# Visualisateur de réseau GTFS

Le visualisateur de réseau a d’abord été conçu pour permettre l’exportation de réseau GTFS (General Transit Feed Specification) vers le format Madigas dans le cadre de l’enquête Origine-Destination prévue pour l’automne 2018. Depuis, l’interface et la structure de base de données ont été recyclées pour servir de visualisateur de réseau. Le visualisateur utilise une approche par tracé, c’est-à-dire que tous les départs sont ramenés à des tracés communs à des fins de représentation géographique et de représentation des horaires. Suite au déploiement progressif du système d’information aux usagers en temps réel Chrono, il est aussi devenu nécessaire d’effectuer un suivi des données publiées sous le format GTFS-Realtime, qui s’arrime lui-même sur les données GTFS publiées par le RTL. Une seconde interface de visualisation permettant à la fois d’explorer les données publiques en temps réels et archivées s’est donc elle aussi greffée à l’application développée.

L’ensemble de l’application est basée sur la plateforme de développement Ruby on Rails et sur les bases de données PostgreSQL/PostGIS, alors que l’interface usager fait usage du langage de script par défaut des navigateurs web, soit javascript. Les librairies permettant de produire les visualisations web sont principalement jQuery, une librairie permettant de simplifier la programmation javascript, ainsi que MapboxGL, une librairie permettant d’afficher des données sur des fonds de cartes avec de très bonnes performances, le rendu étant effectué par la carte graphique du poste où la visualisation s’effectue.

## Visualisateur statique

Le visualisateur de données statiques est somme toute assez simple : les différentes lignes sont présentées à l’utilisateur, qui peut ensuite sélectionner une ligne et un tracé, et voir les horaires liés au tracé en question. Deux tracés peuvent être visualisés de façon simultanée à des fins comparatives, et une section à la droite de l’interface permet de visualiser l’ensemble des tracés liés à une ligne en particulier afin de simplifier l’exploration des données. Finalement, une dernière fonction permet de tracer une ligne géographique à n’importe quel endroit sur la carte et d’obtenir en retour le nombre de départs croisant la forme dessinée, et ainsi faire l’analyse de certains corridors de façon rapide.

L’interface se veut économe en termes de consommation de ressources réseau, celle-ci recevant les tracés de chacune des lignes uniquement au changement de lignes, alors que l’ensemble des tracés pour chaque ligne à des fins d’exploration est stocké immédiatement à l’ouverture de la page. La construction des tables d’intersection demande quant à elle un appel du serveur à chaque dessin, le retour étant toutefois limité en taille (en général moins de 100ko), même si la réponse prend quelque secondes à produire.

Finalement, le visualisateur statique permet d’autres opérations du côté serveur (backend), notamment l’exportation des tracés dans un format approprié à la création de cartes animées sous la plate-forme deck.gl. Cette dernières opération n’est toutefois pas exécutée de façon dynamique, le temps de traitement étant plus important, alors que le flux de données se révèle de son côté plus imposant (~90Mo pour les données d’un jour moyen de service du RTL).

## Visualisateur temps réel

La partie temps réel de l’application se décline en deux interfaces semblables mais distinctes. La première ne fait que récupérer les dernières informations disponibles, alors que la seconde permet d’explorer les données archivées historiques. L’interface présentant les dernières données permet en outre de générer des liens vers la carte historique, permettant ainsi de stocker la visualisation d’un événement spécifique à des fins de partage. Dans les deux cas, les mêmes informations sont fournies pour l’ensemble des véhicules disponibles, soit la position du véhicule, les 3 dernière informations connues pour le véhicule (ce qui permet de montrer une traînée illustrant le déplacement du véhicule) ainsi que les données opérationnelles du véhicule (vitesse, direction, voyage effectué, etc.). L’arrimage se faisant avec les données GTFS déjà stockées pour le visualisateur statique, différentes interactions sont aussi programmées, permettant entre autres de suivre un véhicule lorsque celui-ci est sélectionné via un clic, de montrer le tracé prévu du véhicule selon l’horaire, ainsi que la plage horaire du départ sur lequel celui-ci est affecté.

## Processus en arrière-plan

Afin d’alimenter les diverses interfaces du visualisateur, un certain nombre de processus en arrière-plan doivent être exécuté. Premièrement, l’application est tributaire des données publiques en format GTFS, ce qui implique que les différentes assignations doivent être chargées périodiquement. À l’heure actuelle, l’application ne peut supporter qu’une seule assignation, ainsi qu’un seul type de service. Cette situation pourrait changer et est essentiellement due aux racines de l’application qui visait originalement uniquement à faire l’export des données GTFS vers le format Madigas, qui dépend uniquement des données de semaine.

En outre, l’importation des données GTFS s’accompagne elle-même de quelques autres processus. Le premier consiste en la définition de deux surcouches, soit une couche de tracés (traces) ainsi qu’une de tracé\_arrêt (trace\_stops). Ces deux couches permettent le pré-stockage des informations du GTFS pour un accès plus rapide dans l’interface, notamment la géographie dans le cas des tracés, ainsi que les heures de passages des arrêts dans stop\_traces. Encore ici subsistent quelques artéfacts des racines de l’application comme la présence de champs permettant d’associer un arrêt à un jumeau et de définir un centroïde pour chacune de ces paires d’arrêts, processus essentiel à la création d’un réseau Madigas. De façon générale, le reste de la structure de la base de données respecte les grandes lignes du format GTFS.

Les données GTFS telles que produites par le RTL divisant les boucles en deux départs séparés, les tracés permettent aussi de rassembler ces deux séquences en une seule à des fins d’affichage. Il suffit dans ces cas de faire le joint en utilisant la variable block\_id dans le fichier trips du GTFS. Ce processus, ainsi que la création des surcouche, sont des processus automatisés lors de l’importation des données et ne doivent par conséquent être exécutés qu’une seule fois.

Finalement, un dernier processus en continu d’acquisition des données en temps réel doit être fait. Les fichiers GTFS-R étant produit par le RTM, il suffit, à intervalles réguliers, de faire une requête au serveur pour obtenir les données les plus récentes. En l’absence d’un scheduler efficace sous Windows permettant d’exécuter des utilitaires de ligne de commande, une simple boucle dans une console rails permet d’exécuter une commande spécifique suivie d’un temps de pause.

## Schéma

Schéma de DB ici!

