

# Projet de surveillance du trafic



## Table des matières

1. Introduction .....	2
1.1 Contexte du projet.....	2
1.2 Présentation de l'application.....	2
2. Dispositif de monitoring.....	3
2.1 Outils utilisés .....	3
2.2 Configuration et mise en place .....	3
2.3 Suivi des performances.....	3
2.4 Accès au tableau de bord .....	3
3. Journalisation des incidents.....	4
3.1 Configuration du logging .....	4
3.2 Niveaux de journalisation .....	4
3.3 Enregistrement des erreurs.....	4
3.4 Exemples de logs .....	4
4. Détection et gestion des incidents.....	5
4.1 Définition des seuils d'alerte .....	5
4.2 Détection automatique des incidents .....	5
4.3 Cas concrets d'incidents .....	6
5. Débogage et améliorations .....	7
5.1 Corrections apportées .....	7
5.2 Tests effectués.....	7
5.3 Améliorations et optimisations .....	7
6. Conclusion.....	7
7. Annexes.....	8
8. Bibliographie .....	9

## 1. Introduction

Ce document présente le dérogage, le dispositif de monitoring et de journalisation mis en place pour l'application de surveillance du trafic de Rennes Métropole. L'objectif est de garantir la fiabilité de l'application, d'identifier les incidents et de les résoudre efficacement.

### 1.1 Contexte du projet

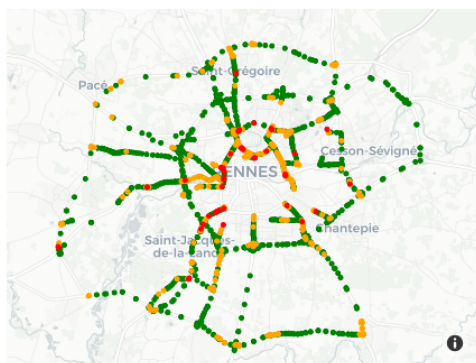
Rennes Métropole dispose d'une application permettant de surveiller en temps réel l'état du trafic routier dans l'agglomération. En tant que développeur IA, la mission est de monitorer, déboguer et mettre en place une journalisation des erreurs dans cette application.

### 1.2 Présentation de l'application

L'application utilise des données de trafic en temps réel pour prédire l'état du trafic aux heures sélectionnées. Les prédictions sont affichées sous forme de carte interactive, permettant aux utilisateurs de visualiser les conditions de circulation dans la ville.

## Traffic Rennes

Traffic en temps réel



### Prédiction d'embouteillage pour le centre ville

Choisissez une heure :

0h

Prédiction : Libre pour 13 h

## 2. Dispositif de monitoring

### 2.1 Outils utilisés

Pour le monitoring de l'application, nous avons utilisé Flask Monitoring Dashboard, un outil intégré qui permet de suivre les performances de l'application Flask, telles que le temps de réponse et l'utilisation des ressources.



Automatically monitor the evolving performance of Flask/Python web services

Login

Login

Password

Login

For advanced documentation, see [this site](#)

### 2.2 Configuration et mise en place

Le tableau de bord de monitoring a été configuré pour surveiller les principales métriques de performance. L'application a été modifiée pour inclure le module de monitoring, avec un accès dédié au tableau de bord via une URL spécifique.

### 2.3 Suivi des performances

Le suivi des performances inclut la surveillance du temps de réponse des routes, l'utilisation de la mémoire, et le nombre de requêtes traitées par l'application. Ces informations sont cruciales pour identifier les points de contention potentiels.

### 2.4 Accès au tableau de bord

Le tableau de bord est accessible via l'URL `http://localhost:5000/dashboard`. Il fournit une interface visuelle permettant de consulter les performances en temps réel ainsi que l'historique des métriques collectées.

➔ Voir 7. Annexes

### 3. Journalisation des incidents

#### 3.1 Configuration du logging

Le module logging de Python a été configuré pour enregistrer les incidents survenus dans l'application. Un fichier de log rotatif est utilisé pour gérer la taille des logs, avec une rotation automatique lorsque la taille maximale est atteinte.

```
# Création d'un fichier log uniquement s'il n'existe pas encore
log_file_path = os.path.join(os.getcwd(), 'app.log')
if not os.path.isfile(log_file_path):
    open(log_file_path, 'w', encoding='utf-8').close()

# Configuration du logger
handler = RotatingFileHandler(log_file_path, maxBytes=10000, backupCount=1, encoding='utf-8')
handler.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)

# Configuration du logger pour l'application
app.logger.setLevel(logging.INFO)
app.logger.addHandler(handler)
```

#### 3.2 Niveaux de journalisation

Les différents niveaux de journalisation utilisés sont :

- INFO : pour les informations générales sur le fonctionnement de l'application,
- WARNING : pour les avertissements qui ne sont pas bloquants,
- ERROR : pour les erreurs qui nécessitent une attention particulière.

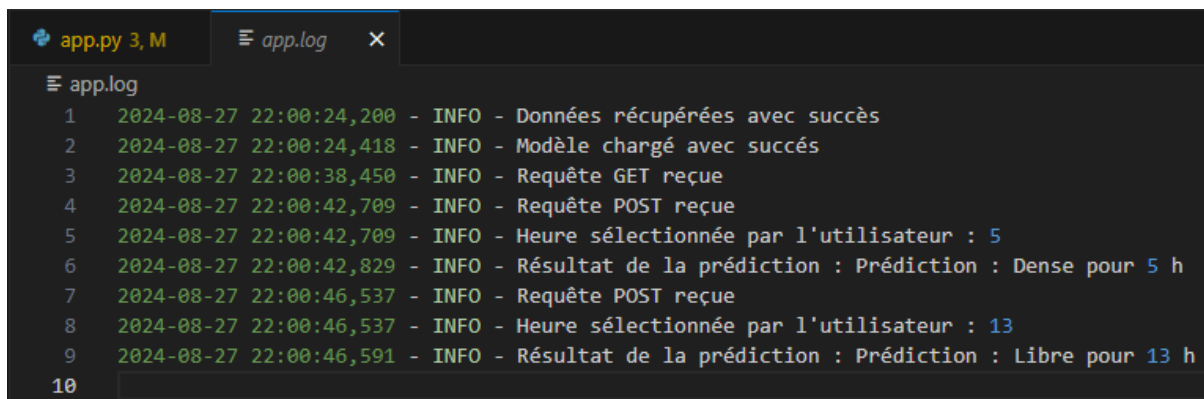
#### 3.3 Enregistrement des erreurs

Les erreurs sont enregistrées avec un horodatage précis, ce qui permet de retracer l'historique des incidents. Seules les erreurs sont enregistrées dans les logs pour minimiser la quantité de données stockées.

#### 3.4 Exemples de logs

Voici quelques exemples de logs générés par l'application :

2023-08-26 12:34:56,789 - ERROR - Erreur lors du chargement du modèle : Model not found,  
2023-08-26 12:35:10,123 - INFO - Requête POST reçue.



```
app.py 3, M  app.log X
app.log
1  2024-08-27 22:00:24,200 - INFO - Données récupérées avec succès
2  2024-08-27 22:00:24,418 - INFO - Modèle chargé avec succès
3  2024-08-27 22:00:38,450 - INFO - Requête GET reçue
4  2024-08-27 22:00:42,709 - INFO - Requête POST reçue
5  2024-08-27 22:00:42,709 - INFO - Heure sélectionnée par l'utilisateur : 5
6  2024-08-27 22:00:42,829 - INFO - Résultat de la prédiction : Prédiction : Dense pour 5 h
7  2024-08-27 22:00:46,537 - INFO - Requête POST reçue
8  2024-08-27 22:00:46,537 - INFO - Heure sélectionnée par l'utilisateur : 13
9  2024-08-27 22:00:46,591 - INFO - Résultat de la prédiction : Prédiction : Libre pour 13 h
10
```

4. Détection et gestion des incidents

4.1 Définition des seuils d'alerte

Les seuils d'alerte ont été définis pour détecter les anomalies dans l'application. Par exemple, un temps de réponse supérieur à 2 secondes ou un taux d'erreurs supérieur à 5% déclenchent une alerte.

```
# Définition des seuils d'alerte
ALERT_THRESHOLD_RESPONSE_TIME = 2 # secondes
ALERT_THRESHOLD_ERROR_RATE = 0.05 # 5%
```

4.2 Détection automatique des incidents

L'application surveille en continu les métriques de performance et les logs pour détecter automatiquement les incidents. Des alertes sont stockés en cas de dépassement des seuils d'alerte.

CODE DE SURVEILLANCE (PYTHON)
<pre>import logging from time import time from functools import wraps from flask import g, abort, Flask  # Définition des seuils d'alerte ALERT_THRESHOLD_RESPONSE_TIME = 2 # secondes ALERT_THRESHOLD_ERROR_RATE = 0.05 # 5%  # Décorateur pour mesurer le temps de réponse des requêtes def monitor_performance(func):     @wraps(func)     def wrapper(*args, **kwargs):         start_time = time()         try:             response = func(*args, **kwargs)         finally:             elapsed_time = time() - start_time             if elapsed_time &gt; ALERT_THRESHOLD_RESPONSE_TIME:                 app.logger.warning(f"Temps de réponse élevé : {elapsed_time:.2f} secondes, seuil dépassé : {ALERT_THRESHOLD_RESPONSE_TIME} secondes")         return response     return wrapper  # Middleware pour surveiller les erreurs @app.before_request def before_request():     if not hasattr(g, 'error_count'):         g.error_count = 0     if not hasattr(g, 'total_requests'):         g.total_requests = 0 @app.after_request def after_request(response):     g.total_requests += 1     if response.status_code &gt;= 500:         g.error_count += 1     # Calculer le taux d'erreurs     if g.total_requests &gt; 0:         error_rate = g.error_count / g.total_requests         if error_rate &gt; ALERT_THRESHOLD_ERROR_RATE:             app.logger.warning(f"Taux d'erreurs élevé : {error_rate:.2%}, seuil dépassé : {ALERT_THRESHOLD_ERROR_RATE:.2%}")     return response</pre>

### 4.3 Cas concrets d'incidents

Durant le développement, un incident majeur a été rencontré lorsque le modèle de prédiction n'a pas pu être chargé. L'erreur a été résolue en corrigeant le chemin d'accès au modèle, et une gestion d'exception a été mise en place pour s'assurer du chargement et de la validité du fichier.

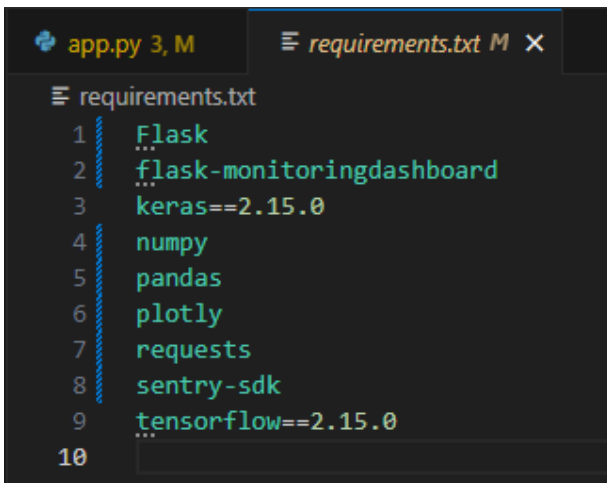
➔ Ci-dessous un tableau récapitulatif des erreurs et des corrections apportées dans le code de l'application.

ERREUR	AVANT CORRECTION ❌	APRÈS CORRECTION ✅
Chemin d'accès	<pre>app.py model = load_model('model.h5')</pre>	<pre># Chargement du modèle avec gestion des erreurs try:     model = load_model('models/model.h5')     app.logger.info('Modèle chargé avec succès') except Exception as e:     app.logger.error(f"Erreur lors du chargement du modèle : {e}")</pre>
Indentation	<pre>get_data.py for data_dict in self.data:     temp_df = self.processing_one_point(data_dict)     res_df = pd.concat([res_df, temp_df])</pre>	<pre># Traite chaque dictionnaire de données et concatène les DataFrames for data_dict in self.data:     # Résolution du problème d'indentation     temp_df = self.processing_one_point(data_dict)     res_df = pd.concat([res_df, temp_df], ignore_index=True)</pre>
Syntaxe	<pre>get_data.py res_df = res_df[res_df.traffic != 'unknown']</pre>	<pre># Filtre les données pour exclure les entrées où le statut de trafic est 'unknown' res_df = res_df[res_df.traffic != 'unknown'] # Manque le crochet fermé ']'  return res_df</pre>
	<pre>utils.py zoom=10 height=500, mapbox_style="carto-positron"</pre>	<pre>zoom=10,          # Niveau de zoom de la carte + Rajout d'une virgule height=500,       # Hauteur de la figure mapbox_style="carto-positron" # Style de la carte )</pre>
Clé	<pre>get_data.py , 'traffic_status', 'geo_point_2d',</pre>	<pre>, 'trafficstatus', 'geo_point_2d',</pre>
	<pre>get_data.py temp.geo_point_2d.map(lambda x : x['latitude']) temp.geo_point_2d.map(lambda x : x['longitude'])</pre>	<pre>temp.geo_point_2d.map(lambda x: x['lat']) # Correction de 'latitude' en 'lat' temp.geo_point_2d.map(lambda x: x['lon']) # Correction de 'longitude' en 'lon'</pre>
Nom de fichier	<pre>app.py return render_template('home.html',</pre>	<pre># Remplacement de 'home.html' par 'index.html' return render_template('index.html',</pre>
Argument	<pre>app.py selected_hour = request.form['hour'] cat_predict = prediction_from_model(model)</pre>	<pre># Rajout de l'heure sélectionnée en paramètre du model cat_predict = prediction_from_model(model, selected_hour)</pre>
Dimension	<pre>utils.py def prediction_from_model(model, hour_to_predict):     input_pred = np.array([0]*25)     input_pred[int(hour_to_predict)] = 1      cat_predict = np.argmax(model.predict(np.array(     return cat_predict  (Taille de '25' au lieu de '24')</pre>	<pre>def prediction_from_model(model, hour_to_predict):     try:         # Conversion de l'heure en entier         hour_to_predict = int(hour_to_predict)     except ValueError:         raise ValueError("hour_to_predict doit être un entier valide entre 0 et 23")      # Crée un tableau d'entrée avec 24 éléments, tous initialisés à 0     input_pred = np.zeros(24)     input_pred[hour_to_predict] = 1      # Effectue la prédiction avec le modèle et trouve la catégorie avec la valeur max     cat_predict = np.argmax(model.predict(np.array([input_pred])))      return cat_predict</pre>

## 5. Débogage et améliorations

### 5.1 Corrections apportées

Plusieurs bugs ont été identifiés et corrigés, notamment des erreurs dans le chargement des données et des problèmes de compatibilité avec certaines versions de bibliothèques. Afin de garantir un environnement cohérent et de résoudre ces problèmes, un fichier "requirements.txt" a été créé pour spécifier les versions précises des bibliothèques nécessaires.



```
app.py 3, M requirements.txt M X
requirements.txt
1 Flask
2 flask-monitoringdashboard
3 keras==2.15.0
4 numpy
5 pandas
6 plotly
7 requests
8 sentry-sdk
9 tensorflow==2.15.0
10
```

### 5.2 Tests effectués

Les tests incluent des vérifications unitaires et des tests d'intégration pour s'assurer que les corrections apportées ne provoquent pas de régressions dans l'application.

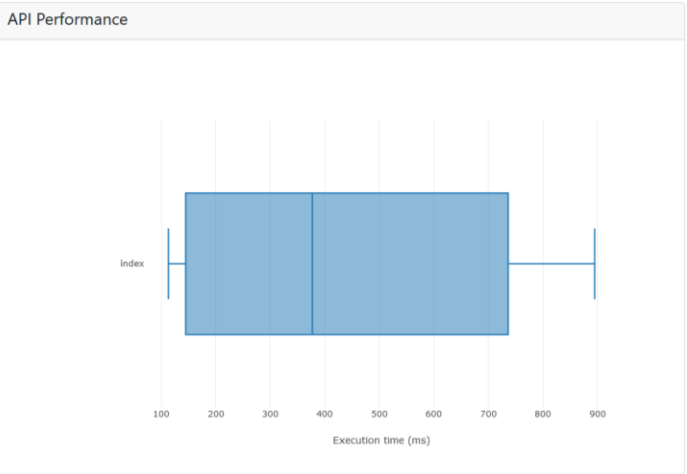
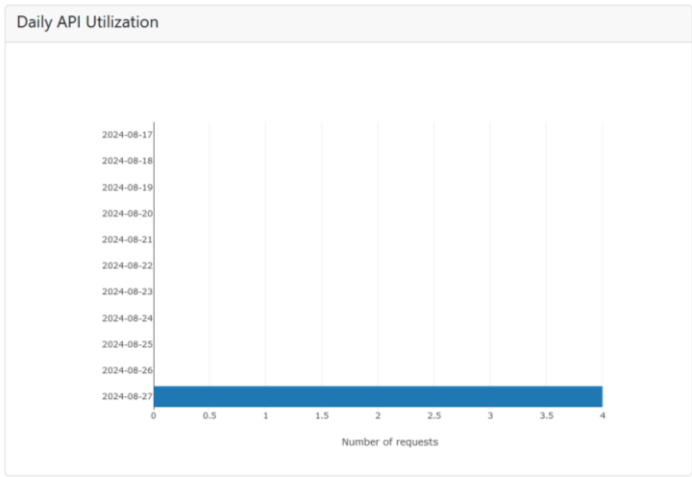
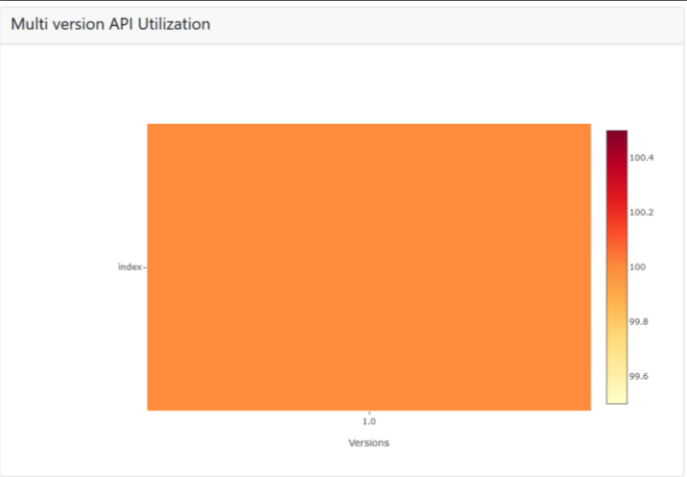
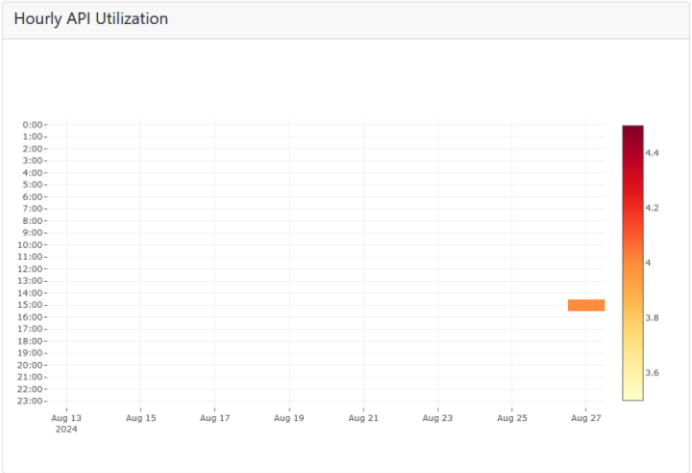
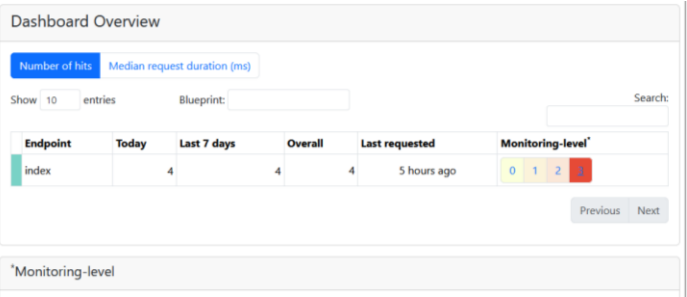
### 5.3 Améliorations et optimisations

Des optimisations ont été faites pour améliorer la performance de l'application, notamment en réduisant le temps de réponse des requêtes et en optimisant l'utilisation de la mémoire.

## 6. Conclusion

Ce projet a permis de renforcer la stabilité et la performance de l'application de surveillance du trafic. Le dispositif de monitoring et la journalisation des incidents fournissent une base solide pour maintenir l'application en production et détecter rapidement les problèmes.

7. Annexes



Reporting

Generate a report to get more insight into how the performance of your service is changing.

Aug 27 vs Aug 26

Week 35 vs Week 34

August vs July

Compare commits

Custom

01 / 08 / 2024 00:00

31 / 08 / 2024 23:59:999

Compared to

01 / 07 / 2024 00:00

31 / 07 / 2024 23:59:999

Generate

☒ Only Show Significant Results

No report generated yet



## 8. Bibliographie

- Le code source de l'application : [https://github.com/bertrandfournel/rennes\\_traffic\\_ko](https://github.com/bertrandfournel/rennes_traffic_ko)
- Le code de l'application débuggée : [https://github.com/roxfr/rennes\\_traffic](https://github.com/roxfr/rennes_traffic)
- [Documentation Flask](#), [Flask Monitoring Dashboard](#), [Module logging de Python](#)