleur communique avec le Modèle par l'intermédiaire des classes de Service qui offrent les fonctionnalités nécessaires.

Les classes de Service se chargent de réaliser des traitements sur les données (logique métier). Il existe une classe de Service par fonctionnalité majeure.

Modèle

Ici sont définies toutes les classes représentant les données, les classes DTO et les outils. Il y a également une classe Data qui contient les données chargées actuellement pour le traitement.

Données

Les données brutes, notamment celles concernant les livraisons et le plan de la ville, sont stockées sous format XML et seront lues par les classes services pour constituer le jeu de données (Data).

III.2. Détails sur nos choix de conception

Classe Data

Afin de stocker les données pertinentes (plan de livraison, carte, entrepôt...) et pouvoir y accéder sur n'importe quel fichier du Backend, nous avons décidé de créer la classe Data. C'est une classe à attributs statiques:

- **planVille :**

C'est un plan chargé à partir d'un fichier XML avec tous ses segments (routes avec un nom, une longueur et les IDs des points aux extrémités) et ses points qui connectent les segments (avec des coordonnées un ID et un type). Le type d'un point indique son rôle: un entrepôt, un lieu de livraison ou destination ou une simple intersection de segments.

- **livraisonsDues :**

La liste des livraisons à réaliser. Chaque livraison est décrite par la durée d'enlèvement et de livraison ainsi que les points correspondants (leurs ids).

- **tourneesPrevues :**

La liste des tournées réalisées par les livreurs. Une tournée contient les livraisons à réaliser par le livreur et le circuit de segments optimal à prendre.

- **idEntrepot :**

L'id du point qui correspond à l'entrepôt de la carte pour le plan de livraisons chargé.

DTOs: Data Transfer Objects

Avant d'envoyer n'importe quel objet du Backend vers le Frontend, nous le transformons d'abord en un objet DTO (exemple: PlanDTO lors de l'envoi de la carte chargée).

Ces classes sont sérialisables et peuvent donc être transformées en JSON directement sans action supplémentaire de notre part lors de l'envoi.

Gestion des imports et des modifications de livraisons

Lors de l'import d'une nouvelle carte, nous ne pouvons pas être sûrs de la cohérence des anciennes positions avec les positions de la nouvelle carte et nous avons donc décidé de supprimer toutes les livraisons présentes dans l'application.

III.3. Designs patterns utilisés

Factory

Certains objets sont générés à partir du parsing d'un fichier XML. Le résultat de ce parsing est un objet JSON avec une structure différente selon le format du fichier lu. Leur création étant plus complexe que pour d'autres classes et pour prendre en compte le format du fichier XML, nous avons décidé de créer des factories (cas des livraisons et du plan).

III.4. Détails sur le TSP

nb = nombre

Définitions

- Nous avons une carte importée qui possède ($n: nb \text{ noeuds}$, $s: nb \text{ segments}$) .
- Nous avons une demande de livraisons importée qui possède
 - $p: nb \text{ de points de pickup et delivery}$.
 - $q: nb \text{ de parcours de noeuds voisins}$, $q < p$
- L'heuristique qui sert de comparateur est la distance à vol d'oiseau (Haversine) $O(1)$ entre un point et l'objectif.
- $d_{\text{vol d'oiseau}} < d_{A^*}$
- Nous utilisons une Priority Queue pour trier les noeuds ouverts du A*:
 - Opérations d'insertion, de mise à jour et d'extraction en $O(\log(n))$
 - Donc A* en $O((n + s)\log(n))$
- Pour les noeuds à visiter (des livraisons) nous utilisons une liste triée en $O(p \log(p))$

Hypothèses

- H1: Les points de livraisons sont parmi les points de la carte, carte assez grande, $p \ll n$
- H2: Les points de livraison sont souvent éloignés: $q \ll p$.

TSP glouton (Nearest Neighbours)

Nous avons fait le choix de prioriser le temps d'exécution sur l'optimalité du parcours. Nous avons décidé d'approcher ce problème avec un algorithme glouton (plus proches voisins).

a) PseudoCode

Fonction trouver_circuit(tournee) :

```

pointActuel = entrepôt
ouverts = {points de pickup des livraisons de tournee};
circuit = {};
faire tant que ouverts not vide:
| //trouver le plus proche voisin
| minCout = +∞;
| meilleurResultat = null
| //On parcourt les voisins par ordre de distance d'oiseau vers pointActuel
| ouverts.ordonner()
Pour chaque pointPossible dans ouverts:
| Si ( DistanceHaversinef(pointActuel, pointPossible) >= minCout ) break;
| // On lance le A* et on regarde s'il minimise l'actuel.
| resultat = runWA(pointActuel, pointPossible, 1.0)
| Si (resultat.trouvé && resultat.longueur < minCout ):
|   minCout = resultat.longueur
|   meilleurResultat = resultat
si meilleurResultat.trouvé :
| // Ajouter circuit du meilleur resultat entre pointActuel et voisin
| circuit += meilleurResultat.circuit;

| Si (minPoint un pickup) ouverts.AjouterDeliveriesAssociées(minPoint);

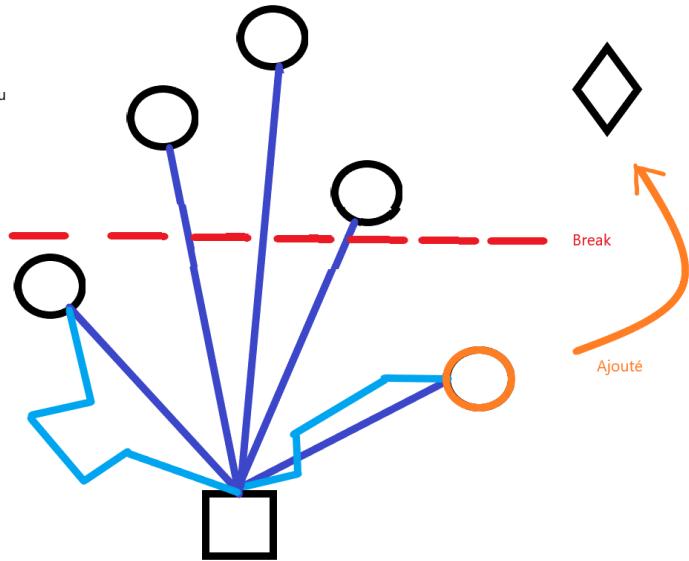
| // Preparer pour trouver le prochain voisin
| pointActuel = minPoint;
| \_ ouverts.enlever(minPoint)
sinon:
| \_ erreur("Pas de résultat trouvé"

\_ // retourner vers l'entrepôt
circuit += runWA(pointActuel, entrepôt, 1.0).circuit;
return circuit;
```

b) détails

Itération n:

- Circuit de A*
- Distance au vol d'oiseau
- Pickup
- Entrepôt
- ◆ Delivery



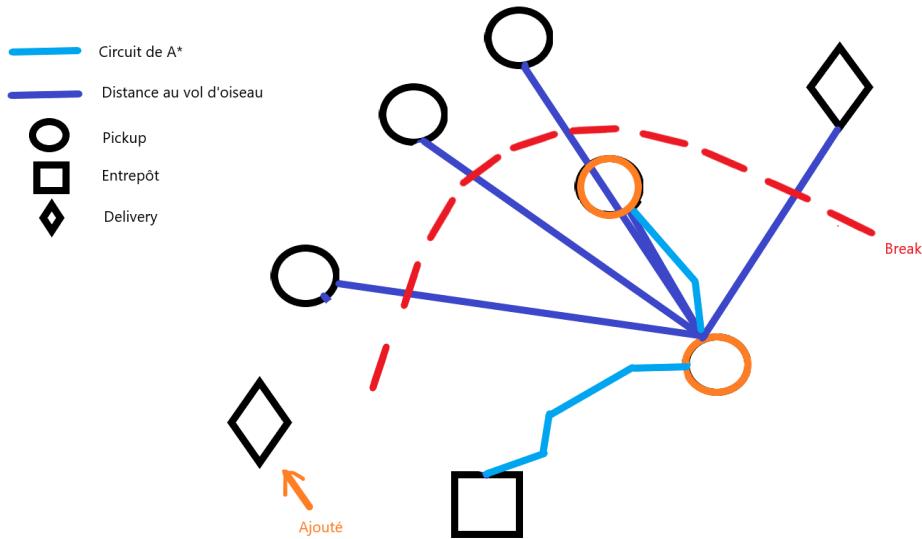
- Nous calculons la distance à vol d'oiseau (en bleu foncé) du pointActuel (ici l'entrepôt) vers les autres points et nous les trions par distance: $O(plog(p))$
- Nous calculons la distance réelle avec A* entre le pointActuel et chaque pointPossible (en bleu clair). On fait cela pour chaque point dans l'ordre de proximité tant que la distance à vol d'oiseau de ce point est inférieure à la distance minimale trouvée (sinon on arrête de parcourir: break).

$$O(q * O(A)) = O(q * (n + s)log(n))$$

- Nous prenons le voisin le plus proche et s'il s'agit d'un pickup, nous ajoutons donc ses deliveries:

Ajout en $O(1)$ car ArrayList.

Au total: $O(plog(p)) + q(n + s)log(n)$

Itération n+1:

Pour l'itération suivante nous répétons la même démarche mais depuis le point voisin le plus proche trouvé à l'itération précédente. Nous continuons jusqu'à ce qu'il n'y ait plus de points à visiter puis nous retournons vers l'entrepôt.

Au total: $O(p * O(\text{iter } n)) = O(p(p \log(p) + q(n + s) \log(n)))$

Conclusion**a) Complexité**

La complexité de cet algorithme est $O(p(p \log(p) + q(n + s) \log(n)))$
avec

- n : nb noeuds .
- s : nb segments
- q : nb de parcours de noeuds voisins
- p : nb de points de pickup et delivery

Or d'après l'hypothèse H1, $p \ll n$ donc

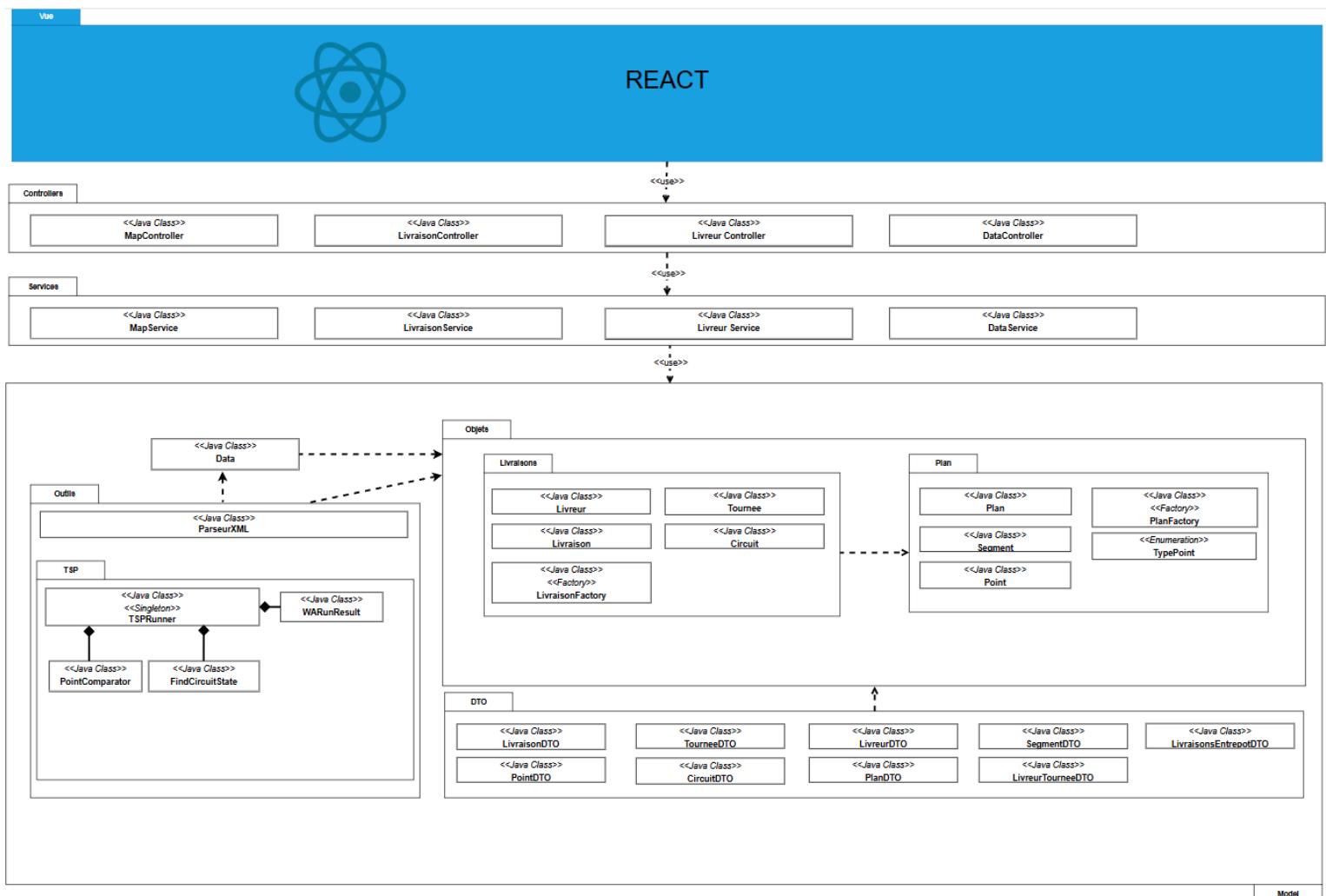
$$O(p(p \log(p) + q(n + s) \log(n))) \approx O(pq(n + s) \log(n))$$

et d'après l'hypothèse H2, $q \ll p$ donc

$$O(p(p \log(p) + q(n + s) \log(n)))_{\text{en moyenne}} \approx O(p(n + s) \log(n))$$

Cas	Complexité
TSP glouton pire des cas	$O(p^2(n + s) \log(n))$
TSP glouton en moyenne	$O(p(n + s) \log(n))$
Dijkstra (sans prendre en compte la contrainte de pickup et delivery qui peut ajouter un facteur p supplémentaire)	$O(p^2(n + s) \log(n))$

IV. Diagramme de classes/packages global



V. Diagramme de classes/packages complet

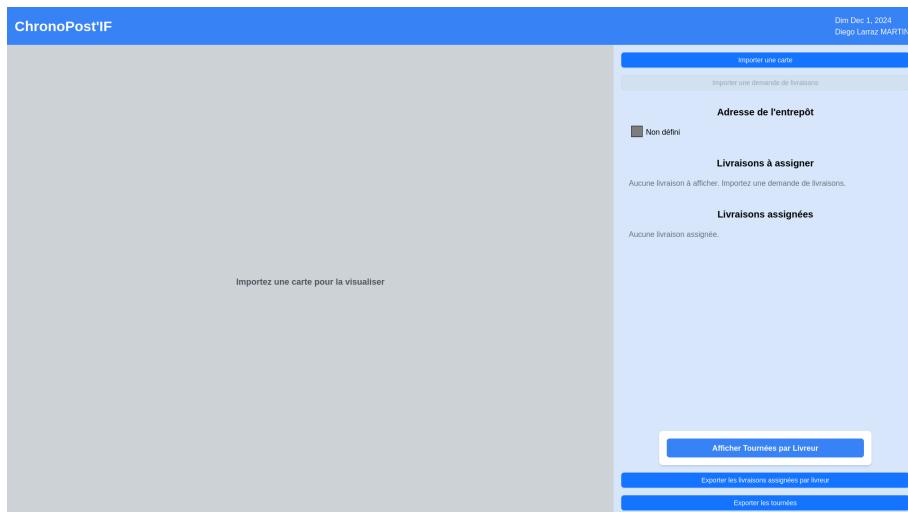
Pour ne pas avoir une version de mauvaise qualité, nous avons décidé de ne pas mettre le diagramme de classes complet sur ce document. Mais, il est disponible sur le github au format svg à l'emplacement : *Documents/Package_And_Class_Diagrams/ClassDiagram.drawio.svg*

VI. Rapport sur la couverture des tests

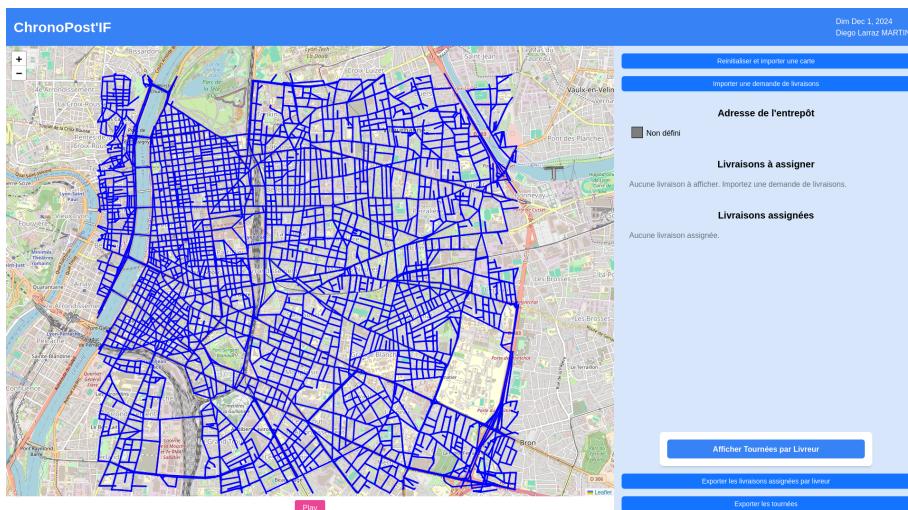
Pour consulter la description des tests que nous avons réalisés, veuillez vous reporter aux fichiers *Description_Unit_Tests.md* présents dans les répertoires enfants de *Documents/Tests*.

VII. Captures d'écran de l'application

Lors de la première visite sur le site, aucune map n'est chargée et la seule action possible est de charger une carte à partir d'un fichier grâce au bouton "Importer une carte"

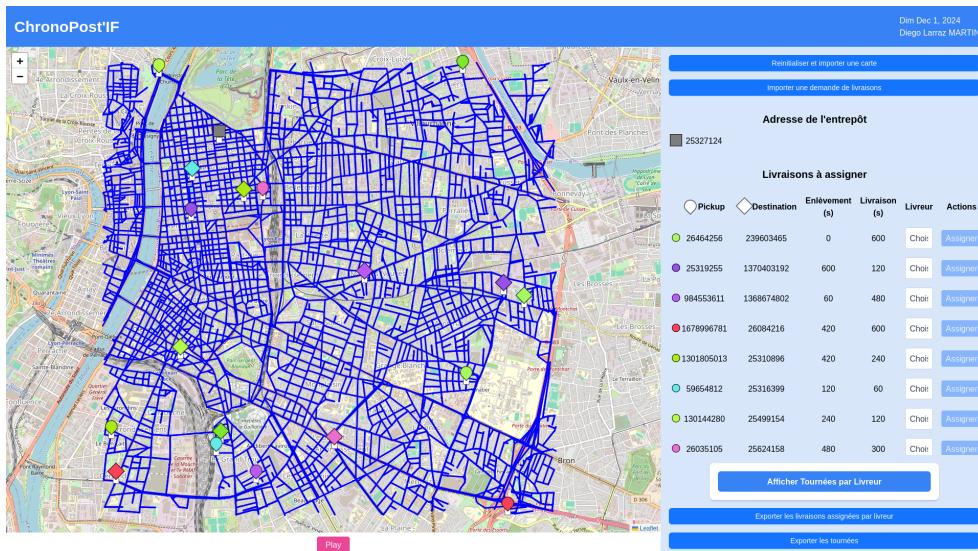


Une fois la carte chargée (cela peut prendre plusieurs secondes pour des grandes cartes), cette dernière s'affiche dans la partie gauche du site et le bouton d'import de livraisons devient accessible.



La position de l'entrepôt et des livraisons importées s'affichent sur la carte et dans le menu de droite. Chaque élément a une couleur et forme distincte visible dans le menu de l'application.

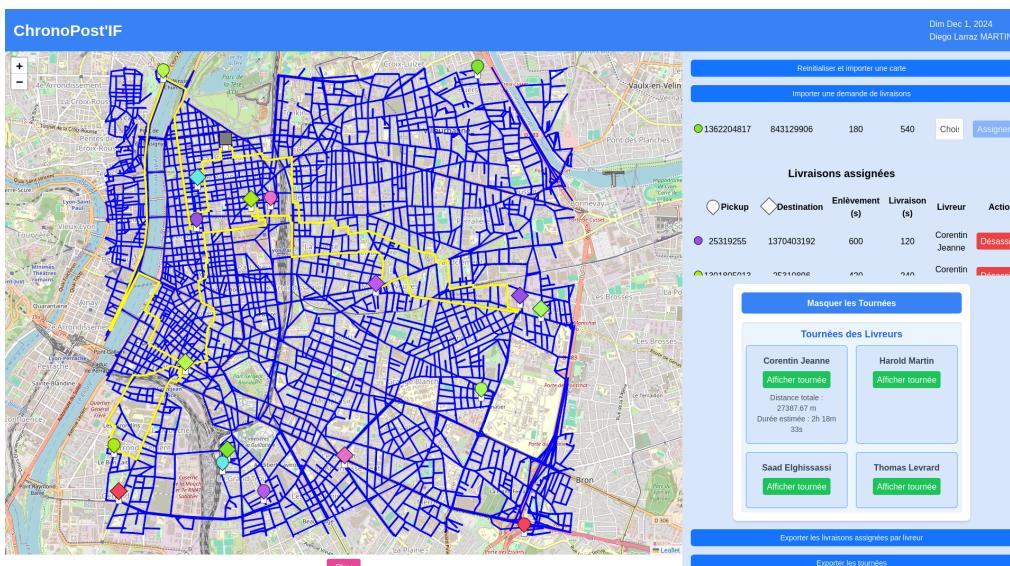
Il est possible de déplacer les points de pickup ou delivery qui ne sont pas assignés en les faisant glisser avec la souris.



Le choix des livreurs se fait avec un drop-down. Après avoir choisi un livreur pour une livraison, il suffit de valider l'assignation en cliquant sur le bouton “Assigner”. Ces livraisons sont listées dans la partie basse du menu et on peut utiliser le bouton “Désassigner” pour les retirer de la tournée du livreur.

Une fois les livraisons assignées, il suffit de cliquer sur le bouton “Afficher tournée” pour calculer et afficher la tournée d'un livreur (circuit en jaune sur la carte). De plus, la distance et le temps pour chaque livreur est indiqué en dessous de son nom et le circuit en jaune sur la carte.

Le bouton play permet de simuler la progression du livreur sur le circuit.



Enfin, il est possible d'exporter (au format .json) les tournées qu'on vient de calculer en cliquant sur le bouton “Exporter les tournées” en bas à droite de la page. Si on avait eu plus de temps, on aurait fait en sorte que l'on puisse exporter en format .xml au lieu de .json.

VIII. Fonctionnalités non implémentées et améliorations possibles

VIII.1. Fonctionnalités non implémentées

VIII.1.a. *Consulter les moments importants d'une tournée*

L'utilisateur peut consulter l'heure de début (8h = départ de l'entrepôt) et l'heure de fin de la tournée (retour à l'entrepôt). L'utilisateur peut également cliquer sur un point d'enlèvement ou de livraison en particulier afin de consulter l'heure d'arrivée ou de départ du livreur à/de celui-ci. Cela permet notamment de prévenir le client.

VIII.1.b. *Modifier le nombre de livreurs : ajouter/supprimer un livreur*

L'utilisateur peut modifier le nombre de livreurs en ajoutant ou en supprimant un livreur déjà existant. Pour créer un nouveau livreur, il suffit de saisir un nom et un prénom de livreur puis de valider

VIII.1.c. *Ajouter une livraison à un programme de Pickup&Delivery*

L'utilisateur vient de recevoir un appel d'un client qui souhaite être livré demain. Il décide alors de rajouter la livraison au programme de Pickup&Delivery. Il choisit d'abord le point d'enlèvement en cliquant sur un point de la carte de la ville. Puis, de la même manière, il choisit le point de livraison. Enfin, il doit saisir une durée d'enlèvement et une durée de livraison.

VIII.1.d. *Supprimer une livraison d'un programme de Pickup&Delivery ou d'une tournée*

L'utilisateur vient de recevoir un appel d'un client qui lui indique qu'il ne sera pas présent demain pour réceptionner son colis, le client lui indique alors le point de livraison correspondant. L'utilisateur décide alors de supprimer la livraison. Pour cela, il commence par rechercher la livraison sur la carte ou sur la liste déroulante des livraisons puis il clique sur le point de livraison correspondant à la livraison et il indique qu'il veut supprimer la livraison. Cela supprime cette livraison de la carte et de la liste des livraisons. De plus, si cette livraison était déjà assignée à un livreur, sa tournée est automatiquement recalculée.

VIII.1.e. *Charger des tournées à partir d'un fichier*

L'utilisateur décide de charger des tournées (préalablement calculées) à partir d'un fichier xml. Pour cela, il clique simplement sur un bouton "charger des tournées", l'application lui demande alors de sélectionner un fichier xml contenant ces tournées. Une fois le fichier chargé, il est possible de consulter ces tournées.

VIII.1.f. *Changer l'ordre de passage aux points*

L'utilisateur vient de recevoir un appel d'un client qui lui indique qu'il doit impérativement être livré avant 10h. L'utilisateur clique alors sur la livraison correspondante et indique qu'elle doit être effectuée avant 10h. Cela recalcule donc la tournée concernée (s'il y en a une).

VIII.2. Autres améliorations possibles

VIII.2.a. *Exportation des tournées*

Actuellement, il est possible d'exporter en format .json les tournées qu'on vient de calculer. Si on avait eu plus de temps, on aurait fait en sorte que l'on puisse exporter en format .xml au lieu de .json.

VIII.2.b. *Comparer nos résultats de TSP avec d'autres groupes pour voir si on a un bon équilibre entre optimalité et temps de calcul*

Nous n'avons pas pu comparer nos résultats de TSP avec tous les groupes pour savoir si on avait un bon équilibre entre temps de calcul et optimalité des tournées obtenues. Cela aurait pu être très intéressant. Nous avons seulement pu comparer le temps d'exécution avec l'hexanome 3 qui nous a dit que leur TSP sur le grand plan avec la grande demande prenait 14 secondes à s'exécuter, ce qui est bien plus élevé que pour nous (presque instantané). Mais, ils n'avaient pas implémenté le calcul de la distance ou de la durée d'une tournée donc on ne peut pas vraiment comparer les qualités des tournées obtenues.

IX. Bilan personnel et humain

IX.1. Bilan technique

Nous avons décidé d'utiliser ReactJS pour le front et SprintBoot pour le Backend. Or, seulement un membre de notre groupe (Corentin) connaissait ces 2 frameworks/bibliothèques avant de commencer le projet. Ainsi, nous avons dû nous former (à la fois en cours mais également chez nous), ce qui a entraîné un peu de retard au début du projet car la majorité d'entre nous ne maîtrisait pas bien ces 2 outils. Cependant, ce projet nous a permis d'acquérir de nouvelles connaissances et compétences sur ces deux outils largement utilisés, ce qui nous sera assurément utile à l'avenir.

IX.2. Bilan organisationnel et humain

Ce projet était, pour la plupart d'entre nous, notre premier projet de développement où nous adoptions une méthode Agile. Voici une partie de nos ressentis :

- Nous avons particulièrement apprécié qu'à chaque itération, nous pouvions alterner entre conception/spécification, développement et tests. L'approche agile permet de diversifier les types de tâches réalisées à chaque itération, ce qui permet de moins s'ennuyer. Alors qu'avec une méthode traditionnelle, comme celle utilisée lors du projet UML de l'année dernière, on doit suivre un déroulement séquentiel : analyse métier, puis conception/spécification, ensuite développement, et enfin tests. Ce mode de fonctionnement traditionnel peut vite devenir monotone et est donc plus susceptible d'être ennuyeux.
- Nous avons également beaucoup apprécié les échanges et validations réguliers avec le/la Product Owner, car cela garantit une meilleure satisfaction des besoins clients. Cette approche permet donc d'éviter de développer des fonctionnalités inutiles ou de les concevoir de manière inappropriée. Cela est bénéfique à la fois pour le client et pour nous, car il est rarement plaisant d'avoir travaillé sur quelque chose d'inutile ou qui devra être refait. Même si, nous aurions dû davantage communiquer avec la PO, notamment concernant les choix de design liés à l'UX/UI (cf. Démo).
- Il était également très appréciable que toute l'équipe participe à la réalisation du planning au début de chaque itération car ça nous permettait de pouvoir choisir ce qu'on voulait faire en fonction de ce qu'on préférait et de là où on était le plus à l'aise. Ainsi, ça a permis d'augmenter notre efficacité. En effet, on travaille évidemment mieux et plus efficacement sur des choses que l'on aime faire.
- Nous avons parfois eu du mal à bien estimer la charge de travail associée aux différentes tâches. D'ailleurs, on le voit bien dans le planning définitif.
- Un membre de notre groupe (Thomas) a été absent pendant 3 séances, nous avons ainsi revu nos objectifs à la baisse mais nous aurions dû le faire davantage au lieu de travailler autant en dehors des séances.

IX.3. Retour sur les itérations

Itération 1

Nous avions surestimé la quantité de travail que nous pouvions achever d'ici la fin de cette itération. En particulier, nous n'avions pas anticipé les nombreuses difficultés et erreurs techniques rencontrées, notamment en raison des différentes versions de Java et des problèmes liés à npm sur Windows, qui ont compliqué l'initialisation de l'application web. De plus, nous n'avions pas prévu le temps de formation à React et SpringBoot. Même si on a quand même réussi à bien définir le sujet et à avoir une application web fonctionnelle pour la suite.

Itération 2

En prévoyant moins de charge, en étant plus coordonnés et en travaillant en dehors des séances, nous avons réussi à atteindre nos objectifs initiaux malgré l'absence de Thomas pour maladie (2 séances). Nous aurions pu revoir encore plus à la baisse nos objectifs au lieu de travailler autant en dehors des séances.

Itération 3

En prévoyant moins de charge, en étant plus coordonnés et en travaillant en dehors des séances, nous avons réussi à atteindre nos objectifs initiaux malgré l'absence de Thomas pour maladie (1 séance). Nous aurions pu revoir encore plus à la baisse nos objectifs au lieu de travailler autant en dehors des séances.

Itération 4

C'est pendant cette itération que nous avons réussi à avoir la meilleure coordination et répartition du travail. Nous avons notamment pu corriger de nombreux bugs.

IX.4. Charges par personne

Le détail des charges par personne est disponible sur le document suivant :

[https://docs.google.com/spreadsheets/d/1tUn0De-s_krnqOxDPwklHsoTilyq7OotppOI_YfjBrg/edit?
gid=2019409002#gid=2019409002](https://docs.google.com/spreadsheets/d/1tUn0De-s_krnqOxDPwklHsoTilyq7OotppOI_YfjBrg/edit?gid=2019409002#gid=2019409002)

IX.5. Planning prévisionnel et définitif

Itération	Tâche	Charge estimée	Charge réelle
Prévues	Create REACT Project	10p	25p
	Design document	1p	1p
	Create Java Project and dependancies	10p	10p
	Parseur XML	10p	10p
	Parsed XML to Java classes into Data(plan)	5p	5p
	Use case diagramm	10p	10p
	Use case descriptions, persona	10p	10p
	Glossary	1p	1p
	Créer les éléments pour afficher la carte	10p	25p
	Afficher la carte	10p	5p (5p restants)
Imprévues	Conception IHM	-	5p
	Total	72	107
Prévues	Pouvoir choisir la carte à afficher xml	10p	10p
	Unit Tests initiaux	10p	10p
	Afficher la carte	5p	5p
	Formation rapide REACT	10p	10p
	afficher livraisons	5p	10p
	parsed XML to Java classes into Data(demandes)	5p	5p
	meilleur chemin service: java lib fournie	5p	10p
	Elements REACT pour charger les deliveries	10p	10p
	IHM	10p	5p
	merge du chargement de la carte	-	5p
2	Total	70	80
	Concevoir les livreurs sur l'architecture	5p	5p
	Unit Tests TSP et Map	5p	-
	TSP avec pick n Delivery	10p	40p
	Mise en place des tournées	25p	25p
Prévues	Afficher une tournée sélectionnée	10p	25p
	Total	55	95
	Rapport	25p	25p
	Corriger diagramme de classes	5p	5p
	Corriger Front	15p	25p
3	To Do	10p	10p
	Documentation	15 p	15p
	Total	70	80
	Total	267	362
End			