Automatisation de l'importation de données ferroviaires à destination d'OpenRails

Johan Brich

Docteur Laurent PHILIPPE

Master 1 en Architecture des Systèmes Informatiques

Année académique 2023-2024

Sommaire

1 Introduction

Ce rapport a été réalisé dans le cadre du cours : « Projets Avancés et Innovation ». Il vise à présenter une solution technique destinée à faciliter la création de cartes pour le simulateur de train OpenRails. Il s'agit d'un simulateur de train open-source lancé en 2009 par une communauté de passionnés. Ce projet est né de la volonté de remplacer le Microsoft Train Simulator dont le développement avait été abandonné par Microsoft en 2004. Grâce à ses capacités avancées, sa flexibilité et son engagement à fournir une simulation réaliste, OpenRails a rapidement gagné en popularité au sein de la communauté des férus de simulation ferroviaire.

La structure de ce rapport suit une logique progressive, retraçant d'abord les recherches et les analyses qui ont été nécessaires pour définir les besoins et les solutions possibles. Ensuite, il décrit les différentes étapes de développement et d'implémentation de l'application. Le rapport se conclut par une réflexion sur les résultats obtenus et les perspectives d'amélioration pour le futur.

2 Objectif du projet

Malgré les nombreux avantages qu'offre OpenRails, la création de nouvelles cartes demeure une tâche complexe et exigeante. Cette complexité réside non seulement dans la manipulation précise des données géographiques et topographiques, mais aussi dans la collecte de ces données, qui peut être une tâche fastidieuse et chronophage. Les informations précises nécessaires pour construire des itinéraires ferroviaires réalistes ne sont pas toujours facilement accessibles, ce qui ajoute une couche supplémentaire de difficulté au processus.

Le Train Simulator Route Explorer (TSRE), bien qu'efficace pour l'intégration de ces données, requiert une précision minutieuse, notamment lors du positionnement manuel des rails et des éléments de décor. Ce processus peut être laborieux et difficile, particulièrement pour les utilisateurs moins expérimentés.

L'objectif de ce projet est donc de concevoir une application capable de simplifier et d'automatiser non seulement certaines des étapes les plus ardues du processus de création de cartes pour OpenRails, mais aussi la collecte et l'intégration des données nécessaires. Cette application permettra aux utilisateurs de sélectionner facilement une zone géographique, d'importer les données requises, puis de les intégrer directement dans TSRE, tout en offrant une interface intuitive qui réduira les risques d'erreurs et accélérera le processus de création.

3 Étapes & Méthodologie

Cette section aborde les différentes étapes qui ont été nécessaires à la réalisation du projet. Avant d'automatiser le processus et de créer l'application finale, il faut d'abord analyser et créer ce processus. Cette analyse a été planifiée et décomposée en différentes étapes afin qu'elle soit cohérente pour que le temps passé sur celle-ci soit optimisé.

Cette analyse a donc été décomposée en 4 étapes majeures : l'obtention des données, le traitement des données, l'importation de ces données dans le simulateur OpenRails et, pour finir, l'automatisation du processus avec la création de l'application.

Concernant la méthodologie, pour chaque étape, il y aura une phase de recherche et d'analyse, une phase sur le fonctionnement et la manière de procéder pour, finalement, valider l'étape par des tests si besoin. Chaque étape devra être validée avant de passer à la suivante.

3.1 Obtention des données

L'étape d'obtention des données concerne la recherche et l'analyse qui sera nécessaire pour obtenir les différentes données géographiques utiles à la réalisation d'un projet OpenRails.

3.2 Traitement des données.

Une fois les données obtenues, il faut pouvoir les traiter, les modifier de manière à assurer la compatibilité avec le simulateur.

3.3 Importation des données

Une fois les données obtenues et adaptées, il faut ensuite trouver un moyen de les utiliser en les important dans OpenRails.

3.4 Automatisation du processus

Lorsque les 3 étapes précédentes sont réalisées et validées, il faut automatiser le processus en réunissant ces différentes étapes au sein d'une application.

3.5 Adaptation de l'ordre des étapes

Une fois les différentes étapes définies, il y a plusieurs possibilités quant à la manière de procéder. Le but étant de planifier la recherche et l'analyse de manière à être le plus cohérent et le plus efficace possible. C'est pourquoi, l'ordre des différentes étapes a été adapté. Celui-ci reste cependant globalement similaire car c'est seulement la partie concernant l'importation des données au sein d'OpenRails qui passe en premier dans les priorités. En effet, c'est cette partie qui définira les contraintes du point d'arrivée.

Ordre initial lors de la définition des étapes :

Obtention → Traitement → Importation → Automatisation

Ordre adapté :

Importation → Obtention → Traitement → Automatisation

4 Recherches et analyses

4.1 Importation des données dans OpenRails

La toute première étape concerne l'importation des données dans OpenRails. Cette étape approfondira ce qu'est OpenRails et, notamment, la structure d'un projet OpenRails ainsi qu'un outil permettant la création de cartes et de projets compatibles avec OpenRails.

4.1.1 Premier projet OpenRails

Avant d'essayer d'importer une carte ou des données dans le simulateur, il est important de se renseigner sur les différents moyens d'importation de données dans celui-ci.

La première étape consistait à consulter le site web officiel d'**OpenRails** qui contient plusieurs informations ; notamment du contenu de la communauté avec des cartes et trains disponibles et accessibles à tous. Après avoir téléchargé l'application et les fichiers du modèle de démo, il suffit de renseigner le chemin vers le répertoire du modèle téléchargé et, enfin, de lancer la simulation.

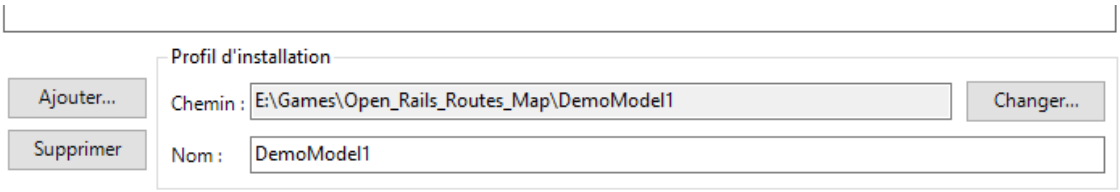


Figure 1 - Lancement projet OpenRails

4.1.1.1 Structure d'un projet OpenRails

Après avoir obtenu la première carte et l'avoir testée, la structure du projet téléchargé a pu être analysée dans le but de la comprendre pour potentiellement pouvoir la reproduire dans une prochaine étape. Ce projet est divisé en plusieurs sous-répertoires, tous contenants une série de fichiers. Parmi ceux-ci, on retrouve des fichiers liés aux emplacement des voies de chemins de fer, des fichiers pour les véhicules (locomotive et wagons), d'autres concernent la topographie, l'élévation du terrain, etc., ainsi que des fichiers de modèles 3D pour tous les objets, qu'ils fassent

partie du décor ou non. Il y a donc énormément de fichiers et de données qui ont tous un but précis.

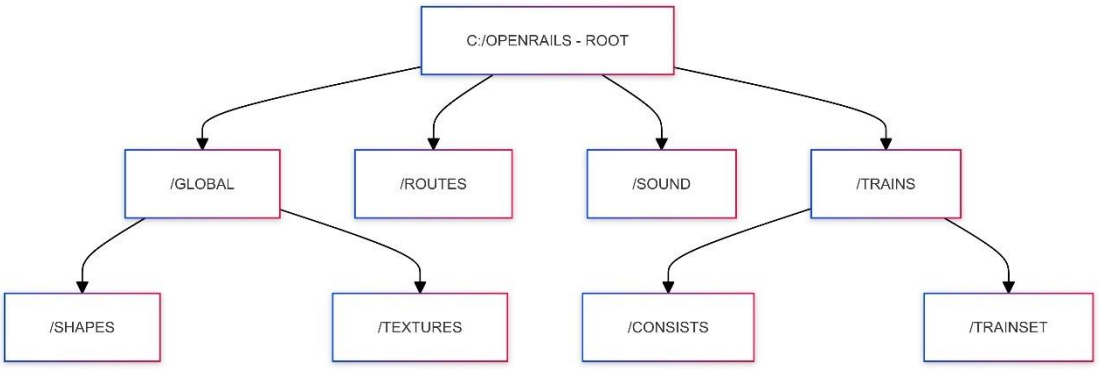


Figure 2 - Structure projet OpenRails

4.1.2 Train Simulator Route Explorer (TSRE)

Á ce stade, après avoir téléchargé le simulateur ainsi qu'un projet de la communauté, il a été possible d'analyser et d'en apprendre plus quant à la structure d'un projet OpenRails. L'étape suivante porte sur la création d'un projet.

Une nouvelle consultation du site d'OpenRails a permis de découvrir une section « *Learn* [1]» où on trouve une page s'intitulant « ***Build your own route*** [2]». Celle-ci renvoie vers un tutoriel qui explique de A à Z comment on peut créer soi-même son réseau ferroviaire pour simulateur, ainsi que tous les éléments autour : décors, terrain, objets, végétation, etc. Jusqu'à l'importation de ce qui a été créé directement dans OpenRails. Toutes ces étapes sont gérées dans une seule et même application : « Train Simulator Route Explorer » souvent abréviée sous l'acronyme TSRE.

TSRE est un outil Open Source et libre d'accès qui permet à la communauté de créer des cartes compatibles avec Open Rails.

[1] Apprendre
[2] Construire sa propre route

4.1.2.1 Mise à jour des étapes

Voici un rappel des différentes étapes qui sont présentes dans l'application finale et ont été définies précédemment :

Obtention des données → Traitement des données → Importation dans OpenRails

Nous sommes, pour l'instant, à l'étape de l'importation des données dans OpenRails. Ces étapes méritent une petite mise à jour ou précision puisque TSRE permet de créer des cartes compatibles avec OpenRails. L'étape d'importation dans OpenRails devient donc Importation dans TSRE.

Obtention des données → Traitement des données → Importation dans TSRE

4.1.2.2 Utilisation de l'outil

Plusieurs fonctionnalités sont disponibles avec cet outil, la principale étant le placement d'éléments sur la carte. Parmi ces éléments, on retrouve une grande sélection de rails, d'arbres, de quais, et tout autre élément de décor.

Le placement des éléments n'est pas limité à certaines zones, contrairement à ce qui pourrait être attendu. En effet, il est possible de placer chaque élément à n'importe quel emplacement avec n'importe quelle orientation, tout cela au centimètre près. Il n'existe également aucune aide pour faciliter le placement des rails et des objets. En effet, les rails doivent être placés à la main avec une grande précision si l'on veut qu'ils soient fonctionnels. Cet outil, s'il permet beaucoup de choses peut donc être compliqué à prendre en main. Pouvoir automatiser la pose des rails faciliterait en fait grandement la création de cartes pour les utilisateurs.

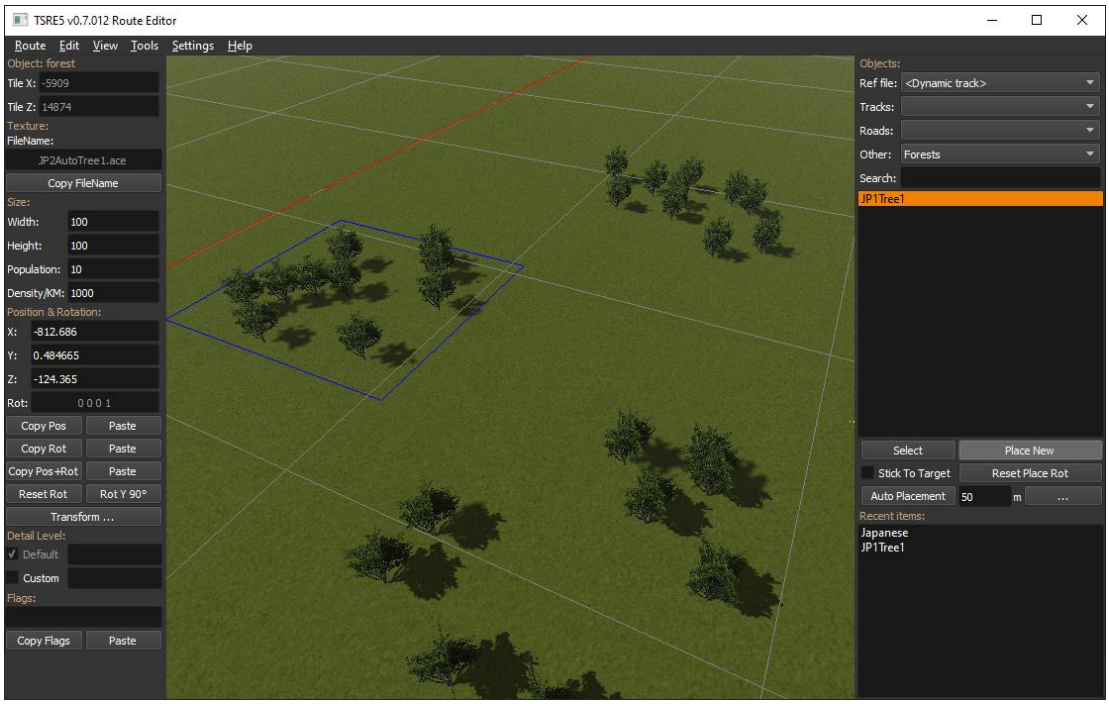


Figure 3 - Utilisation de TSRE

4.1.2.3 Problématique de TSRE

L'objectif du projet est de développer une application visant à faciliter la création de cartes compatibles avec OpenRails. Un élément-clé de cette application serait d'automatiser la pose des rails, étant donné que cet élément est crucial quant à la création de cartes à destination d'un simulateur de train. De plus, cette pose de rails est compliquée à réaliser depuis l'outil TSRE. Cependant, cette automatisation est en réalité impossible ou du moins très compliquée ; et ce, pour plusieurs raisons détaillées au point suivant.

Problématique

En théorie, la pose d'un rail pourrait être envisagée en définissant un point de départ et un point d'arrivée. Ensuite, le logiciel se chargerait de raccorder les 2 points avec un rail qui pourrait donc être droit ou en courbe. Si c'était le cas, il serait relativement facile d'automatiser la pose de ces rails. Cependant, ça n'est pas le cas. En effet, la pose de rails se fait entièrement manuellement. Même les rails disponibles sont limités en termes de format et de distance. Par exemple, réaliser une ligne droite de 32,60 m est impossible car il n'existe pas de rails de 32,60 m. Il est toutefois possible d'aligner un rail de 30m, de 2m et de 0,50 m mais il n'existe pas de rails de 0,10 m. Cela se complique encore lorsqu'il s'agit de réaliser des

courbes puisque les angles et les distances possibles sont également limitées. Ce qui pose un problème dans notre cas puisque le but est d'automatiser la création de réseaux ferroviaires à partir de données réelles.

Solution

Toutefois, cela n'est pas impossible pour autant. Un réseau doit d'abord être scindé en plusieurs sections afin de permettre leur analyse avec un algorithme qui calculerait et déterminerai les rails à utiliser pour une certaine distance. Pour une simple ligne droite, c'est faisable, tant que les valeurs sont réalisables en additionnant différents formats de rails disponibles. Mais quand il s'agit d'intégrer des courbes précises avec des variations d'angle au sein de celles-ci, cela devient bien plus complexe. Une option serait que l'algorithme détermine les formats de pièces manquantes principalement pour les courbes et que l'on automatise la création de ces pièces dans un modèles 3D compatible afin de les utiliser dans TSRE.

C'est la solution théorique que j'ai envisagée. Cependant, cette solution présente 2 défis : le 1er étant la réalisation de l'algorithme en lui-même et le 2ème étant l'automatisation de la création de modèle 3D. Mais créer un modèle 3D demande une certaine puissance de calcul. Donc naturellement, dans un cas concret, on pourrait devoir créer des centaines ou des milliers de modèles 3D à intégrer dans TSRE. Ce qui pourrait demander une très grande puissance de calcul et un temps de création élevé[3].

L'automatisation de la pose des rails n'est donc pas une option envisageable, du moins dans le cadre de ce cours puisque le temps pour la réaliser pourrait prendre de nombreux mois, voire des années.

4.1.2.4 Révision des fonctionnalités

Étant donné que l'automatisation de la pose des rails n'est pas envisageable, cette fonctionnalité est abandonnée au profit d'autres outils permettant de faciliter le positionnement des rails.

[3] N'ayant pas testé cette possibilité, je n'ai malheureusement pas de chiffre ou de données pour appuyer mes propos.

4.1.2.5 Fichiers de marquage géographique

Après plusieurs recherches, j'apprends qu'il est possible d'importer au sein de TSRE des fichiers de marquage géographique. Ces fichiers peuvent avoir plusieurs formats. Parmi ceux-ci : MKR, KML ou encore GPX sont les plus utilisés et sont supportés par TSRE.

Ces formats de fichiers sont utilisés pour stocker et échanger des données géospatiales. Il permettent de représenter des points d'intérêt, des routes, des parcours, des polygones ainsi que d'autres types de données géographiques.

Le format MKR (Marker File) est un format peu courant et n'est utilisé que dans des applications spécifiques.

Le format KML (Keyhole Markup Language) est largement utilisé dans des logiciels comme Google Earth, Google Maps, etc.

Enfin, le format GPX (GPS Exchange Format) est plutôt utilisé pour enregistrer des tracés GPS, des itinéraires et est souvent utilisé avec des appareils GPS.

Ces 3 types de fichiers poursuivent le même but dans le cadre de la création de cartes depuis TSRE : celui d'afficher et de permettre de visualiser des points-clés pour la création de routes/rails.

Voici, ci-dessous, une capture d'écran des repères ajoutés depuis un fichier de marquage géographique :

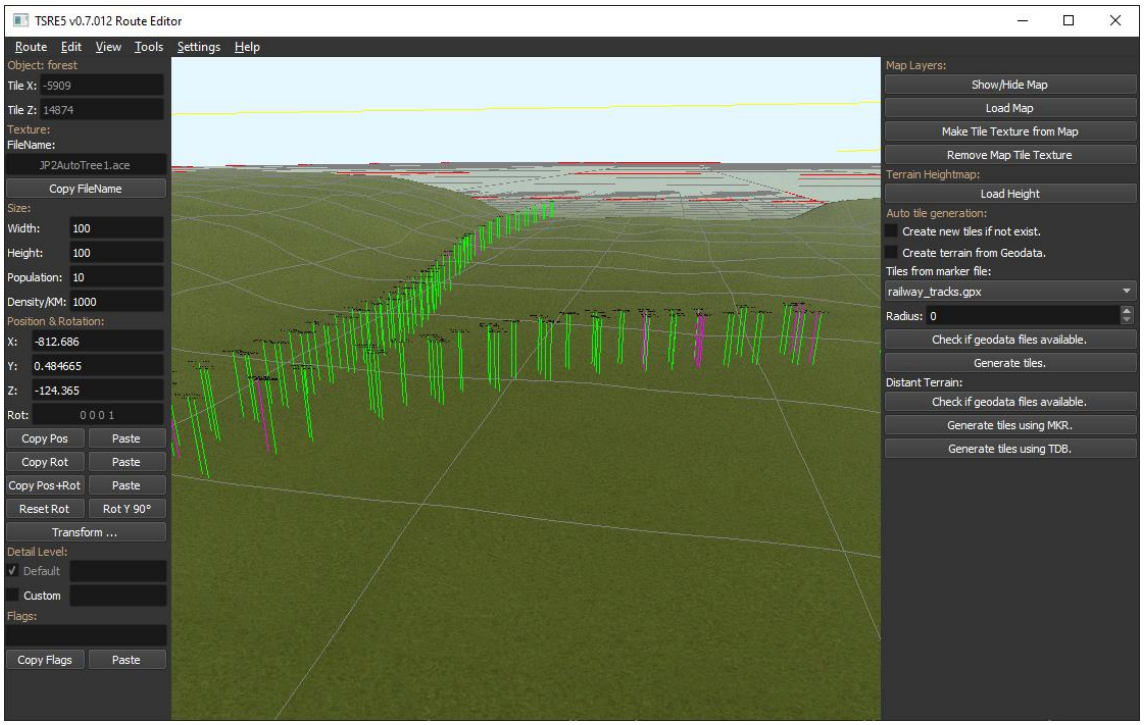


Figure 4 - Affichage des marqueurs

4.1.2.6 Calque Texture

Il existe un autre outil qui, lui, est déjà implémenté dans le logiciel TSRE et qui permet également de faciliter la pose de rails. Cet outil utilise l'API d'Open Street Map afin d'importer des données de texture que l'on peut coller comme un calque sur sa carte. Les correspondances se font via les coordonnées géographiques de la case [4] sélectionnée. Ce calque permet d'être plus précis lorsque l'on place les rails et éléments du décors à la main.

[4] Chaque carte TSRE est décomposée en cases (*Tile*) d'environ 2km de côté.

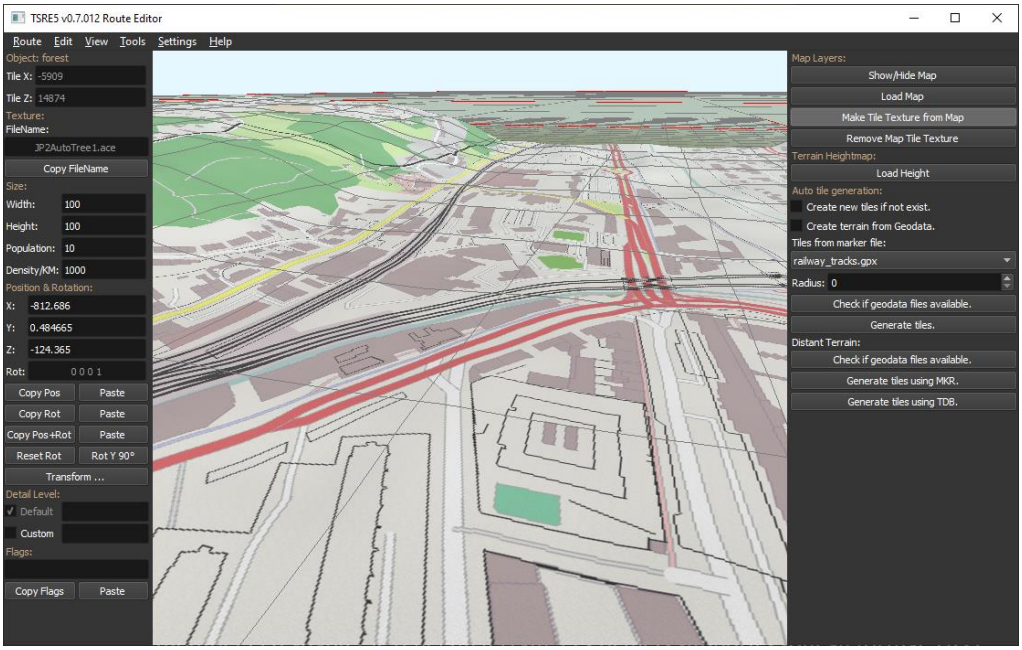


Figure 5 - Affichage du calque

4.1.2.7 Données sur l'élévation du terrain

Il est possible d'importer des fichiers comportant des données quant à l'élévation du terrain d'une zone spécifique. Sur base de ces fichiers, l'application se charge de modifier et d'adapter le relief du terrain. Les fichiers supportés sont au format .HGT qui est un format de fichier qui contient les altitudes du terrain sous forme de grille. Chaque point de la grille représente donc une altitude précise ; ce qui permet de modéliser la topographie du terrain.

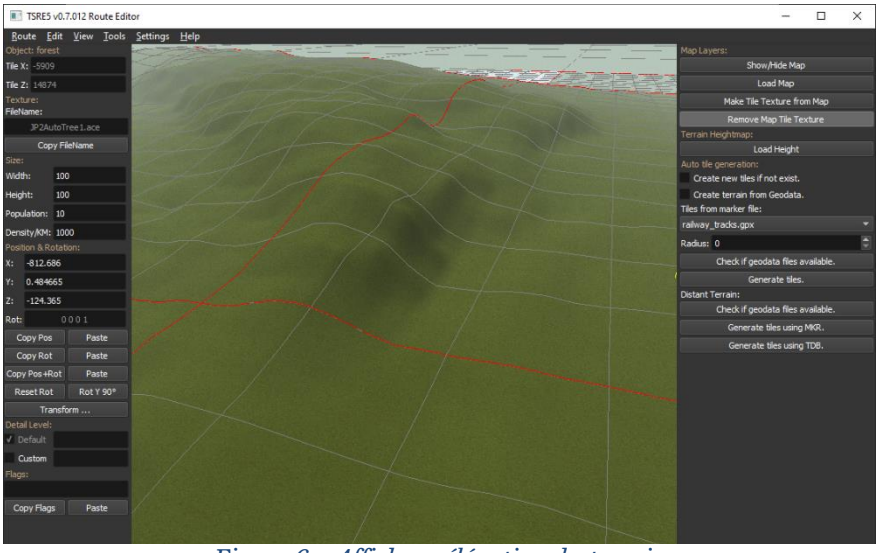


Figure 6 - Affichage élévation du terrain

4.2 Obtention des données géographiques

Après l'analyse et l'identification des formats de données compatibles avec le logiciel TSRE, l'étape suivante correspond à l'obtention de ces données.

4.2.1 Google API

Google possède de nombreuses API permettant de récupérer plusieurs types de données géographiques. L'objectif, dans un premier temps, est d'obtenir une série de points avec les coordonnées géographiques pour chaque point. Ces points doivent représenter les trajets empruntés par les trains (les rails).

Malheureusement, il est impossible d'obtenir directement des informations sur les réseaux ferroviaires avec les API's de Google. Toutefois, lorsque l'on fait une demande d'itinéraire dans Google Maps, on obtient les trajets possibles. Si ces trajets passent par des chemins de fer, ceux-ci sont référencés et le trajet du train est dessiné sur la carte. Il devrait donc être possible d'obtenir les informations du trajet.

Pour y parvenir, il a fallu créer un compte Google Cloud afin de pouvoir utiliser l'API Directions (API utilisée pour les itinéraires). Plusieurs requêtes ont été testées et les réponses ont été analysées. Dans celles-ci, j' ai trouvé, dans les détails des étapes du trajet, un élément nommé : « *polyline* [5] ». Cet élément est en fait la liste des coordonnées GPS des points permettant à Google Map de tracer le trajet du train.

4.2.1.1 Polylines

Cependant, ces *polylines* sont, par défaut, encodées. J'ai pu les décoder en utilisant une bibliothèque python nommé *polyline*. Une fois décodées, j'en ai formaté les données dans un fichier .CSV.

```
"html_instructions": "Train towards Tournai",
"polyline": {
    "points": "gg`sHaut\\~@f@B`@Bv@Bv@DdADzC@zA?R@`FCx@CtAOfBCnB
},
```

Figure 7 - Polylines encodées

[5] Ce qui signifie littéralement : plusieurs lignes.

```
50.46916,4.86241
50.46884,4.86221
50.46882,4.86204
50.4688,4.86176
50.46878,4.86148
50.46875,4.86113
50.46872,4.86035
50.46871,4.85989
50.46871,4.85979
50.4687,4.85866
50.46872,4.85837
50.46874,4.85794
```

Figure 8 - Polylines décodés et formatés

4.2.1.2 Visualisation

Finalement, un outil de visualisation de coordonnées géographiques a été utilisé : « ***GPS Visualizer*** ». Celui-ci a permis de vérifier la cohérence des données obtenues.

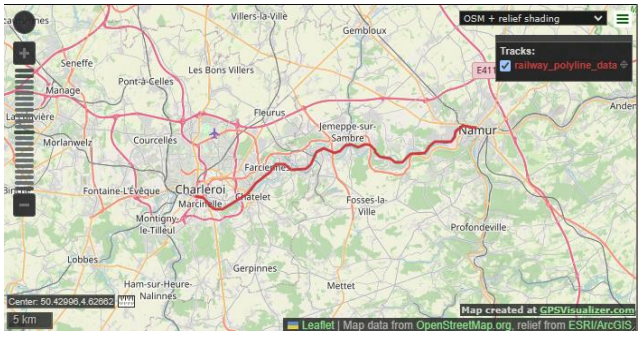


Figure 9 - Visualisation polyline Google

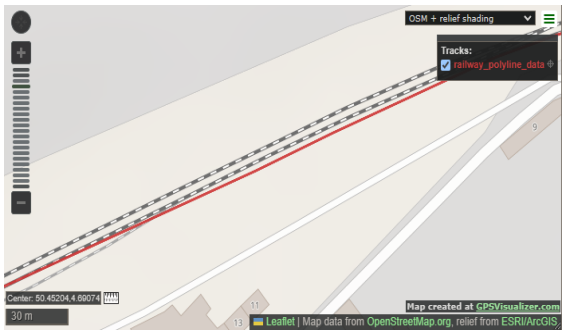


Figure 10 - Imprécision polyline Google

4.2.1.3 Problèmes

Plusieurs problèmes sont à noter quant à l'utilisation de la Google API.

1. **Sélection de zone**

L'utilisation exclusive de l'API Directions de Google ne permet pas de sélectionner directement une zone pour l'extraction de données. En effet, seulement les coordonnées relatives à un trajet spécifique peuvent être obtenues.

2. **Manque de précision**

Depuis GPS Visualizer (**voir image précédente**), on remarque que les coordonnées du trajet ne correspondent pas toujours exactement aux coordonnées réelles des voies.

3. **Accessibilité**

Pour finir, l'API de Google requiert un compte pour pouvoir l'utiliser et est payant à partir d'un certain nombre de requêtes ; ce qui pose problème pour ce projet.

4.2.2 Open Street Map (OSM)

Open Street Map, souvent abrévié OSM, est un projet collaboratif, un projet open source qui a pour but de cartographier la carte du monde de manière à ce qu'elle soit accessible à tous. Ce qui règle le problème d'accessibilité que l'on pouvait avoir en utilisant l'API de Google.

C'est un projet open source, ce qui signifie que n'importe qui peut y contribuer et en modifier les cartes. Il est donc nécessaire de s'interroger sur la manière dont les modifications sont faites et approuvées pour garantir que les changements restent cohérents par rapport à l'environnement dans lequel la modification a été apportée. En d'autres termes, comment le *versioning* [6] est-il géré ? peut-on accéder à d'anciennes versions d'objets ou de zones ?

4.2.2.1 Versioning OSM

Le versioning dans Open Street Map permet de suivre l'historique complet des modifications pour chaque élément, pour chaque objet. Chaque élément, chaque objet possède un identifiant ainsi qu'un numéro de version incrémenté à chaque changement. OSM conserve toutes les versions antérieurs, accessibles via leur API et permet de revenir à des version précédentes en cas d'erreur. Les modifications sont immédiatement visibles sans approbation préalable. C'est donc à la communauté de faire preuve de vigilance et de corriger les erreurs éventuelles.

[6] Terme anglais qui correspond à la manière dont les versions sont gérées.

4.2.2.2 Extraction de données géographiques

OSM met à disposition gratuitement une API permettant d'obtenir et d'automatiser l'extraction d'informations géographiques. Cette API utilise un *wrapper* [7]pour ses requêtes dans le but de les simplifier. Elle permet, en outre, de choisir une zone rectangulaire en fournissant les coordonnées géographiques du Sud-Ouest du rectangle et celles du Nord-Est du rectangle. Elle permet aussi de filtrer les informations pour ne garder que les données relatives aux réseaux ferroviaires.

```
def fetch_railway_data(min_latitude, min_longitude, max_latitude, max_longitude):
    overpass_query = f"""
    [out:json];
    (
      way["railway"]
        ({min_latitude},{min_longitude},{max_latitude},{max_longitude});
    );
    out body;
    >;
    out skel qt;
    """
```

Figure 11 - Requête OSM

4.2.2.3 Visualisation des données

En visualisant les données on remarque que les chemins correspondent parfaitement (au mètre près) aux données réelles.

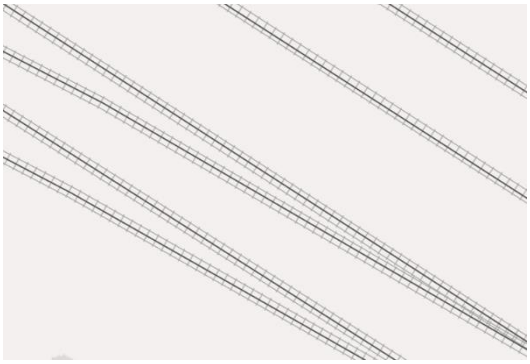


Figure 12 - Visualisation OSM

[7] Terme anglais qui désigne un « emballage ».

4.2.2.4 Résolutions des problèmes

Les 3 problèmes précédents de sélection de zone, de précision et d'accessibilité sont donc résolus seulement avec Open Street Map.

4.2.2.5 Extraction de fichiers GPX

Depuis le site web, il est impossible d'extraire des données de marquage géographique. Pour ce faire, il faut utiliser *Overpass*[8]. Il est possible de tester ses requêtes Overpass depuis un navigateur en utilisant **Overpass Turbo**. Celui-ci présente une interface pour créer sa requête et afficher les résultats de la requête. Une fois les résultats affichés, il est possible de télécharger les données avec plusieurs formats différents dont le format GPX qui peut être utilisé dans TSRE.

Cependant, depuis l'API il est impossible d'obtenir des données géographiques sur les rails aux formats autres que XML et JSON. Il est donc nécessaire de créer un algorithme qui puisse convertir les fichiers au format JSON ou XML en GPX ou KML afin de pouvoir, à terme, automatiser l'importation de ces données. Cette étape sera abordée dans l'étape de **Traitement des données**.

4.2.3 Extraction de données d'élévation de terrain

Pour obtenir les données sur l'élévation de terrain, plusieurs sources sont possibles : **USGS EarthExplorer** (NASA) pour le monde entier, **Copernicus DEM** pour l'Europe. Il en existe pour chaque pays. Pour la Belgique, il y a 2 sites principaux : **geoPunt** & **geoportail** de wallonie.

L'idéal serait de ne pas avoir à contacter des API différentes en fonction des pays, il faut aussi que la plateforme possède une API à disposition et soit idéalement gratuite. Celle qui correspond le mieux à ces critères est USGS EarthExplorer (NASA). Le seul inconvénient est que, pour accéder aux données, il est obligatoire de créer un compte.

Les données d'élévation de terrain y sont répertoriées sous forme de grille. Chaque case de la grille a pour coordonnées un degré de latitude et un degré de longitude qui correspond au coin inférieur gauche de la case.

---

[8] Overpass est une API d'OSM permettant d'extraire des données géographiques spécifiques.

4.3 Traitement des données

Concernant le traitement des données, seulement les données des fichiers géographiques obtenus depuis l'API d'Open Street Map devront être modifiées.

Tout d'abord, il faut choisir le type de fichier que l'on souhaite obtenir depuis l'API, XML ou JSON. Le format JSON étant généralement plus léger, c'est celui-ci qui a été choisi.

Ensuite, il faut identifier la structure du fichier que l'on obtient afin de le convertir correctement et de ne pas perdre d'information cruciale.

Les fichiers obtenus peuvent être décomposés en 3 parties :

1. Les métadonnées

```
{
  "version": 0.6,               // Version de l'API Overpass utilisée
  "generator": "Overpass API...", // Générateur de l'API Overpass
  "osm3s": {                     // Informations supplémentaires sur les métadonnées
    "timestamp_osm_base": "...", // Horodatage des données OSM
    "copyright": "..."           // Informations sur le droit d'auteur
  },
```

Figure 13 - OSM JSON métadonnées

2. Les éléments avec la liste des chemins

```
"elements": [                  // Tableau contenant les éléments OSM
  {
    "type": "way",             // Type de l'élément (ici, une voie)
    "id": 24777894,            // ID unique de la voie
    "nodes": [                 // Tableau d'ID de nœuds constituant la voie
      81953612, 269237848, ...
    ],
    "tags": {                  // Propriétés associées à la voie
      "highway": "residential",
      "lit": "yes",
      "name": "Gielgenstraße"
    }
  },
```

Figure 14 - OSM JSON éléments

3. Les nœuds & leurs coordonnées

```
{
  "type": "node",           // Type de l'élément (ici, un nœud)
  "id": 81953612,           // ID unique du nœud
  "lat": 50.7410694,        // Latitude du nœud
  "lon": 7.2009669          // Longitude du nœud
},
{
  "type": "node",           // Un autre nœud
  "id": 265110170,
  "lat": 50.7407049,
  "lon": 7.1974150
},
```

Figure 15 - OSM JSON noeuds

La spécificité de ces fichiers est que, pour éviter la redondance, chaque point géographique possède un identifiant et les informations de coordonnées de ses points se trouvent à la fin du fichier. Lors de la conversion, il faut donc retrouver les coordonnées de chaque nœud à la fin du fichier sur base de l'identifiant du nœud.

Voici, ci-dessous, un aperçu du fichier obtenu après la conversion :

```
<gpx version="1.1">
  <trk>
    <name>Gielgenstraße</name>
    <trkseg>
      <trkpt lat="50.7410694" lon="7.2009669">
        <ele>100.0</ele>
      </trkpt>
      <trkpt lat="50.7407049" lon="7.1974150">
        <ele>105.0</ele>
      </trkpt>
    </trkseg>
  </trk>
</gpx>
```

Figure 16 - Fichier gpx

5 Solution

La solution ainsi que la procédure d'installation se trouve sur gitlab **https://gitlab.com/DTM-Henallux/MASI/etudiants/brich-johan/tools-for-tsre.git**

5.1 Choix de la solution

Le choix du type d'application ou d'outil se fait selon les contraintes que peuvent poser certaines fonctionnalités. Concernant ces fonctionnalités :

L'utilisateur devra pouvoir sélectionner une zone géographique depuis une carte interactive afin de récupérer les données géographiques du réseaux ferroviaires de la zone sélectionnée. De plus, les données d'élévation du terrain devront aussi être récupérées.

Ensuite, ces données devront pouvoir être importées et utilisées dans le logiciel TSRE pour faciliter la création de cartes.

5.1.1 Script Python Autonome

La première option consiste à développer un simple script Python qui serait exécuté dans un terminal. Cette option présente de nombreux avantages mais également des inconvénients.

5.1.1.1 Avantages

- **Légèreté et simplicité** : Un script Python est léger et facile à exécuter, avec des exigences techniques faibles. Il est également simple à implémenter.
- **Cross-platform** : Python est compatible avec divers systèmes d'exploitation tant que ceux-ci peuvent exécuter une version spécifique de Python.
- **Autonomie** : L'application serait autonome, sans besoin d'un serveur web, sauf pour l'accès aux API d'OpenStreetMap et de la NASA.

5.1.1.2 Inconvénients

- Complexité d'intégration d'un carte interactive : Bien que l'affichage de cartes soit possible, l'intégration d'une carte interactive avec des sélections de zones via des polygones reste complexe à réaliser avec des bibliothèques comme **Folium** qui se limitent à des cartes statiques.

5.1.2 Application Web

La 2ème option consiste à développer un site web s'exécutant dans un navigateur utilisant des technologies telles que JavaScript. Cela faciliterait la manipulation et l'interaction avec une carte.

5.1.2.1 Avantages

- **Facilité d'implémenter un carte interactive** : JavaScript dispose de plusieurs bibliothèques comme **Leaflet** qui permettent d'intégrer des cartes interactives à des applications web.
- **Cross-platform** : Étant donné que cette option utilise les technologies web, il suffirait d'un navigateur pour pouvoir l'utiliser.

5.1.2.2 Inconvénients

- **Nécessite un serveur** : Un serveur web pourrait être nécessaire pour héberger l'application
- **Permissions de l'application :** Pour des raisons de sécurité, les applications web ne peuvent pas accéder facilement aux répertoires de l'ordinateur. Automatiser la modification de fichier locaux devient donc compliqué.

5.1.3 Application Desktop Hybride

La 3ème et dernière option est de créer une application desktop hybride qui combine les avantages des technologies web avec ceux d'une application installée localement sur l'ordinateur.

5.1.3.1 Avantages

- Carte interactive : Les technologies web permettent d'intégrer facilement des cartes interactives.
- Autonomie : Aucune nécessité de serveur web, l'application s'exécute localement sur l'ordinateur avec un navigateur qui peut être intégré à l'application.
- Accès au système de fichiers : L'application peut accéder directement aux répertoires de l'utilisateur (si celui-ci le permet), permettant d'écrire et de gérer les fichiers au bon endroit dans l'arborescence d'un projet TSRE.

5.1.3.2 Inconvénients

- Poids de l'application : Un navigateur est souvent intégré à l'application, ce qui l'alourdit.

5.2 Analyse des technologies

Après avoir déterminé le type d'application idéale pour le projet, il faut encore déterminer les technologies qui vont être utilisées pour la réalisation de celui-ci.

Plusieurs technologies existent pour créer une application hybride. Celles qui ont été comparées sont les suivantes :

Electron

Figure 17 - Logo electron

Tauri

Figure 18 - Logo tauri

NW

Figure 19 - Logo NW.JS

Flutter Desktop

Figure 20 - Logo flutter

	Electron	Tauri	NW.js	Flutter Desktop
Communauté (5)	5	4	2	4
Performance (5)	2	5	2	4
Légèreté (3)	1	3	1	2
Flexibilité (3)	3	2	3	2
Facilité d'intégration(3)	3	3	3	1
Total	14	17	11	13

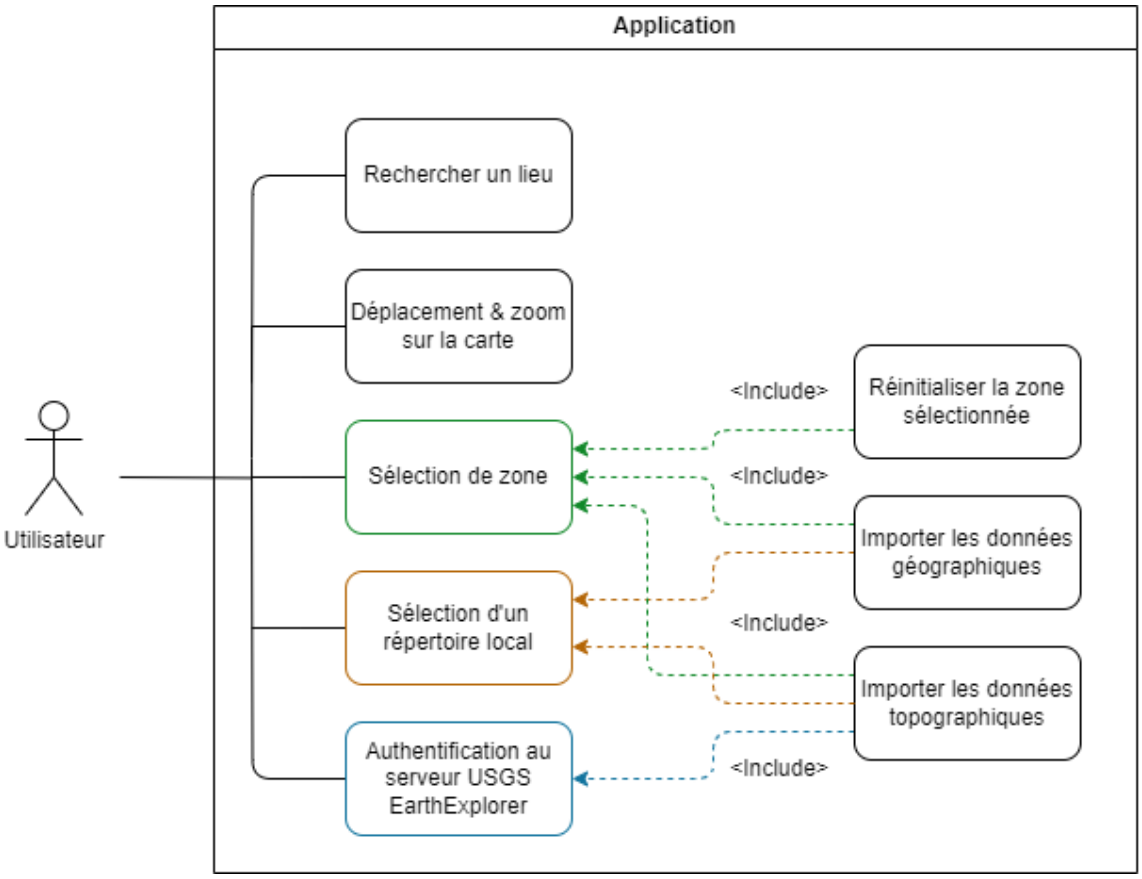
Tauri se distingue avec un total de 17 points selon les critères et la pondération établie, c'est pourquoi Tauri est l'outil qui a été choisi pour la réalisation de cette application hybride.

5.2.1 Tauri

Tauri est un framework léger et performant pour créer des applications desktop multi-plateformes. Il utilise JavaScript/TypeScript pour le frontend et Rust pour le backend. Il se distingue par la petite taille des applications qu'il génère, en utilisant le moteur Web natif de l'OS au lieu d'embarquer un navigateur complet, comme le fait Electron par exemple. Bien que sa communauté soit plus petite, elle est en pleine croissance. De plus, sa documentation est complète et bien réalisée. Il s'intègre également facilement avec des technologies comme React et Vue.js pour le *frontend*[9].

[9] « Facade » d'une application web

5.3 Analyse fonctionnelle



5.3.1 Rechercher un lieu

L'utilisateur a la possibilité d'effectuer sa recherche de lieu via une barre de recherche afin d'afficher la carte correspondante au lieu recherché.

5.3.2 Déplacement & zoom sur la carte

La carte est interactive, ce qui signifie que l'utilisateur peut s'y déplacer avec sa souris et effectuer un zoom et un dézoome à l'aide de boutons ou de sa souris.

5.3.3 Sélection de zone

Pour importer des données géographiques ou topographiques, l'utilisateur doit sélectionner une zone sur la carte en y dessinant un rectangle.

5.3.4 Réinitialiser la zone sélectionnée

Si une zone a été sélectionnée, l'utilisateur peut la réinitialiser à l'aide d'un bouton. S'il décide d'en sélectionner une 2ème, la 1ère se supprimera automatiquement.

5.3.5 Sélection d'un répertoire local

Pour importer des données, l'utilisateur doit renseigner le chemin vers le répertoire de destination. Ceci peut se faire via un bouton qui ouvre l'explorateur de fichiers.

5.3.6 Authentification au serveur USGS Earth Explorer

Afin de pouvoir accéder et importer des données topographiques, il est nécessaire à l'utilisateur d'avoir un compte USGS Earth Explorer et de s'y connecter depuis l'application.

5.3.7 Importer les données géographiques

Une fois la zone et le répertoire sélectionnés, l'utilisateur peut importer les données géographiques correspondantes à sa sélection.

5.3.8 Importer les données topographiques

Une fois la zone, le répertoire sélectionnés et la connexion à son compte établie, l'utilisateur peut importer les données topographiques correspondantes à sa sélection.

5.4 Architecture de l'application

L'application possède une architecture hybride. En effet, le front-end est construit avec des technologies web (JavaScript, HTML, CSS) et utilise le framework **React**. Le back-end est géré par Tauri en **Rust**. Cela permet une interface « légère » tout en ayant un accès aux ressources du système.

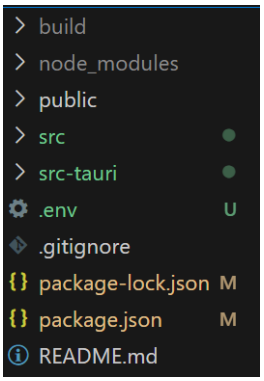


Figure 21 - Architecture de l'application

/build : Dossier contenant les fichiers générés après la compilation et le packaging de l'application.

/node_modules : Répertoire contenant toutes les dépendances et modules Node.js installés pour le projet.

/public : Dossier contenant les fichiers statiques publics, accessibles directement par l'application (ex. : index.html).

/src : Dossier contenant le code source principal de l'application, incluant les composants React et les services JavaScript.

/src-tauri : Dossier contenant les fichiers spécifiques à Tauri, notamment les configurations et le code backend en Rust.

.env : Fichier de configuration des variables d'environnement pour l'application.

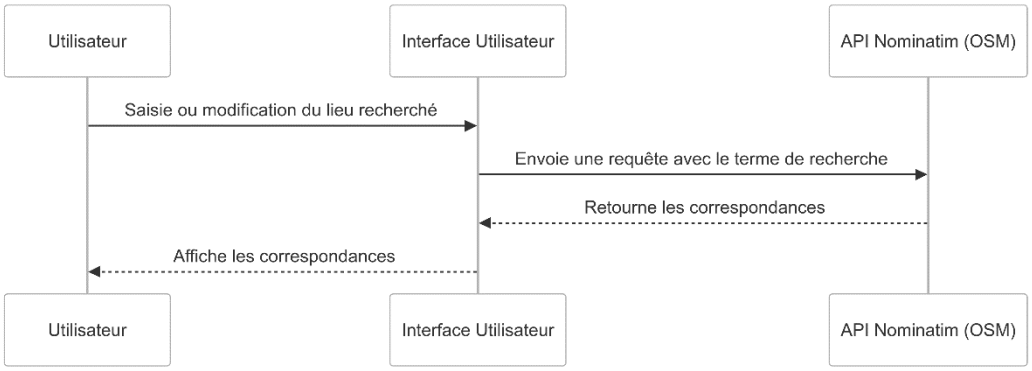
.gitignore : Fichier spécifiant les fichiers et dossiers à ignorer par Git lors des commits.

package.json : Fichier de configuration de Node.js, listant les dépendances du projet et les scripts de commandes.

README.md : Fichier texte contenant la documentation ou les instructions pour le projet.

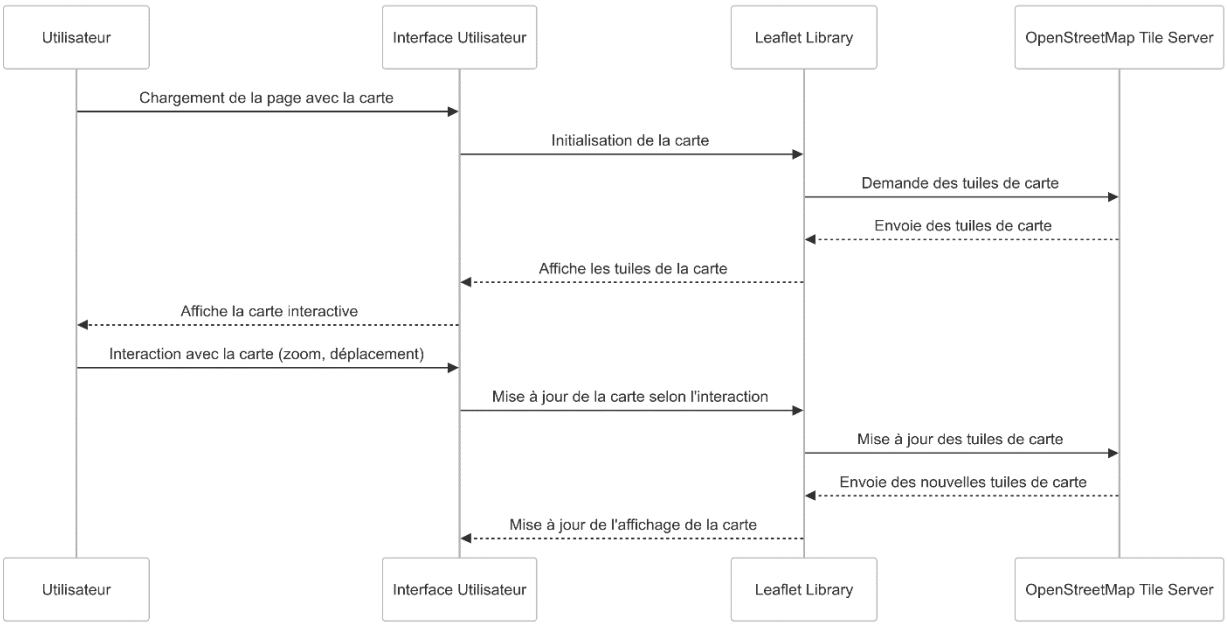
5.5 Implémentation de la solution

5.5.1 Recherche d'un lieu



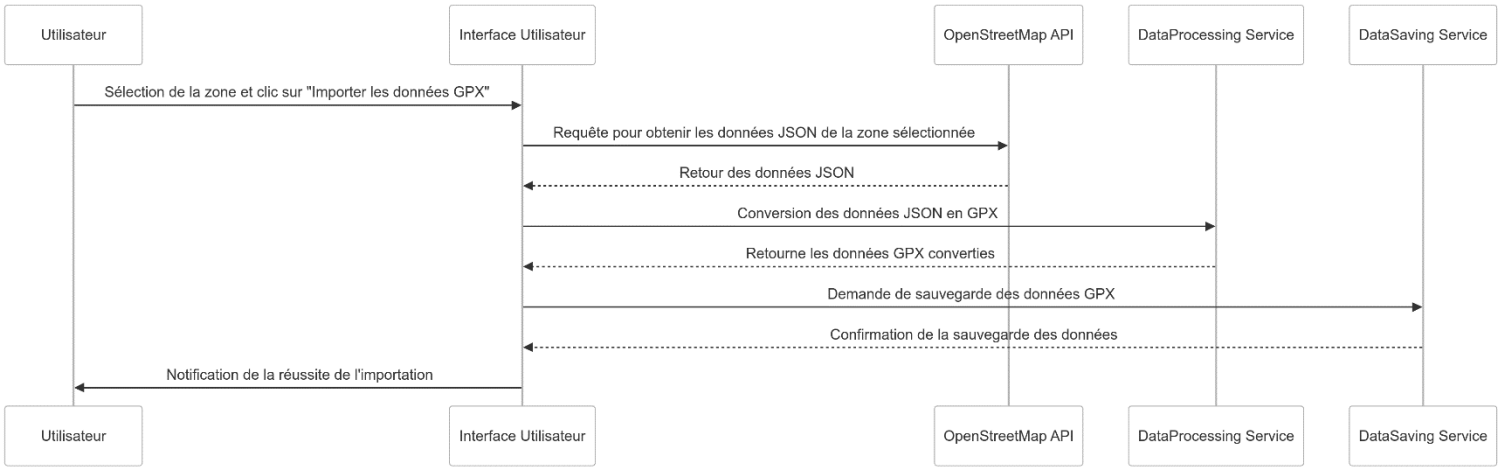
Ce diagramme montre la séquence d'événements qui se déroulent lorsque l'utilisateur saisit ou modifie un lieu à rechercher. L'interface envoie une requête à l'API Nomatim d'OpenStreetMap et affiche les correspondances reçues.

5.5.2 Intégration de la carte



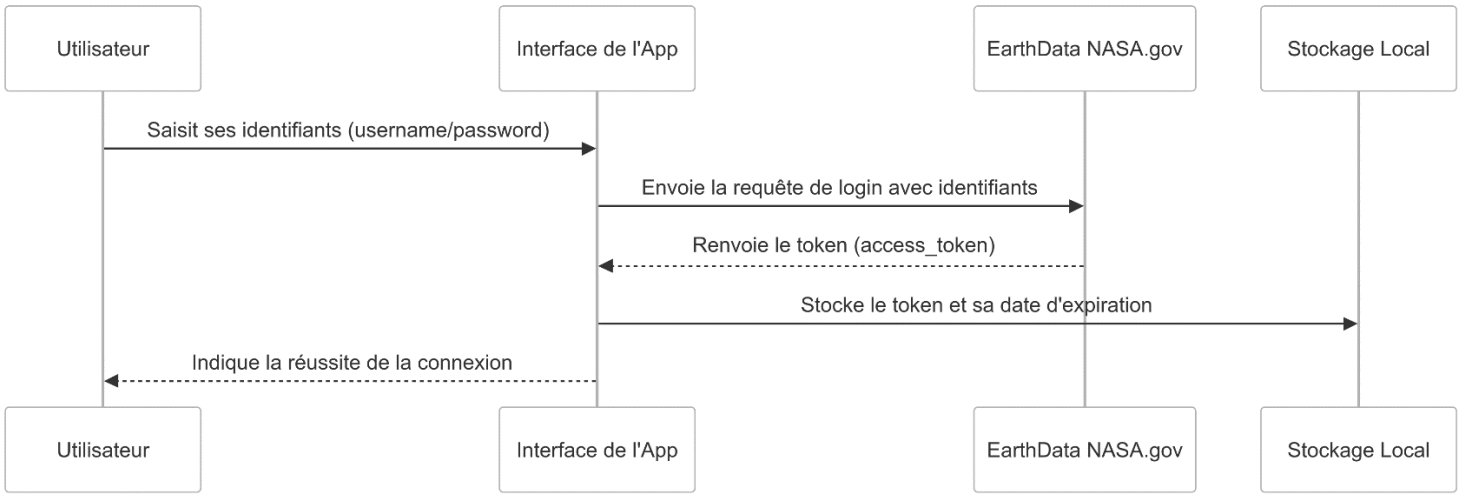
Ce diagramme montre les évènements pour l'intégration de la carte avec Leaflet. Lors du chargement de la page, la carte est initialisée et les tuiles sont récupérées depuis le serveur d'Open Street Map pour être ensuit affichées. Chaque interaction de l'utilisateur avec la carte, telle que le zoom ou le déplacement, entraine une mise à jour de l'affichage.

5.5.3 Importation données GPX



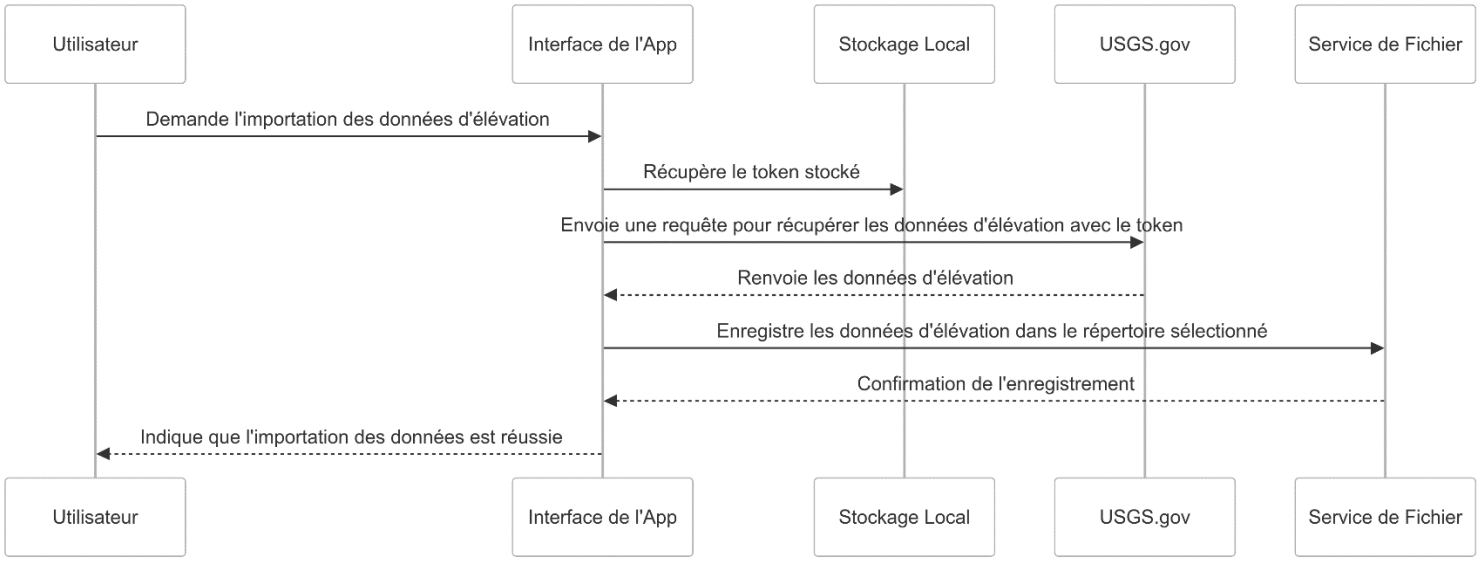
L'utilisateur sélectionne une zone sur la carte et clique sur le bouton « importer les données GPX ». L'application envoie une requête pour obtenir les données de la zone sélectionnée. Les données reçues sont au format JSON, c'est ensuite le service de traitement de données, propre à l'application qui se chargera de convertir ces données au format GPX. Ces données sont ensuite renvoyées et sauvegardées dans le répertoire sélectionné à l'aide du service de sauvegarde des données, lui aussi inclus dans l'application. Pour finir, si toutes les opérations se sont bien déroulées, un message de confirmation de l'importation est envoyé à l'utilisateur.

5.5.4 La connexion, gestion des tokens

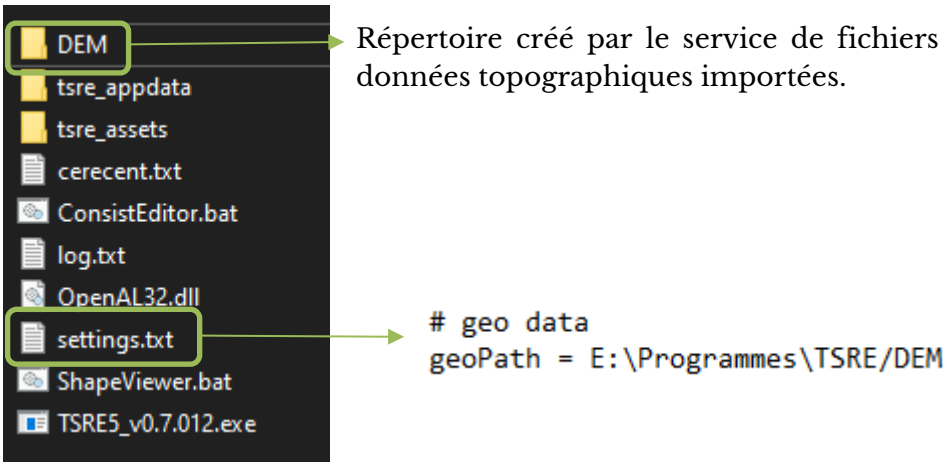


Lorsque l'utilisateur souhaite importer des données sur la topographie mais qu'il n'est pas encore connecté à son compte EarthData NASA, il devra s'y connecter. Il entre donc ses informations : nom d'utilisateur ainsi que son mot de passe. Une requête est envoyée à l'API qui se charge de la gestion des tokens. Le token est ensuite renvoyé à l'application, il est stocké dans le stockage local et pourra être utilisé plus tard pour récupérer les informations sur la topographie.

5.5.5 L'importation des données d'élévations



Pour importer les données d'élévation, l'utilisateur doit s'être précédemment connecté. L'application récupère le token stocké dans le stockage local de l'application et envoie une requête pour récupérer les données d'élévation de la zone sélectionnée. USGS.gov renvoie les données topographiques à l'application. Ensuite, le répertoire renseigné par l'utilisateur doit être le répertoire racine de l'application TSRE, le service de fichier se chargera alors de créer un nouveau répertoire au sein de celui-ci s'il n'existe pas déjà. Ce nouveau répertoire se nomme : « DEM » pour Digital Elévation Model. Le service de fichier se charge également de modifier le fichier settings.txt de TSRE afin de renseigner le chemin vers les données topographiques. Voici un exemple ci-dessous.



Fichier « settings.txt » modifié pour renseigner le chemin vers le répertoire /DEM

6 Conclusion

En réfléchissant sur le chemin parcouru, il est clair que ce projet a offert une opportunité précieuse pour appliquer des compétences techniques tout en répondant à une problématique concrète. Il a également permis d'explorer des solutions innovantes en dehors des sentiers battus. L'intégration de technologies telles que Tauri pour la création d'une application desktop hybride, combinée à l'utilisation des API d'OpenStreetMap et de la NASA pour la collecte de données géographiques et topographiques, m'a permis d'enrichir mes compétences en développement et analyse.

Ce projet a également été l'occasion d'acquérir de nouvelles compétences techniques. La réalisation d'une application hybride avec Tauri a renforcé la compréhension du développement d'applications desktop en utilisant des technologies web. La gestion des API, ainsi que des tokens d'authentification, a amélioré la maîtrise des aspects liés à la sécurité et à la gestion des données. La manipulation et la conversion de données géographiques ont permis de rendre ces informations compatibles avec des outils de simulation.

En termes d'améliorations, plusieurs aspects pourraient être envisagés pour perfectionner l'application. L'ajout d'une fonctionnalité multilingue permettrait de rendre l'outil accessible à un public plus large. L'amélioration de l'interface utilisateur pour la rendre encore plus intuitive, notamment pour les utilisateurs moins expérimentés, pourrait également être un axe de développement. De plus, l'intégration d'une fonctionnalité de prévisualisation des données avant l'importation pourrait aider à éviter les erreurs et à optimiser le processus de création de cartes.

En résumé, ce projet a permis de répondre à une problématique concrète tout en offrant un terrain d'apprentissage riche, avec des perspectives d'amélioration et d'évolution pour le futur.

7 Sources

1. OpenRails . [Online]. Available: https://www.openrails.org/
2. Build your own route, OpenRails. [Online]. Available: https://www.openrails.org/learn/build-route/.
3. OpenRails Tutoriel, Coalstone Newcastle. [Online]. Available: https://www.coalstonewcastle.com.au/physics/route-build-tutorial/.
4. TSRE Manuel, GitHub. [Online]. Available: https://raw.githubusercontent.com/pwillard/TSRE5-Document/master/book.pdf.
5. Google Map. [Online]. Available: https://www.google.com/maps.
6. Google Cloud. [Online]. Available: https://cloud.google.com/.
7. Python polyline, PyPI. [Online]. Available: https://pypi.org/project/polyline/.
8. GPS Visualizer. [Online]. Available: https://www.gpsvisualizer.com/.
9. GeoJson. [Online]. Available: https://geojson.io/.
10. Open Street Map. [Online]. Available: https://www.openstreetmap.org/.
11. Open Street Map Wiki. [Online]. Available: https://wiki.openstreetmap.org/.
12. Overpass Turbo. [Online]. Available: https://overpass-turbo.eu/.
13. USGS EarthExplorer. [Online]. Available: https://earthexplorer.usgs.gov/.
14. Copernicus DEM. [Online]. Available: https://spacedata.copernicus.eu/collections/copernicus-digital-elevation-model.
15. Geopunt. [Online]. Available: https://www.geopunt.be/.
16. Geoportail de Wallonie. [Online]. Available: https://geoportail.wallonie.be/.
17. Folium, Python Visualization. [Online]. Available: https://python-visualization.github.io/folium/latest/.
18. Leaflet. [Online]. Available: https://leafletjs.com/.
19. Electron. [Online]. Available: https://www.electronjs.org/.
20. Tauri. [Online]. Available: https://tauri.app/.
21. NW.js. [Online]. Available: https://nwjs.io/.
22. Flutter Desktop. [Online]. Available: https://flutter.dev/multi-platform/desktop.

23. React. [Online]. Available: https://react.dev/.
24. Rust. [Online]. Available: https://www.rust-lang.org/.

7.1 Images

1. Electron, "Electron Software Framework Logo," Wikipedia. [Online]. Available: https://en.m.wikipedia.org/wiki/File:Electron_Software_Framework_Logo.svg.
2. Tauri, "Tauri CSS Examples," GitHub. [Online]. Available: https://github.com/topics/tauri?l=css.
3. NW.js, "NW.js (a.k.a. Nodewebkit) Logo," SeekLogo. [Online]. Available: https://seeklogo.com/vector-logo/273759/nw-js-a-k-a-nodewebkit.
4. Flutter Desktop, "Flutter Brand Assets," Flutter.dev. [Online]. Available: https://flutter.dev/brand

8 Table des figures