

# Debugger et monitorer

## brief cas pratique E5

28/08/2024 - Marc-Antoine Abadie

### Introduction

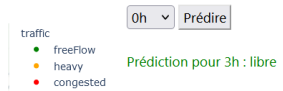
Ce document présente la correction et le monitoring d'une application fictive (Rennes Traffic) comprenant plusieurs erreurs non diagnostiquées. Il vise à montrer la démarche employée et les outils utilisés pour corriger et optimiser le projet étudié.

### Débogage de l'application

Cette application ne possède aucun outil de monitoring ou de logging. Pour détecter les erreurs de syntaxe, l'application est directement lancée via la commande **flask --app app.py --debug run** puis l'application accédée via l'url <http://127.0.0.1:5000/>. Les premières traces de messages d'erreur sont remontées via le terminal d'exécution de l'application (erreurs de syntaxe, etc.), analysées et corrigées les unes après les autres. Les erreurs plus complexes sont découvertes via navigation dans l'application.

Voici le récapitulatif des erreurs identifiées :

Incident	Périmètre impacté	Compréhension de la panne	Résolution / test
Erreurs techniques au lancement de l'application. Ex : File "C:\Users\Utilisateur\Documents\Briefs\Brief_E5_DEBUG\Brief_E5_debug_mabadie\rennes_traffic_ko-main\src\get_data.py", line 30 res_df = res_df[res_df.traffic != 'unknown' ^ SyntaxError: '[' was never closed	Front-end / Back-end	Lecture du message d'erreur en terminal. Problèmes identifiés : <ul style="list-style-type: none"><li>• Erreurs de syntaxe</li><li>• Indentation</li><li>• Problèmes de ponctuation</li><li>• Clés / variables / fichiers mal nommés (index.html)</li><li>• Colonnes de dataframe erronées</li></ul>	Correction du code assistée par le message d'erreur remonté. Test par lancement de l'application pour contrôler la disparition du message d'erreur, puis test bout en bout.

<p>Erreur technique lors de la navigation.</p> <p>Ex : Input 0 of layer "dense_93" is incompatible with the layer: expected axis -1 of input shape to have value 24, but received input with shape (1, 25).</p>	Backend	<p>Lecture du message d'erreur remonté par le terminal. Problème : la dimension de l'objet attendue est 24 (24 heures), non 25.</p>	<p>Correction du code assistée par le message d'erreur remonté. Test par navigation.</p>
<p>Erreur de conception : L'heure de la prédiction n'est jamais affichée.</p>	Frontend	<p>Aucun champ n'est visible sur l'IHM.</p>	<p>Ajout de l'heure lors de l'affichage de la prédiction :</p> 
<p>Erreur de conception : Le nom de l'application dans l'onglet est "Scatter Plot".</p>	Frontend	<p>Le nom est mal défini dans la balise &lt;title&gt; du fichier html.</p>	<p>On change la valeur pour le champ &lt;title&gt; en remplaçant par "Traffic Rennes". Le contenu de l'onglet est ensuite vérifié.</p>
<p>Erreur de conception : L'application est en français, or les libellés de la carte sont en anglais (congested, freeflow, dense).</p>	Frontend	<p>Les valeurs utilisées proviennent directement de la source de données.</p>	<p>On ajoute une map de traduction dans la méthode create_figure().</p>
<p>Erreur de syntaxe : &lt;h4&gt;Choisissez une heure :&lt;/h1&gt;</p>	Frontend	<p>Détectée via les fonctions d'inspection du navigateur.</p>	<p>On corrige la balise avec h4.</p>
<p>Erreur de conception : Lorsqu'on ouvre l'url pour la première fois, la map ne s'affiche pas.</p>	Backend	<p>L'objet contenant la map s'initialise lors de la sollicitation de l'url (dans les routes).</p>	<p>On crée la map à l'initialisation de l'application.</p>
<p>Erreur de conception : Oubli de modules python non utilisés (surcharge inutile).</p>	Backend	<p>Imports python inutiles identifié grâce à l'IDE.</p>	<p>Suppression des imports.</p>

## Mise en place d'une solution de journalisation pour l'application

On utilise le module python logging que l'on importe et configure dans le fichier racine de l'application :

```
# logging configuration - can be s
logging.basicConfig(
    level=logging.ERROR,
    format="{asctime} - {levelname} - {message}",
    style="{",
    filename="logs/flask_app.log",
    encoding="utf-8",
    filemode="a")
```

Ce module permet avec cette seule configuration de ne remonter dans le journal que les avertissements de niveau ERROR ou plus grave. On peut l'utiliser par exemple pour contrôler la récupération des données en ligne.

Flask a son propre système de journalisation que l'on paramètre aussi sur ERROR pour ne pas encombrer le journal :

```
# Set the logging level for the werkzeug logger from Flask to ERROR
logging.getLogger('werkzeug').setLevel(logging.ERROR)
```

On définit ensuite, dans le fichier utils.py, un logger que l'on pourra ensuite exploiter partout dans l'application :

```
def logger_error(name):
    """
    logging method to use through all app
    """
    logger = logging.getLogger(name)
    console_handler = logging.StreamHandler()
    file_handler = logging.FileHandler("logs/flask_app.log", mode="a", encoding="utf-8")
    logger.addHandler(console_handler)
    logger.addHandler(file_handler)
    return logger
```

## Mise en place d'une solution de monitoring

Flask-MonitoringDashboard est un module qui facilite le débogage et l'optimisation d'applications python Flask avec un dashboard de monitoring. Son premier intérêt est de surveiller les performances pour chaque point d'accès. Il permet la détection automatique des exceptions en référençant leur trace pour faciliter leur correction.

Le module fournit aussi un suivi des performances de l'application web (requêtes, mémoire, utilisation du processeur). S'il est configuré de façon adaptée, il peut suivre les requêtes et l'IP

par utilisateur. Enfin sa fonctionnalité de profilage permet d'identifier et optimiser les fonctions et points d'accès de l'application qui posent problème.

Flask-MonitoringDashboard est facilement personnalisable. On peut ainsi modifier l'affichage des visualisations existantes ou en ajouter de nouvelles. Le dashboard est développé et donc livré / conteneurisé conjointement avec l'application.

A titre d'exemple, voici l'interface qui donne une vue d'ensemble sur les différents points d'accès de l'application sollicités. Pour chaque point d'accès on obtient quelques informations essentielles :

Dashboard Overview

Number of hits

Median request duration (ms)

Show  entries

Blueprint:

Search:

Endpoint	Today	Last 7 days	Overall	Last requested	Monitoring-level*
index	28	28	28	12 minutes ago	<div><div>0</div><div>1</div><div>2</div><div>3</div></div>

Previous

Next

Le module est configuré ainsi dans le projet :

```
[dashboard]
APP_VERSION=1.0
# GIT=/<path to your project>/..git/
CUSTOM_LINK=dashboard
MONITOR_LEVEL=3
# If request takes more time than (OUTLIER_DETECTION_CONSTANT * average_request_time)
# then extra information is logged into the dashboard
OUTLIER_DETECTION_CONSTANT=2
SAMPLING_PERIOD=20
ENABLE_LOGGING=True
BRAND_NAME=Rennes Traffic Monitoring Dashboard
TITLE_NAME=RennesTraffic-MonitoringDashboard
DESCRIPTION=Automatically monitor the evolving performance of Flask/Python web services
SHOW_LOGIN_BANNER=True
SHOW_LOGIN_FOOTER=True

[authentication]
USERNAME=admin
PASSWORD=admin
SECURITY_TOKEN=cc83733cb0af8b884ff6577086b87909

[database]
TABLE_PREFIX=fmd
DATABASE=sqlite:///dashboard/dashboard.db

[visualization]
TIMEZONE=Europe/Amsterdam
COLORS={'main': '[0,97,255]',
        'static': '[255,153,0]'}
```

Il est ensuite initialisé à la racine de l'application :

```
if __name__ == '__main__':  
    app.run(debug=True)  
  
# monitoring dashboard configuration  
dashboard.config.init_from(file='config/dashboard_config.cfg')  
dashboard.bind(app)
```

Une fois l'application lancée on peut accéder au dashboard à l'url suivante : <http://localhost:5000/dashboard>

Selon le niveau de monitoring (`MONITOR_LEVEL`) choisi dans le fichier de configuration, l'utilisateur aura accès à des informations plus ou moins complètes sur l'application monitorée. On distingue 4 niveaux, de 0 à 3, chaque niveau implémentant les fonctionnalités proposées par les niveaux précédents en plus de ses propres fonctionnalités.

- Niveau 0 : seules les dates auxquelles les points d'accès de l'application sont sollicités sont présentées
- Niveau 1 : est accessible une gestion des performances et de la fréquence d'utilisation pour chaque point d'accès
- Niveau 2 : ajout de visualisations des outliers lors de la sollicitation des points d'accès (temps de traitement hors norme)
- Niveau 3 : via un thread parallèle à celui gérant les requêtes de l'application, le niveau 3 donne des informations sur la vitesse d'exécution des requêtes (temps moyen, temps total)

Le niveau 3 est particulièrement intéressant pour identifier puis optimiser les point faibles de l'application. Il permet également de relativiser l'importance d'une fonctionnalité en fonction de sa fréquence d'utilisation. C'est donc un outil de choix pour prioriser les futures évolutions / correctifs applicatifs.

Par ailleurs, pour chaque niveau de monitoring, plusieurs visualisations sont proposées en complément que nous ne détaillerons pas ici.

## Conclusion

L'application Rennes trafic, sujette à de nombreux incidents, a été corrigée par débogage puis optimisée en ajoutant un outil de journalisation et un système de monitoring. Ces solutions ont l'avantage d'être développées dans le projet lui-même et sont hautement paramétrables. Elles permettent également de faciliter et contrôler les évolutions futures du projet : nouveaux points d'accès et/ou nouvelles fonctionnalités.

