Plan de Tests

AlpesTransport

Mohammed ROUABAH
Ilyes ZEGHDALLOU
William MAILLARD

01/03/2023

1 Intro et rappel des objectifs du projet

Ce documents regroupe les tests qui seront réalisés dans le cadre du projet AlpesTransport, afin de vérifier son bon fonctionnement.

Le projet a pour objectif d'extraire des données de sources hétérogènes afin de remplir une wikibase, centralisant alors des informations sur une activité locale, permettant à un utilisateur de rechercher des informations sur ces activités dans un endroit centralisé.

Dans cette optique là nous pouvons distinguer deux catégories, qui feront chacune l'objet d'un plan de test, détaillés dans les sections suivantes de ce document :

1. insertion des données
2. interrogation des données

2 Plan de tests d'insertion des données

Cette partie s'intéresse aux données, de leurs extraction à partir de sources diverses à leurs transformation en un modèle intermédiaire, à leurs insertion dans la wikibase.

Pour que le remplissage de notre wikibase réussisse, il est primordiale que chacune de ces étapes fonctionne correctement. Ainsi, un test spécifique, basé sur le scénario de la section suivante, est réalisé pour chacune d'entre elles.

2.1 Scénario

Pour réaliser le plan de tests d'insertion de données nous avons besoin de créer des données de tests en se basant sur les formats des sources de données choisies, afin de tester le bon fonctionnement de chaque parseur.

Ainsi chaque jeu de données de tests associé à un parseur sera composé des documents suivants :

1. j1_source : un document comprenant des données simples au format de la source de données considéré par le parseur,
2. j1_df : un *dataframe* correspondant aux données extraites et nettoyer que le parseur doit fournir,
3. j1_dict : un dictionnaire python représentant la structure obtenu après transformation des données extraites,
4. j1_request : un scrip python pour questionner la wikibase à propos des données que l'on vient d'insérer,

   (avec j1 le nom du jeu de test)

Ces documents vont donc nous permettre de simuler la chaîne de traitement des données et de comparer les résultats obtenus avec ceux présent dans les documents pour valider ou non le bon fonctionnement de cette partie du projet.

De plus ces tests doivent permettre d'observer le comportement du programmes dans le cas d'entrées non conformes, ainsi chaque module sera exécuté sur différents jeux de données

permettant de couvrir le plus de valeurs possible pour les intrants des programmes.

Afin de corriger les éventuels bug rencontrés lors de ces tests, et d'en garder une trace écrite, tous les résultats obtenus seront consignés dans des fichiers de log organisés de la manière suivant :

— Les valeurs de colonnes suivantes :
  * jour et date du test,
  * fichier d'intrant,
  * fichier d'extrant,
  * code du type du test,
  * status du test,
  * raisons de l'échec du test (stocke la différences entre le résultat obtenus et attendus, ainsi que la ligne du fichier concerné).

— Les valeurs des lignes correspondent à un résultat d'exécution de test.

2.2 Test d'extraction des données

intrants : j1_source
extrants : j1_df

Déroulement :

Chaque parseur est exécuté sur différents sources de données de test détaillés ci-dessous. On exécute le programme pour l'extraction et le nettoyage des données des documents de tests, puis on compare le résultat obtenus aux document contenant le résultat attendu.

Chaque résultat est enregistré dans le fichier de log associé au parseur nommé parseurN_resultatsextraction.log

---

Cas de tests :

1. j1_source avec des données normales pour valider le fonctionnement du parseur,
2. j1_source avec un en-tête manquant (par exemple pas d'arrêts de bus), pour s'assurer que ce manque d'information génère une exception sans arrêter le programme ni insérer les données incomplètes dans la wikibase,
3. j1_source avec certaines valeurs manquantes (par exemple certains arrêts sont sans horaires, des horaires sont associés à un arrêt inexistant), afin de vérifier que les lignes incomplètes soit supprimés et répertoriés dans un fichier de log,
4. j2_source provenant d'un jeu de données d'une source différentes pour constacter le levé d'une exception et le rejet du document sans que le programme s'arrête ni n'extrait des données d'un format qu'il ne connait pas (ce qui pourrais mener à l'insertion de données erronées dans la wikibase)

2.3 Test de la transformation des données

intrants : j1_df
extrants : j1_dict

Déroulement :

Chaque parseur possède un module de transformation du *dataframe* des données chargées en un format de donnée intermédiaire (dictionnaire python). On exécute donc ces module avec les *dataframe* des jeux de données de tests, puis, de la même manière que précédemment, on compare les résultat avec le dictionnaire attendus et on répertories les résultats dans un fichier de log intitulé
parseurN_resultats_transformation.log

Cas de tests :

1. *dataframe* complet et bien structuré afin de valider sa transformation en un modèle de donnée intermédiaire,
2. *dataframe* avec des en-têtes manquantes pour s'assurer que ce manque d'information génère une exception sans arrêter le programme ni insérer les données incomplètes dans la wikibase,
3. *dataframe* avec des valeurs manquantes et des types incorrectes, afin de vérifier que les lignes incomplètes soit supprimés et répertoriés dans un fichier de log,
4. *dataframe* null afin de vérifier que cela ne fait pas bugger le programmes est est notifié à l'utilisateur par un *warning*,
5. *dataframe* avec des colonnes inconnus en plus, pour vérifier qu'elles sont ignorés et que cela est bien notifié à l'utilisateur,
6. *dataframe* provenant d'un autre jeu de données, qui doit lever une exception de format incorrecte sans terminer le programme.

2.4 Test de l'insertion des données

intrants : j1_dict
extrants : entités dans la wikibase

Déroulement :

Différents dictionnaire erronés vont êtres fournis au programme d'insertion de données à la wikibase. Ensuite le programme python j1_request associé est exécuté et permet de tester la réalisation ou non de l'insertion des entités dans la wikibase. De la même manière que précédemment ces résultats seront consignés dans le fichier de log du plan de test d'insertion des données.

Pour les résultats de ces tests nous pourrons aussi nous aider de la page de log de la wikibase afin de constater l'insertion des entités présentent dans le dictionnaire.

Cas de tests :

1. un dictionnaire complet pour chaque parseur afin de constater que l'insertion des données est valide quel que soit la source grâce à la structure de données intermédiaire,
2. un dictionnaire null pour vérifier que cela n'interomp pas l'exécution du programme,
3. un dictionnaire avec des clés erronés pour vérifier que les entités correspondantes ne sont pas insérés, et que cela n'empêche pas le programme d'insérer les autres entités correctes présentent,

4. un dictionnaire contenant des entités déjà présente mais avec des valeurs différentes (par exemple une modification des horaires), pour vérifier que les entités sont bien mises à jour.

3 Plan de tests d'interrogation des données

Cette partie concerne l'utilisation finale de l'utilisateur, des données insérées dans la wikibase par l'intermédiaire d'une application.

Notre application va permettre à l'utilisateur de rechercher 3 types d'informations, qui feront chacune l'objet d'un test :

— demande d'informations sur un arrêts
— demande d'information sur les horraires d'un itinéraire
— demande de calcul d'un itinéraire entre un point A et B (suivant le temps disponible)

3.1 Scénario

Prérequis :

1. La wikibase est bien alimentée avec les données nécessaires pour répondre aux requêtes utilisateur,
2. L'application est lancée et en état de marche,
3. L'utilisateur (ou le bot) est sur la page de recherche.

NB : Dans la suite un utilisateur désigne sans distinction un utilisateur humain ou un robot. Chaque réalisation de test sera accompagné d'un fichier de log du même format que la section précédente.

3.1.1 Scénario de test pour la requête "Arrêts"

Action de l'utilisateur :

1. L'utilisateur renseigne le nom de l'arrêt qu'il souhaite trouver dans le champ de recherche,
2. L'utilisateur sélectionne l'option "Arrêts" dans le menu déroulant,
3. L'utilisateur clique sur le bouton "Rechercher".

Résultats attendus :

1. L'application affiche une liste d'arrêts correspondant à la recherche de l'utilisateur,
2. Les arrêts proposés sont proches de la localisation de l'utilisateur ou de l'adresse qu'il a renseignée,
3. Les arrêts affichés sont pertinents et correspondent bien à la recherche de l'utilisateur,
4. Les informations relatives aux arrêts sont complètes et précises (nom de l'arrêt, lignes qui y passent, horaires).

Procédure de test :

1. Vérifier que les arrêts proposés correspondent bien à la recherche de l'utilisateur.
2. Vérifier que les informations relatives aux arrêts sont complètes et précises.
3. que les horaires affichés sont pertinents et permettent de prendre un transport en commun rapidement.
4. Vérifier que l'application propose des arrêts proches de la localisation de l'utilisateur ou de l'adresse qu'il a renseignée.
5. Vérifier que l'application signale clairement les arrêts en travaux ou fermés.
6. Vérifier que l'interface graphique est ergonomique et facile à utiliser.
7. Vérifier que les messages d'erreur éventuels sont clairs et compréhensibles.

3.1.2 Scénario de test pour la requête "Tarifs"

Action de l'utilisateur :

1. L'utilisateur sélectionne l'option "Tarifs" dans le menu déroulant,
2. L'utilisateur choisit le type de billet qu'il souhaite acheter (par exemple, un ticket à l'unité ou un abonnement mensuel),
3. L'utilisateur renseigne les informations nécessaires (nombre de voyages, durée de validité, etc.),
4. L'utilisateur clique sur le bouton "Rechercher".

Résultats attendus :

1. L'application affiche une liste de tarifs correspondant à la recherche de l'utilisateur,
2. Les tarifs proposés sont clairs et précis,
3. Les tarifs affichés sont cohérents avec les tarifs pratiqués dans la région,
4. Si des réductions sont disponibles pour certains profils d'utilisateurs (étudiants, seniors, etc.), l'application les signale clairement à l'utilisateur.

Procédure de test :

1. Vérifier que les tarifs affichés correspondent bien à la recherche de l'utilisateur.
2. Vérifier que les informations relatives aux tarifs sont complètes et précises.
3. Vérifier que les réductions pour certains profils d'utilisateurs sont bien proposées le cas échéant.
4. Vérifier que les tarifs promotionnels sont bien signalés le cas échéant.
5. Vérifier que l'interface graphique est ergonomique et facile à utiliser.
6. Vérifier que les messages d'erreur éventuels sont clairs et compréhensibles.

3.1.3 Scénario de test pour la requête ”Itinéraires”

Action de l'utilisateur :

1. L'utilisateur renseigne l'arrêt de départ et l'arrêt d'arrivée souhaités,
2. L'utilisateur sélectionne l'option ”Itinéraire” dans le menu déroulant,
3. L'utilisateur clique sur le bouton ”Rechercher”.

Résultats attendus :

1. L'application calcule un itinéraire entre les deux arrêts renseignés,
2. L'itinéraire affiché est cohérent (correspond bien à un trajet possible entre les deux arrêts),
3. Les horaires proposés sont réalistes et permettent de rejoindre la destination à l'heure souhaitée,
4. Les correspondances proposées sont cohérentes et permettent d'arriver à destination.

Procédure de test :

1. Vérifier que les arrêts renseignés sont bien pris en compte par l'application.
2. Vérifier que l'itinéraire proposé est cohérent avec les données de la wikibase.
3. Vérifier que les correspondances proposées sont logiques et efficaces.
4. Vérifier que les informations relatives aux changements de ligne sont claires et précises.
5. Vérifier que l'interface graphique est ergonomique et facile à utiliser.
6. Vérifier que les messages d'erreur éventuels sont clairs et compréhensibles.

4 Test de connexion

Action de l'utilisateur :

1. L'utilisateur lance l'application.

Résultats attendus :

1. L'application affiche un message de confirmation si la connexion est établie avec succès,
2. L'application affiche un message d'erreur si la connexion n'est pas établie avec succès,
3. Si la connexion est établie avec succès, l'application affiche les informations de base sur la base de données Wikibase, telles que le nombre d'entités et le nombre de propriétés.

Procédure de test :

1. Vérifier que l'application peut se connecter à l'API Wikibase avec succès,
2. Vérifier que les informations de base sur la base de données Wikibase sont correctement affichées,
3. Vérifier que l'application affiche des messages d'erreur clairs et compréhensibles en cas de problème de connexion.

5 Test de performance

Prérequis

1. L'application est installée et configurée sur un serveur de test,
2. Le logiciel POSTMAN est installé sur un ordinateur de test.
3. Le serveur de test est accessible via une URL publiques,
4. Des scripts POSTMAN sont créés pour exécuter les différentes requêtes utilisateur.

Action de l'utilisateur :

1. L'utilisateur lance POSTMAN sur son ordinateur de test,
2. L'utilisateur sélectionne le script correspondant à la requête utilisateur à tester,
3. L'utilisateur définit le nombre de requêtes à exécuter et la fréquence d'exécution.
4. L'utilisateur lance l'exécution des requêtes.

Résultats attendus :

1. L'application est capable de gérer le nombre de requêtes utilisateur défini sans défaillance,
2. Les temps de réponse de l'application restent acceptables même avec un nombre élevé de requêtes,
3. Les requêtes s'exécutent sans erreur.

Procédure de test :

1. Tester la capacité de l'application à gérer un nombre élevé de requêtes utilisateur en augmentant progressivement le nombre de requêtes et en vérifiant les temps de réponse de l'application (environ 5s),
2. Vérifier que les requêtes s'exécutent correctement même avec un grand nombre de requêtes,
3. Vérifier que les messages d'erreur sont clairs et compréhensibles en cas de problème.
4. Vérifier que l'application répond rapidement et de manière efficace, même en cas de forte charge.

6 Test de compatibilité

Action de l'utilisateur :

Procédure de test :

— Vérifier que l'application fonctionne correctement sur les navigateurs courants (chrome, firefox, bing)
— Tester l'application sur des écrans de différentes tailles pour s'assurer que l'interface utilisateur est bien adaptée à tous les formats d'affichage.

7 Intégration des tests au planning

En accord avec ce qui est énoncé dans ce document de nouvelle tâches ont été créé pour effectuer ces différents tests.

Concernant les test sur l'insertion des données, la répartition a été faites de sorte que la personne qui teste un parseur ne l'aura pas codé, afin de faire émerger des situation auquel le développeur du parseur n'aurait pas pensé.