AUBAIN MBOKOU

Master1 - IA

Rapport du Projet 2024-2025

Application Web pour la Visualisation des Stations de Vélos en Temps Réel

Introduction

La mobilité durable est un enjeu majeur pour les villes modernes. Ce projet vise à développer une application web interactive permettant de visualiser les stations de vélos en libre-service en temps réel. Le projet combine des techniques de collecte de données, de gestion via une base SQL et de visualisation interactive sur une carte.

L'objectif est de fournir une solution intuitive qui aide les utilisateurs à identifier les stations de vélos proches, leur disponibilité, et les places libres. Nous réaliserons cet objectif sur Jupyter notebook et VSCode. Ce rapport documente les étapes clés du projet et présente les résultats obtenus.

Étape 1 : Collecte des Données

Nous avons utilisé des API ouvertes pour récupérer les informations des stations de vélos en temps réel pour les villes suivantes :

- **Paris**

  URL de l'API utilisée :
  "https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/velib-disponibilite-en-temps-reel/records?"

- **Lille**

  URL de l'API utilisée :
  "https://data.lillemetropole.fr/data/ogcapi/collections/vlille_temps_reel/items?"

- **Toulouse**

  URL de l'API utilisée :
  "https://data.toulousemetropole.fr/api/explore/v2.1/catalog/datasets/stationnement-velo/records?limit=20"

Grace à des fonctions python, nous collectons toutes les données depuis les API. Ces données collectées incluent :

- Le Nom de la station
- Les Coordonnées GPS (latitude, longitude)
- Le Nombre de vélos disponibles

- Le Nombre de places disponibles
- Le Statut de la station

Extrait de deux lignes de données collectées : cas de Paris

	stationcode	name	is_installed	capacity	nombre_places_disponibles	nombre_velos_disponibles	mechanical	ebike	is_renting
0	44015	Rouget de L'isle - Watteau	OUI	20	13	6	0	6	OUI
1	14111	Cassini - Denfert-Rochereau	OUI	25	22	3	2	1	OUI

is_returning	duedate	longitude	latitude	arrondissement_commune	code_insee_commune
OUI	2024-12-08T19:27:04+00:00	2.396302	48.778193	Vitry-sur-Seine	94081
OUI	2024-12-08T19:30:14+00:00	2.336035	48.837526	Paris	75056

Au terme de cette collecte, nous avons eu 3 dataframes : dfParis, dfToulouse et dfLille qui recensent les données en temps réel collectées depuis les API. Cela étant fait, nous sommes passées au nettoyage des données.

Étape 2 : Préparation et Nettoyage des Données

Objectif : Harmoniser les données pour faciliter leur stockage et leur traitement.

Actions réalisées :

1. Ajout de la colonne ‘Ville’ dans chaque dataframe.
2. Normalisation des noms des colonnes en français pour une meilleure lisibilité.
3. Concaténation des 3 dataframes.
4. Gestion des valeurs manquantes.
5. Récupération du dataframe nettoyé avec uniquement les colonnes pertinentes (filtrage du dataframe nettoyé).

Extrait des 5 lignes du dataframe final obtenu :

	arrondissement_commune	longitude	latitude	nombre_places_disponibles	nombre_velos_disponibles	ville
0	Vitry-sur-Seine	2.396302	48.778193	13	6.0	Paris
1	Paris	2.336035	48.837526	22	3.0	Paris
2	Paris	2.343335	48.819428	34	23.0	Paris
3	Paris	2.353468	48.835093	34	11.0	Paris
4	Paris	2.351966	48.843893	11	12.0	Paris

Nous avons à présent notre dataframe final, qui nous donne en temps réel les informations pertinentes dont nous avons besoin pour notre application, Cependant, nous faisons face à plusieurs défis :

La Performance : en effet, le traitement des données directement en mémoire via notre dataframe peut devenir inefficace à mesure que la taille des données augmente, la recherche et le filtrage deviennent plus lents.

La Fiabilité et persistance : En cas de panne ou d'arrêt du programme, les données en mémoire sont perdues. Il faut une solution qui permette de sauvegarder les données de manière permanente.

Par ailleurs, les données doivent pouvoir être partagées avec d'autres outils ou utilisateurs. Cela exige un format standardisé et compatible, comme une base SQL, pour faciliter l'accès et les requêtes. C'est pour toutes ces raisons que nous avons stocké nos données dans une base SQL.

Étape 3 : Stockage des Données dans une Base SQL

Nous avons utilisé SQLite pour stocker les données nettoyées et permettre des requêtes efficaces.

Étapes :

1. Création d'une base de données nommée info_velos.db.
2. Exportation des données nettoyées sous forme de table SQL nommée clean_data.

Code utilisé :

```
# Connexion à la base de données SQLite
conn = sqlite3.connect('info_velos.db')
cursor = conn.cursor()

# Obtenir la liste des tables dans la base de données
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

# Afficher les noms des tables
print("Tables dans la base de données :")
for table in tables:
    print(table[0])

conn.close()
```

Affichage de quelques lignes de la table :

```
Premières lignes de la table 'clean_data' :
('Vitry-sur-Seine', 2.3963020229163, 48.778192750803, 13, 6.0, 'Paris')
('Paris', 2.3360354080796, 48.837525839067, 22, 3.0, 'Paris')
('Paris', 2.3433353751898, 48.819428333369, 34, 23.0, 'Paris')
('Paris', 2.3534681351338, 48.835092787824, 34, 11.0, 'Paris')
('Paris', 2.3519663885235786, 48.84389286531899, 11, 12.0, 'Paris')
```

La base SQL fournit une fondation robuste pour organiser et manipuler les données. Dans l'étape qui suit, nous avons rendu ces données **visuellement compréhensibles** et **interactives**, répondant ainsi au besoin final d'une application accessible aux utilisateurs.

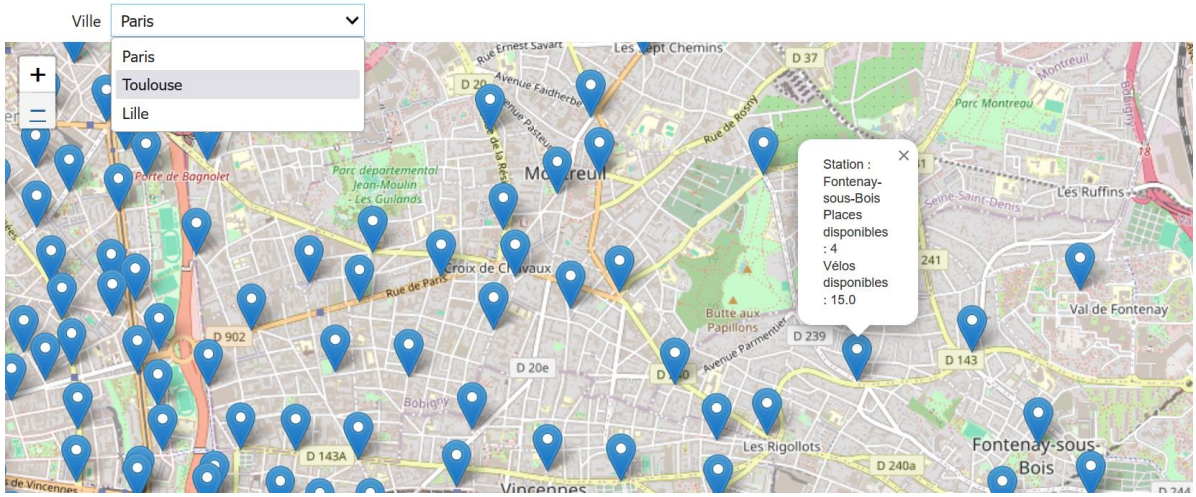
Étape 4 : Visualisation Interactive

Nous avons développé une interface interactive en Python, utilisant **Folium** pour afficher les stations sur une carte.

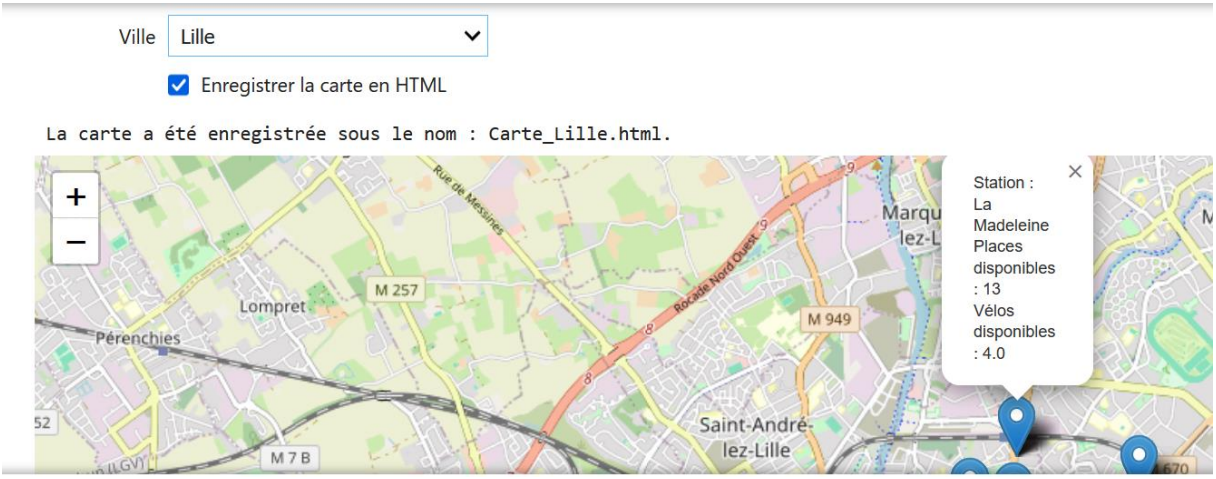
Fonctionnalités :

1. **Carte interactive :** Visualisation des stations avec des marqueurs dynamiques.
2. **Sélection de la ville :** Les utilisateurs peuvent choisir leur ville parmi Paris, Lille ou Toulouse.
3. **Informations sur les stations :** Nombre de vélos et places disponibles visibles au survol d'un marqueur.

Résultat :



Nous avons également enrichi le projet en intégrant une nouvelle fonctionnalité permettant d'enregistrer, à tout moment et en temps réel, **une carte HTML** affichant les disponibilités des vélos en fonction de la ville sélectionnée. Cette fonctionnalité offre une flexibilité supplémentaire aux utilisateurs, leur permettant de générer et de sauvegarder une visualisation interactive adaptée à leurs besoins, comme illustré ci-dessous :



Après avoir mis en place une interface interactive permettant de visualiser en temps réel la disponibilité des vélos, l'étape suivante consiste à enrichir cette expérience utilisateur en ajoutant des fonctionnalités de filtrage avancé. En particulier, nous allons permettre aux utilisateurs de restreindre l'affichage des stations de vélos à celles situées dans un rayon défini autour de leur position actuelle.

Étape 5 : Analyse Spatiale

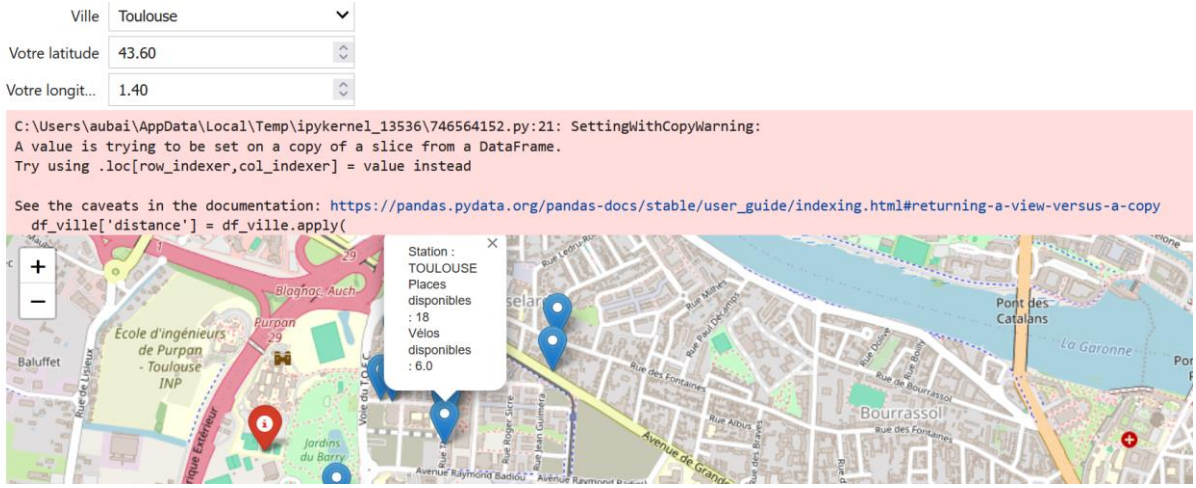
Nous avons ajouté une fonctionnalité pour filtrer les stations dans un rayon de 1 km autour de la position de l'utilisateur, en utilisant la distance haversine.

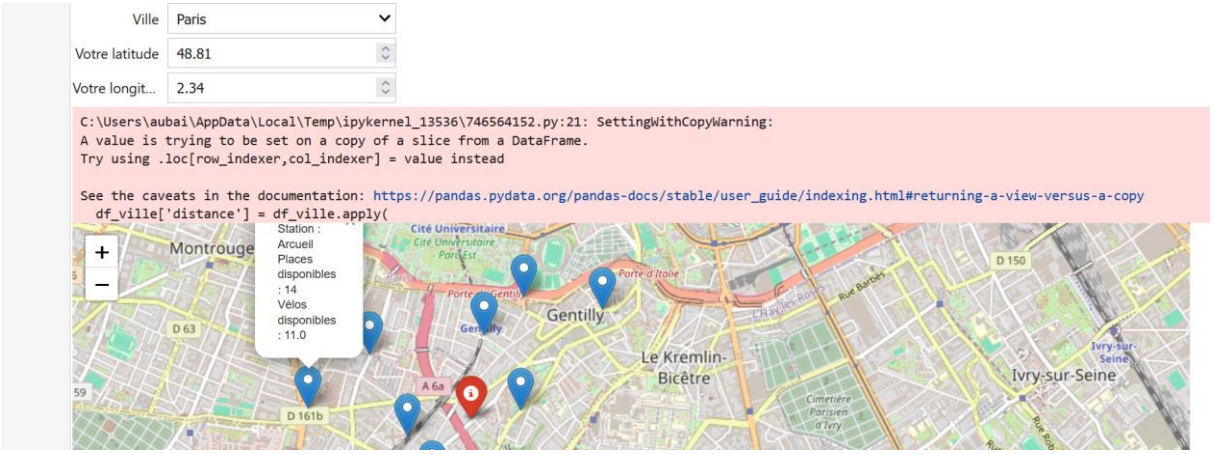
L'utilisateur sélectionne une ville et saisit ses coordonnées.

Une carte s'affiche montrant :

- La position de l'utilisateur **(marqueur rouge).**
- Les stations dans un rayon de 1 km **(marqueurs interactifs).**

Chaque marqueur de station affiche les détails (nom, places disponibles, vélos disponibles) lorsqu'on clique dessus.



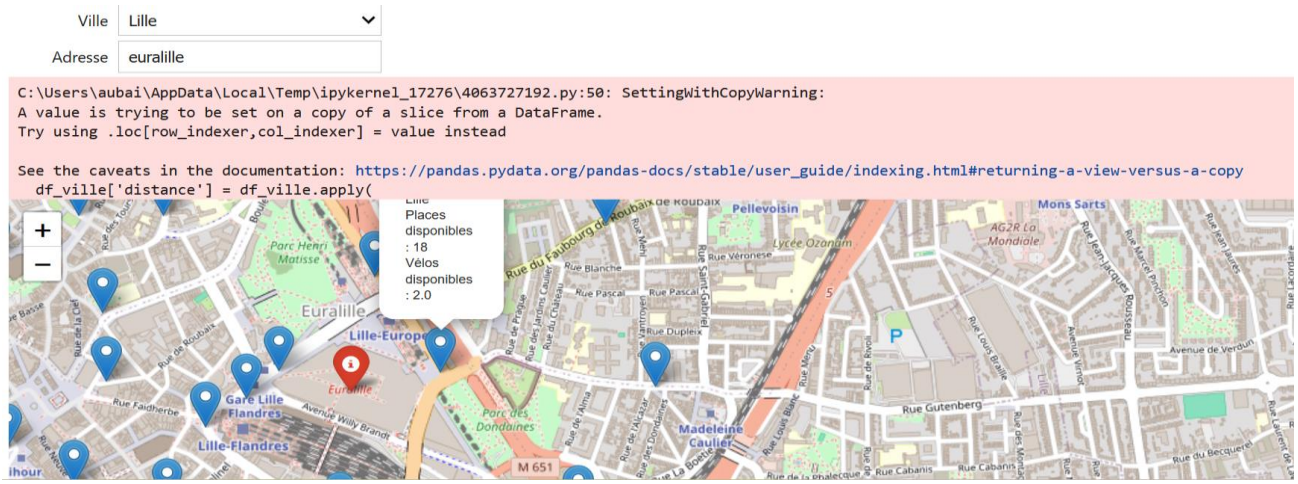


Transformation des adresses

Pour rendre les données plus interactives, intuitives et utiles, nous avons intégré une fonctionnalité de géocodage permettant :

- De convertir une adresse donnée en latitude et longitude à l'aide de l'API OpenCage.
- D'utiliser ces coordonnées pour filtrer les stations dans un rayon de 1 km autour de l'utilisateur.

Cette fonctionnalité a été validée via des widgets interactifs dans Jupyter Notebook, permettant aux utilisateurs de tester et visualiser directement les résultats.



Tout a été fait sur Jupyter avec soin mais comment déployer notre application, comment la mettre en place de manière concrète ? pour le faire, nous utiliserons une application web flask dans vs code.

Étape 6 : Transition vers une application web dans VS Code

a) Développement de l'interface Flask

L'application web Flask a été développée pour fournir une interface conviviale. Elle inclut :

- **Un formulaire interactif** pour sélectionner une ville et saisir une adresse.
- **Une carte interactive générée avec Folium**, affichant :
    - Les stations de vélos disponibles.
    - Leur nombre de vélos et de places disponibles.
    - Un rayon de 1 km autour de l'adresse saisie par l'utilisateur.

b) Automatisation de la collecte des données

Pour garantir l'actualisation des données en temps réel sans intervention manuelle :

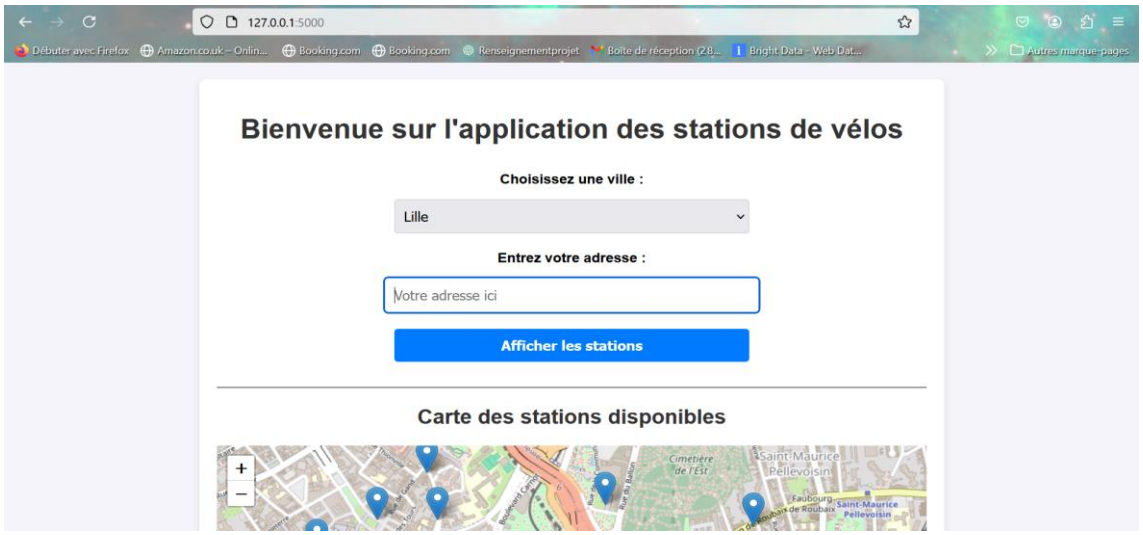
- **Une fonction d'actualisation automatique** a été mise en place à l'aide d'APScheduler.
- Les données sont collectées toutes les 5 minutes et mises à jour dans la base SQLite.
- L'architecture repose sur des fonctions modulaires, facilitant la maintenance.

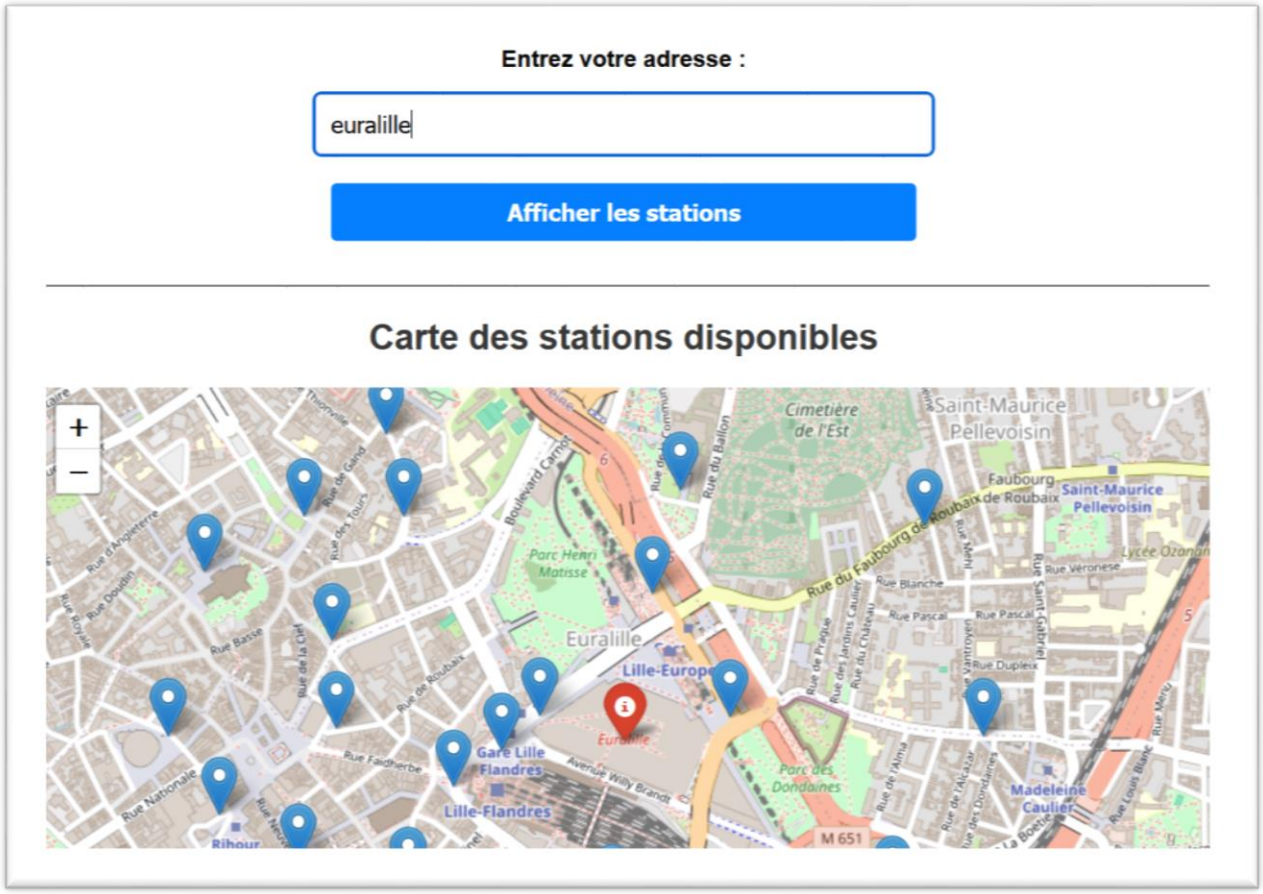
c) Intégration des fonctionnalités

Les fonctionnalités développées dans Jupyter Notebook ont été transférées dans VS Code et réorganisées :

- Le calcul de la distance haversine a été intégré pour filtrer les stations proches.
- La gestion des erreurs a été améliorée pour garantir la robustesse de l'application.
- Une base de données SQLite est utilisée pour stocker les données entre deux actualisations.

<u>Résultat final</u>





Conclusion

Ce projet a permis de développer une application complète pour la gestion et la visualisation des stations de vélos en temps réel. Le projet est passé avec succès d'une simple exploration de données sur Jupyter à une application web interactive et automatisée. En combinant Flask

pour l'interface utilisateur, APScheduler pour l'automatisation, et SQLite pour le stockage, nous avons conçu une solution robuste et évolutive. Ce projet illustre l'importance de la gestion des données en temps réel et de leur visualisation intuitive pour répondre aux besoins d'une mobilité urbaine moderne et intelligente. Une extension future pourrait inclure des prédictions sur la disponibilité des vélos en utilisant des algorithmes de machine Learning. Avec des améliorations supplémentaires, cette application peut être déployée à grande échelle pour des cas d'usage variés.

Annexes :

- Le notebook Projet_API_Aubain.ipynb
- Le dossier API du projet exécuté sur VS Code, app.py est le programme principal de l'application