Travel Recorder

Rapport technique

Ludovic Tagnon Baptiste Beroual Xavier Schaefer Alexis Collignon

19 février 2026

Table des matières

Page de garde

Projet : Travel Recorder
Formation : EPITECH
Équipe : Ludovic Tagnon, Baptiste Beroual, Xavier Schaefer, Alexis Collignon

Résumé

Ce rapport présente le projet Travel Recorder, une application qui transforme une phrase en langage naturel en proposition de trajet ferroviaire. Le système suit une chaîne simple : extraction NLP (départ/arrivée), résolution des gares candidates, calcul de chemin sur un graphe ferroviaire, puis visualisation dans une interface Dash.

Le projet est fonctionnel pour une démonstration de soutenance. Il couvre les briques principales demandées par le sujet (NLP + pathfinding), avec une architecture lisible et modulaire. Les limites actuelles concernent surtout la robustesse, la reproductibilité et l'évaluation quantitative standardisée.

1. Contexte et objectif

1.1 Contexte

Le sujet demande de traiter une phrase utilisateur de type chatbot pour :

1. identifier un départ et une destination ;
2. calculer un trajet ferroviaire cohérent ;
3. fournir une sortie interprétable.

L'application cible une démonstration locale, sans infrastructure cloud.

1.2 Objectif pratique

Objectif fonctionnel principal :

1. entrée utilisateur : phrase libre en français ;
2. extraction NLP : `départ` et `arrivée` (ou phrase invalide) ;
3. calcul du chemin le plus court ;
4. affichage du résultat (trajet + durée) sur une carte.

Objectif secondaire :

1. conserver une architecture modulaire, compréhensible et présentable.

2. Périmètre du projet livré

Structure actuelle du repository :

1. `main.py` : application Dash et callbacks.
2. `back/extract_gares.py` : pipeline NLP (spaCy + CamemBERT + NER).
3. `back/train_ner.py` : apprentissage du modèle spaCy NER.
4. `back/phrase_controller.py` : orchestration NLP -> matching -> pathfinding.
5. `back/stations.py` : mapping noms de gares et identifiants.
6. `back/path_finding.py` : construction du graphe et Dijkstra.
7. `back/dataframe.py` : chargement de la base ferroviaire via `.env`.
8. `back/database/` : données `dataset.csv` et `res.csv`.

Le projet est livré comme une application complète de bout en bout.

3. Données et préparation

3.1 Données NLP

Fichier : `back/database/dataset.csv`

Caractéristiques :

1. nombre de lignes : 60 ;
2. colonnes : `sentence`, `arrival`, `departure`, `nlp_tuple` ;
3. rôle : entraîner un NER spaCy sur les labels `DEP` et `ARR`.

Lecture :

1. le dataset couvre le besoin minimal d'entraînement ;
2. le volume reste réduit pour une généralisation forte.

3.2 Données ferroviaires

Fichier : `back/database/res.csv`

Caractéristiques principales :

1. nombre de lignes : 362624 ;
2. `trip_id` uniques : 42507 ;
3. `parent_station` uniques : 3515 ;

4. `stop_name` uniques : 3510.

Ces données servent de base pour la génération du graphe et le calcul de chemin.

3.3 Configuration

Le chargement de la base est piloté par `.env` :

1. variable `path_to_database` ;
2. lecture dans `back/dataframe.py`.

Exemple présent dans le repo :

```
path_to_database = "back/database/res.csv"
```

4. Architecture technique

4.1 Vue d'ensemble

Flux complet d'exécution :

1. l'utilisateur saisit une phrase dans Dash ;
2. `phrase_controller.phrase_to_trip` est appelé ;
3. extraction NLP des entités `DEP` et `ARR` ;
4. recherche des stations candidates ;
5. calcul Dijkstra pour chaque couple candidat ;
6. sélection du meilleur trajet ;
7. affichage du parcours sur la carte.

4.2 Couplage entre modules

Couplage principal :

1. `main.py` dépend de `phrase_controller` ;
2. `phrase_controller` dépend de `extract_gares`, `stations`, `path_finding` ;
3. `stations` et `path_finding` dépendent de `dataframe`.

Cette organisation facilite la lecture, mais implique des imports en cascade à maîtriser.

5. NLP : extraction départ/arrivée

5.1 Pipeline actuel

Le pipeline NLP de `extract_gares.py` suit 3 étapes :

1. préparation texte via spaCy (`fr_core_news_sm`) ;
2. encodage CamemBERT (`camembert-base`) ;
3. extraction des labels `DEP` et `ARR` via modèle NER spaCy.

5.2 Entraînement du NER

Script : `back/train_ner.py`

Principe :

1. lecture de `dataset.csv` ;
2. parsing des offsets d'entités via `nlp_tuple` ;
3. fine-tuning d'un composant `ner` spaCy ;
4. export du modèle dans `back/model_ner`.

5.3 Validation logique locale

Le système retourne :

1. un dictionnaire `raw_input_depart` / `raw_input_arrivee` si extraction valide ;
2. un message de type "phrase invalide" sinon.

Le contrôle invalide repose sur une heuristique de mots-clés transport.

6. Résolution des gares

6.1 Matching actuel

La résolution des noms de gares est faite en deux niveaux :

1. présélection des candidats avec recherche partielle dans `stations.py` ;
2. score fuzzy (`RapidFuzz`) dans `phrase_controller.py`.

6.2 Données renvoyées

Pour chaque phrase :

1. liste de gares candidates de départ ;
2. liste de gares candidates d'arrivée ;
3. scores de matching associés ;
4. identifiants candidats ;
5. meilleur trajet retenu.

Cette partie est utile pour la démonstration car elle rend le traitement interprétable.

7. Pathfinding

7.1 Construction du graphe

Implémentation dans `back/path_finding.py`.

Étapes :

1. tri par `trip_id` et `stop_sequence` ;
2. normalisation des horaires (gestion du passage minuit) ;
3. agrégation en arêtes orientées ;
4. conservation du meilleur poids par couple origine/destination.

Statistiques observées sur ce graphe dérivé :

1. nœuds effectifs : 749 ;
2. arêtes dédoublonnées : 3100 ;
3. poids moyen : 6042.97 secondes.

7.2 Algorithme

L'algorithme de recherche utilisé est Dijkstra :

1. priorité par distance cumulée ;
2. reconstruction du chemin via prédécesseurs ;
3. sortie : `path` et `total_s`.

Complexité classique :

1. temps : O((V+E) log V) ;
2. mémoire : O(V+E).

8. Interface et démonstration

8.1 Interface Dash

L'application Dash :

1. affiche les gares sur fond OpenStreetMap ;
2. propose une entrée texte pour la phrase ;
3. affiche le trajet retenu ;
4. affiche la durée estimée.

8.2 Scénario de démo jury

Scénario recommandé :

1. lancer `python main.py` ;
2. tester une phrase nominale ;
3. tester une phrase ambiguë ;
4. tester une phrase invalide ;
5. montrer la différence entre résultat valide et invalide.

9. Points forts

Le projet présente plusieurs points solides :

1. intégration NLP + pathfinding complète ;
2. architecture modulaire compréhensible ;
3. exploitation de données ferroviaires volumineuses ;
4. visualisation claire pour la soutenance ;
5. extraction interprétable (candidats + score).

10. Limites et risques techniques

10.1 Reproductibilité NLP

Le dossier `back/model_ner` n'est pas versionné :

1. il faut entraîner localement avant usage ;
2. sinon `extract_gares.py` ne peut pas charger le modèle.

10.2 Robustesse des cas limites

Cas à surveiller :

1. absence de chemin (`None`, `inf`) pas toujours filtrée strictement ;
2. sélection du meilleur trajet basée sur `total_s`, pouvant écraser des cas ex æquo ;
3. risque de faux positifs avec matching partiel permissif.

10.3 Simplification du graphe

Le choix "premier arrêt -> dernier arrêt par trip" simplifie fortement le réseau :

1. moins d'étapes intermédiaires explicites ;
2. couverture partielle des correspondances complexes.

10.4 Industrialisation

Il manque une couche qualité complète :

1. pas de tests unitaires automatisés ;
2. pas de pipeline CI ;
3. benchmark NLP comparatif initial disponible, à étendre sur un jeu plus large.

11. Résultats NLP et comparaison des parseurs

Une évaluation comparative a été exécutée sur `back/database/dataset.csv` (60 phrases) avec :

1. parseur A : pipeline principal (`spaCy + CamemBERT + NER`) ;
2. parseur B : parseur alternatif `Stanza` rule-based (règles de formulation + référentiel gares SNCF).

Fichiers de sortie :

1. `reports/nlp_parser_comparison.json`
2. `reports/nlp_parser_comparison.md`

Métriques principales :

Parseur	Samples	Invalid	Dep Acc	Arr Acc	Pair Acc
Principal (`spaCy+CamemBERT+NER`)	60	2	0.9167	0.8667	0.8333
Alternatif (`stanza_rule_based_parser`)	60	4	0.9000	0.8833	0.8500

Lecture :

1. le parseur principal est plus précis sur le départ ;
2. le parseur alternatif `Stanza` est plus stable sur l'arrivée ;
3. le parseur `Stanza` obtient une meilleure exactitude sur le couple complet (`pair_accuracy`) ;
4. ce résultat reste à reconfirmer sur un jeu d'évaluation plus large.

12. Couverture des attentes du sujet

État synthétique :

1. extraction départ/arrivée : faite ;
2. invalidation des phrases hors scope : couverte sur le périmètre de démonstration ;
3. pathfinding sur données ferroviaires : fait ;
4. intégration bout en bout : faite ;
5. robustesse fautes/ambiguïté : prise en charge par normalisation et matching fuzzy, avec axe d'extension ;
6. évaluation quantitative : base de mesure disponible, consolidable sur un jeu élargi.

13. Plan d'amélioration post-keynote

Améliorations prioritaires :

1. augmenter le dataset NER (objectif 300+ phrases annotées) ;
2. ajouter un parser alternatif et comparer les métriques ;
3. renforcer la gestion des erreurs en sortie NLP/pathfinding ;
4. enrichir le graphe avec segments consécutifs ;
5. ajouter des tests, une CI et des scripts d'évaluation.

14. Conclusion

Travel Recorder répond au cœur du sujet avec une implémentation complète de démonstration : interprétation NLP d'une phrase utilisateur et calcul de trajet ferroviaire visualisé.

En l'état, le projet est une preuve de concept fonctionnelle, techniquement cohérente, avec des limites connues et des pistes d'évolution claires.

La contribution principale de l'équipe est d'avoir assemblé des briques hétérogènes (NLP, données GTFS, graphe, UI) dans une chaîne de valeur opérationnelle et présentable.

Annexe A - Commandes de reproduction

A.1 Installation

```
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

A.2 Préparation des données

```
mkdir -p back/database
# copier res.csv et dataset.csv dans back/database/
```

A.3 Entraînement NER

```
python back/train_ner.py
```

A.4 Lancement application

```
python main.py
```

URL locale :

`http://localhost:8090`

A.5 Évaluation NLP comparative

`.venv/bin/python scripts/evaluate_nlp_parsers.py`

Sorties :

1. `reports/nlp_parser_comparison.json`
2. `reports/nlp_parser_comparison.md`

Annexe B - Fichiers de référence

1. `main.py`
2. `back/extract_gares.py`
3. `back/phrase_controller.py`
4. `back/path_finding.py`
5. `back/stations.py`
6. `back/train_ner.py`
7. `back/dataframe.py`
8. `back/parser_stanza_rules.py`
9. `scripts/evaluate_nlp_parsers.py`
10. `reports/nlp_parser_comparison.md`
11. `reports/nlp_parser_comparison.json`
12. `back/database/dataset.csv`
13. `back/database/res.csv`