Projet Plate eyes
Identification des plaques d'immatriculation pour les garages

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

6 septembre 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 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 (par exemple 72 heures). 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'arrêt 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 respecte les exigences fonctionnelles suivantes :

Serveur Frontend :

- être capable de naviguer entre les pages de l'interface graphique utilisateur ou administrateur,
- être capable d'adapter son affichage à l'appareil utilisé (mobile, tablette, ou ordinateur) selon le principe du *mobile first*.

Les utilisateurs peuvent :

- 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, et
- mettre à jour leur mot de passe.

L'administrateur peut :

- modifier les tarifs horaires,
- consulter les logs, et
- consulter les factures ouvertes.

Serveur Backend :

Le serveur Backend est constitué d'une **API** dont les exigences sont :

- interroger et modifier une base de données relationnelle,
- calculer des montants à payer [1],
- fermer une facture ouverte lorsque celle-ci est payée,
- 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 sont :

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

1.3 Exigences non-fonctionnelles

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

Serveur Frontend :

L'utilisateur peut :

- utiliser l'interface de paiement sans compte sans aucune aide dans plus de 99.9% des cas, et
- 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), et
- reconnaître le numéro d'une plaque d'immatriculation en moins de 5 secondes avec un taux de succès de 90% au moins.

2 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 le dépôt GitHub du projet et en hyperlien dans ce document.

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

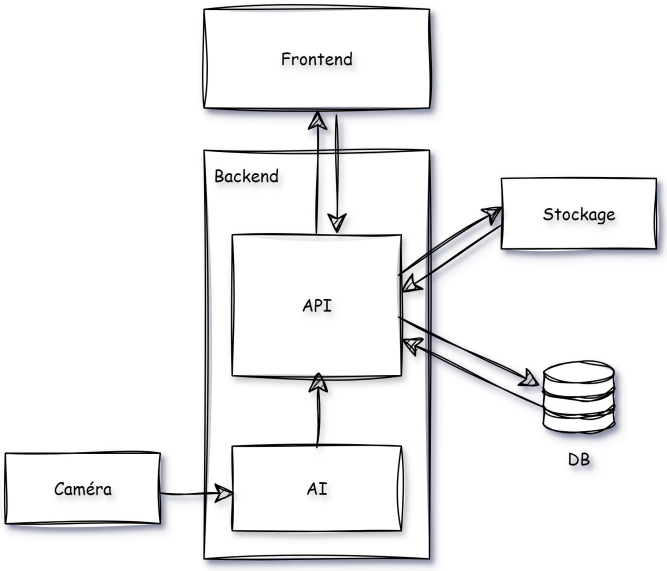

FIGURE 1 – Architecture de Plate eyes

4 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 *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 est entièrement containerisé à l'aide de *Docker* et les conteneurs sont publiés sur *DockerHub* comme étape de livraison, en découpant l'architecture en containers frontend (latest), backend (latest), et DB (Postgres version 16). L'ordre et les paramètres de lancement de ces containers sont spécifiés sur le *readme* du dépôt *GitHub* du projet.

Le projet peut également être lancé localement grâce à un script *docker-compose.yml* situé sur le dépôt *GitHub* du projet.

5 Processus de travail et outils de développement collaboratifs

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

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

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

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

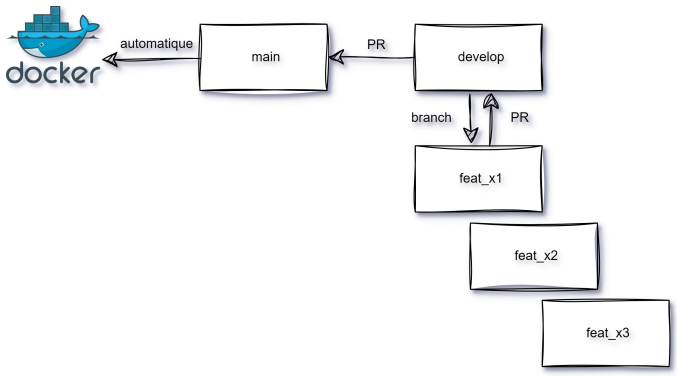

FIGURE 2 – Architecture de travail employée dans ce projet avec des *pull request* (PR) incluant des scripts au format .yml, ainsi que des tests

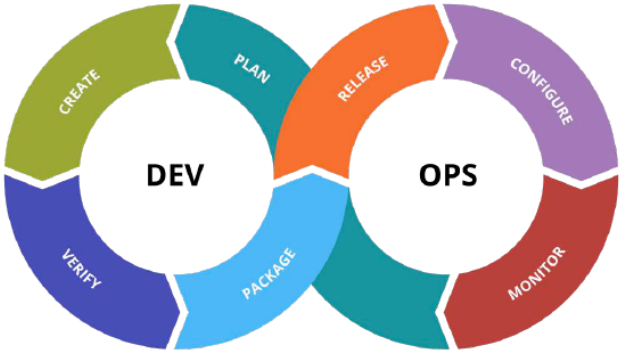

FIGURE 3 – Schéma DEVOPS

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

Le travail a été réparti entre deux équipes distinctes de deux personnes. JB et JMü ont été responsables du module Python pour le traitement des images des plaques et le backend, tandis que JMi et SNS se sont concentrés sur la partie *frontend*. L'équipe backend a travaillé principalement en mode partagé a l'aide de l'extension liveShare de Visual Studio Code, et JMü s'est chargé de synchroniser les mises à jour avec le dépôt GitHub.

Chaque équipe a fonctionné de manière autonome, en suivant un modèle de Sprints avec des Sprint Backlogs compatibles entre équipes. 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* pour l'échange de documents lors des jours de travail à distance, ainsi qu'un groupe *WhatsApp* pour la coordination.

Le travail a été organisé selon le principe des *issues* de GitHub, chaque *issue* devant être créée avant d'entreprendre une tâche. Les objectifs des sprints ont été discutés avec l'ensemble de l'équipe.

Le travail a débuté par la définition des modes de communication entre le backend et le frontend, ainsi que des routes nécessaires pour le frontend. Le premier sprint a porté sur les requêtes de récupération et de recherche des plaques d'immatriculation dans la base de données. Le deuxième sprint a été consacré à l'implémentation des fonctionnalités liées aux données des utilisateurs. Le troisième sprint s'est focalisé sur l'inscription et l'authentification, la modification du mot de passe des abonnés, et des fonctions d'édition (suppression de plaques). Le quatrième et dernier sprint a traité des dernières exigences, notamment la gestion des tarifs via l'interface administrateur.

6 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*

2. Plan, Do, Check, et Act

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 déposer sur DockerHub, rendant ainsi disponible le Backend, le Frontend, et la Base de données pour un futur déploiement.

6.1 Tests de fonctionnalité

Le fonctionnement des fonctions JavaScript a été systématiquement testé après leur implémentation. Cependant, faute de temps, il n'a pas été possible d'écrire des tests préalables spécifiques pour valider leur bon fonctionnement, comme le recommande la bonne pratique du développement logiciel.

Durant le développement de la partie backend, des tests ont été écrits pour chaque route ouverte sur le serveur. Cela a permis de corriger rapidement une grande partie des *bugs* avant la mise en commun du code avec l'équipe frontend.

La plateforme *DockerHub* a été utilisée pour mettre à disposition de tous les membres de l'équipe une image docker de la base de données PostgreSQL contenant des données de test. Cela a permis à l'ensemble de l'équipe de travailler sur un schéma de base de données commun.

7 Perspectives d'améliorations

Plusieurs pistes d'amélioration peuvent être envisagées afin d'optimiser le projet.

Tout d'abord, la mise à jour des informations des utilisateurs est essentielle pour garantir l'exactitude des données et devrait être implémentée.

Ensuite, il serait pertinent d'améliorer la précision de la lecture des caractères des plaques d'immatriculation, en augmentant les taux d'*accuracy* et de *recall* du système de reconnaissance (easyOCR).

Il est également nécessaire de résoudre rapidement les alertes urgentes liées aux plaques non identifiées, tout en permettant de consulter et de traiter les alertes non urgentes lorsque les ressources le permettent.

Une amélioration du module IA est souhaitable pour étendre la prise en charge des plaques d'immatriculation provenant d'autres pays, notamment des pays d'Europe.

L'implémentation d'une fonctionnalité de suppression de compte pour les abonnés serait également à prévoir pour mieux se conformer à la loi fédérale sur la protection des données (LPD). De plus, il serait bénéfique de détecter le véhicule associé à la plaque lors de la reconnaissance afin d'éviter les fraudes avec des dispositifs factices.

Sur le plan technologique, l'utilisation d'une solution plus robuste pour le serveur *backend*, comme Django en remplacement de Flask, permettrait d'améliorer la stabilité et les performances.

En ce qui concerne l'amélioration de l'interface utilisateur, une attention particulière devrait être accordée à l'ergonomie et à l'expérience utilisateur. Une interface plus intuitive et réactive permettrait de simplifier l'utilisation du système, en réduisant le temps nécessaire pour accomplir des tâches courantes et en facilitant la navigation entre les différentes fonctionnalités. Cela inclurait notamment l'optimisation de l'affichage des alertes, une meilleure organisation des menus et du design afin de rendre l'application plus attractive et fonctionnelle.

Enfin, la mise en place d'un environnement de déploiement, par exemple via Google App Engine avec une orchestration gérée par Kubernetes, offrirait un meilleur contrôle des performances et de la scalabilité du projet. L'amélioration de l'interface utilisateur devrait également être prise en compte pour une expérience plus fluide et ergonomique.