Projet Plate eyes
Livrables semaine 1

Johan Mikami[JMi], Stéphane Nascimento Santos[SNS],
Julien Mühlemann[JMu], Dr. Ing. Julien Billeter[JB]

26 août 2024

1 Description du projet

Les parkings font face à plusieurs défis majeurs qui compromettent la fluidité et l'efficacité de leur gestion. Les systèmes traditionnels, reposant souvent sur des barrières, des badges ou des tickets, entraînent des ralentissements significatifs aux points d'entrée et de sortie. Cela se traduit par des files d'attente, un stress accru pour les conducteurs, ainsi qu'une consommation excessive de carburant et une augmentation des émissions de gaz polluants dues aux moteurs au ralenti. Par ailleurs, le bruit généré par les véhicules en attente crée une ambiance sonore désagréable, perturbant le confort des usagers.

C'est dans ce contexte que notre projet s'inscrit, avec pour ambition de révolutionner l'expérience des parkings en éliminant ces inefficacités.

1.1 Objectif

Le projet vise à développer un système innovant de reconnaissance automatique des plaques d'immatriculation pour les véhicules entrant et sortant des parkings, spécialement conçu pour les institutions. Ce système, fonctionnant en continu, permettra d'optimiser la circulation en éliminant les obstacles physiques. En effet, l'entrée du parking sera équipée d'une caméra de lecture des plaques et d'une barrière afin de s'assurer que le véhicule entrant dispose d'une plaque d'immatriculation. La sortie quant à elle sera équipée d'une caméra sans barrières afin d'accélérer les flux de véhicules. Un ralentisseur sera probablement nécessaire comme un feu ou un dos d'âne.

Les véhicules pourront sortir du parking à leur guise et ainsi procéder au paiement jusqu'à un certain délai. Le parking proposera également un système d'abonnement. Une interface WEB permettra aux utilisateurs de s'acquitter de leurs frais de parking ou de s'inscrire et de recevoir ainsi une facture mensuelle.

En outre, le système propose des solutions aux problèmes environnementaux en diminuant non seulement le bruit des moteurs dû aux temps d'arrêts mais aussi en réduisant les émissions polluantes, offrant ainsi un environnement plus serein et durable pour les usagers.

1.2 Exigences fonctionnelles

Le système développé dans ce projet devra respecter les exigences fonctionnelles suivantes :

Serveur Frontend :

- être capable de naviguer entre les pages de l'interface graphique utilisateur ou administrateur

Les utilisateurs devront pouvoir :

- chercher leur plaque, afficher leurs factures ouvertes et afficher un moyen de paiement,
- créer un compte et saisir leurs informations (y compris leurs identifiants),
- saisir leurs identifiants pour accéder à leur compte,
- saisir un moyen de paiement dans leur compte,
- mettre à jour leurs informations dans leur compte.

L'administrateur devra pouvoir :

- modifier les tarifs horaires,

- consulter les logs,
- résoudre les alertes urgentes (numéro de plaque non identifié),
- consulter les factures ouvertes,
- consulter les alertes non urgentes et les régler quand il en a la possibilité (optionnel).

Serveur Backend :

Le serveur Backend sera constitué d'une **API** dont les exigences seront :

- interroger et modifier une base de données relationnelle,
- calculer des montants à payer [1],
- vérifier les transactions monétaires,
- récupérer les requêtes d'une intelligence artificielle de reconnaissance de numéros de plaques,
- stocker sur un disque les images des véhicules dont les conducteurs n'ont pas encore réglé leurs frais,

et d'un module d'**intelligence artificielle** dont les exigences seront :

- détecter un véhicule dans un flux vidéo,
- identifier le numéro de plaque d'un véhicule sur un flux vidéo,
- envoyer en temps réel le numéro de plaque à l'API.

1.3 Exigences non-fonctionnelles

Le système résultant de ce projet respectera les exigences non-fonctionnelles suivantes :

Serveur Frontend :

L'utilisateur devra pouvoir :

- utiliser l'interface de paiement sans compte sans aucune aide dans plus de 99.9% des cas,
- utiliser l'interface avec compte sans aucune aide dans plus de 99% des cas.

L'administrateur devra pouvoir :

- utiliser l'interface administrateur après 30 minutes de formation, et ainsi ne nécessiter un support que dans 1 cas par semaine, au plus.

Serveur Backend :

- mettre en oeuvre les règles appropriées de respect de la vie privée (selon la Loi fédérale sur la protection des données, LPD).
- reconnaître le numéro d'une plaque d'immatriculation en moins de 5 secondes avec un taux de succès de 90% au moins.

2 Description préliminaire de l'architecture

L'architecture du projet suit le schéma classique d'une interface web, avec un serveur Frontend et un serveur Backend dédié aux tâches spécifiques du projet. Ce serveur Backend est chargé de la reconnaissance des plaques d'immatriculation à partir d'un flux vidéo. Il est connecté à une base de données relationnelle ainsi qu'à un système de stockage, comme illustré par la Figure 1.

3 Mockup et Landing page

Les maquettes dynamiques (Mockups) des interfaces graphiques du projet pour l'utilisateur et pour l'administrateur ont été réalisées avec Figma et sont accessibles comme hyperliens.

La landing page du projet est située sur notre dépôt GitHub à l'adresse https://jmuhleman.github.io/PDG_gr_12.

1. à partir de données extraites d'une base de données

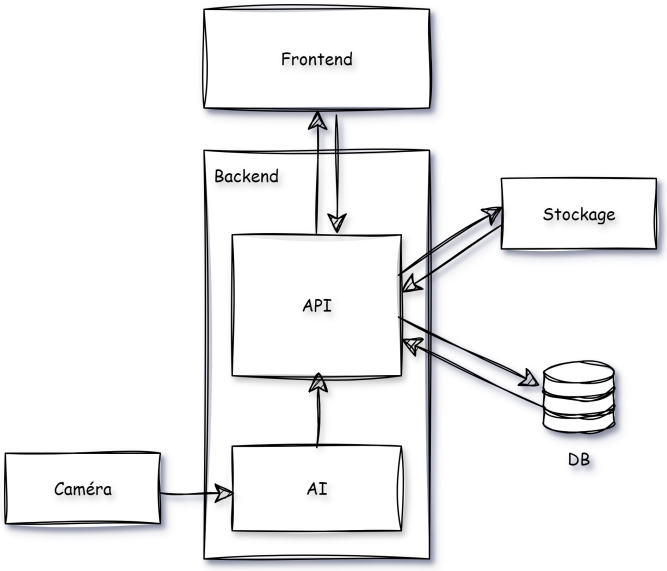

FIGURE 1 – Architecture préliminaire

4 Description des choix techniques

Les choix technologiques concernant le module d'**intelligence artificielle** sont les suivants :

- développement du traitement des images de véhicules du flux vidéo avec un script écrit en Python et interprété avec la version 3.11,
- utilisation d'un modèle open source d'intelligence artificielle de détection d'objets et de segmentation d'images en temps réel nommé *YOLOv8* de chez *Ultralitics* pré-entraîné sur un ensemble de données de chez *Roboflow*,
- utilisation de la bibliothèque *easyOCR* pour la lecture du texte situé sur les images des plaques d'immatriculation,
- utilisation de la bibliothèque *sort* pour le tracking d'objets 2D,
- utilisation de la bibliothèque *cv2* (OpenCV) pour la manipulation des images en Python.

Les choix technologiques concernant les serveurs **Frontend** et **Backend** sont les suivants :

- utilisation du framework *React* pour le *Frontend*,
- utilisation du framework *Flask* en Python pour le *Backend*,
- utilisation de *Postgres SQL* pour la base de données,
- utilisation de la plateforme de paiement *Stripe*.

Le projet sera entièrement containerisé à l'aide de *Docker* et les conteneurs seront publiés sur *DockerHub* comme étape de livraison.

5 Description du processus de travail

Nous adopterons une approche simplifiée du processus *Gitflow* car le processus original est trop complexe pour un projet de petite envergure comme celui-ci. Ainsi, nous aurons une organisation de notre dépôt comme suit dont le processus de travail est schématisé dans la Figure 2 :

- une branche *main* pour le code testé
- une branche *develop* pour le code en développement
- autant de branches *feature* qu'il y aura d'apport de fonctionnalités ou de corrections

Les branches *main* et *develop* seront protégées et des *pull requests* (PR) seront nécessaires pour fusionner les branches *feature* une fois qu'elles seront terminées. A chaque *pull request*, des tests automatiques seront lancés sur le code du Frontend et du Backend.

Dans ce projet, nous suivrons les pratiques *DevOps* présentés dans la Figure 3 qui seront pertinents afin de travailler de la manière la plus efficace possible. En particulier, les pratiques de sécurité et de monitoring ne seront pas abordées.

Nous travaillerons également selon les principes d'agilité de Scrum, avec, en particulier, un Product Owner (JMü), un Scrum Master (JB), un product backlog, et des sprints backlog. Nous adhérons aux 12 principes du Manifeste Agile et aux 4 piliers de l'amélioration continue [2].

2. Plan, Do, Check, et Act

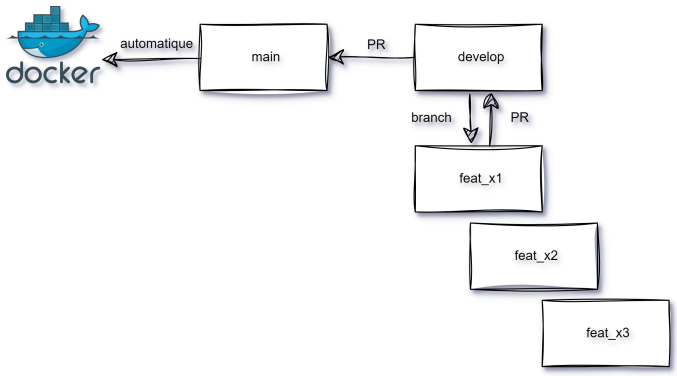

FIGURE 2 – Architecture de travail employée dans ce projet avec des *pull request* (PR) incluant des scripts au format .yml, ainsi que des tests

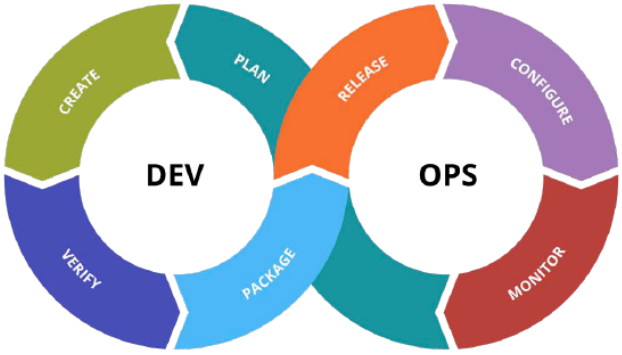

FIGURE 3 – Schéma DEVOPS

6 Mise en place des outils de développement collaboratifs

Nous avons décidé de répartir le travail entre deux équipes distinctes. JB et JMü seront responsables du module Python pour le traitement des images des plaques, tandis que JMi et SNS se concentreront sur la partie web. Chaque équipe fonctionnera de manière autonome, en suivant un modèle de Sprints avec des Sprint Backlogs compatibles entre équipes. Au sein de chaque équipe, le développement se fera en binômes, permettant ainsi une collaboration étroite entre deux personnes. Pour faciliter la communication avec les assistants et professeurs, nous avons mis en place un canal *Microsoft Teams*. En ce qui concerne la communication interne au sein du groupe, nous avons créé un serveur *Discord* ainsi qu'un groupe *WhatsApp*.

7 Mise en place d'un environnement de déploiement

En cas de déploiement, nous envisageons de l'effectuer sur Google App Engine, avec une orchestration gérée par *Kubernetes*. Cette configuration nous offrirait un contrôle plus précis sur le suivi des performances et la scalabilité.

8 Mise en place d'un pipeline de livraison et de déploiement

Un pipeline de livraison CI/CD a été mis en place à l'aide de GitHub Actions. Le pipeline fonctionne de la manière suivante : à chaque *pull request* sur les branches *develop* ou *main*, une action est déclenchée pour exécuter l'*API backend* en Python et réaliser des tests à l'aide de *pytest*. Des tests sont également exécutés sur le *frontend* à l'aide d'un *linter*.

Chaque *pull request* est ensuite mise en attente pour une revue de code par un autre développeur.

Enfin, le pipeline se charge de créer une image *Docker* et de la pousser sur *DockerHub*, rendant ainsi disponible le script *Backend* ainsi que le *Frontend* pour un futur déploiement.