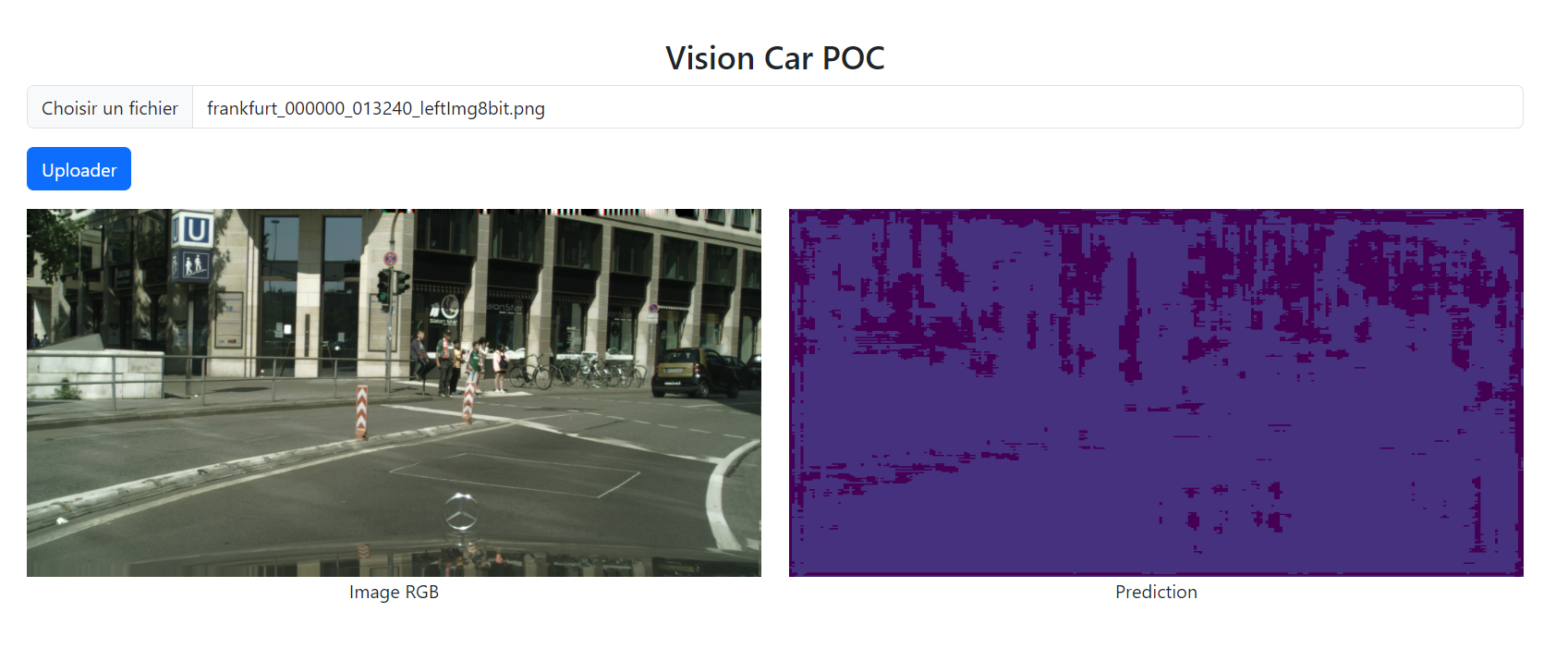
**Livrable vision-car**

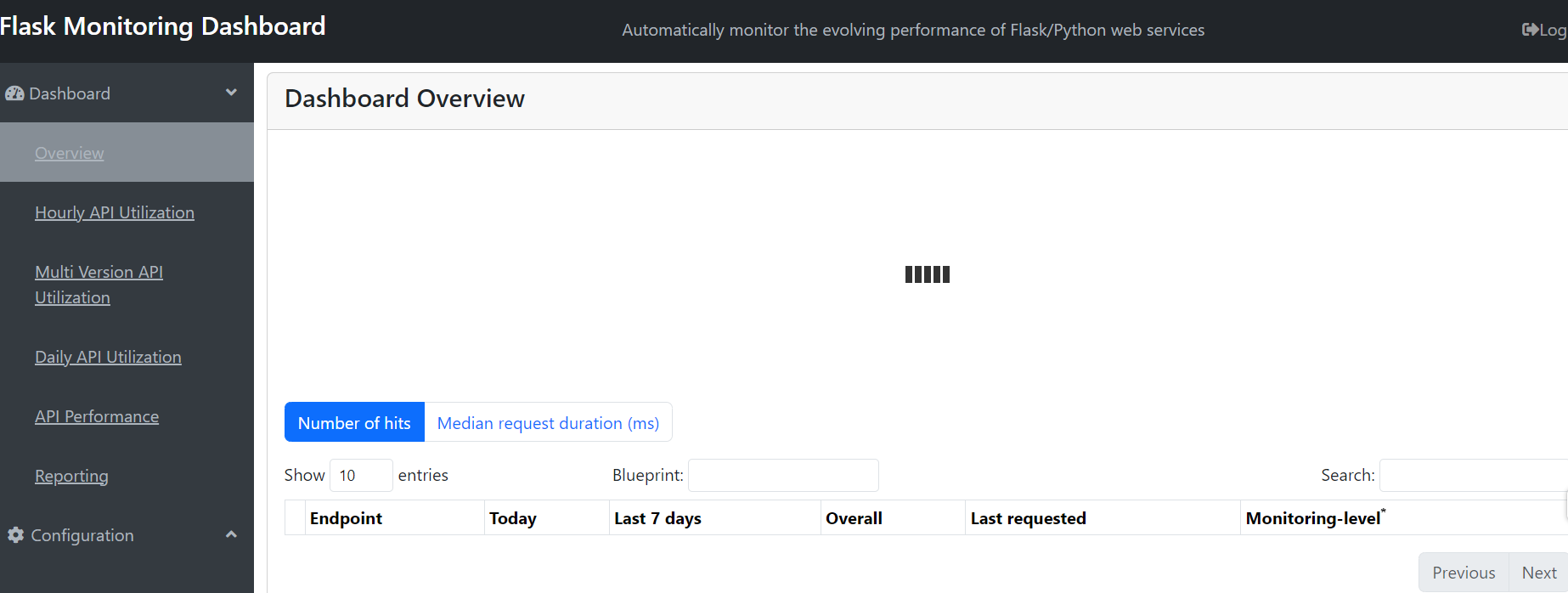
****

Dans le cadre du projet Vision-Car, j'ai mis en place un dispositif de monitoring applicatif utilisant Flask Monitoring Dashboard.

Ce système permet de surveiller en temps réel les performances de notre application et d'identifier rapidement les problèmes. Les principales fonctions incluent :

**1. Présentation du Dispositif de Monitoring applicatif**

**Fonctions Principales**

* **Collecte et Visualisation des Statistiques** : Le dispositif de monitoring, en l’occurrence **Flask Monitoring Dashboard**, permet de surveiller en temps réel les statistiques de l'application, telles que les temps de réponse, le nombre de requêtes par minute, et la consommation de ressources.
* **Détection des Erreurs** : Le monitoring permet de suivre les erreurs levées dans l'application, ce qui aide à identifier rapidement les problèmes.
* **Alertes sur les Seuils** : Le système peut être configuré pour envoyer des alertes lorsque certaines conditions sont rencontrées, comme un temps de réponse trop long ou un taux d'erreurs élevé.
* 

**Seuils d'Alerte et Méthodes de Détection des Incidents**

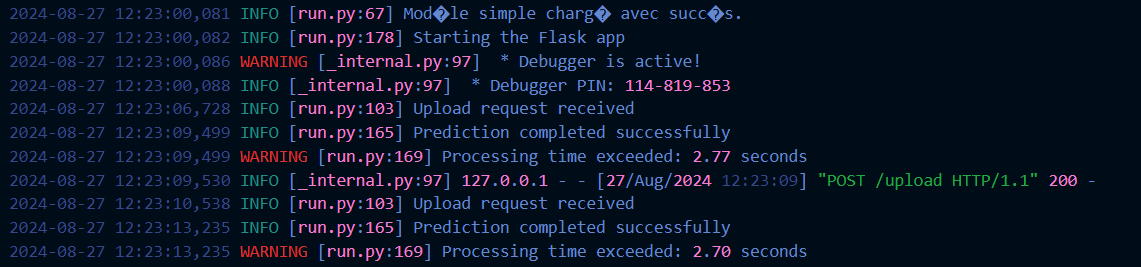
* **Temps de Réponse Supérieur à 1 Seconde** : Une alerte est générée si le temps de traitement d'une requête dépasse 1 seconde. Ceci est essentiel pour garantir une bonne expérience utilisateur.
* **Taux d'Erreur Élevé** : Une alerte est émise si plus de 5% des requêtes rencontrent une erreur dans une période définie (par exemple, sur une heure).
* **Surveillance des Erreurs** : Le module logging enregistre toutes les erreurs critiques dans un fichier de log pour une analyse ultérieure.

**2. Configuration du Système de Journalisation**

Pour assurer un suivi précis des erreurs, j'ai configuré le module logging de Python pour n'enregistrer que les erreurs avec horodatage. Chaque entrée de log contient des informations détaillées comme le fichier concerné et le numéro de ligne, ce qui facilite le diagnostic et la résolution rapide des problèmes.

**Méthode d’Enregistrement des Erreurs avec Horodatage**

La journalisation est configurée pour n’enregistrer que les erreurs (ERROR level) avec des informations détaillées comprenant l'horodatage, le fichier concerné, et le numéro de ligne.



**3. Récapitulatif des Bugs, Corrections et Monitoring pour le POC de Reconnaissance de Véhicules Autonome**

Récapitulatif des différents bugs rencontrés lors du développement du POC de reconnaissance de véhicules autonomes, les solutions apportées pour les corriger, ainsi que les outils et méthodes mis en place pour le monitoring de l'application :

**Problème : Environnement virtuel non créé ou non activé**

**Symptôme :**

* Les modules requis ne sont pas disponibles, ou les modules installés ne sont pas détectés par l'interpréteur Python utilisé.

**Solution :**

1. **Création de l'environnement virtuel :**
   * Créez un environnement virtuel dans votre projet pour isoler les dépendances :

python -m venv env

1. **Activation de l'environnement virtuel :**
   * Activez l'environnement virtuel avant d'installer des dépendances :
     + **Sous Windows** :

.\env\Scripts\activate

**Problème : Fichier requirements.txt manquant**

**Symptôme :**

* Difficulté à reproduire l'environnement de développement, ou besoin de réinstaller manuellement les dépendances.

**Solution :**

1. **Génération du fichier requirements.txt :**
   * Après avoir installé toutes les dépendances nécessaires dans l'environnement virtuel, générez un fichier requirements.txt qui liste toutes les dépendances avec leurs versions exactes :

pip freeze > requirements.txt

1. **Utilisation du fichier requirements.txt :**
   * Ce fichier peut ensuite être utilisé pour installer toutes les dépendances dans un nouvel environnement virtuel ou par d'autres développeurs travaillant sur le projet :

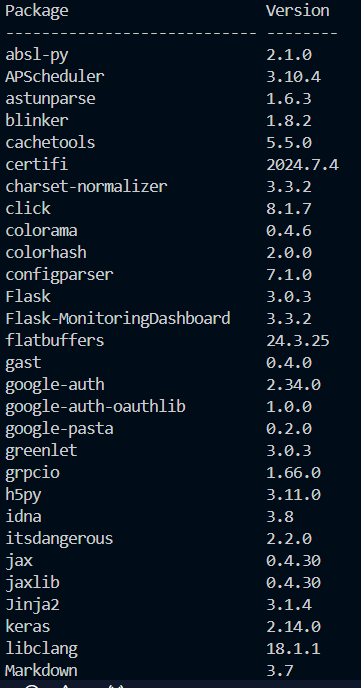
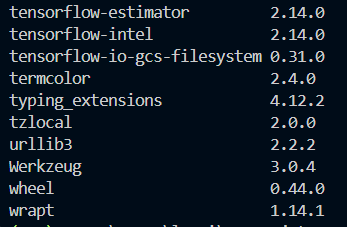
pip install -r requirements.txt

**Problème : Installation des dépendances**

**Symptôme :**

* Erreurs lors de l'importation de modules Python, telles que ModuleNotFoundError, en raison de l'absence de certaines bibliothèques.

**Solution :**

1. **Installation des dépendances nécessaires :**
   * Utilisez pip list pour vérifier que toutes les dépendances ont été correctement installées dans l'environnement virtuel.
   *   

**Erreur : Python n'est pas trouvé dans le terminal**

**Symptôme :**

* Lors de l'exécution du script Python, une erreur indiquait que Python n'était pas installé ou que le chemin d'accès n'était pas configuré correctement.

**Solution :**

1. **Installation de Python 3.11 :**
   * Installer Python à partir du site officiel si nécessaire, et s'assurer de cocher l'option "Add Python to PATH" lors de l'installation.

**Erreur de chargement du modèle Keras**

**Symptôme :**

* Lors de l'importation de keras.models, une erreur ModuleNotFoundError pour le module tensorflow est apparue.

**Solution :**

1. **Installation de Keras 2 :**

pip install keras==2.4.3

**6. Installer une version compatible de TensorFlow**

version actuelle de Python (3.11.7) et installation de TensorFlow 2.14.0 :

pip install tensorflow==2.14.0

**Erreur de Template introuvable (TemplateNotFound)**

**Symptôme :**

* Une erreur jinja2.exceptions.TemplateNotFound: home.html ou index.html est survenue lorsque l'application a tenté de rendre un template inexistant ou mal placé.

**Solution :**

1. **Vérification de l'emplacement du fichier :**
   * Assurez-vous que le fichier index.html est dans le dossier templates (avec un "s" à la fin) à la racine du projet.
2. **Ajustement de la route Flask :**
   * Si le nom du fichier a été modifié (par exemple, de home.html à index.html), ajustez la route Flask pour refléter ce changement :

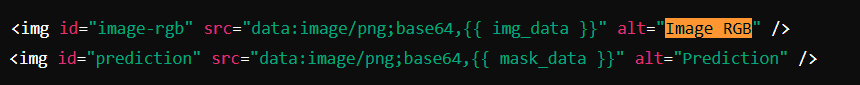
@app.route('/')

def index():

return render\_template('index.html')

**Vérification du chemin des images**

Les images encodées en base64 sont bien intégrées dans le HTML. Une modification du template HTML pour afficher les images en utilisant des balises <img> avec les sources en base64.

Voici un exemple de modification à effectuer dans votre template HTML (index.html) :**2. Vérification du retour JSON**

Le JSON retourné par le serveur contient bien les données encodées en base64 :



**Erreur d'Indentation et de Logique**

**Symptôme :**

* Le code contenait des erreurs d'indentation et de logique, ce qui empêchait certaines parties du code de s'exécuter correctement.

**Solution :**

1. **Réajustement de l'Indentation :**
   * Correction de l'indentation dans la route upload\_image pour s'assurer que toutes les opérations sont exécutées correctement
2. La ligne img = Image.open(image) est dans le mauvais bloc et ne sera jamais exécutée si le nom de fichier est valide. Elle doit être placée après le bloc if image.filename == ''.

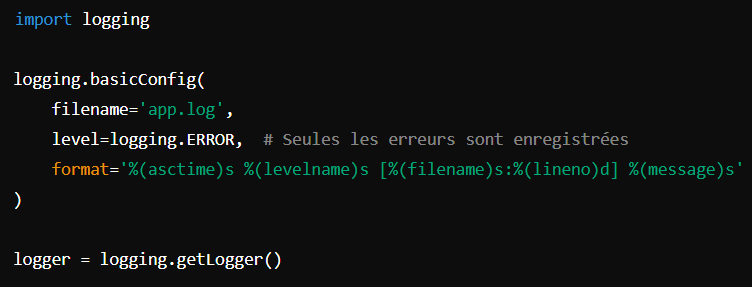
**Problèmes de Journalisation et de Monitoring**

**Symptôme :**

* L'application ne contenait aucun mécanisme de journalisation pour capturer les erreurs, ni de monitoring pour surveiller les performances en temps réel.

**Solution :**

1. **Mise en place de la Journalisation :**
   * Utiliser le module logging pour capturer les erreurs avec horodatage :



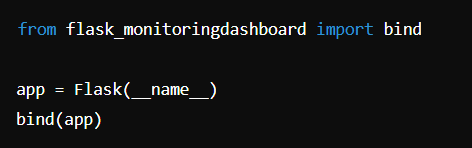
1. **Configuration du Monitoring :**
   * **Arrêter l'application Flask** si elle est en cours d'exécution
   * **Créer un compte administrateur** manuellement en utilisant une commande dans le terminal. Vous pouvez utiliser la commande suivante :



* + Installer et configurer Flask-MonitoringDashboard pour surveiller les performances de l'application en temps réel :



* + Intégrer Flask-MonitoringDashboard dans l'application Flask :



* + Accédez au dashboard via http://127.0.0.1:8000/dashboard/ pour surveiller les performances et les requêtes



**Voici quelques points clés concernant le comportement de votre application :**

1. Modèle Simple Chargé : Le journal montre que le modèle simple a été chargé avec succès. Cela signifie que les modèles unet\_vgg16\_categorical\_crossentropy\_raw\_data.keras et unet\_v2.keras n'ont pas pu être chargés, ce qui a entraîné le recours au modèle simple.
2. Temps de Traitement : Le temps de traitement de l'image a dépassé le seuil défini (1 seconde) et a pris environ 3,34 secondes. Cela est signalé par un avertissement dans les logs.
3. Flask Monitoring Dashboard : Accès à l'interface de monitoring. Le journal indique des requêtes HTTP pour le dashboard, y compris le login et le chargement de ressources statiques.