Document de conception final du projet

AlpesTransport

Mohammed ROUABAH
Ilyes ZEGHDALLOU
William MAILLARD

22/03/2023

Table des matières

1 Introduction

1.1 Présentation du sujet : WikibaseLocale

L'intérêt croissant pour les **informations locales** et la vie de quartier a conduit à la création de nombreux projets et applications visant à regrouper et **centraliser** ces données.

FIGURE 1 – Secteurs d'activités WikibaseLocale

Cependant, la **multiplicité** et l'**hétérogénéité** des sources de données rendent l'accès et l'exploitation de ces dernières difficiles *(format non reconnu, manque de données ou au contraire duplication, ...)*.

Le projet WikibaseLocale vise à résoudre ce problème en créant une **base de connaissances** centralisée et vivante regroupant des informations diverses sur un territoire. Cette base sera alimentée par différentes entités telles que les citoyens, les entreprises, les collectivités, les associations, les bibliothèques et les commerces.

L'objectif du projet est donc de choisir un secteur d'activité afin de promouvoir une ou plusieurs **activités locales**, qui lui sont associées.

En conséquent le projet nécessitera d'identifier des **sources de données**, de **modéliser** les relations entre ces données, afin de les **insérer** dans une wikibase tout en prenant en compte l'exploitation finale des données collectées par l'utilisateur, via une **application**.

Une fois la WIKIBASE constituée, elle sera maintenue à jour avec l'intégration de nouvelles sources de données ou la synchronisation des données avec les modifications réalisées par les sources dont elles proviennent. Et utilisée par des utilisateurs finaux qui l'interrogeront via l'application développée.

1.2 Le projet : AlpesTransport

AlpesTransport
organisez vos déplacements

FIGURE 2 – Logo du projet

AlpesTransport est un projet de type WikibaseLocale, qui se concentre sur les activités de transports en communs au sein de la région Auvergne Rhône-Alpes.

Ainsi la WIKIBASE de ce projet **centralise les informations** relatives aux transports en communs tels que les informations sur les **arrêts**, les **horaires** et les **tarifs** ; pour chaque entreprise de la région officiant dans ce domaine (stas, sncf, bus région).

Ces informations sont collectées à partir de 3 sources qui possèdent chacune un format de données structurées différents :

Lien source	Format de données
SNCF	*.csv* , GTFS
STAS	*.xml* , web-scrapping
RegionRhoneAlpes	GTFS , *.txt*

TABLE 1 – Sources et formats des données

Du fait de l'**hétérogénéité** des structures des **sources de données**, chacune d'elle est associée à un *parseur* chargé d'extraire les informations des données collectés et de les transformer en un **format intermédiaire homogène** permettant de les insérer dans la WIKIBASE.

En effet, la WIKIBASE est alimenté en respectant un **schéma rdf** définit par nos soins, permettant de mettre en exergue les **relations** entre les entités créées.

Ces relations seront ensuite utilisées pour rechercher les informations requêtées par l'utilisateur via une application. Cette application permettra entre autres de consulter les transports en communs passant à un lieu donné, les différents tarifs, et les horaires, afin de permettre à l'utilisateur d'organiser au mieux son déplacement dans la région Auvergne-Rhône-Alpes.

1.3 Objectifs du projet

1. Collecte et extraction de données hétérogènes,
2. Modélisation et remplissage d'une WIKIBASE,
3. Exploitation des données à travers une interface utilisateur.

1.4 Vue d'ensemble

Suite à cette partie qui introduit l'étendu du projet AlpesTransport les parties suivantes détaillent sa réalisation lors de la période janvier-mars 2023.

La deuxième partie s'intéresse à la **conception** des différentes composantes du projet, en particulier l'**insertion**, la **modélisation** et l'**exploitation** des données dans la WIKIBASE. La troisième partie présente la **réalisation** de ce projet, notamment au travers de son **organisation**, des **technologies utilisées** ainsi que les problèmes rencontrés et les solutions apportées pour y faire face.

La dernière partie de ce document résume les **fonctionnalités implémentées** dans le cadre du projet, propose une **rétrospective** du travail accompli en comparant les résultats obtenus avec la conception initiale du projet, et présente les perspectives d'évolution. Elle souligne également les efforts déployés pour garantir la **maintenabilité du système**.

2 Conception du projet

2.1 Architecture

Le projet est constitué des trois parties principales suivantes :

1. **Insertion** de données dans le WIKIBASE, qui comprend aussi la collecte et l'extraction de ces données,
2. **Modélisation** d'une WIKIBASE, qui met en lien les données centralisées,
3. **Interface utilisateur**, pour exploiter les données de la WIKIBASE.

La WIKIBASE est la partie centrale du projet, où chaque **évènement** reçu déclenche une séquence d'**activités** qui finit par faire appelle à elle. Pour mieux comprendre la place de la WIKIBASE dans notre projet et son intéraction avec les différentes composantes, voici la modélisation de processus du projet :

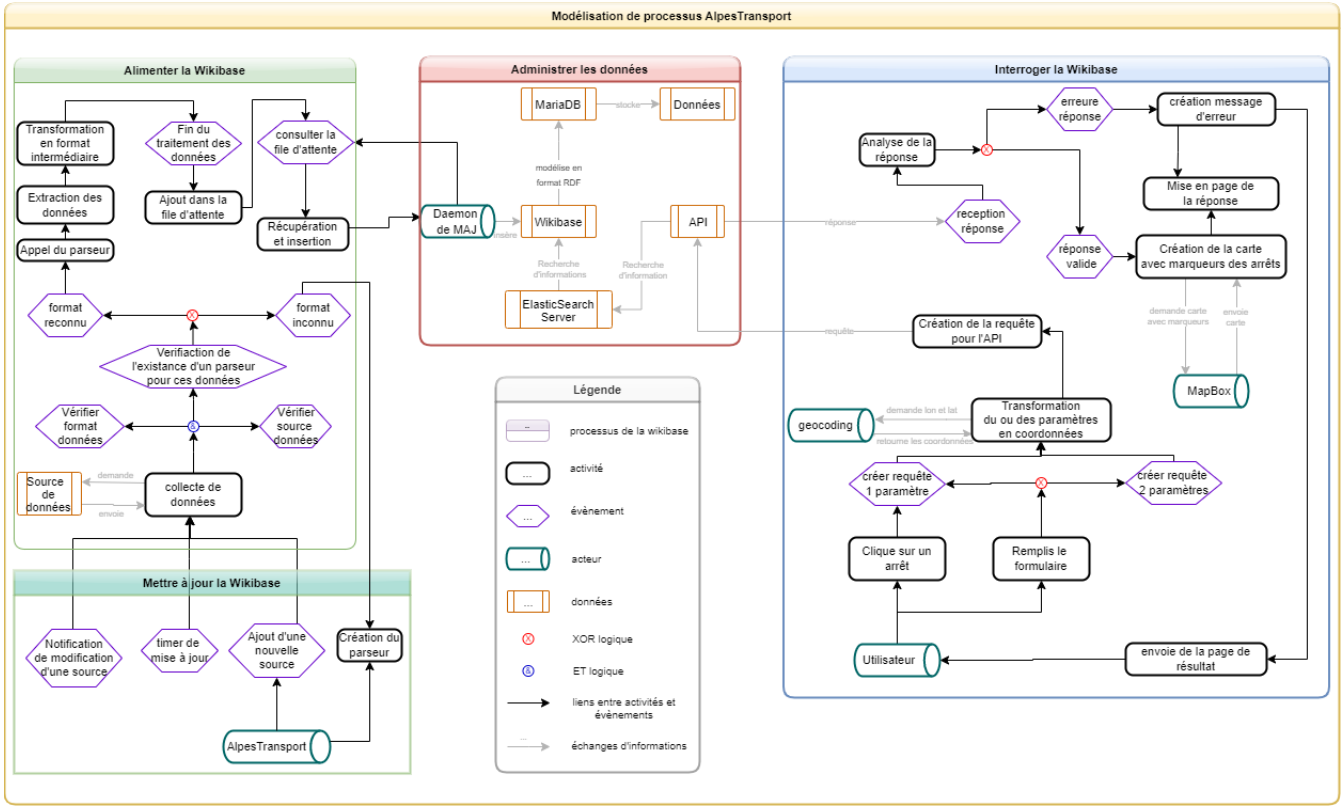


FIGURE 3 – Modélisation ARIS du projet AlpesTransport

2.2 Collecte et Extraction de données

2.2.1 Les sources de données

Nous disposons des **trois sources** de données suivantes (avec leurs format de données respectifs) :

1. SNCF
   - *.csv* pour les référentiels des gares et les tarifs
   - GTFS pour les horaires
2. STAS
   - *.xml* pour les informations sur les horaires et les arrêts des lignes de bus
   - *.html* : webscrapping sur le site de la STAS pour récupérer les tarifs
3. RegionRhoneAlpes
   - GTFS , *.txt* pour les arrêts et les horaires des lignes de bus de la région

2.2.2 Principe général de l'extraction de données

Du fait de l'**hétérogénéité** des formats des données, chaque type de fichier de **chaque source** est associé à un *parseur* qui extrait les informations pertinentes pour le projets (i.e lié aux : trajets, arrêts, horaires et tarifs).

Nous établissons donc dans un premier temps un schéma *global as view*, dans lequel notre WIKIBASE **s'adapte** aux sources de données **locales**. Nous verrons dans la partie *"Insertion des données dans la* WIKIBASE*"* comment ce modèle à été inversé pour devenir un *local as view* afin de faciliter l'**intégration de nouvelles sources** de données à la WIKIBASE.

Ces *parseurs* effectuent les actions ordonnées suivantes :

1. Chargement des données,
2. Nettoyage des données,
3. Transformation des données en un format intermédiaire unique.

2.2.3 Chargement des données par un parseur

Pour **charger** nos documents structurés (*.csv* , GTFS ,*.xml* , *.txt*) collectés à partir des sources les *parseurs* utilisent la fonction *read* correspondant au type du document (read_csv, read_gtfs, read_xml, read_txt) de la librairie python **pandas**.

Cette fonction permet de charger les données dans des *dataframes*, qui sont des tableaux structurés possédants des **colonnes nommés**. Ces noms de colonnes sont utilisés lors du processus d'extraction de données.

Pour le *webscrapping*, le *parseur* récupère la page à l'aide d'une requête de type GET au site et en extrait le contenus (dans notre cas le tableau des tarifs), qui peut ensuite est convertis en *dataframe* comme le reste des données (en le considérant comme un document xml).

Les sous-parties ci-dessous donnent un apperçut de la structure des **données brutes** et des **données chargée**s dans un *dataframe* pour chaque source de données.

2.2.3.1 SNCF

Le *parseur* de la SNCF traite deux types de formats de données : GTFS et *.csv* . Les fichiers *.csv* référentiel des gares et des tarifs permettent de récupérer les **nom des gares** ainsi que leurs **localisation**s et leurs **tarifs** identifiées par leurs **code uic**.

```
Code plate-forme;Code gare;Code UIC;Intitulé plateforme;Code postal;Code Commune;Commune;Code département;Département;Longitude;Latitude;RG;Intitulé gare;UT;TVS;WGS 84
00151-1;151;87764290;Donzère;26290;116;Donzère;26;Drôme;4.7046262;44.4433646;GARES C RHONE;Donzère;DONZERE GARE;DOE;44.4433646, 4.7046262
00162-1;162;87763029;Valence TGV Rhône-Alpes Sud;26958;4;Alixan;26;Drôme;4.978703;44.991545;GARE VALENCE TGV;Valence TGV Rhône-Alpes Sud;VALENCE TGV RHONE ALPES SUD;VAL
00166-1;166;87761841;Die;26150;113;Die;26;Drôme;5.3632053;44.7582549;GARES C RHONE;Die;DIE GARE;DIE;44.7582549, 5.3632053
00167-1;167;87761817;Saillans;26340;289;Saillans;26;Drôme;5.1940397;44.6944365;GARES C RHONE;Saillans;SAILLANS GARE;SIA;44.6944365, 5.1940397
00176-1;176;87761247;Livron-sur-Drôme;26250;165;Livron-sur-Drôme;26;Drôme;4.8305412;44.7796035;GARES C RHONE;Livron-sur-Drôme;LIVRON GARE;LIV;44.7796035, 4.8305412
```

FIGURE 4 – Données brutes référentiels sncf, format csv

Ces informations viennent compléter celles du fichier GTFS sur les **itinéraires** des trains, et peut être rassemblé grâce au code UIC présent dans les fichiers.

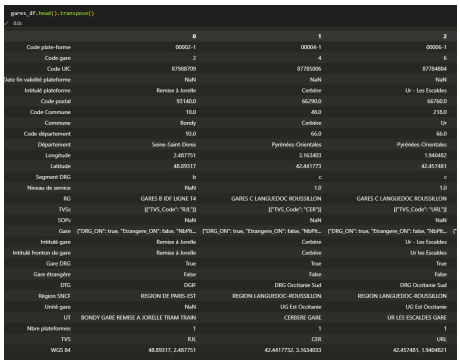
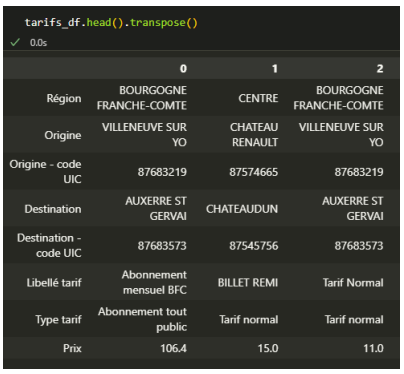


FIGURE 5 – Chargement des référentiels des gares de la sncf



	0	1	2
Région	BOURGOGNE FRANCHE-COMTE	CENTRE	BOURGOGNE FRANCHE-COMTE
Origine	VILLENEUVE SUR YO	CHATEAU RENAULT	VILLENEUVE SUR YO
Origine - code UIC	87683219	87574665	87683219
Destination	AUXERRE ST GERVAI	CHATEAUDUN	AUXERRE ST GERVAI
Destination - code UIC	87683573	87545756	87683573
Libellé tarif	Abonnement mensuel BFC	BILLET REMI	Tarif Normal
Type tarif	Abonnement tout public	Tarif normal	Tarif normal
Prix	106.4	15.0	11.0

FIGURE 6 – Chargement des données des tarifs de la sncf

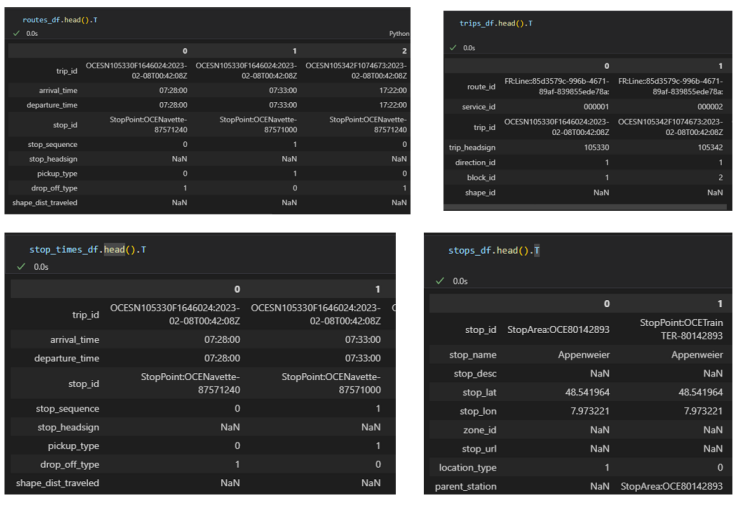


FIGURE 7 – Chargement des données du fichier gtfs sncf

Le format GTFS est une archive *.zip* de fichier *.txt* , dont les données sont reliées entre elles par des id, qui établissent la liaison entre les fichier.
Voici ces liens :

- route.txt est lié à trips.txt par route_id
- trips.txt est lié a stop-times.txt par trip_id
- stop-times.txt est lié à stop.txt par stop_id

2.2.3.2 Région Auvergne Rhône-Alpes

Les données des bus de la région sont aussi au format GTFS , mais il sont plus complet que la sncf, en effet ils contiennent directement les noms et coordonnées des arrêts.

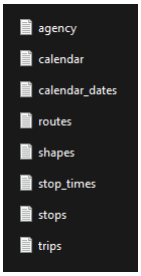


FIGURE 8 – Données brutes gtfs région

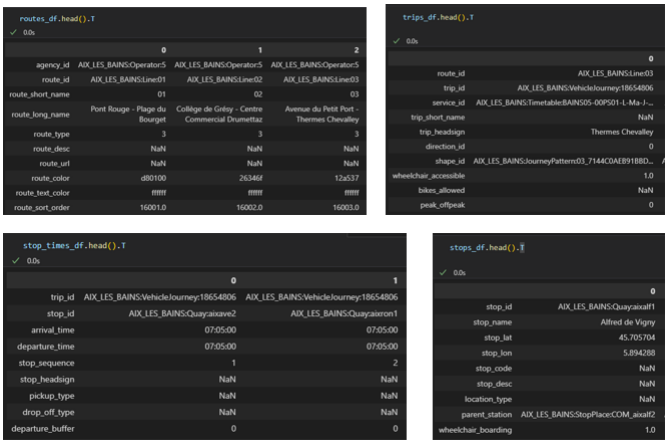


FIGURE 9 – Données format gtfs région

2.2.3.3 STAS

Le *parseur* de la stas traite des données au format *.xml* , à la manière des fichiers GTFS , les données sont composés de plusieurs fichiers *.xml* reliés entre eux.

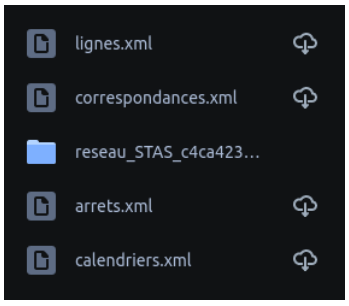


FIGURE 10 – Décomposition de l'archive

Les documents *.xml* sont regroupés au sein d'une archive ZIP composé de 4 fichiers et d'une archive supplémentaire contenant les fichier*.xml* d'itinéraire de chaque lignes.

Chaque fichier *.xml* est composé d'une balise **<dataObject>** et d'une **<frame>**, les données étudiées sont structurées autour de ce balisage.

Prenons l'exemple du fichier "lignes.xml" :

- les balises **<lines>** et **<line>** sont utilisées pour délimiter les caractéristiques d'une ligne, ici une ligne de bus.

```
<Line id="FR:Line:10:" version="any">
        <Name>PLACE JEAN JAURES &lt;&gt; BOURG</Name>
        <TransportMode>bus</TransportMode>
        <PublicCode>10</PublicCode>
</Line>
```

FIGURE 11 – Représentation d'une ligne de bus STAS

- les sous-balises de **<line>** définissent ces caractéristiques, tel que :
  - **<Name>** qui représente le nom de la ligne en question
  - **<TransportMode>** qui représente le mode de transport
  - **<PublicCode>** qui indique le code de ligne public

Les liens entre les fichiers *.xml* sont établis par des **identifiants uniques** contenus dans chaque fichiers, associé à une **ligne** ou un **arrêt** par exemple, qui permet de faire le lien entre chaque contenu de chaque fichier.

Pour comprendre ces liens, prenons l'exemple de l'itinéraire d'un bus qui comprend :

- un identifiant **<fr :Route :48>**
- des caractéristiques tel que :
  - un nom : **<Name>**
  - une distance : **<Dist>**
  - une ligne **<LineRef>** (qui fait référence a l'identifiant unique du fichier lignes*.xml*)

- une séquence de points **<pointsInSequence>**, du fichier arrets*.xml* , qui contient des référence aux points d'arrêt **<RoutePointRef>** entre le point de départ et le point d'arrivée.

```
<Route id="FR:Route:48:" version="any">
        <Name>MAIRIE CHAGNON - CH. DE GAULLE</Name>
        <Distance>0</Distance>
        <LineRef ref="FR:Line:48:">
        </LineRef>
        <DirectionType>inbound</DirectionType>
        <pointsInSequence>
                <PointOnRoute id="FR:PointOnRoute:48_1:" order="1" version="any">
                        <RoutePointRef ref="FR:RoutePoint:48_1:">
                        </RoutePointRef>
                </PointOnRoute>
                <PointOnRoute id="FR:PointOnRoute:48_2:" order="2" version="any">
                        <RoutePointRef ref="FR:RoutePoint:48_2:">
                        </RoutePointRef>
                </PointOnRoute>
```

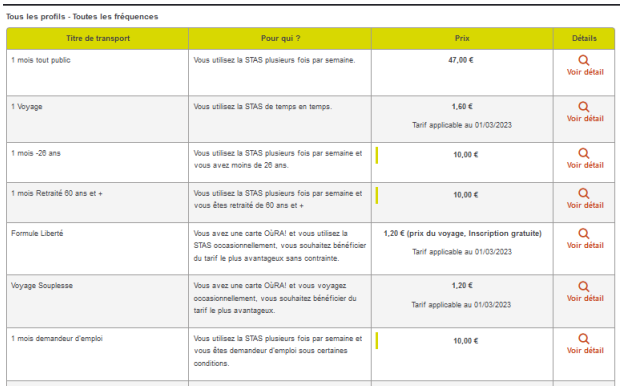
FIGURE 12 – Exemple de route

Pour charger les données du fichier *.xml* dans un *dataframe*, on utilise la fonction *read_csv* du module *pandas* et une requête *xpath* pour charger les noeuds qui nous intéresse dans chaque fichier.

0	1	1	M1	BELLEVU...	3	005BAA	nan	E1-0-1-31	1	Eglise c	0	nan
1	1	1	M1	BELLEVU...	3	005BAA	nan	E1-0-1-31	10	Eglise co...	0	nan
2	1	1	M1	BELLEVU...	3	005BAA	nan	E1-0-1-31	100	Bellevue	1	nan
3	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1000	Bellevue	1	nan
4	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1001	Bellevue	1	nan
5	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1002	Bellevue	1	nan
6	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1003	Bellevue	1	nan
7	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1004	Bellevue	1	nan
8	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1005	Bellevue	1	nan
9	1	1	M1	BELLEVU...	3	005BAA	nan	V1-0-1-31	1006	Bellevue	1	nan

FIGURE 13 – Chargement des données de la stas

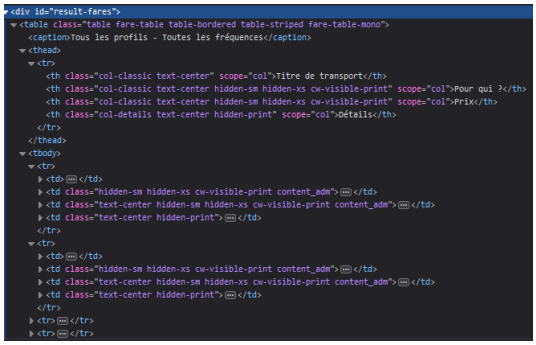
Pour les **tarifs**, on extrait le tableau des tarifs à partir de la page web, puis on insérer son contenu dans un *dataframe* avant de le transformer dans le format intermédiaire.



Tous les profils - Toutes les fréquences

Titre de transport	Pour qui ?	Prix	Détails
1 mois tout public	Vous utilisez la STAS plusieurs fois par semaine.	47,00 €	Voir détail
1 Voyage	Vous utilisez la STAS de temps en temps.	1,60 € Tarif applicable au 01/03/2023	Voir détail
1 mois -26 ans	Vous utilisez la STAS plusieurs fois par semaine et vous avez moins de 26 ans.	10,00 €	Voir détail
1 mois Retraité 60 ans et +	Vous utilisez la STAS plusieurs fois par semaine et vous êtes retraité de 60 ans et +	10,00 €	Voir détail
Formule Liberté	Vous avez une carte OURA! et vous utilisez la STAS occasionnellement, vous souhaitez bénéficier du tarif le plus avantageux sans contrainte.	1,20 € (prix du voyage, Inscription gratuite) Tarif applicable au 01/03/2023	Voir détail
Voyage Souplesse	Vous avez une carte OURA! et vous voyagez occasionnellement, vous souhaitez bénéficier du tarif le plus avantageux.	1,20 € Tarif applicable au 01/03/2023	Voir détail
1 mois demandeur d'emploi	Vous utilisez la STAS plusieurs fois par semaine et vous êtes demandeur d'emploi sous certaines conditions.	10,00 €	Voir détail

FIGURE 14 – Tableau des tarifs sur la page web de la stas



```
<div id="result-fares">
▼<table class="table fare-table table-bordered table-striped fare-table-mono">
  <caption>Tous les profils - Toutes les fréquences</caption>
  ▼<thead>
    ▼<tr>
      <th class="col-classic text-center" scope="col">Titre de transport</th>
      <th class="col-classic text-center hidden-sm hidden-xs cw-visible-print" scope="col">Pour qui ?</th>
      <th class="col-classic text-center hidden-sm hidden-xs cw-visible-print" scope="col">Prix</th>
      <th class="col-details text-center hidden-print" scope="col">Détails</th>
    </tr>
  </thead>
  ▼<tbody>
    ▼<tr>
      ▶<td>…</td>
      ▶<td class="hidden-sm hidden-xs cw-visible-print content_adm">…</td>
      ▶<td class="text-center hidden-sm hidden-xs cw-visible-print content_adm">…</td>
      ▶<td class="text-center hidden-print">…</td>
    </tr>
    ▼<tr>
      ▶<td>…</td>
      ▶<td class="hidden-sm hidden-xs cw-visible-print content_adm">…</td>
      ▶<td class="text-center hidden-sm hidden-xs cw-visible-print content_adm">…</td>
      ▶<td class="text-center hidden-print">…</td>
    </tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
```

FIGURE 15 – Structure html du tableau des tarifs extrait

2.3 Nettoyage des données par le parseur

Les données ainsi chargés sont ensuite **filtrés** afin de ne conserver que les informations qui seront insérées dans la WIKIBASE. Lors de cette étape, les *parseurs* vont aussi **nettoyer les données**, pour que chaque valeur ait une valeur correcte selon son type.

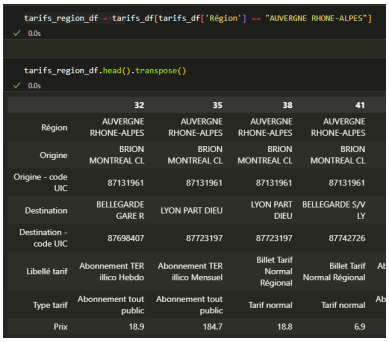
Concrètement, les *parseurs* vont supprimmer les colonnes contenant des données qui ne sont pas utilisées et remplacer par -1 les nombre qui ont la valeur *NaN*. Comme cette étapes est la même pour chaque *parseur*, un exemple pour une seule source de données est fournit pour que cela ne soit pas trop redondant.

2.3.0.1 Un exemple avec les documents de la SNCF

```
gare_region_df.head().transpose()
✓ 0.2s
```

	11	87
Code plate-forme	00062-1	00422-1
Code gare	62	422
Code UIC	87783175	87734707
Intitulé plateforme	Saint-Flour - Chaudes-Aigues	Lavoûte-sur-Loire
Code postal	15100.0	43800.0
Code Commune	187.0	119.0
Commune	Saint-Flour	Lavoûte-sur-Loire
Code département	15.0	43.0
Département	Cantal	Haute-Loire
Longitude	3.106286	3.90541
Latitude	45.035012	45.12145
TVSs	[{"TVS_Code": "SFC"}]	[{"TVS_Code": "LVL"}]
SOPs	NaN	NaN
Gare	{"DRG_ON": true, "Etrangere_ON": false, "NbPlt...	{"DRG_ON": true, "Etrangere_ON": false, "NbPlt...
Intitulé gare	Saint-Flour - Chaudes-Aigues	Lavoûte-sur-Loire
Intitulé fronton de gare	Saint-Flour - Chaudes-Aigues	Lavoûte-sur-Loire
Gare DRG	True	True
Gare étrangère	False	False
DTG	DRG AURA-BFC	DRG AURA-BFC
Région SNCF	REGION AUVERGNE	REGION AUVERGNE
Unité gare	UG Auvergne	UG Auvergne

FIGURE 16 – Nettoyages des données des gares de la sncf



```
tarifs_region_df = tarifs_df[tarifs_df['Région'] == "AUVERGNE RHONE-ALPES"]
✓ 0.0s

tarifs_region_df.head().transpose()
✓ 0.0s
```

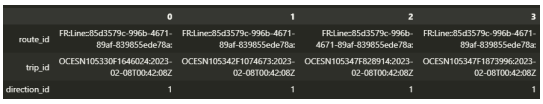
	32	35	38	41
Région	AUVERGNE RHONE-ALPES	AUVERGNE RHONE-ALPES	AUVERGNE RHONE-ALPES	AUVERGNE RHONE-ALPES
Origine	BRION MONTREAL CL	BRION MONTREAL CL	BRION MONTREAL CL	BRION MONTREAL CL
Origine - code UIC	87131961	87131961	87131961	87131961
Destination	BELLEGARDE GARE R	LYON PART DIEU	LYON PART DIEU	BELLEGARDE S/V LY
Destination - code UIC	87698407	87723197	87723197	87742726
Libellé tarif	Abonnement TER illico Hebdo	Abonnement TER illico Mensuel	Billet Tarif Normal Régional	Billet Tarif Normal Régional
Type tarif	Abonnement tout public	Abonnement tout public	Tarif normal	Tarif normal
Prix	18.9	184.7	18.8	6.9

FIGURE 17 – Nettoyages des données des tarifs de la sncf

Voici les données nettoyés d'un fichier gtfs de la sncf :

	0	1	2	3	4
route_id	FR:Line::00F2577A-6A87-42E0-95F3-0735 1E48C2F6:	FR:Line::00F7208C-CEBC-4521-A792-6EC3ABB65811:	FR:Line::0128E1D5-9183-4D58-B1CF-F5AASA64A037:	FR:Line::0202671B-7107-429E-A378-473C55E0254C:	FR:Line::022B77D9-D121-4DCB-B808-FB2F7931866B:
route_long_name	Bening - Sarreguemines	Saint-Étienne - Roanne	Marseille - Toulon - Hyeres	Montpellier Saint-Roch - Avignon Centre	Angers St Laud - Cholet

FIGURE 18 – données du fichier routes.txt



	0	1	2	3
route_id	FR:Line::85d3579c-996b-4671-89af-839855ede78a:	FR:Line::85d3579c-996b-4671-89af-839855ede78a:	FR:Line::85d3579c-996b-4671-89af-839855ede78a:	FR:Line::85d3579c-996b-4671-89af-839855ede78a:
trip_id	OCESN105330F1646024:2023-02-08T00:42:08Z	OCESN105342F1074673:2023-02-08T00:42:08Z	OCESN105347F828914:2023-02-08T00:42:08Z	OCESN105347F1873996:2023-02-08T00:42:08Z
direction_id	1	1	1	1

FIGURE 19 – données du fichier trips.txt

	0	1	2
trip_id	OCESN105330F1646024:2023-02-08T00:42:08Z	OCESN105330F1646024:2023-02-08T00:42:08Z	OCESN105342F1074673:2023-02-08T00:42:08Z
arrival_time	07:28:00	07:33:00	17:22:00
stop_id	StopPoint:OCENavette-87571240	StopPoint:OCENavette-87571000	StopPoint:OCENavette-87571240

FIGURE 20 – données du fichier stop.txt

	0	1	2	3
stop_id	StopArea:OCE80142893	StopPoint:OCETrain TER-80142893	StopArea:OCE80142901	StopPoint:OCETrain TER-80142901
stop_name	Appenweier	Appenweier	Legelshurst	Legelshurst
stop_lat	48.541964	48.541964	48.558617	48.558617
stop_lon	7.973221	7.973221	7.913533	7.913533
parent_station	NaN	StopArea:OCE80142893	NaN	StopArea:OCE80142901

FIGURE 21 – données du fichier stops_times.txt

On peut voir ici que seules les **colonnes pertinentes** ont été conservé et que les lignes avec des **id manquant** ont été supprimés.

2.3.0.2 Transformation en un format intermédiaire

Après avoir été chargées et nettoyées, les données sont ensuite **transformées dans un format commun** avant de pouvoir être insérées dans la WIKIBASE. Cela permet d'être plus modulaire et de faciliter l'insertion de données provenant de sources diverses.
Le **format intermédiaire** choisi est un **dictionnaire python** avec les objets suivants :

- "entity" : "item"
  - "label" : clé contenant le label de l'item
  - "property" : clé contenant une liste de *statements* de l'item.

- "entity" : "property", il y en a deux types une dont les valuers sont des string, d'autres dont les valeurs sont des items.
  - "label" : le label de la propriété
  - "type" : string, item ou listItem suivant le type de la valeur de la propriété.
  - "value" : la valeur de la propriété selon sont type.

Remarque : grâce à ce format intermédiaire il est plus facile d'ajouter de **nouvelles sources** de données, ou de modifier le schéma de la wikibase. En effet, s'il l'on veut rajouter des propriétés ou des items ou les deux, il suffit de les rajouter dans le format intermédiaire et c'est tout. Ce format médiateur permet donc une meilleure **flexibilité** du modèle de la WIKIBASE.

FIGURE 22 – Exemple de données au format intermédiaire

2.4 Modélisation des données

2.4.1 Conception du modèle de données

Le projet utilise une WIKIBASE pour stocker les informations. Cette technologie nécessite de réaliser une représentation des données sous forme de graphes RDF.

Nous avons donc créé un modèle de données à partir des **entités** et de leurs **relations**, extraites de nos différentes sources d'informations. Les entités identifiées comprennent les **entreprises de transport** (STAS, SNCF, région Auvergne-Rhône-Alpes), les **lignes de transport** (bus, train) et les lieux des **arrêts**.

Nous avons utilisé le modèle Sujet<Prédicat>Objet en utilisant trois éléments de base : Entités, Propriétés, Littéraux.

Nous avons étudié les différents documents dont nous disposons pour identifier les entités et littéraux apparaissant dans ces documents. Ensuite, nous avons identifié les relations entre ces différents éléments pour créer un **graphe** qui représente la structure des données stockées dans notre wikibase.

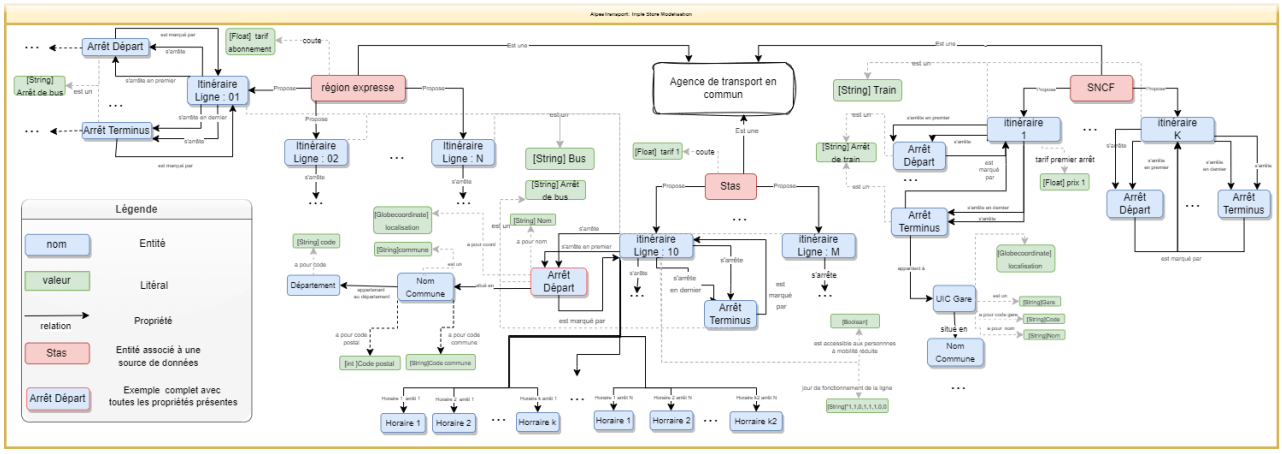


FIGURE 23 – Modélisation de la Wikibase

2.4.2 Choix de représentation de la sémantique des données

Notre WIKIBASE contient les données de trois **agences de transport en commun**, chacune proposant aux usagers un ensemble d'**itinéraires** à emprunter. Chacun de ces itinéraires est représenté par un item itinéraire, identifié par son numéro de ligne.

Chaque itinéraire possède trois propriétés qui seront exploitées pour l'application utilisateur : des **arrêts**, des **horaires** et un **coût**. Les itinéraires possèdent deux types de propriétés pour les relier aux arrêts : *s'arrête*, qui permet de retrouver la liste des arrêts auxquels s'arrête la ligne, et s'arrête en n-ième, qui permet d'ajouter une sémantique d'ordre aux arrêts.

Chaque arrêt possède des horaires pour chaque ligne. Pour cela, chaque itinéraire est relié à un item horaire par une propriété *horaire k arrêt n-ième* qui permet d'ajouter une sémantique quant à l'arrêt et le numéro de l'horaire considéré. Ces **horaires** possèdent une propriété qui les lient aux jours de services de la ligne, ce qui permet de stocker des horaires différents suivant le jour considéré (par exemple différencier les horaires de la semaine et du week-end).

Les tarifs sont un peu plus particuliers car ils peuvent correspondre à des **tarifs globaux** (abonnement) ou un **tarif unique** (prix d'un voyage). Ainsi, les tarifs de la STAS sont constants et donc associés (par une propriété) à cet item. Pour les autres transports (train et bus région), le tarif dépend du trajet effectué, ce qui est représenté par une propriété *tarif debute n-ième arrêt, s'arrête p-ième arrêt* qui correspond au **coût entre l'arrêt n et l'arrêt p** de l'itinéraire.

Le fait d'utiliser une **combinaison propriétés-item** pour les arrêts et propriétés-littérale pour les horaires et les tarifs, au lieu de les stocker sous forme de liste, va permettre d'exploiter tout le potentiel du modèle RDF pour questionner les données.

En effet, l'utilisation de propriétés nommés permet d'associer une sémantique d'ordre pour les horaires et les arrêts, et ainsi permettre de les **retrouver directement** grâce à une requête SPARQL à l'API, sans avoir à parcourir tous les arrêts ou horaires à chaque fois, comme c'est le cas pour des listes.

2.5 Insertion des données dans la Wikibase

Une fois le *preprocessing* des données complétés, nous disposons de ces dernières dans un **format homogène** (i.e dictionnaire python). Le programme d'insertion parcourt ensuite ces structures de données et **insère** les items et propriétés qu'elles contiennent dans la *wikibase* à l'aide de la librairie *wikibaseIntegrator*.

Cette librairie, agit comme un médiateur entre notre programme et l'API WIKIBASE, ce qui nous simplifie l'interaction avec cette dernière, notamment pour la connexion. *(on n'a pas besoin de se préoccuper du crsf token, le médiateur s'en occupe lors de l'appelle à la fonction de connexion)*

Lors du parcours du dictionnaire, le programme d'insertion **parcours item par item** et vérifie les conditions suivantes :

1. propriété de type item ou listItem ?
2. Non
   (a) L'item existe ? Si non on le crée
   (b) On crée la propriété
3. Oui
   (a) Pour chaque item on vérifie s'il existe, et on le crée si non.
   (b) ensuite on vérifie que l'item de base existe ou on le crée si non
   (c) et enfin on crée la propriété

Ainsi, le programme d'insertion des données est vue comme un médiateur entre les **sources** de données et notre **wikibase** en établissant un schéma *local as view*. En effet les sources de données doivent **s'adapter au format intermédiaire** pour pouvoir être intégré dans notre wikibase, cela permet de simplifier l'insertion des données.

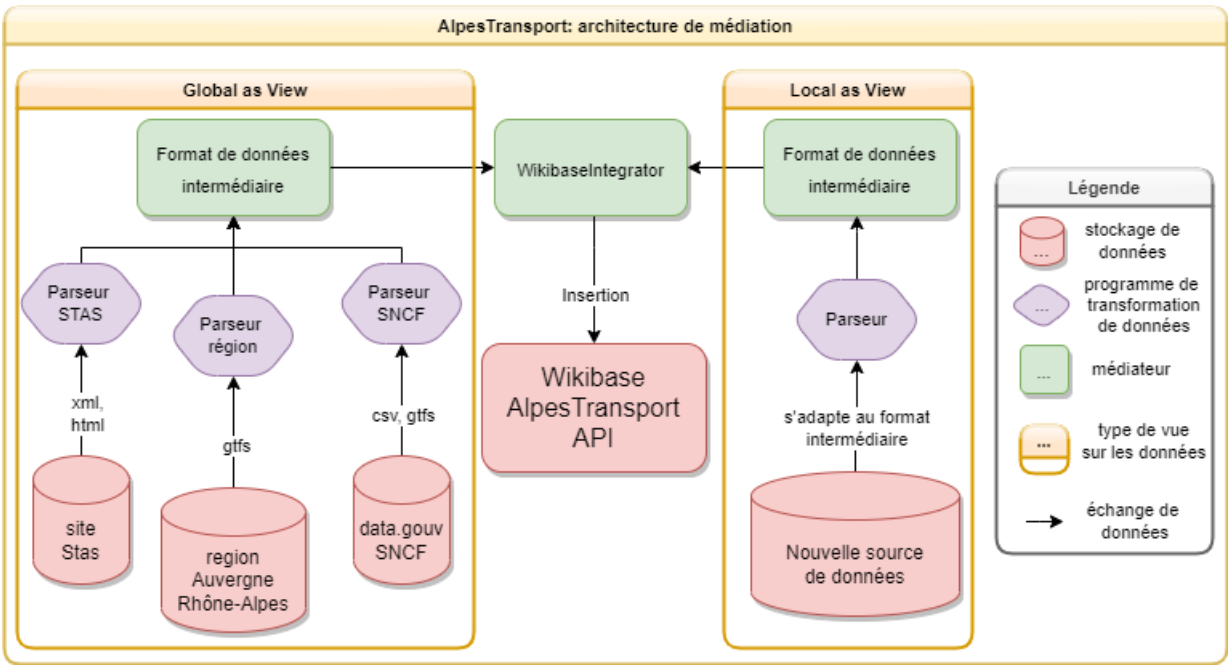


FIGURE 24 – Architecture de médiation de l'insertion de données

2.6 Exploitation des données

L'**objectif final** pour l'utilisateur et de lui permettre d'obtenir des informations sur les transports en communs. Cette échange d'information entre la WIKIBASE et l'utilisateur va se faire au travers d'une **application web**.

Elle permet de questionner la WIKIBASE via un formulaire pour demander les informations suivantes :

- consulter les horaires des transports en communs à un arrêt,
- obtenir les tarifs d'un déplacement,
- les différentes options de transport en commun pour aller d'un point A à un point B.

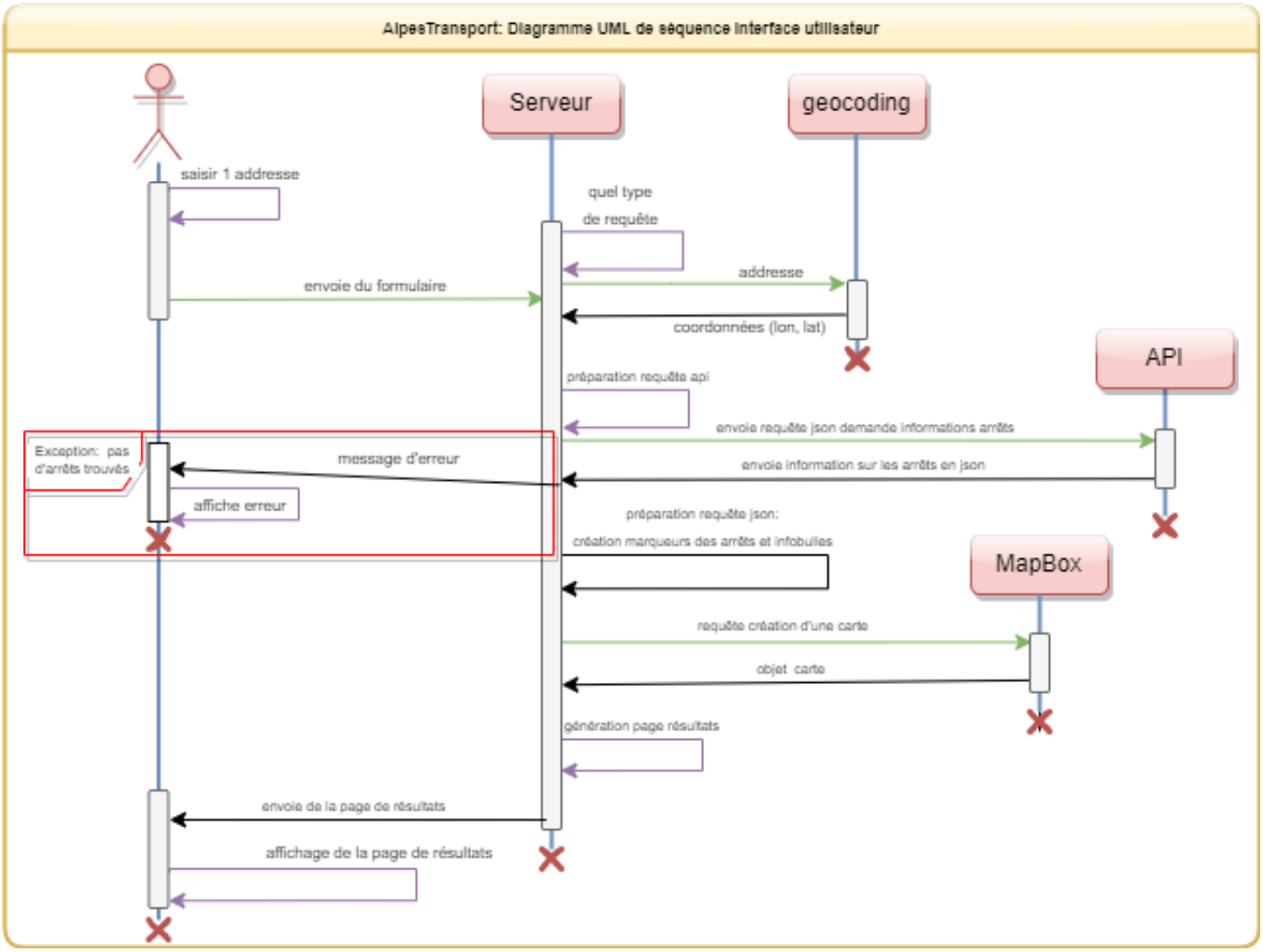


FIGURE 25 – UML, diagramme de séquence d'une requête de l'interface utilisateur

3 Réalisation du projet

3.1 Organisation du projet

3.1.1 Présentation des outils utilisés

Les outils suivants on été utilisés pour réaliser l'organisation du projet :

Jira

Cet outil nous a permis de créer un ***taskboard*** afin de nous répartir les tâches, puis de le synchroniser avec un diagramme de Gant afin de réaliser une **planification** de la réalisation de ces tâches.

GitLab

Cet outil nous a permis de réaliser le **partage** et la **gestion des version** des **codes sources** du projets.

Overleaf

Cet outil a été utiliser pour partager et créer, en LaTeX, les différents documents de documentations demandés.

3.1.2 Gestion du planning du projet

Le **planning** a été réalisé avec Jira, qui nous a permis de le créer à partir des **tickets**, représentant une **tâche à réaliser**, présent dans notre *taskboard*.

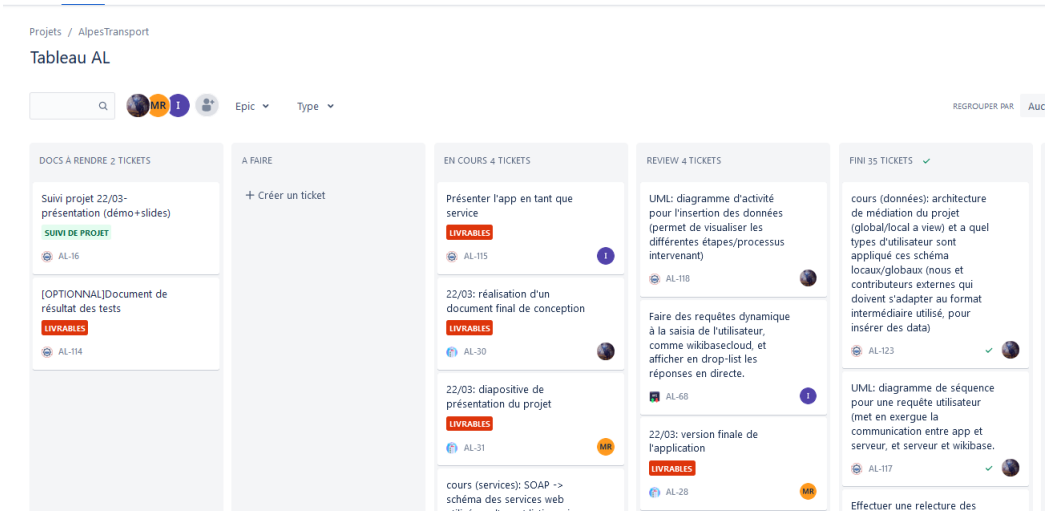


FIGURE 26 – Jira : aperçut du taskboard

Ce ***taskboard*** nous a permis de créer des tickets pour diviser et nous répartir le travail parmi les tâches du projet à faire. Il a évolué tout au long du projet, au fil de l'avancement du projet permettant de faire état de celui-ci, avec les différentes colonnes auxquelles appartiennent les tickets.

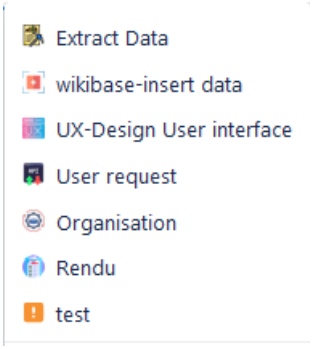


FIGURE 27 – Jira : type de ticket

FIGURE 28 – Jira : epics

Chacun de ces tickets est associé à un **type de tâche**, et un **epic** permettant de l'associer à un objectif à accomplir par rapport au projet et une **date d'échéance** pour celui-ci, identifié dans le planning par des couleurs différentes.

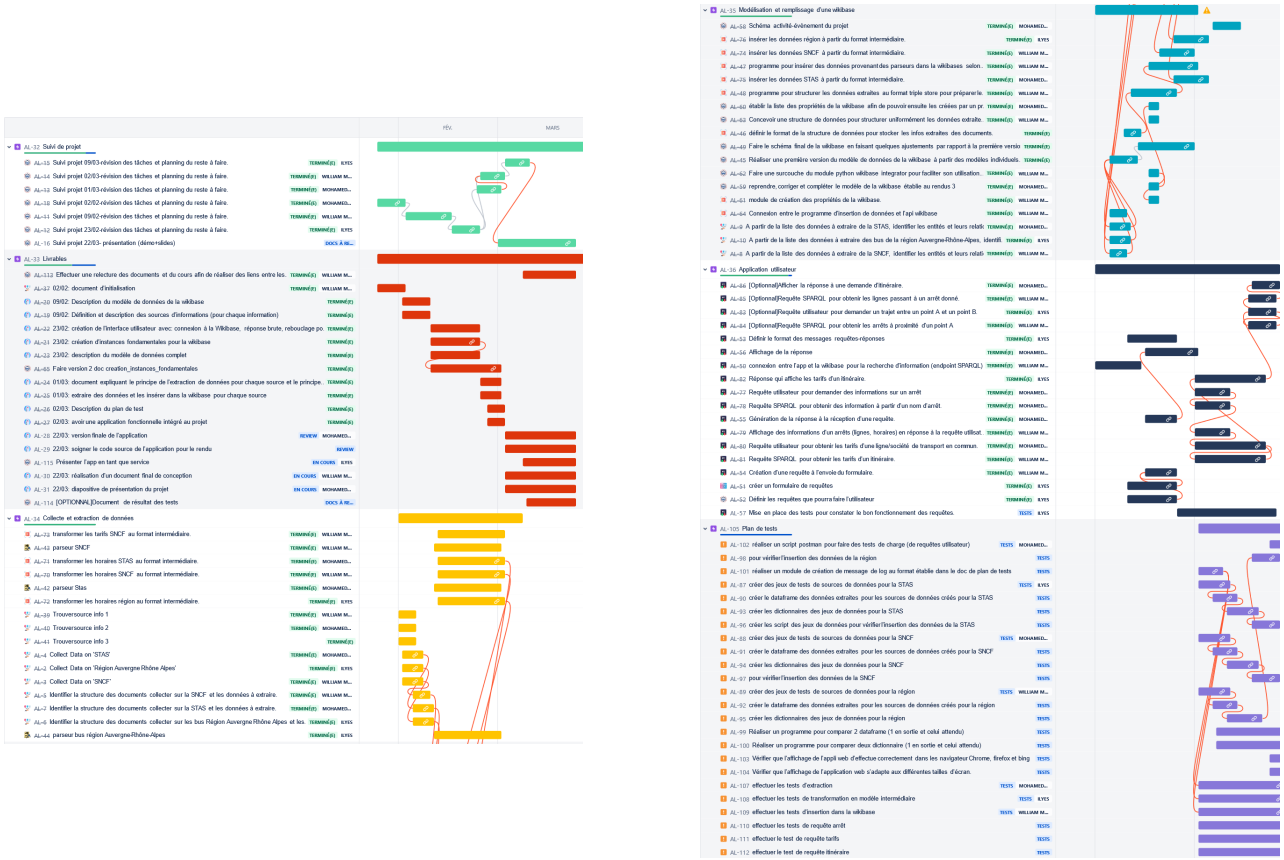


FIGURE 29 – Jira : aperçut du planning final

3.2 Ingestion de données dans la wikibase

L'ingestion de données est réalisé par des *informaticiens*, le programme d'insertion n'a donc pas pas d'interface graphique et interagit avec l'utilisateur directement via un **terminal**.

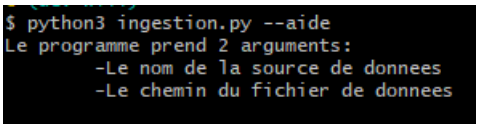


FIGURE 30 – Programme d'ingestion de données : aide utilisation

Ce programme tient un **index des sources de données** associé avec un ou plusieurs format de fichier qu'il est possible de consulter.

FIGURE 31 – Programme d'ingestion de données : afficher les sources enregistrées

Lorsque le programme est appelé avec une source de données inconnu, il est proposé à l'utilisateur d'ajouter cette source de données à l'**index** avec le format du fichier de données donné. Une fois la nouvelle source ajouté, il devra placer un parseur éponyme dans le dossier */parseur*, qui sera appelé pour charger et transformer les données suivant le format intermédiaire imposé par notre wikibase.

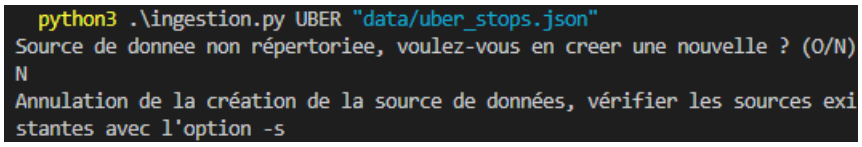


FIGURE 32 – Programme d'ingestion de données : refus création nouvelle source

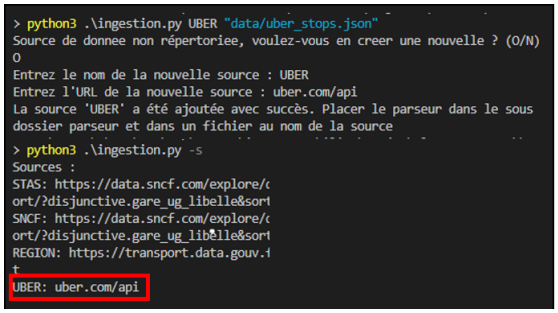


FIGURE 33 – Programme d'ingestion de données : ajout d'une nouvelle source

Lorsque le programme est appelé avec une source de données existante et un **format de fichier compatible** avec celui enregistré, alors l'insertion des données est réalisée et le programme affiche à la fin les **changements réalisés** par l'exécution du programme.

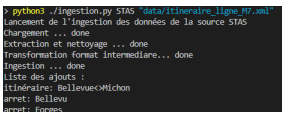


FIGURE 34 – Programme d'ingestion de données : ingestion de données

Le flux d'exécution de cette application d'insertion de données est détaillé dans le digramme d'activité suivant.

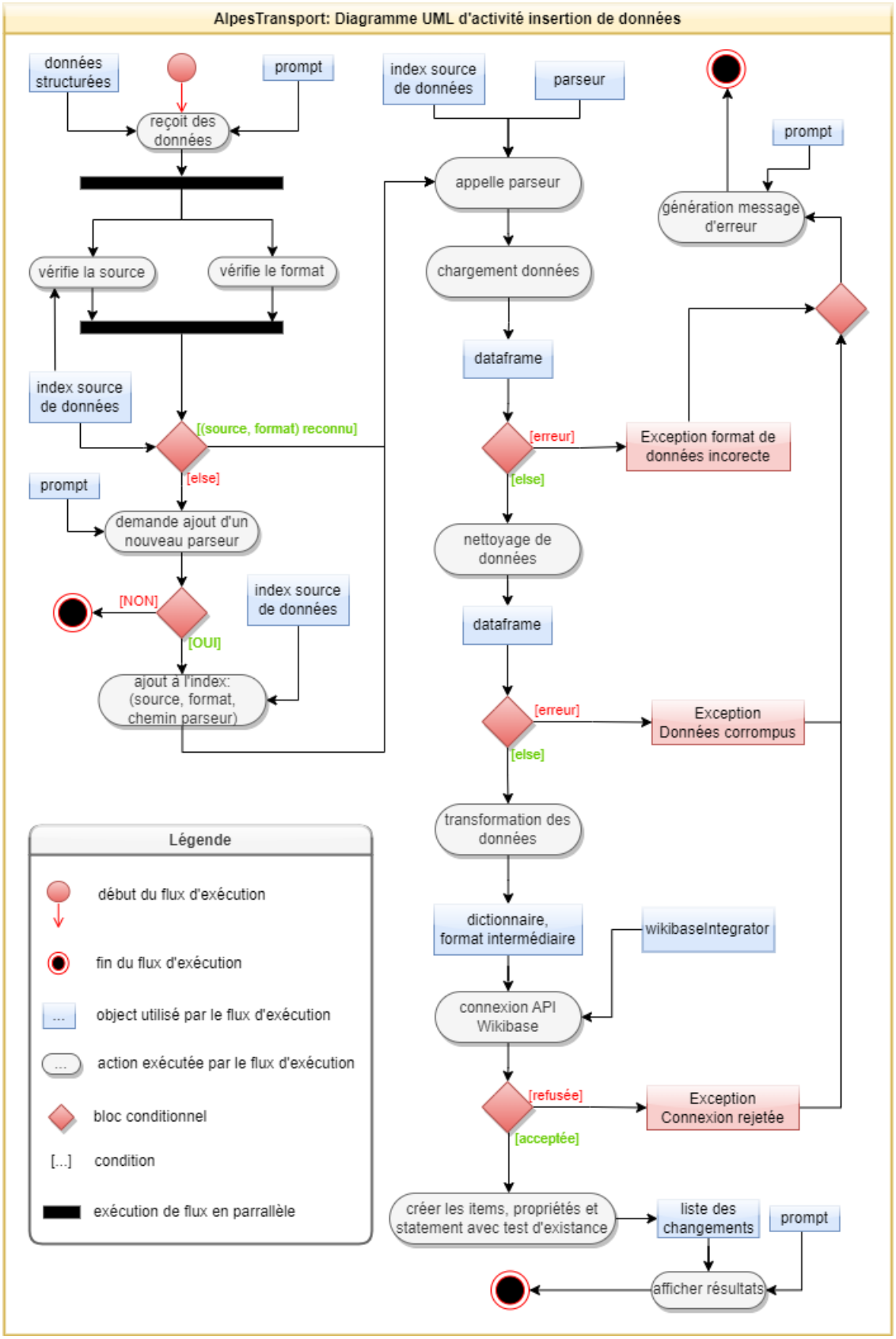


FIGURE 35 – Diagramme UML d'activité : ingestion de données

3.3 Utilisation de l'application utilisateur

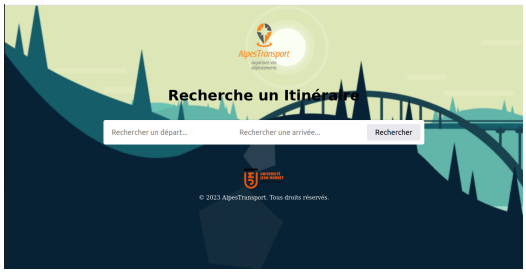


FIGURE 36 – Application : page d'accueil

L'application utilisateur se compose de trois parties distinctes :

1. l'**interface utilisateur**, au travers de laquelle l'utilisateur effectue ses recherches d'informations,
2. la partie **serveur** qui va communiquer avec la WIKIBASE pour en **extraire les informations**,
3. la partie **serveur** qui va communiquer avec l'api MapBox afin de constituer la **page de résultat**.

L'interface utilisateur permet de sélectionner soit un **point de départ** pour rechercher des **informations sur les arrêts** en ce point, soit **un point de départ et un point d'arrivée** pour rechercher l'**itinéraire** ou la combinaison d'itinéraires pour effectuer le déplacement.

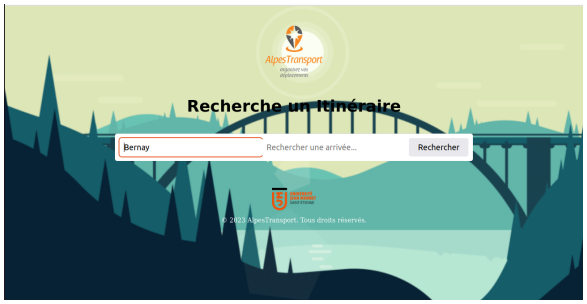


FIGURE 37 – recherche d'arrêts à un point donné

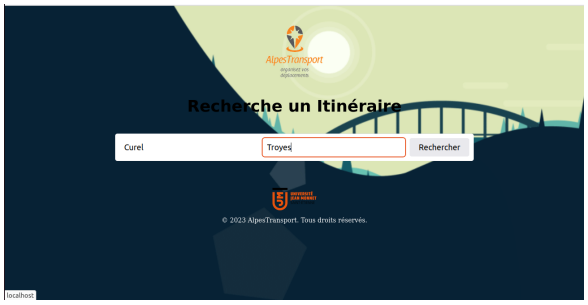


FIGURE 38 – recherche pour se déplacer entre deux points

FIGURE 39 – Application : recherche avec l'interface utilisateur

Ensuite, une fois que l'utilisateur valide l'envoie du formulaire, ce dernier est reçu par le serveur qui va créer un requête destiné à l'API WIKIBASE avant de l'envoyer.

Si une erreur est retournée par l'API, cela signifie que l'endroit entré par l'utilisateur ne possède pas d'arrêt, le serveur redirige l'utilisateur sur une page d'erreur.

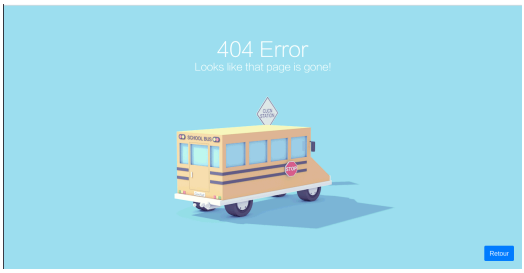


FIGURE 40 – Application : page d'erreur

Si l'arrêt est bien situé dans la WIKIBASE, une deuxième requête est effectuée à l'API WikibaseCloud pour rechercher une propriété précise : **l'itinéraire passant par cet arrêt**, qui permet

d'obtenir les itinéraires passant par cet arrêt.

Ensuite, la valeur de cette propriété est récupérée, puis une nouvelle recherche est effectuée sur l'élément correspondant qui s'avère être un autre item représentant un itinéraire. Avec ce nouvel item, les *claims* qui sont les propriétés de l'item sont récupérés grâce à une requête à l'API.

Maintenant, passons à la partie de la construction de la réponse. Elle se fait avec une requête spécifique pour récupérer, pour chaque propriété "s'arrête en...", la valeur de la propriété qui est elle-même un item représentant **un arrêt**.

Puis, avec cette valeur, une nouvelle recherche est effectuée pour **récupérer l'item arrêt**, ce qui permet d'obtenir ces **coordonnées** ainsi que son **type** et son **nom**.

Ensuite, une mise en forme est effectuée et les valeurs sont stockées dans des listes. En utilisant Flask et Jinja2, les variables sont ensuite utilisées pour construire la requête à l'API MapBox afin de construire la **carte** qui sera afficher sur la **page de résultat**.

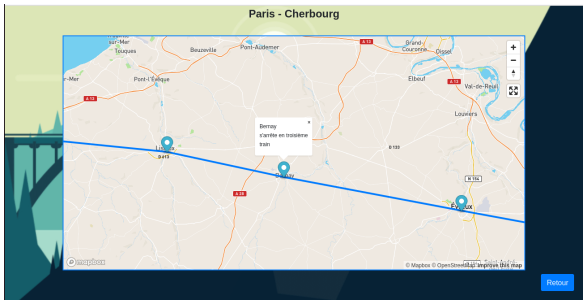


FIGURE 41 – recherche d'arrêts à un point donné

FIGURE 42 – recherche pour se déplacer entre deux points

FIGURE 43 – Application : visualisation des résultats avec l'interface utilisateur

Le diagramme d'activité ci-dessous permet de visualiser le flux d'exécution d'une requête utilisateur de sa saisie à l'affichage de sa réponse.

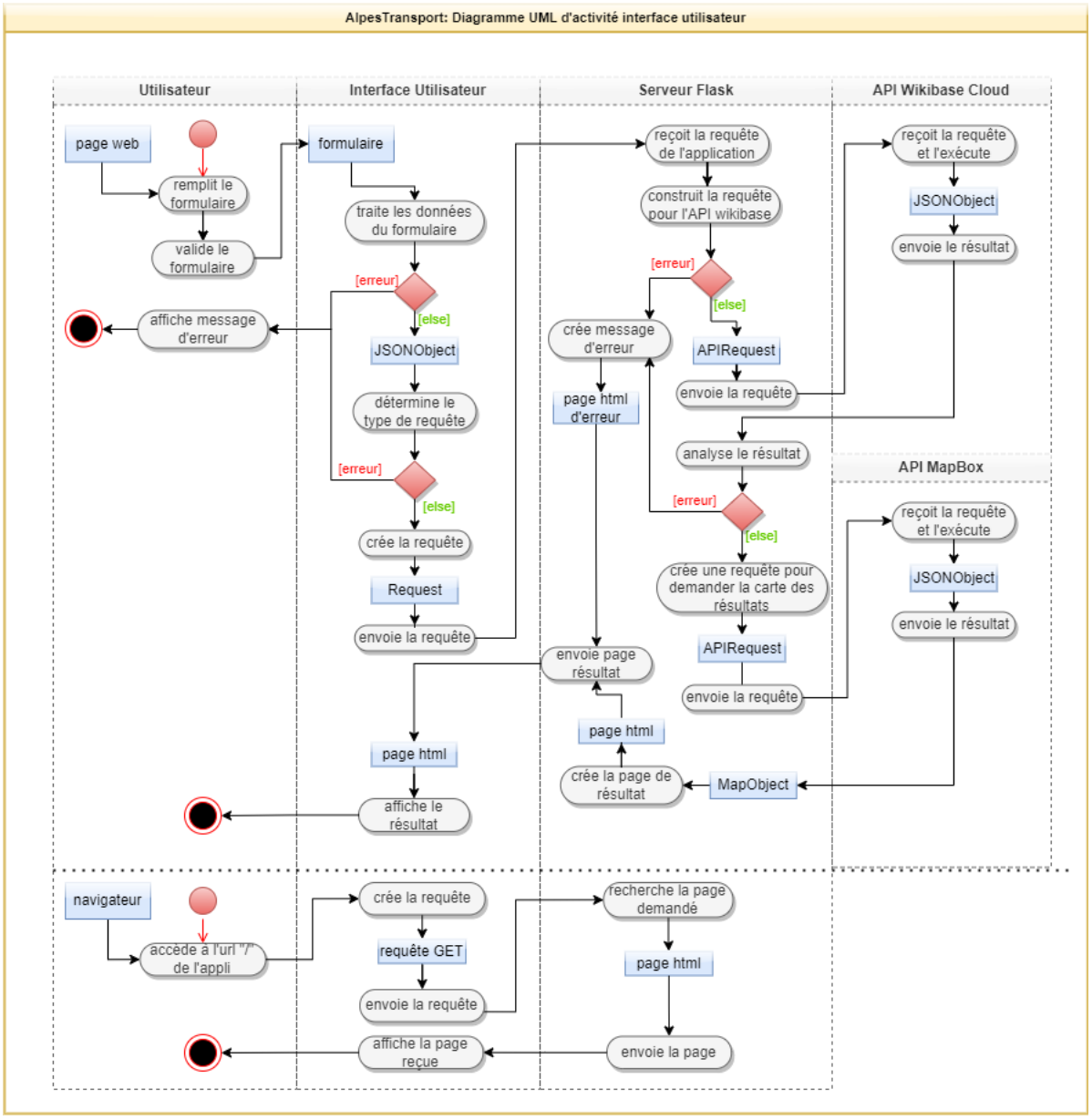


FIGURE 44 – Diagramme UML d'activité de l'utilisation de l'interface utilisateur

3.4 Problèmes rencontrés et solutions apportées

3.4.1 Trou dans les données

Une étape important de l'ingestion de données est le **nettoyage des données**. En effet, avant de les insérer, il faut s'assurer que les données considérées soit **complètes**. Pour cela nous avons supprimer dans les fichier les lignes ayant des données manquantes, au profit de la fiabilité des données insérées.

3.4.2 Problème serveur elasticsearch

Durant le début du projet nous avons fait face à un problème margeur avec notre WIKIBASE locale : il était impossible de rechercher des items ou des propriétés.

Après une étude approfondis des fichiers de *logs* de l'image docker nous avons pu identifier l'origine du problème : le serveur **elastic search** responsable de toutes les recherches de données dans la wikibase ne se lançait pas à cause du manque de mémoire **RAM virtuelle alloué**.

Après avoir effectué quelques recherches sur se problème, nous avons qu'il fallait augmenter la valeur de la variable globale ***vn.max_max_count*** pour autoriser au serveur d'utiliser plus de RAM.

Cependant, en augmentant cette valeur le serveur **elastic search** à tendance à saturer la RAM. Nous avons donc fait une demande d'accès à wikibasecloud pour résoudre ce problème. Cette demande à été accéder, et nous avons pu continuer la suite du projet en ayant accès à une WIKIBASE dans le ***cloud***.

3.4.3 Masse de données à insérer

L'algorithme d'insertion parcours les données d'une source et crée ses items et propriétés. Du fait de la masse de données présent dans les sources *(plus de trente mille lignes pour les données de la stas par exemple)* cette algorithme peut prendre beaucoup de temps à s'exécuter.

Nous avons donc décider de séparer l'insertion en plusieurs petit *batch* à exécuter en parallèle. Pour les faire tourner en continu sans interruption, nous avons fait tourner cette algorithme sur *colab*

(plateforme collaboratrice d'édition de upyter notebook permettant l'exécution de 4 programme en parallèle)

ce qui a permis de faire fasse à une **ingestion massive de données** *(qui a pris plus de 10h à faire tourner).*

4 Conclusion

4.1 Fonctionnalités implémentées dans le cadre du projet

Voici une liste récapitulative des principales fonctionnalités implémenté dans le projet ALpesTransport :

- ingestion de données hétérogènes issue de 3 sources distinctes,
- modélisation RDF des données dans une WIKIBASE,
- interface utilisateur pour chercher des informations sur un arrêt ou un itinéraire d'un point A à un point B.

4.2 Rétrospective

4.2.1 Analyse des écarts par rapport aux objectifs

Nous n'avons finalement traiter **ni les tarifs ni les horaires** des arrêts pour un itinéraire donnée dû fait du manque de données d'une part, et d'autre part de la complexité des données déjà traités pour organiser les déplacements d'un utilisateur..

Cela nous a permis de nous consacrer pleinement aux arrêts et itinéraires, et ainsi dédier notre projet au **déplacement avec les transport en communs**.

4.2.2 retours d'expérience

Ce projet nous a permis de **mettre en pratique les notions vue en cours** d'intéropérabilité comme : la sémantique web avec le schéma RDF, l'utilisation de médiateurs pour simplifier les échanges de données, , la délégation de service avec l'utilisation d'une API comme MapBox, la mise en place d'une architecture activité-évènement.

Il a aussi été l'occasion de **découvrir de nouvelles technologies** comme WIKIBASE/WIKIBASECLOUD et SPARQL ainsi que le questionnement d'un schéma RDF pour trouver des informations.

A travers ce projet, nous avons aussi pû améliorer nos **compétences organisationnelles** notamment avec la gestion évolutive d'un planning en utilisant jira.

Par ailleurs son utilisation sera à approfondir pour exploiter pleinement les fonctionnalités offertes par cet outil, en particulier l'intégrer à gitlab pour synchroniser nos tickets avec les développement en cours.

Il est aussi intéressant de considérer l'utilisation de l'**intégration continu** pour automatiser les tests réaliser lors du projet, avec les *pipelines* CI/CD proposées par gitlab, pour nos futures projets.

4.3 Perspectives d'évolution

Comme établie lors de la conception, l'application web pourrait étendre ses fonctionnalités en fournissant à l'utilisateur les **horaires** et **tarifs** des itinéraires en plus des calculs d'itinéraires déjà proposés.Pour cela, il faudra trouver des données plus complètes sur ces deux informations que ce que nous avons pu récolter.

4.4 Efforts déployés pour garantir la maintenabilité du système

Tout au long du développement du projet, une attention particulière à été porté à la **maintenabilité** et la **modularité** du code afin de simplifier les évolutions futures.

Concernant l'ingestion de données, chaque *parseur* est **décomposé** en une **série de fonctions** qui effectue une tâche spécifique :

1. chargement des données
2. extraction de données
3. nettoyage des données
4. transformation en format intermédiaire

L'intérêt ici est qu'il est possible de **réutiliser des fonctionnalités** des *parseurs* existants si le format d'une nouvelle source de données rejoint le format utilisé par ce *parseur* à une de ses étapes.

D'autre part, l'utilisation d'un **format intermédiaire** pour insérer les données permet de faire **abstraction** des techniques employés pour **insérer des données** dans la WIKIBASE en permettant aux *parseurs* de seulement fournir un format de données structuré en entré du programme d'insertion, et ainsi simplifier l'ajout de nouvelles sources de données.

Concernant l'application utilisateur, le serveur crée un ***wrapper*** pour la création de requêtes SPARQL à l'API WIKIBASE, ce qui permet d'encapsuler son utilisation et ainsi simplifier l'**ajout de paramètre** pour permettre à l'utilisateur de réaliser de nouvelles requêtes.

Ainsi l'**ajout de nouvelles fonctionnalités** à l'application sera plus rapide car il ne sera pas nécessaire de connaître comment fonctionne l'API ou SPARQL.