1. Présentation du Dispositif de Monitorage Applicatif : Flask Monitoring Dashboard

Le dispositif de monitorage que nous avons intégré utilise Flask Monitoring Dashboard pour surveiller les performances de l'application et détecter les incidents. Il fournit des fonctionnalités telles que la surveillance du temps de réponse des requêtes, le comptage des erreurs, et la génération d'alertes lorsque des seuils critiques sont dépassés. Le tableau de bord affiche des graphiques et des statistiques sur les performances de l'application, permettant une analyse en temps réel des données.

Technologies Utilisées

Le système de monitorage de l'application utilise plusieurs technologies clés pour assurer une surveillance efficace. **Flask** sert de framework principal pour le développement de l'application web, tandis que **Flask Monitoring Dashboard** est intégré pour surveiller les performances en temps réel. La bibliothèque **Plotly** est employée pour créer des graphiques interactifs qui visualisent les données de trafic, et **Keras** est utilisé pour charger et appliquer un modèle de machine learning afin de faire des prédictions sur l'état du trafic. Enfin, le module **logging** est configuré pour gérer la journalisation des erreurs et des événements importants.

Architecture

L'architecture du dispositif de monitorage est intégrée dans l'application Flask, où le serveur web héberge les fonctionnalités de monitoring. **Flask Monitoring Dashboard** est configuré pour suivre les performances de l'application, afficher des graphiques, et gérer les alertes. Le modèle de machine learning, chargé via **Keras**, permet d'effectuer des prédictions sur les données de trafic. Le module de **journalisation** assure un suivi des erreurs et des événements dans un fichier dédié, facilitant ainsi la gestion des incidents.

Principales Fonctions

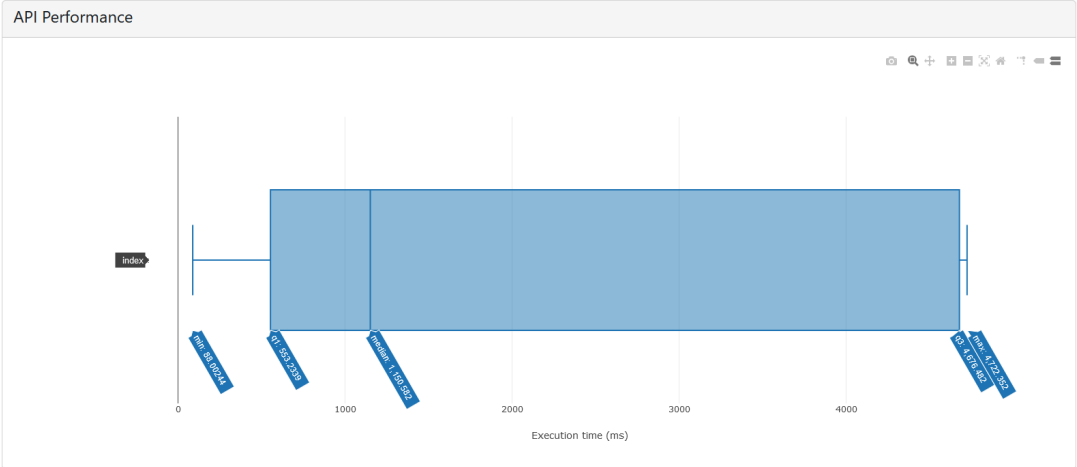
Le dispositif de monitoring propose plusieurs fonctionnalités essentielles :

- **Surveillance des performances** : Grâce à Flask Monitoring Dashboard, les temps de réponse et le nombre de requêtes sont surveillés en temps réel pour garantir des performances optimales.
- **Alertes et notifications** : Le système génère des alertes automatiques lorsqu'un seuil critique est atteint, comme des erreurs ou des ralentissements importants.
- **Visualisation des données** : Plotly permet la création de graphiques dynamiques pour afficher les données de trafic, facilitant l'analyse visuelle des tendances.
- **Analyse des erreurs** : Le module logging capture les erreurs et les enregistre dans un fichier de log, avec des horodatages précis, offrant une vue détaillée des incidents pour une résolution rapide.

2. Seuils d'Alerte et Méthodes de Détection des Incidents

Les seuils d'alerte sont définis pour surveiller les performances de l'application et détecter les anomalies. Par exemple, un seuil de 2 secondes a été fixé pour le temps de réponse des requêtes, au-delà duquel une alerte est déclenchée. Les incidents sont détectés automatiquement grâce à des vérifications périodiques des métriques et à des règles basées sur ces seuils. Lorsque les seuils sont dépassés, des alertes sont envoyées par email aux administrateurs du système.

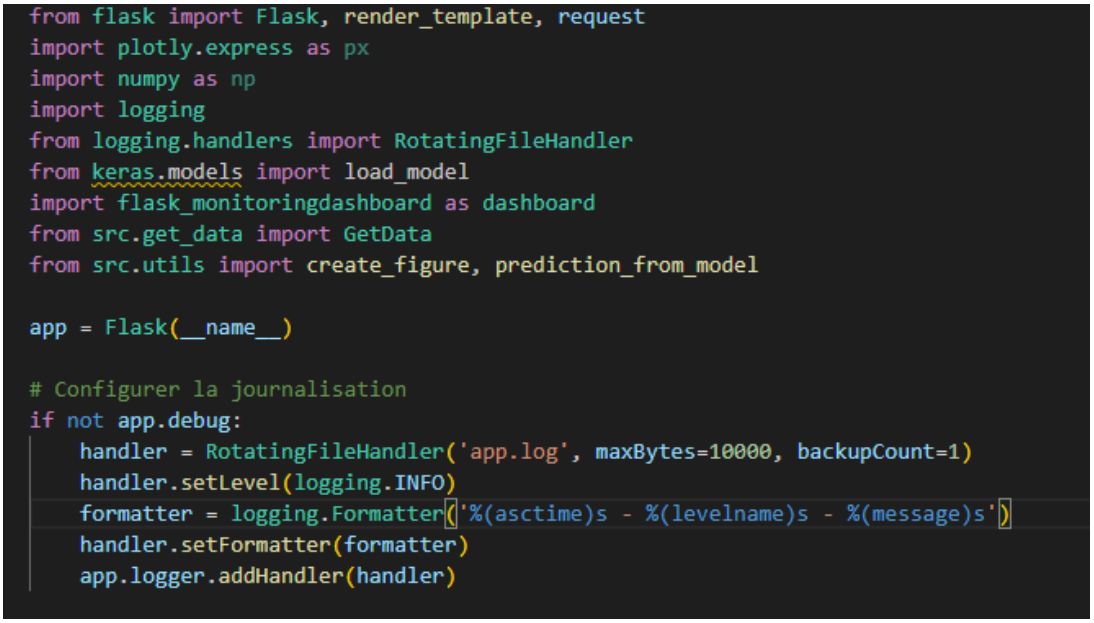
```python
def monitor_response_time(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        start_time = time.time()
        response = f(*args, **kwargs)
        duration = time.time() - start_time

        # Check if the duration exceeds the threshold
        if duration > 2:  # seuil de 2 secondes
            app.logger.warning(f"High response time: {duration} seconds for {request.path}")

        return response

    return decorated_function
```



3. Méthode pour Enregistrer les Erreurs dans un Fichier de Journalisation

Pour enregistrer les erreurs, nous avons utilisé le module logging de Python. La configuration est faite pour écrire les erreurs dans un fichier nommé app.log, avec un niveau

de gravité ERROR. Les messages sont formatés avec un horodatage pour faciliter le suivi des erreurs au fil du temps.

- 

```python
from flask import Flask, render_template, request
import plotly.express as px
import numpy as np
import logging
from logging.handlers import RotatingFileHandler
from keras.models import load_model
import flask_monitoringdashboard as dashboard
from src.get_data import GetData
from src.utils import create_figure, prediction_from_model

app = Flask(__name__)

# Configurer la journalisation
if not app.debug:
    handler = RotatingFileHandler('app.log', maxBytes=10000, backupCount=1)
    handler.setLevel(logging.INFO)
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    handler.setFormatter(formatter)
    app.logger.addHandler(handler)
```

4. Description des Incidents, Méthode de Résolution, et Périmètre Impacté

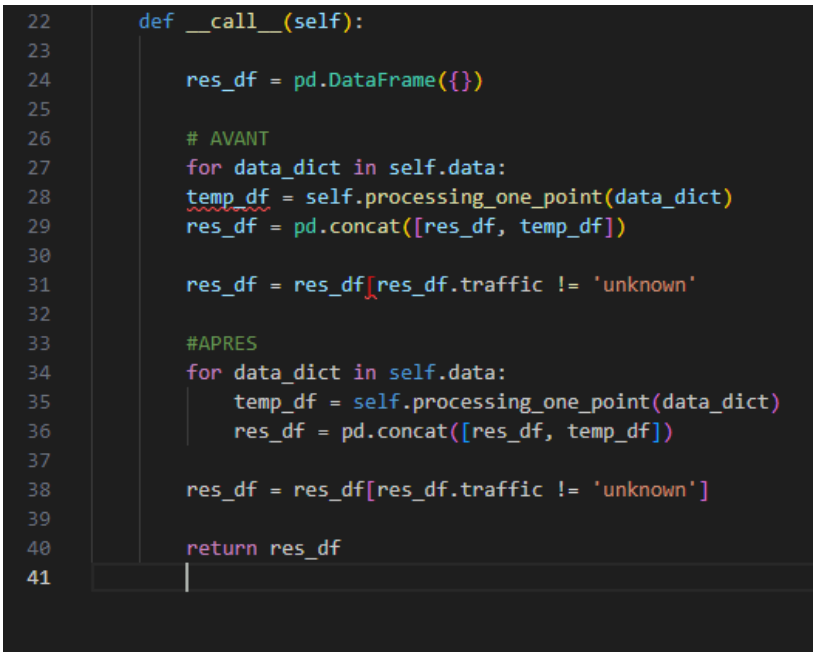
1. L'erreur rencontrée est un `IndentationError`

Analyse de l'erreur :

- Problème : L'instruction `temp_df = self.processing_one_point(data_dict)` n'est pas correctement indentée. En Python, l'indentation est cruciale pour définir les blocs de code, notamment pour les boucles, les conditions, et les fonctions. L'absence d'indentation indique à Python que le bloc de code après le `for` est manquant.
- Impact : Cette erreur empêche le script de s'exécuter, ce qui bloque l'importation de la classe `GetData` dans votre fichier `app.py`, et par conséquent, l'application Flask ne démarre pas.

Correction proposée :

Pour corriger cette erreur, il suffit d'ajouter une indentation correcte après la déclaration de la boucle `for` :

- 

```
22      def __call__(self):
23
24          res_df = pd.DataFrame({})
25
26          # AVANT
27          for data_dict in self.data:
28          temp_df = self.processing_one_point(data_dict)
29          res_df = pd.concat([res_df, temp_df])
30
31          res_df = res_df[res_df.traffic != 'unknown'
32
33          #APRES
34          for data_dict in self.data:
35              temp_df = self.processing_one_point(data_dict)
36              res_df = pd.concat([res_df, temp_df])
37
38          res_df = res_df[res_df.traffic != 'unknown']
39
40          return res_df
41
```

2. L'erreur rencontrée ici est une `SyntaxError`

Analyse de l'erreur :

- Problème : La ligne de code ci-dessus commence une opération de filtrage sur le DataFrame res_df, mais le crochet ouvrant [n'a pas de crochet fermant], ce qui est nécessaire pour compléter l'expression.
- Impact : Cette erreur syntaxique empêche le script de se compiler et de s'exécuter, bloquant ainsi le bon fonctionnement de l'application.

Correction proposée :

Pour corriger cette erreur, il suffit de fermer correctement le crochet] à la fin de la ligne :

```
# AVANT
res_df = res_df[res_df.traffic != 'unknown'

# APRES
res_df = res_df[res_df.traffic != 'unknown']

return res_df
```

3. L'erreur rencontrée ici est une autre `SyntaxError`

Analyse de l'erreur :

- Problème : Il y a une syntaxe incorrecte dans le code, où une ligne manque d'une virgule après l'argument zoom=10.
- Cause probable : Le message d'erreur indique que la syntaxe est invalide, probablement en raison d'une virgule manquante après cette ligne. Dans le contexte de la création d'une figure avec Plotly, chaque argument de la fonction doit être séparé par une virgule si d'autres arguments suivent.
- Impact : Cette erreur empêche l'importation correcte du module utils.py dans le fichier app.py, bloquant ainsi l'exécution de l'application.

Correction proposée :

Pour résoudre ce problème, il est nécessaire d'ajouter une virgule après l'argument zoom=10, surtout si d'autres paramètres sont définis après. Cela garantira que la syntaxe est correcte et que le code peut s'exécuter sans erreur.

-
```
fig_map = px.scatter_mapbox(
        data,
        title="Traffic en temps réel",
        color="traffic",
        lat="lat",
        lon="lon",
        color_discrete_map={'freeFlow':'green', 'heavy':'orange', 'congested':'red'},
        zoom=10
        height=500,
        mapbox_style="carto-positron"
)
```

4. L'erreur rencontrée : Clé Inexistante dans le Dictionnaire

Analyse de l'erreur :

- Problème : Le code tente d'accéder à une clé appelée 'traffic_status' dans un dictionnaire data_dict. Cependant, cette clé n'existe pas dans les données.
- Cause probable : Il semble y avoir une confusion entre les noms des clés. Le code tente d'accéder à 'traffic_status', mais la clé correcte dans les données pourrait être 'trafficstatus' (comme observé dans la liste des clés utilisées dans le dictionnaire).
- Impact : Cette erreur provoque un plantage du programme, empêchant l'application de traiter les données et de les afficher correctement.

Correction proposée :

Pour résoudre ce problème, il faut s'assurer que toutes les clés référencées dans le dictionnaire existent réellement dans les données. Dans ce cas, la clé 'traffic_status' doit être remplacée par 'trafficstatus' dans le code pour correspondre à la clé présente dans le dictionnaire data_dict.

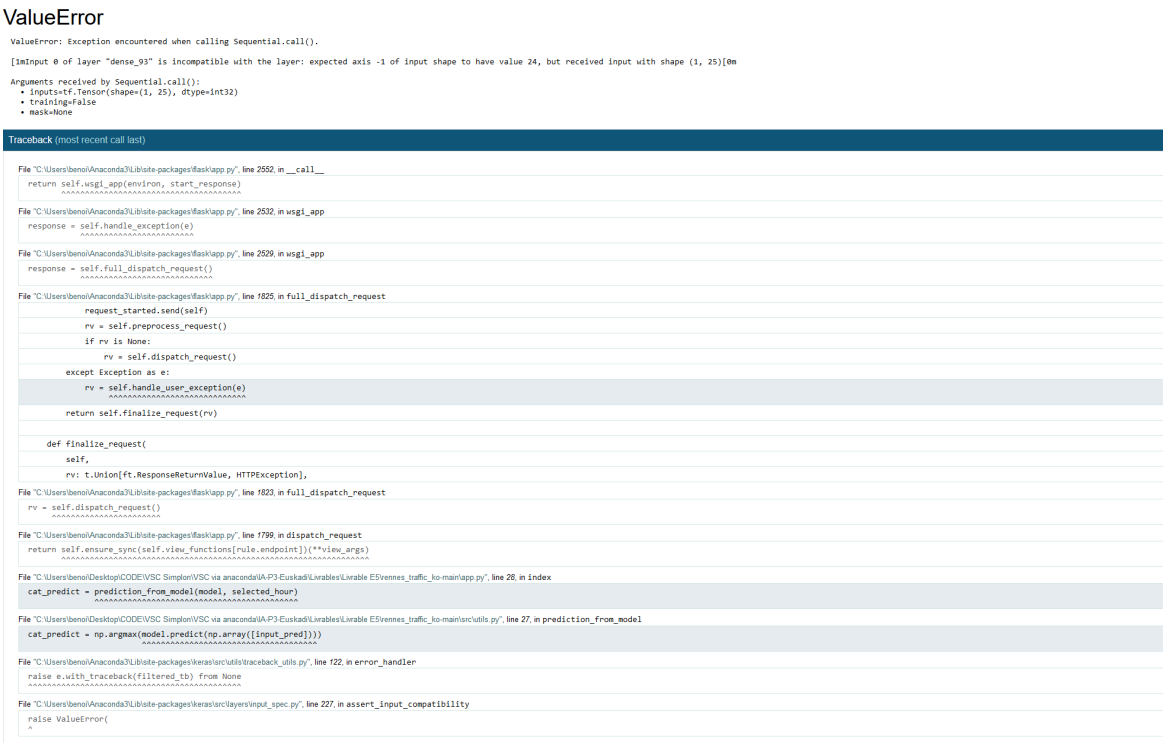
```
temp = pd.DataFrame({key:[data_dict[key]] for key in ['datetime', 'traffic_status',
```

5. La même erreur pour les clés latitude et longitude inexistante

Cette erreur est due à une faute de frappe dans le nom de la clé 'lattitude' et ‘longitude’, qui devraientt être 'lat' et ‘lon’. Le message d'erreur "KeyError: 'lattitude'" indique que le code tente d'accéder à une clé inexistante dans le dictionnaire geo_point_2d, ce qui provoque l'arrêt de l'exécution. Pour corriger cette erreur, il faut vérifier le nom correct de la clé utilisée pour stocker les coordonnées de latitude et le mettre à jour dans le code.

```
temp['lat'] = temp.geo_point_2d.map(lambda x : x['lattitude'])
temp['lon'] = temp.geo_point_2d.map(lambda x : x['longitude'])
```

6. L’erreur rencontrée ici est une `Sequential`



Analyse de l'erreur :

L'erreur que vous rencontrez indique que votre modèle `Sequential` attend une entrée avec une dimension spécifique de `(1, 24)`, mais il reçoit une entrée de forme `(1, 25)`. Cela signifie que le modèle a été conçu pour fonctionner avec un vecteur d'entrée de taille 24, et non 25.

Correction proposée :

Vérifier la Dimension des Données d'Entrée : Assurez-vous que les données d'entrée ont exactement 24 caractéristiques avant de les passer au modèle. Voici un exemple pour vérifier et ajuster la taille de l'entrée

```
def prediction_from_model(model, hour_to_predict):

    input_pred = np.array([0]*24)
```

5. Conclusion

En conclusion, le dispositif de monitorage applicatif mis en place utilise Flask Monitoring Dashboard pour assurer une surveillance en temps réel des performances de l'application,

incluant la détection des incidents et la génération d'alertes automatiques lorsque des seuils critiques sont dépassés. Grâce à l'intégration des technologies telles que Flask, Plotly, Keras, et le module logging, le système offre des fonctionnalités essentielles comme le suivi des temps de réponse, l'analyse des erreurs, et la visualisation des données de trafic.

Les seuils d’alerte ont été définis pour détecter rapidement les anomalies, et un mécanisme de journalisation des erreurs permet un suivi précis des incidents. Les principales erreurs rencontrées, allant des problèmes de syntaxe à des incompatibilités de données pour le modèle machine learning, ont été identifiées, analysées et corrigées efficacement, démontrant une approche rigoureuse dans la résolution des problèmes.

Ainsi, le dispositif de monitorage est pleinement opérationnel, répond aux exigences demandées, et assure une gestion proactive des performances de l'application, contribuant à la stabilité et à la fiabilité de l'ensemble du système.