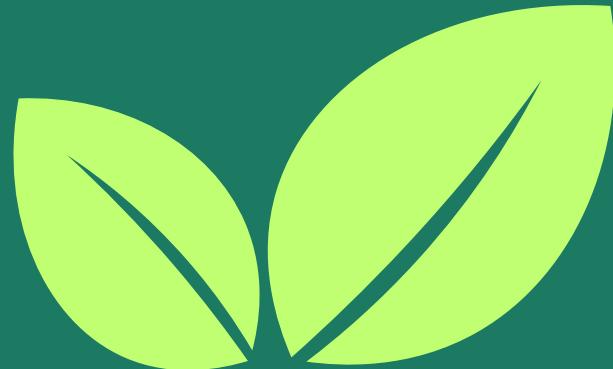


Sous la supervision de :

Robert Tomczak



Rapport de projet

Application de Reconnaissance d'Images pour le Tri des Déchets

2024 -- 2025

ADIDO Juste-Medis  KUGATHAS Jeathusan

SOMMAIRE

INTRODUCTION

Page de titre

Résumé

I. PREMIÈRE PARTIE

Présentation du projet

Analyse des besoins

Planification et répartition des tâches

II. PARTIE PERSONNELLE : JEATHUSAN

Introduction

Conception de l'interface utilisateur

Implémentation du Frontend

III. PARTIE PERSONNELLE : JUSTE-MEDIS

Introduction

Architecture backend

Implémentation du Backend

IV. CONCLUSION ET PERSPECTIVES

Bilan du projet

Axes d'amélioration

ANNEXES

Code source

Captures d'écrans

Résultats des tests

BIBLIOGRAPHIE ET RÉFÉRENCES

Sources et technologies utilisées



INTRODUCTION

Ce rapport présente le projet d'une application de reconnaissance d'images pour le tri des déchets. Ce projet a été réalisé par deux étudiants, ADIDO Juste-Medis et KUGATHAS Jeathusan, sous la supervision de Robert Tomczak. L'objectif principal est de développer une solution technologique innovante pour améliorer la gestion des déchets en facilitant leur tri grâce à la reconnaissance d'images.

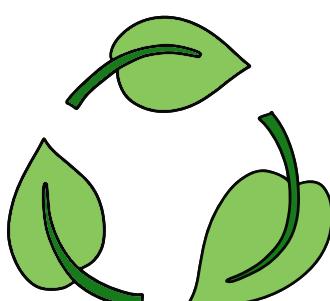
Page de titre

- **Titre du projet :** App Reconnaissance d'Images pour le Tri des Déchets
- **Date de soumission :** 24/02/2025
- **Nom du cours et de l'enseignant :** Technologie du web

Résumé

Ce projet vise à créer un système de classification d'images capable d'identifier différents types de déchets (plastique, métal, verre, papier, organique) en temps réel. L'application permet aux utilisateurs de télécharger des images de déchets et d'obtenir instantanément des classifications précises, accompagnées d'un pourcentage de confiance.

L'objectif est d'éduquer et de sensibiliser les utilisateurs aux enjeux du tri et du recyclage, tout en rendant le processus plus accessible et intuitif.



I. Première partie : Partie commune

1. Présentation du contexte

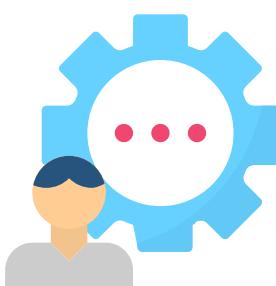
La gestion des déchets est devenue une priorité mondiale en raison de l'augmentation alarmante de leur production. Environ un tiers des déchets solides générés dans le monde ne sont pas correctement traités, ce qui contribue à la pollution, à la dégradation des écosystèmes et aux émissions de gaz à effet de serre. Le recyclage apparaît comme une solution efficace pour réduire la pression sur nos ressources naturelles, préserver l'énergie et atténuer les impacts environnementaux.

Cependant, une grande partie de la population manque d'informations sur les bonnes pratiques de tri, ce qui limite l'efficacité des systèmes de recyclage.

2. Description générale du projet

Notre application offrira aux utilisateurs, qu'ils soient des particuliers ou des entreprises, la possibilité de télécharger des images de déchets et d'obtenir instantanément des classifications précises.

L'objectif principal de ce projet est d'éduquer et de sensibiliser les utilisateurs aux enjeux du tri et du recyclage, tout en rendant le processus plus accessible et intuitif. En intégrant des fonctionnalités interactives et des recommandations personnalisées, nous visons à encourager une participation active des citoyens dans la gestion des déchets, contribuant ainsi à la transition vers une société plus durable.



3. Objectifs et finalités du projet

Offrir une solution automatisée de classification des déchets :

Développer un système capable de reconnaître et de classer automatiquement différents types de déchets à partir d'images.

Sensibiliser les utilisateurs aux bonnes pratiques de tri :

Sensibiliser les utilisateurs à l'importance du tri des déchets et les inciter à adopter des gestes écoresponsables.

Contribuer à une meilleure gestion des déchets dans les villes et sociétés de collecte :

Faciliter le processus de tri pour les particuliers et les entreprises, améliorant ainsi la gestion des déchets à grande échelle.

4. Importance et justification du projet

La mauvaise gestion des déchets entraîne des problèmes environnementaux majeurs, tels que la pollution et la dégradation des écosystèmes. En facilitant le tri des déchets à la source, ce projet vise à réduire ces impacts négatifs et à promouvoir des pratiques durables. En offrant une solution technologique accessible, nous espérons encourager une participation active des citoyens dans la gestion des déchets, contribuant ainsi à un environnement plus propre et plus sain.



5. Définition de ce qui est inclus et exclu du projet

Inclus :

Reconnaissance et classification automatique des déchets : Développement d'un algorithme capable d'identifier et de classer les déchets à partir d'images.

Interface utilisateur intuitive : Créer une interface simple où les utilisateurs peuvent télécharger des images et recevoir une classification rapide.

Traitement recommandé : Recommandations sur le traitement approprié des déchets identifiés.

Exclus :

Disponibilité et qualité des images pour l'entraînement du modèle : La qualité et la disponibilité des images utilisées pour entraîner le modèle ne sont pas garanties.

Performance des algorithmes de classification en environnement réel : Les performances peuvent varier en fonction des conditions réelles d'utilisation.

Accès à une connexion Internet stable pour l'utilisation de l'application : L'application nécessite une connexion Internet stable pour fonctionner correctement, ce qui peut ne pas être disponible dans toutes les situations.

Limites et contraintes du projet

Disponibilité et qualité des images pour l'entraînement du modèle : La performance du modèle de classification dépend fortement de la qualité et de la diversité des images utilisées pour son entraînement. Une disponibilité limitée ou une mauvaise qualité des images peut affecter la précision et la fiabilité du système.

Performance des algorithmes de classification en environnement réel : Les algorithmes peuvent rencontrer des défis en conditions réelles, notamment en raison de variations dans l'éclairage, la position des objets, ou la présence de déchets similaires. Ces facteurs peuvent influencer la précision des classifications.

Contraintes matérielles et coûts d'hébergement : Le projet nécessite des ressources matérielles pour le traitement des données et l'hébergement du système. Les coûts associés à l'infrastructure, tels que les serveurs et le stockage, peuvent limiter la scalabilité et l'accessibilité de l'application. De plus, les contraintes matérielles peuvent affecter les performances, notamment sur des appareils moins puissants.



6. Analyse des besoins exigences fonctionnelles

Description détaillée des besoins du logiciel

L'application doit offrir aux utilisateurs la possibilité de capturer ou de télécharger des images de déchets pour obtenir une classification instantanée, accompagnée d'un pourcentage de confiance. Elle doit être accessible via un navigateur web et une application mobile, garantissant ainsi une large accessibilité et une utilisation flexible.

7. Rédaction du cahier des charges fonctionnelles

Téléchargement d'images :

- Permettre aux utilisateurs de capturer des photos de déchets directement depuis l'application mobile ou de télécharger des images depuis leur appareil.
- Assurer la compatibilité avec divers formats d'image pour une utilisation optimale.

Classification automatique des déchets :

- Analyser les images téléchargées à l'aide d'un algorithme de reconnaissance d'images pour identifier le type de déchet (plastique, métal, verre, papier, organique, etc.).
- Fournir un pourcentage de confiance pour chaque classification afin d'indiquer la fiabilité du résultat.

Interface utilisateur intuitive et accessible :

- Concevoir une interface conviviale et facile à naviguer, adaptée à différents types d'utilisateurs.
- Offrir des fonctionnalités claires et bien organisées pour le téléchargement, la classification et la consultation des résultats.



8. Spécifications fonctionnelles

Interactions utilisateur :

- Un utilisateur peut importer une image de déchet via l'application.
- L'application analyse l'image et fournit une classification du type de déchet avec un pourcentage de confiance.
- Les résultats sont affichés de manière claire et compréhensible pour l'utilisateur.

Scénarios d'utilisation :

Particulier : Un utilisateur prend en photo un déchet avec son smartphone. L'application analyse l'image et propose une suggestion de tri (par exemple, bac de recyclage approprié).

Société de recyclage : Une entreprise de recyclage utilise l'application pour scanner et trier automatiquement les déchets reçus, optimisant ainsi leur processus de tri et améliorant l'efficacité opérationnelle.

9. Spécifications non fonctionnelles

Performance :

Afin de garantir une expérience utilisateur fluide et réactive, le système doit traiter chaque image en moins de 3 secondes. Cela implique l'optimisation du prétraitement des images, l'utilisation d'algorithmes efficaces de deep learning, ainsi que des méthodes de gestion des ressources pour minimiser les latences. L'architecture du système devra être capable de gérer une montée en charge importante, avec des temps de réponse constants, même en période de forte utilisation.

Sécurité :

La sécurité du système doit se reposer sur plusieurs couches de protection : les données des utilisateurs doivent être chiffrées et protégées, tandis que les images téléchargées seront anonymisées et débarrassées de toute donnée personnelle ou métadonnée. Le stockage des informations sensibles sera sécurisé, et des mesures de validation des entrées seront mises en place pour prévenir les attaques malveillantes. Enfin, une surveillance continue et des tests de pénétration assurent la détection et la correction rapide des vulnérabilités.

Ergonomie et accessibilité :

Interface adaptée : Conception d'une interface utilisateur qui est accessible aux personnes en situation de handicap, en respectant les normes d'accessibilité.

Normes et réglementations :

Respect des réglementations environnementales : Assurer que l'application est conforme aux lois et régulations environnementales en vigueur.

Conformité au RGPD : Garantir que l'application respecte le Règlement Général sur la Protection des Données (RGPD) pour protéger les données personnelles des utilisateurs.

10. Contraintes techniques :

Plateformes :

L'application sera développée pour être accessible sur le web et sur mobile, offrant ainsi une flexibilité d'utilisation selon les préférences des utilisateurs.

Technologies utilisées :

Backend : Python avec le framework Flask pour créer une API RESTful qui gère les requêtes et les données.

Frontend : TypeScript avec React Native et Expo pour développer une interface utilisateur réactive et multiplateforme.

Modèle de classification : Utilisation de TensorFlow et OpenCV pour le traitement et la classification des images, assurant une reconnaissance précise des déchets.

Compatibilité :

L'application sera conçue pour être compatible avec les navigateurs web récents, garantissant une expérience utilisateur optimale sur différents appareils.

Elle sera également compatible avec les systèmes d'exploitation Android et iOS, permettant une utilisation fluide sur smartphones et tablettes.

Phases, planification et répartition des tâches

Découpage du projet en différentes phases ou étapes

Analyse des besoins

Objectif : Comprendre les attentes et les exigences du projet, tant sur le plan fonctionnel que technique.

Actions : Étudier les utilisateurs cibles, définir les fonctionnalités essentielles, et rédiger un cahier des charges détaillé.

Livrable : Document de spécifications fonctionnelles et techniques.

Conception

Objectif : Structurer le projet en définissant l'architecture côté front-end et back-end.

Actions : Choisir les technologies adaptées pour chaque composant du système et élaborer un plan de développement.

Livrable : Diagrammes d'architecture et plan de développement.

Développement Front-end

Objectif : Créer une interface utilisateur intuitive et réactive.

Actions : Utiliser React Native et Expo pour développer l'interface, en mettant l'accent sur une navigation fluide et une optimisation de l'expérience utilisateur.

Livrable : Prototype fonctionnel de l'interface utilisateur

Développement Back-end

Objectif : Mettre en place le serveur et intégrer le modèle d'intelligence artificielle.

Actions : Développer l'API avec Flask, gérer les requêtes et les données, et intégrer le modèle de classification basé sur TensorFlow et OpenCV.

Livrable : Serveur opérationnel avec API fonctionnelle.

Tests et validation

Objectif : Assurer la stabilité et le bon fonctionnement de l'application.

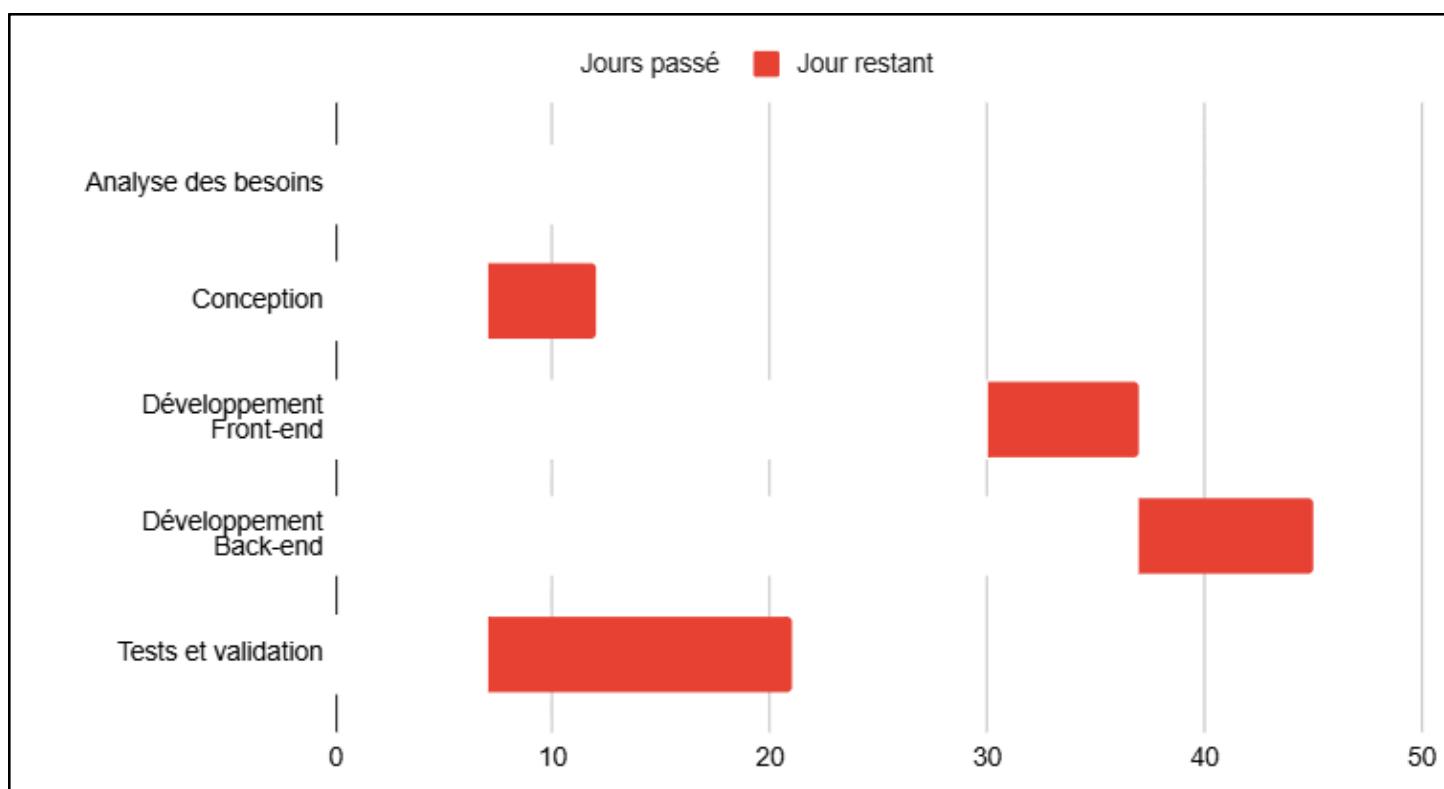
Actions : Réaliser des tests unitaires et d'intégration, corriger les bugs identifiés, et valider les performances du système.

Livrable : Rapport de tests et validation de l'application.

11. Estimation de la durée de chaque phase

Phase	Estimation de la durée
Analyse des besoins	2 semaines
Conception	1 semaines
Développement Front-end	4 semaines
Développement Back-end	4 semaines
Tests et validation	3 semaines

- **Diagramme de gantt**



Répartition des tâches

ADIDO Juste-Medis (Responsable Backend)

Conception et développement du back-end.

Implémentation du modèle de classification des déchets.

Tests de performance et d'intégration.

KUGATHAS Jeathusan (Responsable Frontend)

Conception et développement de l'interface utilisateur.

Intégration du back-end avec le front-end.

Conception du UX/UI.

Cette organisation permet de tirer parti des compétences spécifiques de chaque membre de l'équipe, assurant ainsi une réalisation efficace et cohérente du projet.



II . Partie personnelle : une partie par étudiant

Introduction

KUGATHAS JEATHUSAN (Frontend)

Dans cette section, je présente les principales tâches à accomplir pour la conception et le développement de l'interface utilisateur de l'application. L'objectif est d'assurer une expérience utilisateur fluide et intuitive, en mettant l'accent sur l'accessibilité et l'ergonomie.

1.Rappel des tâches à réaliser

Analyser les besoins des utilisateurs en matière d'ergonomie et d'accessibilité : Comprendre les attentes et les besoins des utilisateurs pour concevoir une interface qui soit à la fois facile à utiliser et accessible à tous.

Concevoir une interface utilisateur adaptée aux différentes plateformes (web et mobile) : Créer une interface qui fonctionne de manière optimale sur les navigateurs web ainsi que sur les appareils mobiles, en assurant une cohérence visuelle et fonctionnelle.

Implémenter les interactions et garantir une navigation intuitive : Développer des fonctionnalités interactives qui permettent aux utilisateurs de naviguer facilement dans l'application, tout en assurant une expérience utilisateur agréable.



2. Positionnement

Le développement du frontend repose sur les principes d'accessibilité afin de maximiser l'adoption de l'application par un large public. L'interface devra être moderne, minimaliste et fonctionnelle, offrant ainsi une expérience utilisateur optimale. En mettant l'accent sur l'accessibilité, nous visons à rendre l'application utilisable par tous, y compris les personnes ayant des besoins spécifiques.

Fin de l'Analyse

1. Identification des Scénarios et Cas d'Utilisation :

L'application de reconnaissance et de tri des déchets propose plusieurs scénarios d'utilisation courants :

Utilisateur particulier :

- Un utilisateur prend en photo un déchet via l'application mobile.
- L'application analyse l'image et fournit une classification du déchet avec un pourcentage de confiance.
- Une recommandation de tri est affichée, indiquant le bac de recyclage approprié.

Entreprise de recyclage ou municipalité :

- Un employé utilise l'application pour scanner des déchets en grande quantité.
- L'application fournit une classification automatique pour optimiser le tri.
- Un historique des classifications est accessible pour le suivi des déchets traités.

Ces scénarios permettent de mieux comprendre les interactions entre les utilisateurs et l'application, et aident à affiner les exigences fonctionnelles et techniques du projet.

2. Scénario : Une entreprise de recyclage optimise son tri (Cas critique)

Contexte : Une entreprise de recyclage reçoit une grande quantité de déchets quotidiennement et souhaite améliorer son efficacité en automatisant le processus de tri. L'objectif est de réduire les erreurs humaines et d'accélérer le traitement des déchets pour une gestion plus efficace et durable.

Acteurs :

Employé de l'entreprise de recyclage
Utilisateurs potentiels
Universités et écoles

3. Scénario principal :

Préparation :

- L'entreprise installe des caméras connectées à l'application sur les lignes de tri des déchets.
- Les caméras sont configurées pour capturer des images en continu des déchets passant sur le convoyeur.

Analyse en temps réel :

- Les images capturées par les caméras sont envoyées en temps réel au backend de l'application via une connexion sécurisée.
- L'algorithme de classification, basé sur TensorFlow et OpenCV, analyse chaque image pour identifier le type de déchet (plastique, métal, verre, papier, organique, etc.).

Classification et enregistrement :

- Chaque déchet est classifié automatiquement par l'algorithme, et les résultats sont enregistrés dans la base de données de l'application.
- Les données incluent le type de déchet, le pourcentage de confiance de la classification, et l'heure de l'analyse.

Tri automatique :

- Les déchets sont automatiquement dirigés vers le circuit de traitement approprié en fonction de leur classification.
- Par exemple, les déchets en plastique sont envoyés vers une zone de recyclage spécifique, tandis que les déchets organiques sont dirigés vers une unité de compostage.

Optimisation des processus :

- L'entreprise accède aux statistiques et rapports générés par l'application pour analyser les tendances et optimiser ses processus de tri.
- Les rapports incluent des informations sur les volumes de chaque type de déchet traité, les taux de précision de la classification, et les périodes de pointe.

Amélioration continue :

Les données collectées sont utilisées pour affiner l'algorithme de classification, améliorant ainsi sa précision et son efficacité.

4. Diagrammes de séquence pour illustrer ces scénarios

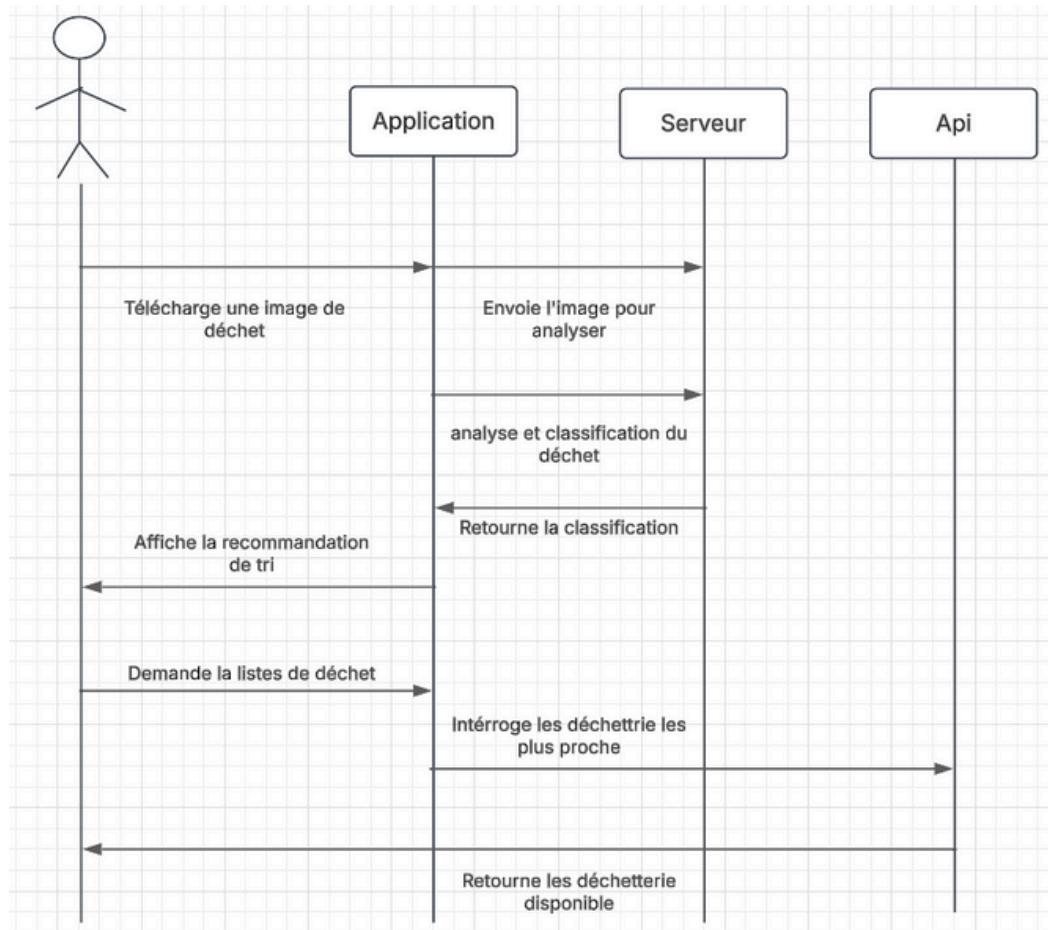


Figure: Diagramme de séquence associé aux scénarios

5. Recherche de solutions

Lors du développement de l'application, il est essentiel d'explorer différentes options technologiques pour choisir celles qui répondent le mieux aux besoins du projet en termes de performance, compatibilité, et simplicité d'implémentation. Voici une comparaison des solutions disponibles et les raisons qui ont motivé mon choix :

Choix du Front-end

Technologie retenue : **React Native + Expo + TypeScript**

Pourquoi ?

Développement multiplateforme :

React Native permet de créer une application mobile qui fonctionne sur plusieurs plateformes (Web, Android, iOS) à partir d'une seule base de code. Cela réduit le temps et les coûts de développement tout en assurant une cohérence entre les différentes versions de l'application.

Gestion simplifiée des dépendances avec **Expo** :

Expo est un framework et une plateforme qui facilite le développement d'applications React Native. Il simplifie la gestion des dépendances et offre des outils intégrés pour le déploiement et le test, ce qui accélère le processus de développement.

Robustesse du code avec **TypeScript** :

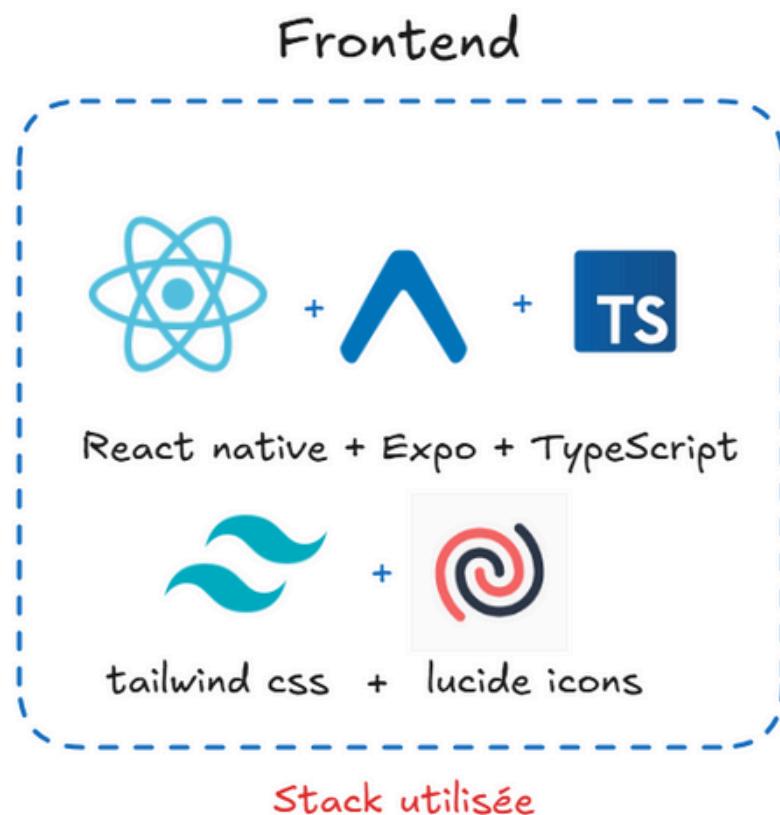
TypeScript apporte un typage statique à JavaScript, ce qui permet de détecter les erreurs à la compilation plutôt qu'à l'exécution. Cela améliore la robustesse du code et réduit les bugs potentiels, tout en facilitant la maintenance et l'évolution du projet.

Performance et expérience utilisateur :

L'utilisation de ces technologies permet d'assurer une performance optimale de l'application, tout en offrant une expérience utilisateur fluide et réactive. La compatibilité multiplateforme garantit que l'application fonctionne de manière optimale sur différents appareils et systèmes d'exploitation.

En conclusion, le choix de React Native, Expo, et TypeScript pour le développement du front-end de l'application repose sur leur capacité à offrir un développement rapide et efficace, une gestion simplifiée des dépendances, et une robustesse accrue du code. Ces choix technologiques permettent de répondre aux exigences du projet tout en assurant une expérience utilisateur de haute qualité.

6 . Architecture générale du logiciel



Difficultés rencontrées et solutions apportées

1. Difficultés rencontrées

Limites des connaissances :

Manque d'expertise sur certains concepts avancés : Le projet demandait une bonne compréhension de la reconnaissance des déchets et de l'optimisation du tri, des sujets complexes et spécialisés.

Difficultés à trouver des sources fiables et récentes : Il est difficile de trouver des études récentes sur la reconnaissance d'images et le tri des déchets, ce qui complique l'apprentissage.

Langage et compréhension :

Problèmes de compréhension liés aux termes techniques : La terminologie technique liée à la reconnaissance d'images et à l'intelligence artificielle est parfois complexe, rendant la compréhension plus difficile.

Difficultés à structurer et organiser les informations : La quantité d'informations à traiter pour le projet était importante, nécessitant une organisation claire pour en faciliter la compréhension et l'implémentation.

2. Solutions apportées

Amélioration des connaissances :

Recherche approfondie : Réaliser des recherches détaillées en consultant des articles scientifiques, des études de cas et des forums spécialisés pour approfondir la compréhension des concepts avancés

Optimisation de la communication :

Utilisation de schémas UML : Créer des diagrammes UML pour visualiser les processus et les interactions au sein du système, facilitant ainsi la compréhension et la communication des idées.

Simplification du langage technique : Adopter un langage plus simple et des explications claires pour faciliter la compréhension des concepts techniques par tous les membres de l'équipe.

Ces solutions ont permis de surmonter les difficultés rencontrées, en améliorant la compréhension des concepts techniques et en structurant efficacement les informations pour une mise en œuvre réussie du projet.

Conception :

Présentation de l'interface:

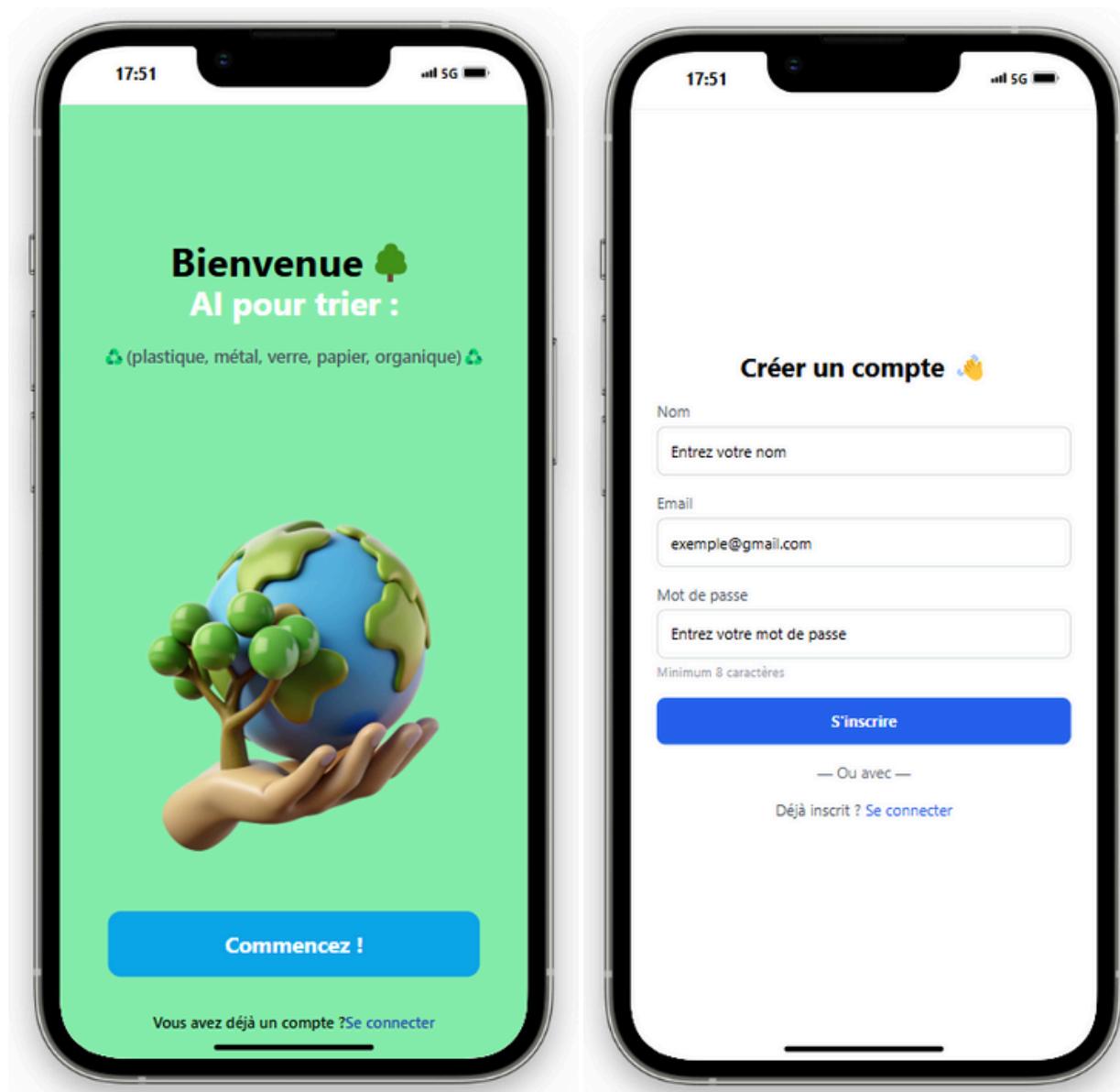


Figure: Capture d'écran 1

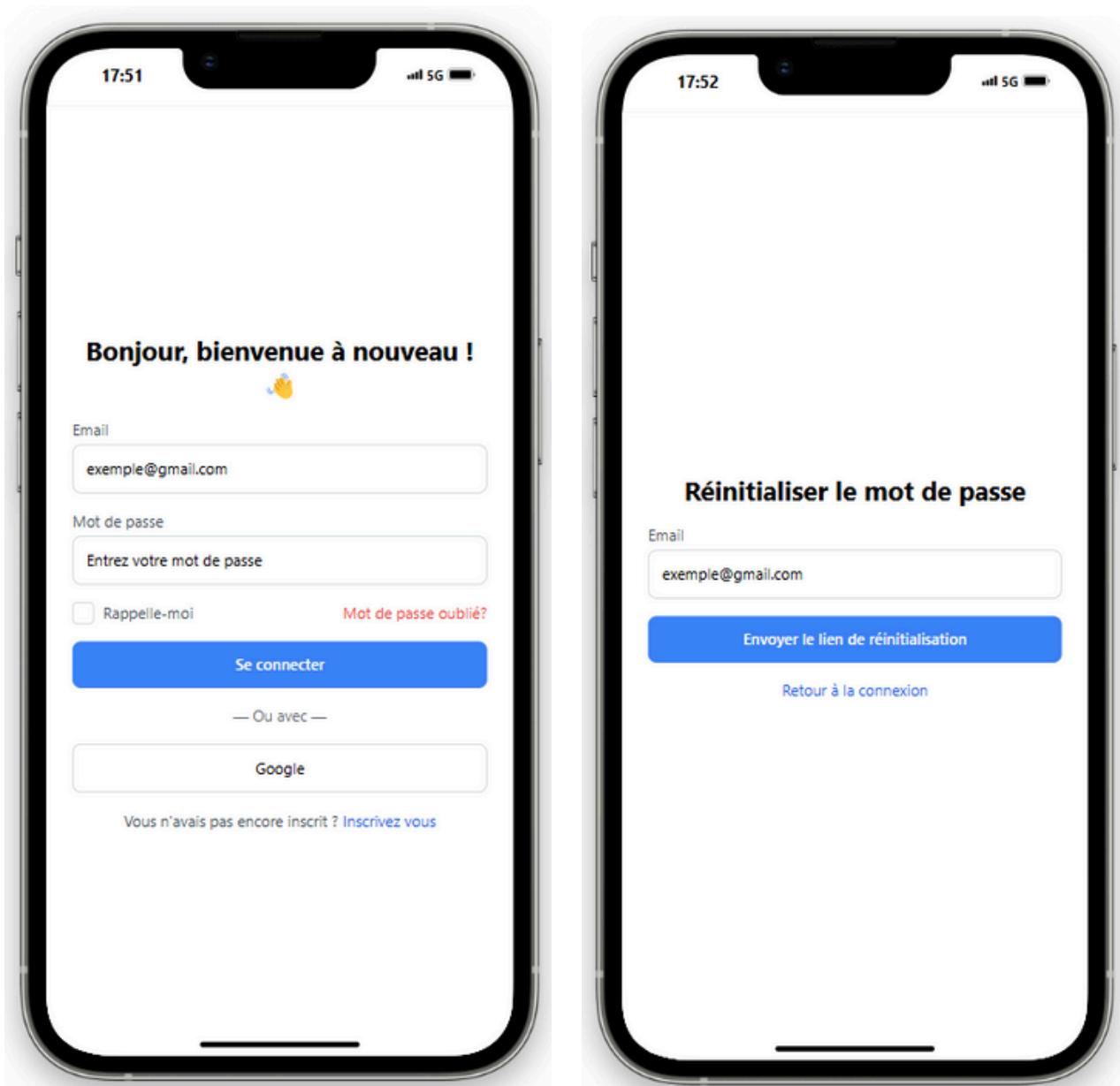
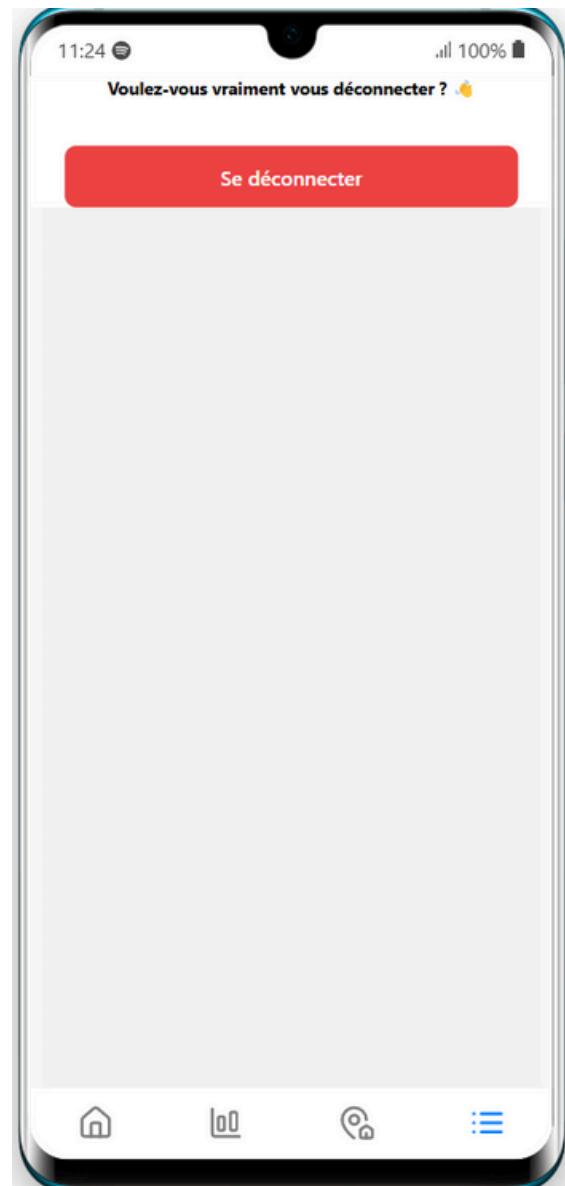
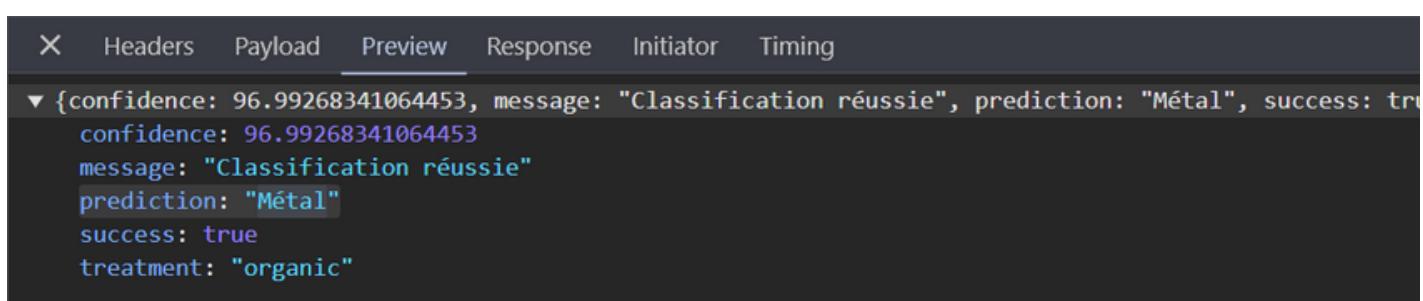
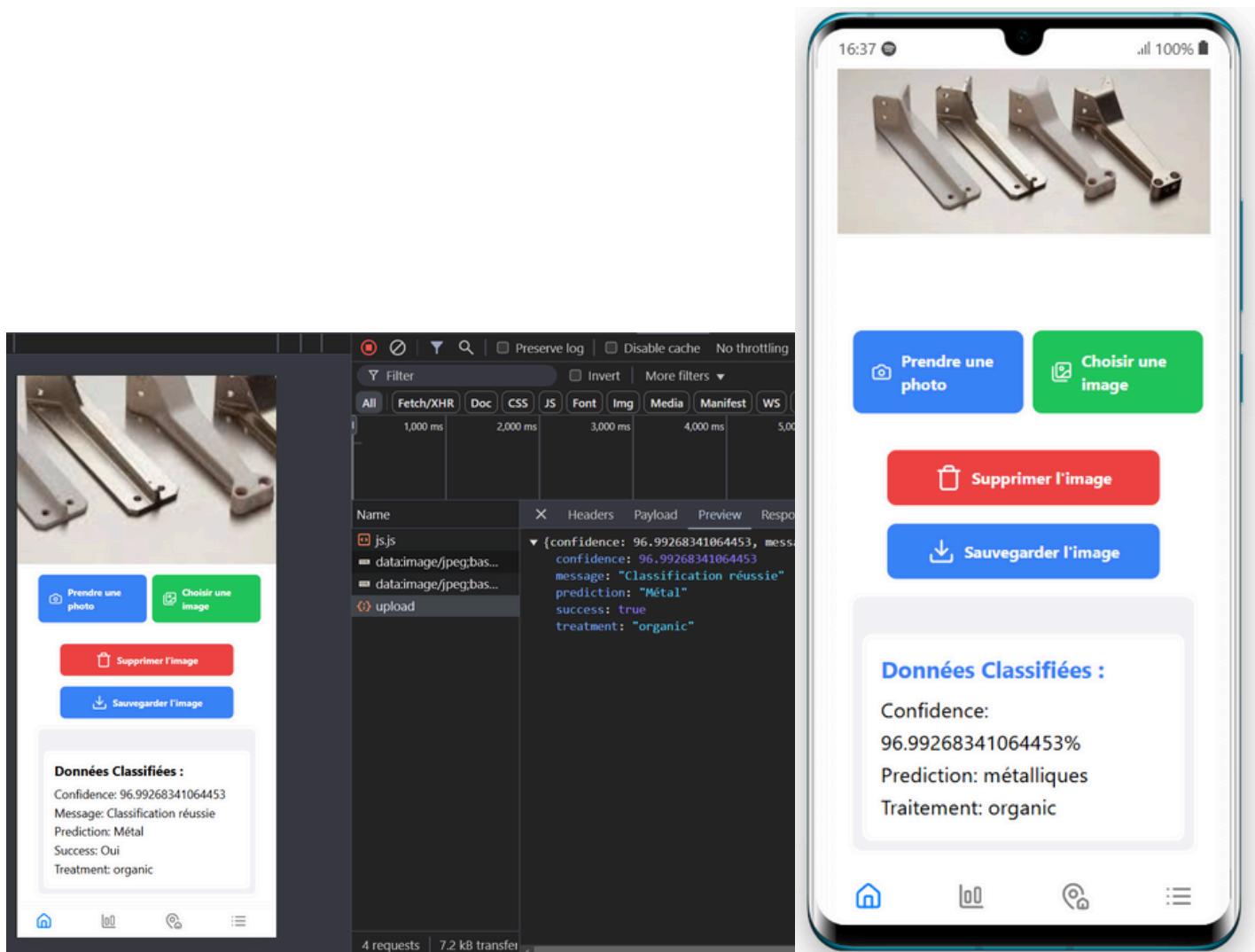


Figure: Capture d'écran 2



- L'utilisateur choisit l'option d'inscription.
- L'application affiche le formulaire d'inscription.
- L'utilisateur remplit le formulaire avec ses informations.
- L'utilisateur choisit l'option de connexion.
- L'application affiche le formulaire de connexion.
- L'utilisateur saisit ses identifiants.
- L'utilisateur choisit pour déconnecté

Figure: Capture d'écran 3



- L'utilisateur peut insérer une image, qui est ensuite envoyée au backend pour être classifiée à l'aide de l'intelligence artificielle. Le système renvoie ensuite un fichier JSON contenant le pourcentage de confiance et la prédiction du type de matériau (plastique, métal, papier, etc.).

Figure: Capture d'écran 4



Prendre une photo

Choisir une image

Supprimer l'image

Sauvegarder l'image

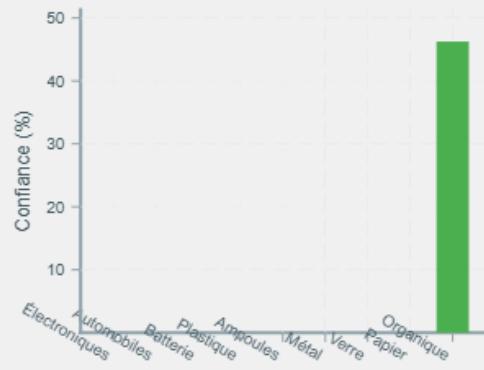
Données Classifiées :

Confidence: 46.25029265880585%

Prediction: organiques

Traitement: organic

Résultats de Classification



Électroniques : 0.00%

Automobiles : 0.00%

Batterie : 0.00%

Plastique : 0.00%

Ampoules : 0.00%

Métal : 0.00%

Verre : 0.00%

Papier : 0.00%

Organique : 46.25%



- Ici, l'utilisateur peut consulter, à l'aide d'un graphique, le pourcentage de confiance concernant le type de déchet, comme on peut le voir dans l'image. L'image de la carotte affiche une précision d'environ 46.25%. Comme illustré dans le graphique, nous avons une représentation visuelle de ce pourcentage.

Figure: Capture d'écran 5

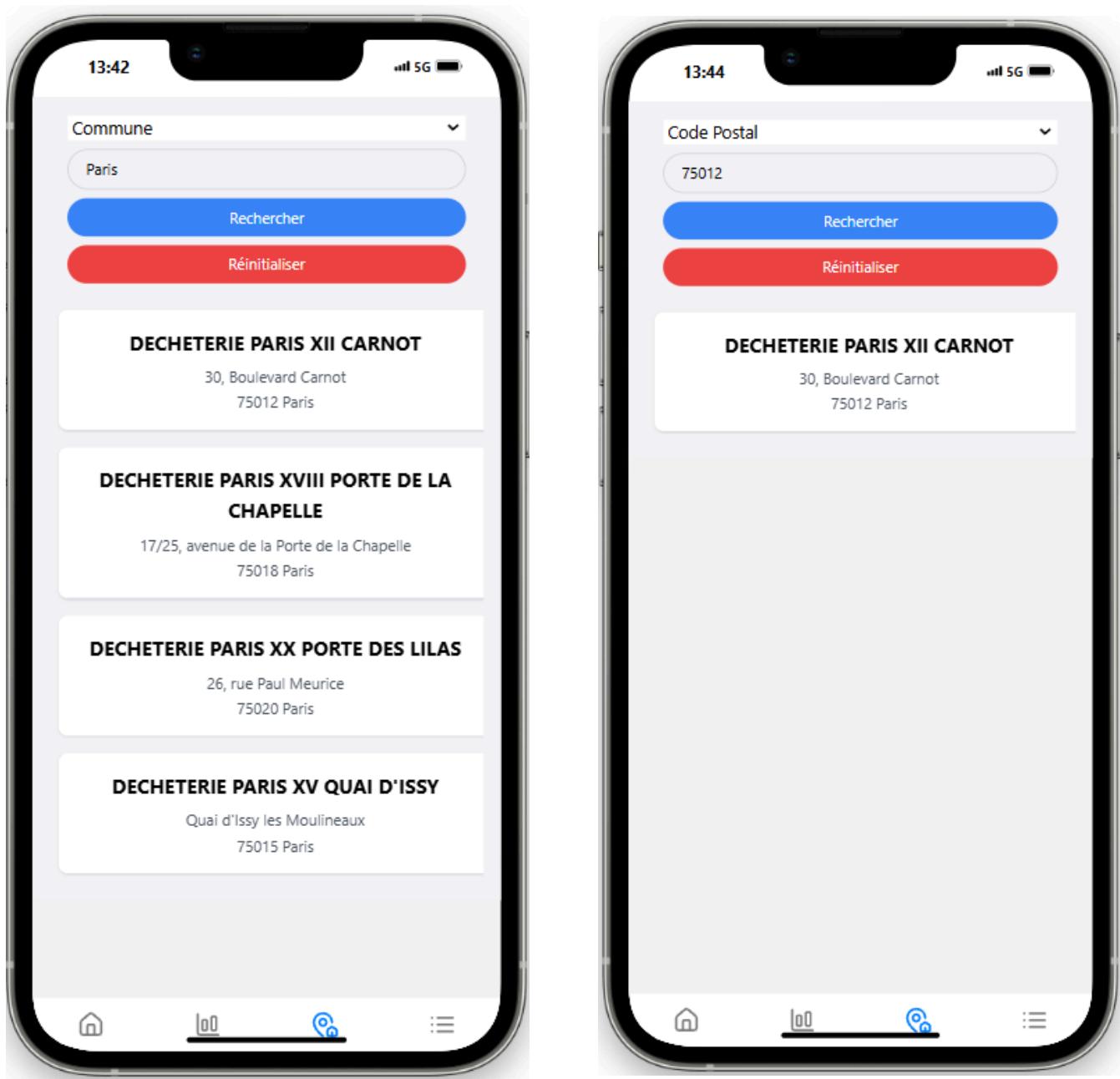
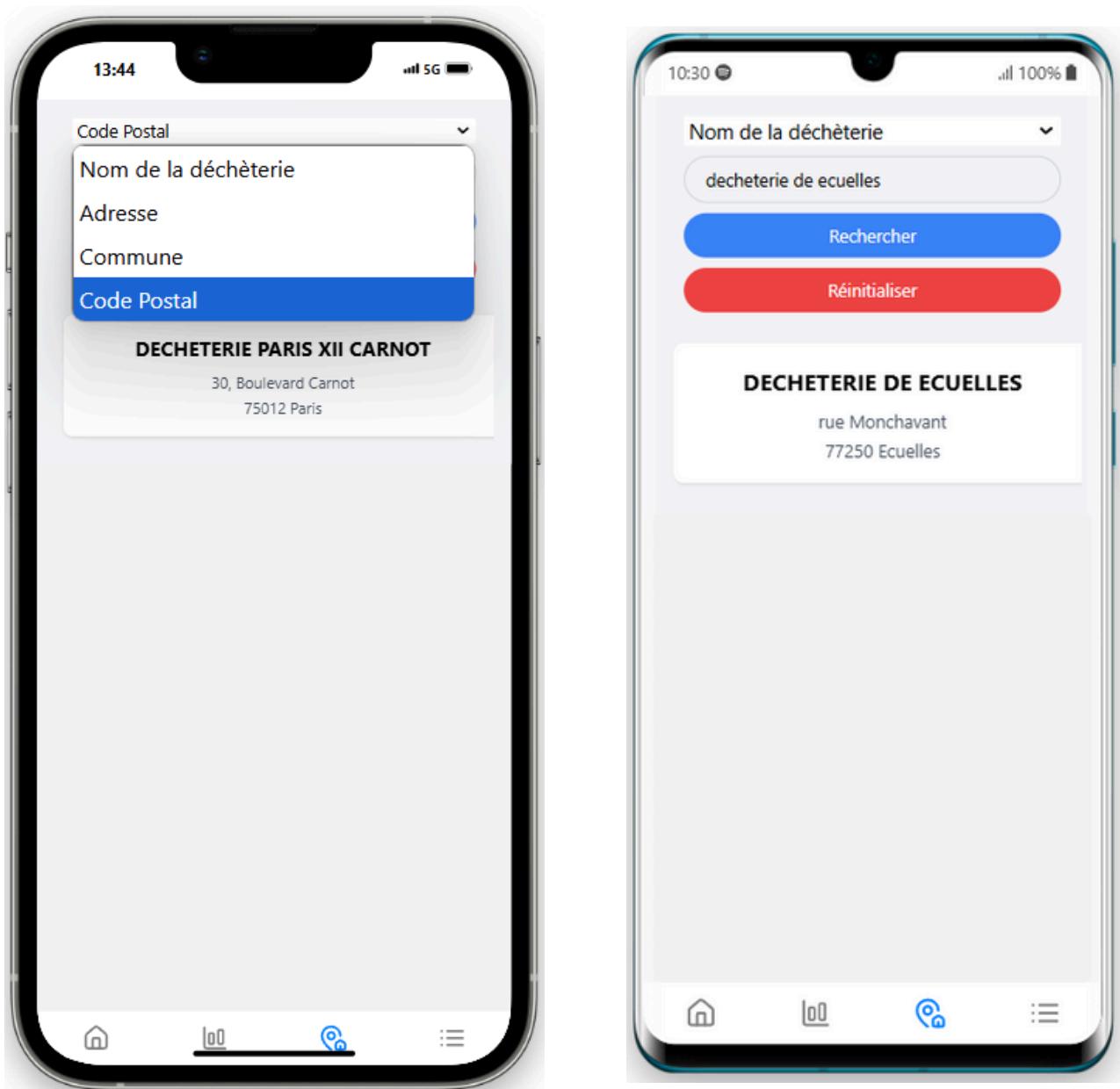


Figure: Capture d'écran 6



- L'utilisateur peut rechercher les déchetteries les plus proches. Les résultats sont obtenus via une API et peuvent être filtrés par nom, adresse, commune ou code postal.

Figure: Capture d'écran 7

3. Justification des choix de design d'interface et de l'expérience utilisateur

L'objectif principal du design de l'interface est d'offrir une expérience fluide, intuitive et accessible à tous les utilisateurs. Pour atteindre cet objectif, plusieurs principes clés ont guidé les choix de conception :

4. Simplicité et clarté

Interface épurée : Une interface avec des éléments espacés aide à réduire la surcharge cognitive, permettant aux utilisateurs de se concentrer sur les tâches importantes sans être distraits par une trop grande quantité d'informations.

Langage clair et concis : Utiliser un langage clair et simple pour guider l'utilisateur de manière précise. Cela rend l'application facile à comprendre et à utiliser, même pour ceux qui ne sont pas habitués à la technologie.

Icônes et visuels explicites : Les icônes et visuels sont conçus de manière intuitive et claire, ce qui aide les utilisateurs à comprendre rapidement les fonctionnalités de l'application. Cela leur permet de naviguer facilement et de trouver rapidement ce qu'ils recherchent.

5. Fluidité et rapidité

Accès rapide aux fonctionnalités essentielles : Le design est pensé pour offrir un accès rapide aux fonctionnalités principales, comme l'ajout d'images et la recherche de déchetteries. Cela réduit le temps nécessaire pour accomplir les tâches courantes.

Structure bien pensée : Une structure claire et bien organisée simplifie les étapes nécessaires pour accomplir une tâche, rendant l'application plus efficace et agréable à utiliser.

6. Feedback utilisateur

Messages de confirmation et d'erreur clairs : Fournir des messages simples et précis pour valider les actions de l'utilisateur ou indiquer des erreurs. Cela permet aux utilisateurs de comprendre ce qui se passe et de corriger rapidement les problèmes.

Retour visuel immédiat : Offrir un retour visuel immédiat sur les interactions, comme des boutons changeant d'état au clic ou des indicateurs de chargement. Cela assure aux utilisateurs que leurs actions sont prises en compte et traitées.

7. Expérience utilisateur : l'accessibilité et l'ergonomie

Accessibilité

Compatibilité avec les lecteurs d'écran : L'application fonctionne avec tous les lecteurs d'écran.

Contraste élevé et tailles de police adaptées : Un contraste élevé entre le texte et l'arrière-plan, ainsi que des tailles de police ajustables, sont utilisés pour assurer une bonne lisibilité.

Navigation optimisée : La navigation est facile pour les utilisateurs mobiles et ceux avec des difficultés physiques, avec des éléments de navigation clairs et accessibles.



Conception détaillée

1. Architecture Logicielle Détailée :

L'architecture adoptée pour ce projet suit le modèle MVC (Modèle-Vue-Contrôleur) combiné à une approche microservices. Cette combinaison vise à garantir la scalabilité et la modularité du système, permettant ainsi une gestion efficace des différentes composantes de l'application.

2. Modèle-Vue-Contrôleur (MVC)

Frontend (Vue) :

- **Technologies utilisées :** React Native avec Expo et TypeScript.
- **Description :** L'interface utilisateur est développée en utilisant React Native, qui permet de créer une application mobile multiplateforme (Web, Android, iOS). Expo facilite la gestion des dépendances et simplifie le déploiement, tandis que TypeScript améliore la robustesse du code grâce au typage statique.

Backend (Contrôleur) :

- **Technologies utilisées :** API REST en Flask (Python).
- **Description :** Le backend est conçu pour gérer les requêtes et les données via une API RESTful. Flask est choisi pour sa légèreté et sa flexibilité, permettant de créer rapidement une API performante et évolutive.

Microservices :

- **Technologies utilisées :** OpenCV et TensorFlow.
- **Description :** Un microservice dédié est utilisé pour le système de classification des images. OpenCV est employé pour le traitement d'image, tandis que TensorFlow est utilisé pour le modèle de reconnaissance d'images. Cette approche permet de séparer les responsabilités et d'assurer une scalabilité optimale.

Implémentation : Détails techniques de la réalisation

Le développement du frontend de l'application repose sur des technologies modernes qui garantissent une interface fluide et réactive, offrant ainsi une expérience utilisateur optimale. Voici les principales technologies utilisées et leurs rôles dans le projet :

1. Framework : React Native

Description : React Native est utilisé pour créer une interface dynamique et modulaire permet de développer des applications mobiles multiplateformes (Web, Android, iOS) à partir d'une seule base de code.

Avantages :

- **Réutilisation du code** : Le code peut être partagé entre différentes parties grâce aux composants, ce qui réduit le temps et les efforts de développement.
- **Performance** : Offre des performances proches du natif grâce à l'utilisation de composants natifs.
- **Communauté active** : Une large communauté de développeurs et une riche collection de bibliothèques et d'outils disponibles.

2. Styling : Tailwind CSS

Description : Tailwind CSS est utilisé pour le styling de l'application, offrant un design moderne et adaptable aux différents écrans. Il s'agit d'un framework CSS "utility-first" qui permet de créer des interfaces utilisateur réactives.

Avantages :

- **Rapidité de développement** : Permet de styliser rapidement les composants grâce à des classes utilitaires prédéfinies.
- **Responsive design** : Facilite la création de designs adaptatifs qui s'ajustent automatiquement à différentes tailles d'écran.
- **Cohérence visuelle** : Assure une apparence uniforme et moderne à travers toute l'application.

3. API Communication : Axios

Description : Axios est utilisé pour interagir avec le backend et récupérer les résultats d'analyse. Il s'agit d'une bibliothèque JavaScript qui permet de faire des requêtes HTTP de manière simple et efficace.

Avantages :

- **Simplicité** : Facilite les requêtes HTTP avec une syntaxe claire et concise.
- **Support des promesses** : Gère les promesses pour les opérations asynchrones, ce qui simplifie la gestion des requêtes réseau.
- **Compatibilité** : Fonctionne aussi bien dans les navigateurs que dans les environnements Node.js.

4. Gestion des images : TensorFlow.js et OpenCV

Description : L'intégration de TensorFlow.js et OpenCV permet l'analyse en temps réel des images et l'affichage des résultats. Ces technologies sont utilisées pour le traitement et la classification des images directement dans le navigateur ou sur l'appareil mobile.

Avantages :

- **Performance** : Permet une analyse rapide et efficace des images grâce à des algorithmes de machine learning avancés.
- **Flexibilité** : Offre une grande flexibilité pour le traitement d'image, y compris le redimensionnement, la normalisation et la classification.
- **Accessibilité** : Permet d'exécuter des modèles de machine learning directement dans le navigateur, réduisant ainsi la dépendance à un backend puissant.

En utilisant ces technologies, l'application est conçue pour offrir une expérience utilisateur fluide et réactive, tout en assurant des performances optimales et une grande flexibilité dans le traitement des images.

5. Problèmes rencontrés et solutions adoptées

Optimisation des performances du rendu

Problème : Le traitement des images entraînait un ralentissement de l'interface, affectant ainsi l'expérience utilisateur.

Solution : Implémentation d'un chargement asynchrone avec Lazy Loading. Cette technique permet de charger les images uniquement lorsqu'elles sont sur le point d'apparaître à l'écran, réduisant ainsi la charge initiale et améliorant la fluidité de l'interface.

Compatibilité multi-plateforme (web et mobile)

Problème : Certains composants React Native ne s'adaptaient pas correctement sur les appareils mobiles, entraînant des problèmes d'affichage et d'interaction.

Solution : Utilisation de composants responsifs qui s'adaptent automatiquement à différentes tailles d'écran et orientations. Cela garantit une expérience utilisateur cohérente et agréable, que l'application soit utilisée sur un navigateur web ou un appareil mobile.

Gestion des erreurs API et affichage utilisateur

Problème : Les erreurs de requête API n'étaient pas bien gérées, ce qui pouvait entraîner des interruptions ou des messages d'erreur peu clairs pour l'utilisateur, affectant ainsi l'expérience globale.

Solution : Implémentation d'un système de gestion d'erreurs robuste avec des messages d'erreur clairs et explicites. En cas d'erreur, une interface de fallback est affichée pour informer l'utilisateur de la situation et lui proposer des actions alternatives, améliorant ainsi la transparence et la réactivité de l'application.

En adoptant ces solutions, l'application est devenue plus performante, compatible avec différentes plateformes, et offre une meilleure gestion des erreurs, améliorant ainsi l'expérience utilisateur globale.

Bonnes pratiques de codage et patterns de conception utilisés

Utilisation de hooks React

Hooks utilisés : useState, useEffect, useContext

- **useState** : Permet de gérer l'état local des composants de manière simple et efficace.
- **useEffect** : Utilisé pour gérer les effets secondaires dans les composants fonctionnels, comme les appels API ou les abonnements.
- **useContext** : Facilite l'accès aux valeurs de contexte, permettant de partager des données entre les composants sans avoir à passer les props manuellement à chaque niveau.

Modularisation du code

Découpage des fonctionnalités : Le code est organisé en composants réutilisables, chaque composant étant responsable d'une fonctionnalité spécifique. Cela améliore la lisibilité et la maintenabilité du code.

Avantages : Facilite la réutilisation des composants dans différentes parties de l'application et simplifie les tests unitaires.

3. TypeScript

Sécurisation du code : TypeScript apporte un typage statique à JavaScript, ce qui permet de détecter les erreurs à la compilation plutôt qu'à l'exécution.

Réduction des erreurs runtime : Le typage strict réduit les erreurs liées aux types, améliorant ainsi la robustesse et la fiabilité du code.

Tests d'intégration (collectif)

Tests d'intégration dans un contexte collectif : Les tests d'intégration sont essentiels pour s'assurer que toutes les parties de l'application fonctionnent bien ensemble. Ils montrent l'importance de la collaboration et de l'intégration continue dans l'équipe de développement. Voici comment ces tests ont été réalisés :

Collaboration avec le backend

Utilisation de Postman : Avant d'intégrer les endpoints API dans le frontend, Postman a été utilisé pour tester chaque endpoint séparément. Cela a permis de vérifier que les requêtes et réponses étaient correctes et conformes aux spécifications. Cette étape est importante pour détecter et corriger les problèmes avant qu'ils n'impactent le système.

Tests d'intégration

Simulations de parcours utilisateur : Des scénarios de test ont été créés pour simuler les parcours utilisateur typiques, validant ainsi les interactions avec l'API. Ces simulations incluent des tests pour des fonctionnalités clés telles que :

Téléchargement d'images : Vérification que les images sont correctement envoyées au backend et que les réponses sont traitées comme prévu.

Classification des déchets : Assurer que les résultats de classification sont reçus et affichés correctement dans l'interface utilisateur.

Affichage des résultats : Valider que les résultats d'analyse sont présentés de manière claire et compréhensible pour l'utilisateur.

En mettant en oeuvre ces pratiques de tests d'intégration, on a pu s'assurer que le frontend et le backend fonctionnent de manière cohérente, offrant ainsi une expérience utilisateur fluide et fiable.

```

const handleFilter = (attribute: keyof Dechetterie, searchTerm: string) => {
  if (!searchTerm) {
    setFilteredDechetteries(dechetteries);
  } else {
    const filtered = dechetteries.filter((dechetterie) =>
      String(dechetterie[attribute])
        .toLowerCase()
        .includes(searchTerm.toLowerCase())
    );
    setFilteredDechetteries(filtered);
  }
};

```

Filtrage dynamique

Problème : Le filtrage doit être dynamique et capable de s'adapter à différents attributs des déchèteries (nom, adresse, type de déchets, etc.).

Solution : Utiliser une fonction générique qui accepte un attribut et un terme de recherche, permettant ainsi de filtrer n'importe quel attribut de l'objet Dechetterie.

```

// Validation du formulaire
const validateForm = () => {
  if (!username.trim()) {
    setError("Le nom est requis");
    return false;
  }
  if (!email.trim() || !/\S+@\S+\.\S+/.test(email)) {
    setError("Email invalide");
    return false;
  }
  if (password.length < 8) {
    setError("Le mot de passe doit contenir au moins 8 caractères");
    return false;
  }
  if (password !== repeatPassword) {
    setError("Les mots de passe ne correspondent pas");
    return false;
  }
  return true;
};

```

Validation du nom d'utilisateur :

Problème : S'assurer que le nom d'utilisateur est fourni et n'est pas vide.

Solution : Utiliser `trim()` pour supprimer les espaces superflus et vérifier que la chaîne n'est pas vide.

Validation de l'email :

Problème : S'assurer que l'email est au bon format peut être complexe en raison des nombreuses variations possibles.

Solution : Utiliser une expression régulière pour vérifier que l'email contient au moins un caractère, suivi de `@`, puis d'un domaine valide.

Validation du mot de passe :

Problème : Imposer des règles de complexité pour les mots de passe (longueur minimale, caractères spéciaux, etc.) peut rendre la validation plus complexe.

Solution : Commencer par une règle simple (longueur minimale) et ajouter progressivement des règles supplémentaires si nécessaire.

```
if (Platform.OS === "web") {
  const response = await fetch(image);
  const blob = await response.blob();
  formData.append("file", new File([blob], "upload.jpg", { type: blob.type }));
} else {
  const fileInfo = await FileSystem.getInfoAsync(image);
  if (!fileInfo.exists) {
    throw new Error("Fichier introuvable !");
  }
  const mimeType = getMimeType(image) || "image/jpeg";
  formData.append("file", {
    uri: image,
    type: mimeType,
    name: "upload.jpg",
  } as any);
}
```

Compatibilité multiplateforme :

Problème : La gestion des fichiers diffère entre les plateformes web et mobiles.

Solution : Utiliser des approches spécifiques pour chaque plateforme. Sur le web, récupérer le fichier en tant que **Blob** et le convertir en **File**. Sur mobile, utiliser l'URI du fichier avec le type MIME approprié.

Détermination du type MIME :

Problème : Déterminer le type MIME correct pour le fichier est crucial pour le téléchargement.

Solution : Utiliser une fonction **getMimeType** pour déterminer le type MIME basé sur l'extension du fichier.

```

export interface Dechetterie {
    nom_de_la_dechetterie: string;
    adresse_1: string;
    code_postal: string;
    commune: string;
    departement: string;
    dechets_non_dangereux?: string;
    dechets_dangereux?: string;
    dechets_inertes?: string;
    accueil_des_professionnels?: string;
    accueil_des_services_techniques?: string;
    zone_dediee_au_reemploi?: string;
    jours_d_ouverture_particuliers?: string;
    jours_d_ouverture_artisans?: string;
    tout_venant?: string;
    gravats?: string;
    dechets_dangereux0?: string;
    dechets_verts?: string;
    ferrailles?: string;
    bois?: string;
    pneus?: string;
    dechets_d_equipements_electriques_et_electroniques?: string;
    textiles?: string;
    cartons?: string;
    papiers_graphiques?: string;
    dechets_d_activites_de_soin_a_risques_infectieux?: string;
    amiante?: string;
    piles_et_accumulateurs_et_batteries?: string;
    huiles?: string;
    autres?: string;
    wgs84?: {
        lon: number;
        lat: number;
    };
}

```

Définition des propriétés :

Problème : Définir une interface qui couvre toutes les propriétés nécessaires pour représenter une déchèterie peut être complexe, surtout si les données proviennent de différentes sources ou peuvent évoluer.

Solution : Utiliser des propriétés optionnelles (?) pour les champs qui peuvent ne pas être disponibles dans toutes les instances de **Dechetterie**.

Gestion des données manquantes :

Problème : Certaines propriétés peuvent ne pas être présentes dans les données réelles, ce qui peut entraîner des erreurs lors de l'accès à ces propriétés.

Solution : Marquer les propriétés potentiellement manquantes comme optionnelles et gérer leur absence dans le code qui utilise cette interface.

```

// Gérer la connexion
const handleLogin = async () => [
  setError("");
  if (!validateForm()) return;

  setLoading(true);
  try {
    const response = await axios.post("https://adidome.com/visionrec/auth/login", {
      email: email.trim(),
      password,
    });

    if (response.status !== 200) {
      throw new Error(response.data.message || "Erreur de connexion");
    }

    Alert.alert("Succès", "Connexion réussie !");
    router.replace("/(tab)/HomeScreen");
  } catch (error) {
    if (axios.isAxiosError(error)) {
      setError(error.response?.data.message || "Erreur de connexion");
      Alert.alert("Erreur", error.response?.data.message || "Erreur de connexion");
    } else {
      setError("Une erreur inconnue s'est produite");
      Alert.alert("Erreur", "Une erreur inconnue s'est produite");
    }
  } finally {
    setLoading(false);
  }
];

```

Gestion des erreurs réseau :

Problème : Les erreurs réseau peuvent survenir pour diverses raisons (pas de connexion, serveur inaccessible, etc.).

Solution : Utiliser **axios.isAxiosError** pour vérifier si l'erreur provient d'une requête HTTP et afficher des messages d'erreur spécifiques basés sur la réponse du serveur.

Validation des entrées utilisateur :

Problème : La validation côté client peut être contournée. Il est essentiel de valider également les données côté serveur.

Solution : Implémenter une validation côté serveur pour s'assurer que les données reçues sont valides et sécurisées.

Discussion

Retour sur les choix effectués

React Native + Tailwind CSS :

- **Gain de temps significatif** : React Native a permis de créer une application mobile multiplateforme avec une seule base de code, ce qui réduit le temps de développement. Tailwind CSS a simplifié la création d'un design moderne et réactif, offrant une interface fluide et agréable.
- **Interface fluide et design réactif** : L'utilisation de ces technologies a permis de créer une application qui s'adapte à toutes les tailles d'écran et offre une expérience utilisateur uniforme sur toutes les plateformes.

TypeScript :

- **Amélioration de la robustesse du code** : L'intégration de TypeScript a permis d'ajouter un typage statique à JavaScript, ce qui a amélioré la robustesse du code en détectant les erreurs à la compilation plutôt qu'à l'exécution.
- **Réduction des bugs liés aux types** : Le typage strict a réduit les bugs liés aux types, rendant le code plus fiable et plus facile à maintenir.

Axios + API REST :

- **Simplicité et efficacité** : Axios a simplifié les requêtes HTTP, permettant une communication fluide et efficace avec le backend via une API RESTful. Cette approche a été choisie pour sa simplicité et son efficacité dans la gestion des interactions entre le frontend et le backend.

Limitations du logiciel actuel

Performances sur appareils bas de gamme :

Problème : Certains traitements d'image, comme ceux avec des algorithmes de machine learning, peuvent utiliser beaucoup de ressources. Cela peut ralentir les appareils moins puissants et affecter l'expérience utilisateur.

Solution potentielle : Optimiser les algorithmes de traitement d'image pour les rendre plus légers, ou offrir des options pour désactiver certaines fonctionnalités lourdes sur les appareils moins puissants.

Dépendance à Internet :

Problème : L'application a besoin d'une connexion Internet pour fonctionner, car elle utilise des requêtes API pour traiter les images et récupérer les résultats. Cela limite son utilisation sans accès au réseau.

Solution potentielle : Ajouter un mode hors ligne qui permettrait de traiter certaines images localement avec des modèles de machine learning intégrés, et de synchroniser les données quand la connexion est rétablie.

Dépendance à Internet :

Problème : La gestion du cache peut être améliorée pour réduire la consommation de bande passante et améliorer les performances de l'application.

Solution potentielle : Implémenter une meilleure stratégie de cache qui garde les résultats des analyses d'images fréquemment utilisées, réduisant ainsi le besoin de les récupérer à chaque fois depuis le serveur.

Comparaison avec d'autres solutions existantes

Critère	React Native	Kotlin	Swift	Flutter
Langage	TypeScript / JavaScript	Kotlin	Swift	Dart
Performance	Bonne, proche du natif	Excellente	Excellente	Compilé en natif
Développement cross-platform	iOS & Android avec un seul code	Android uniquement	iOS uniquement	iOS & Android avec un seul code
Temps de développement	Rapide	Plus long	Plus long	Rapide
Facilité d'apprentissage	Facile	Moyen	Moyen	Moyen
Interface utilisateur (UI)	Native avec composants bridgés	Très fluide	Très fluide	widgets personnalisés
Ecosystème & Plugins	Riche	limité	limité	Riche

II . Partie personnelle : une partie par étudiant

ADIDO Juste-Medis (Backend)

1. Introduction

Cette section vise à décrire l'architecture, la conception et les décisions techniques prises pour le développement de la plateforme backend de l'application Vision Rec. La plateforme est destinée à la reconnaissance et au tri des déchets via des images fournies par les utilisateurs, afin d'améliorer le recyclage et la gestion des déchets dans un contexte de ville intelligente.

1.1 Rappel des tâches à réaliser

- Traitement des images uploadées par les utilisateurs
- Classification des déchets (plastique, métal, verre, papier, organique)
- Gestion des historiques de classifications
- Fourniture d'API pour les échanges avec le frontend
- Mise à disposition des déchetteries à proximité via des API tiers

1.2 Positionnement

Le backend étant crucial pour la stabilité, la sécurité, et la performance de l'application , il gère toute la logique serveur, la communication avec la base de données, l'intégration des modèles de machine learning et l'API. Il veille à ce que l'application fonctionne efficacement en arrière-plan tout en permettant une interaction fluide avec le frontend et d'autres systèmes externes.

2. Fin de l'Analyse

2.1 Identification des Scénarios et Cas d'Utilisation :

Le client (Frontend) consomme l'API pour divers cas d'utilisation, permettant une interaction fluide et personnalisée avec le système.

Le backend propose plusieurs scénarios d'utilisation courants pour répondre aux besoins des différents acteurs. Parmi ces scénarios, on retrouve l'analyse et classification d'image, destinée aux utilisateurs particuliers, où une application backend Flask interagit avec un modèle de reconnaissance d'image pour identifier et classer les déchets. Un autre cas d'utilisation est l'affichage des déchetteries disponibles, qui s'appuie sur une API externe de localisation des centres de recyclage, permettant aux utilisateurs particuliers de trouver facilement les points de collecte les plus proches. Pour les entreprises de recyclage, le backend propose une classification en masse des déchets, où un employé utilise une caméra connectée et l'application Flask pour traiter de grandes quantités de déchets via le modèle de reconnaissance d'image. Le système inclut également des fonctionnalités d'inscription et d'authentification pour les utilisateurs particuliers, les employés de recyclage et les administrateurs, garantissant un accès sécurisé et personnalisé. Les rapports analytiques sont générés pour les administrateurs et les entreprises de recyclage, offrant des insights précieux sur les activités de tri et de recyclage. Enfin, le système envoie des notifications push personnalisées aux utilisateurs particuliers et conserve un historique de classification des déchets accessible à la fois aux utilisateurs particuliers et aux employés de recyclage, facilitant le suivi et la gestion des déchets.

2.2 Développement des scénarios d'utilisation typiques ou critiques.

Le client (Frontend) consome l'API pour diverses cas d'utilisation. Le backend propose plusieurs scénarios d'utilisation courants :

- **Analyse et classification d'image (Utilisateur particulier)**

Acteur: Utilisateur particulier, Application backend Flask, Modèle de reconnaissance d'image.

Description: L'utilisateur télécharge une image d'un déchet via l'application mobile. L'image est envoyée au backend Flask qui utilise un modèle d'apprentissage automatique pour classifier le déchet.

Scénario principal:

- L'utilisateur sélectionne une image via l'application.
 - L'application envoie l'image au backend via une requête HTTP.
 - Le backend traite l'image et renvoie la catégorie détectée (plastique, métal, verre, etc.) avec un pourcentage de confiance.
 - L'application affiche la catégorie du déchet et propose une recommandation de tri.
- Affichage des déchetteries disponibles (API externe)**

Acteur: Utilisateur particulier, Application backend Flask, API de localisation des centres de recyclage

Description: Une fois le déchet identifié, l'application propose une carte des centres de tri les plus proches.

Scénario principal:

- L'utilisateur demande la localisation des centres de tri.
 - L'application appelle le Backend Flask.
 - Le backend consulte une API externe pour récupérer les données de localisation.
 - L'application affiche les centres sur une carte interactive.
- Classification en masse (Entreprise de recyclage)**

Acteur: Employé d'une entreprise de recyclage , Caméra connectée, Application backend Flask, Modèle de reconnaissance d'image.

Description: Un employé utilise une caméra connectée pour analyser des déchets en continu. Le système identifie chaque déchet et enregistre les résultats.

Scénario principal:

- L'employé démarre le flux vidéo.
- Les images capturées sont envoyées au backend Flask.
- Chaque image est classifiée automatiquement.
- Les résultats sont enregistrés dans la base de données.
- Le système affiche un rapport de classification.

2.3 Diagrammes de séquence pour illustrer ces scénarios.

• Analyse et classification d'image (Utilisateur particulier)

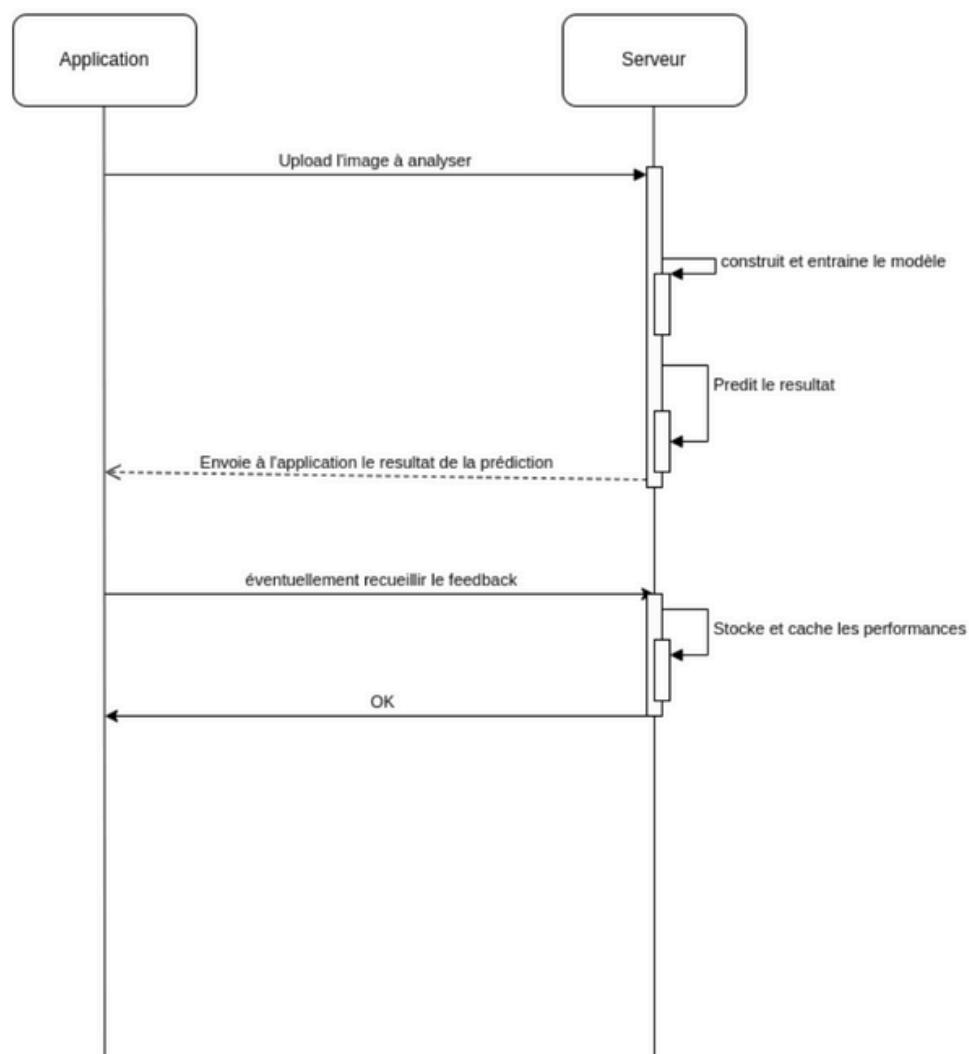


Image: Diagramme de séquence associé aux scénarios d'une analyse simple

- Affichage des déchetteries disponibles (API externe)

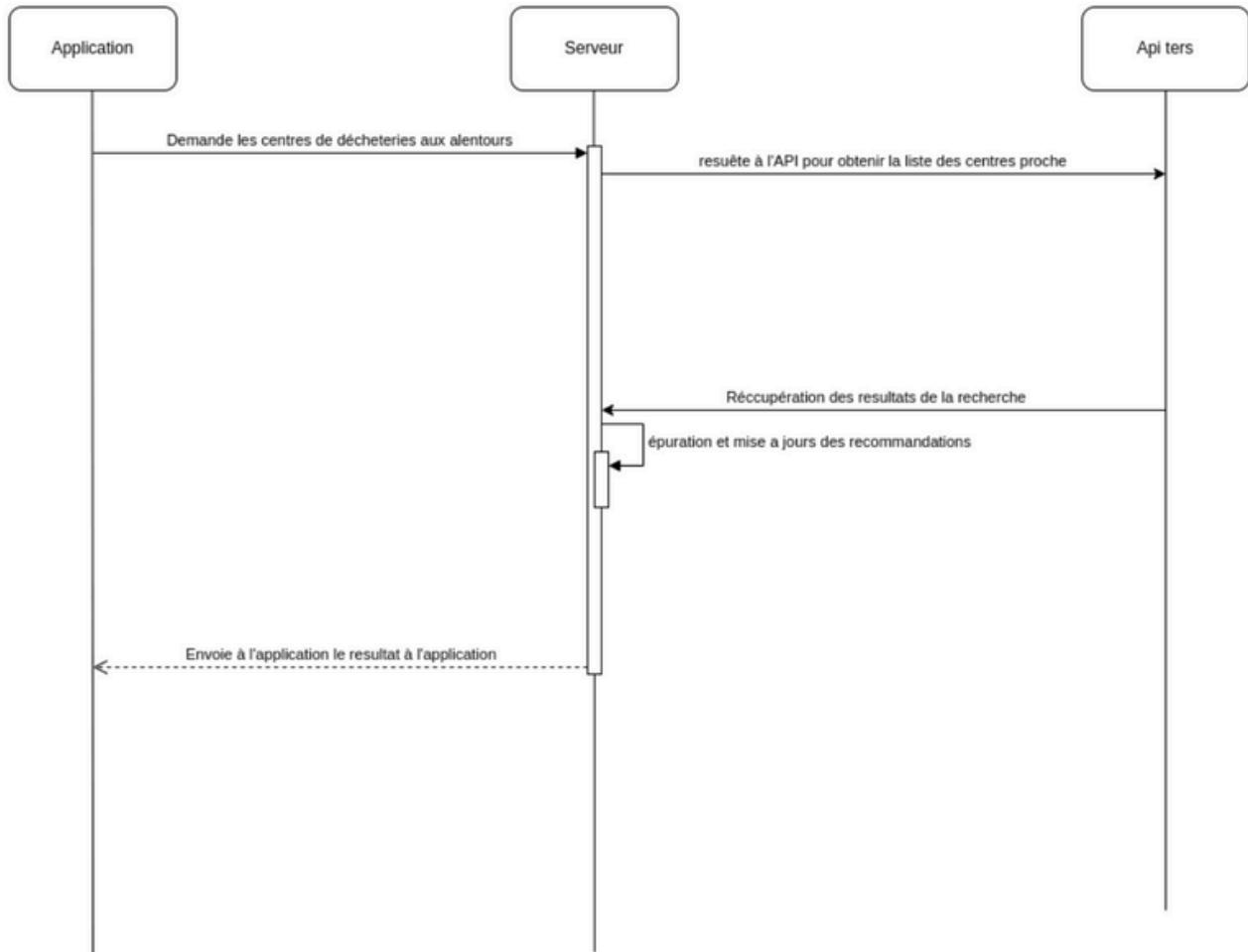


Image: Diagramme de séquence associé aux scénarios de recherche de déchetteries disponible

- Classification en masse (Entreprise de recyclage)

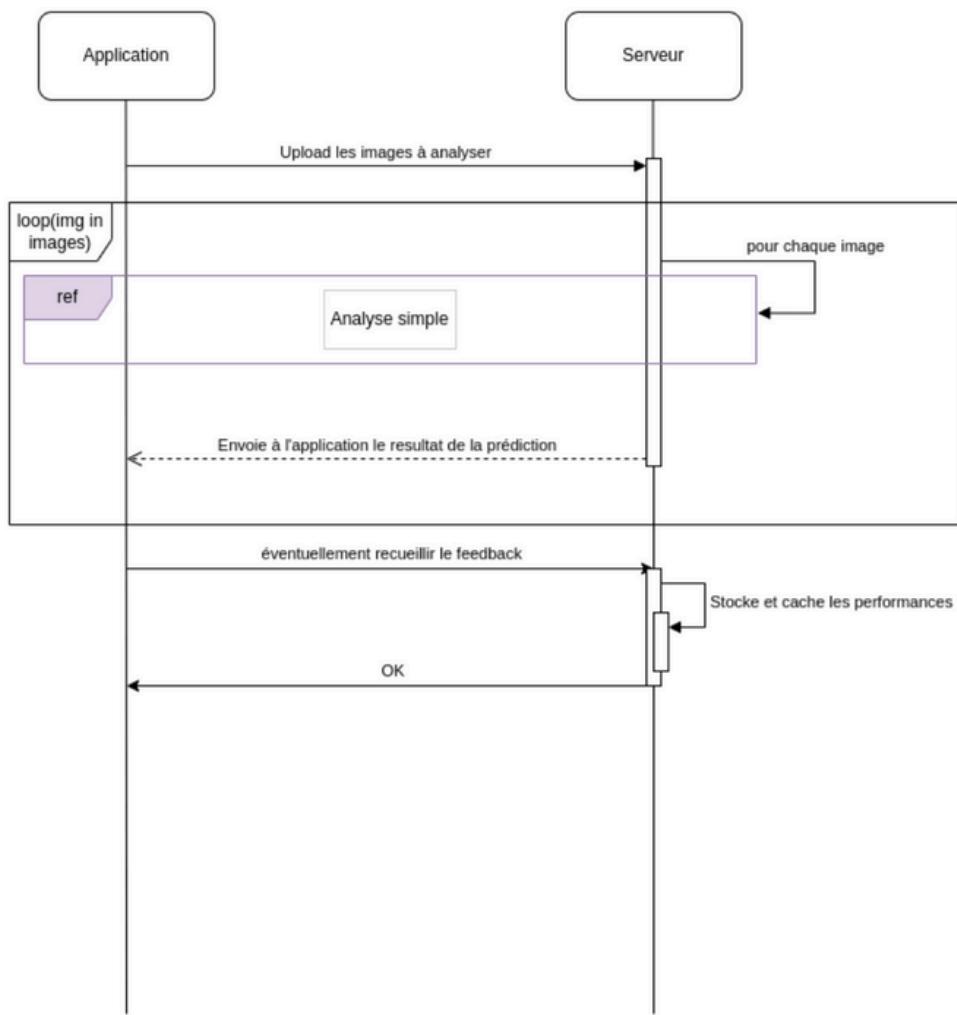


Image: Diagramme de séquence associé aux scénarios de classification multiples

2.4 Recherche de solutions :

Il existe plusieurs approches techniques pour concevoir et développer l'application. Une comparaison des solutions disponibles a été réalisée, en tenant compte des performances, de la compatibilité et de la simplicité d'implémentation.

Technologie retenue : Flask + TensorFlow + Opencv

Pourquoi:

- Flask**

Flask est un framework léger, idéal pour un projet qui ne nécessite pas une complexité excessive. Il permet de créer une API simple pour interagir avec les utilisateurs, leur permettre de télécharger des images et leur fournir des réponses instantanées sur les types de déchets via la classification. Il est bien adapté à la création d'API RESTful pour gérer des interactions simples et efficaces entre le front-end et le back-end, ce qui est essentiel pour un système qui traite des images et fournit des résultats en temps réel.

- Tensorflow**

TensorFlow est l'un des frameworks les plus utilisés pour la création de modèles d'apprentissage automatique, notamment pour les tâches de classification d'image. Grâce à TensorFlow, vous pouvez facilement entraîner un modèle sur un ensemble de données pour reconnaître les différents types de déchets à partir des images téléchargées.

Il facilite le traitement des données d'image, comme le redimensionnement, la normalisation et l'entraînement de réseaux neuronaux convolutifs (CNN), qui sont idéaux pour ce type de tâche.

- Opencv**

OpenCV est un outil très complet pour le traitement d'image. Avant d'envoyer une image à un modèle TensorFlow pour classification, il peut être nécessaire de réaliser des opérations comme le redimensionnement, la rotation, le recadrage, ou encore la correction de la couleur. OpenCV fournit des fonctions robustes pour ces opérations, ce qui permet de préparer correctement les images pour l'analyse.

OpenCV et TensorFlow peuvent être utilisés de manière complémentaire. OpenCV peut servir à manipuler les images, tandis que TensorFlow peut se concentrer sur la classification basée sur des modèles d'apprentissage automatique.

3. Conception

A. Conception générale

a) les diagrammes UML

- *Diagramme de cas d'utilisation*

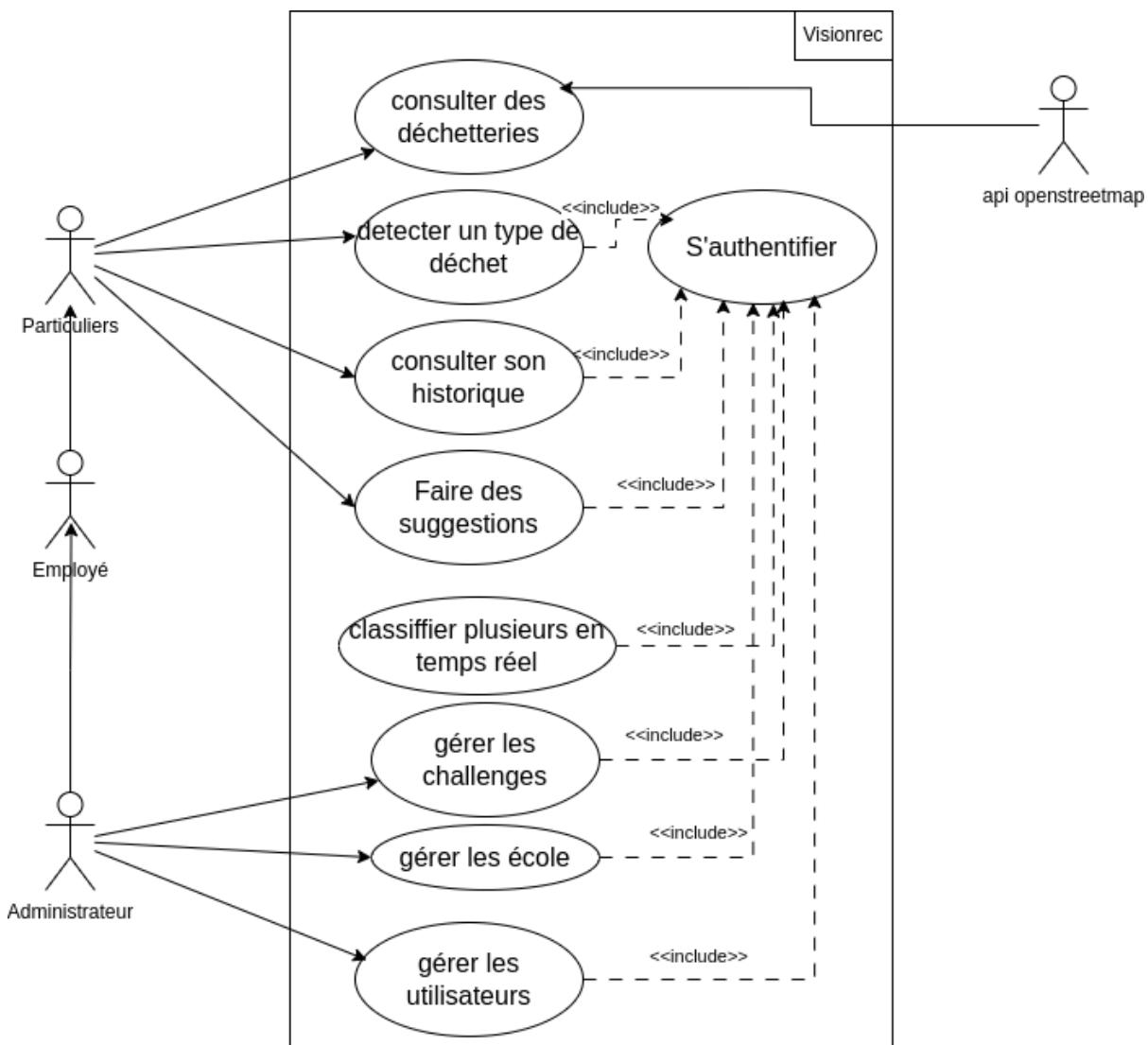


Figure: Diagramme de séquence associé aux scénarios de classification multiples

• Diagramme d'activités

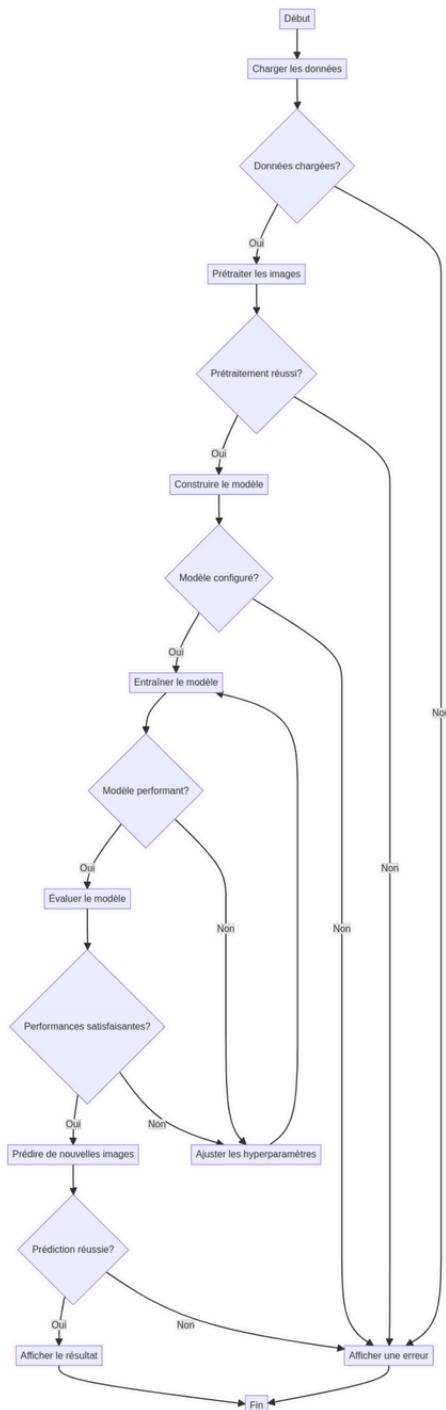


Figure: Diagramme d'Activité du Système de Classification des Déchets

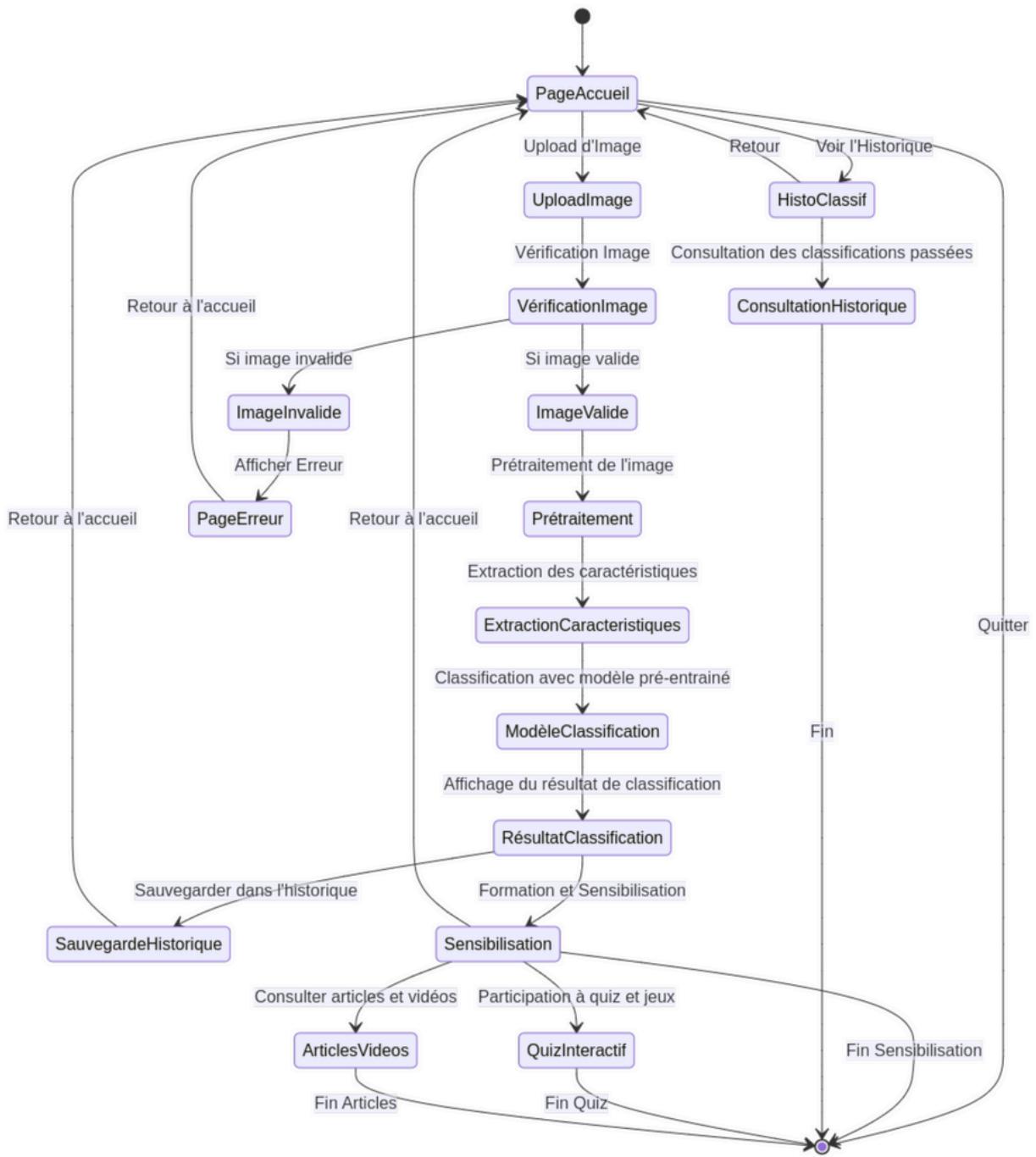


Figure: Diagramme d'Activité de visionrec

• Diagramme de classes

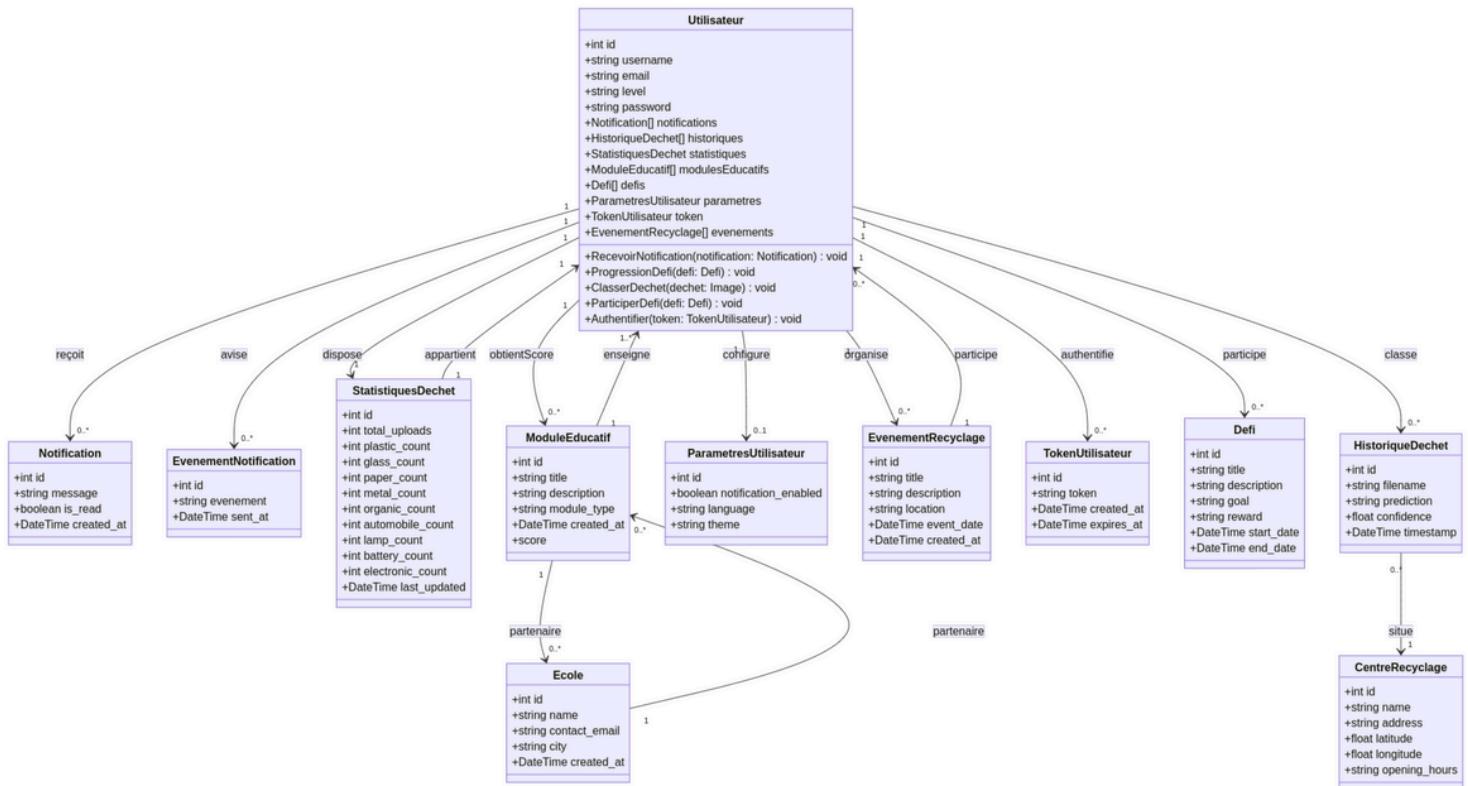


Figure: Diagramme de classe

• Modèle Conceptuel de Données, MCD

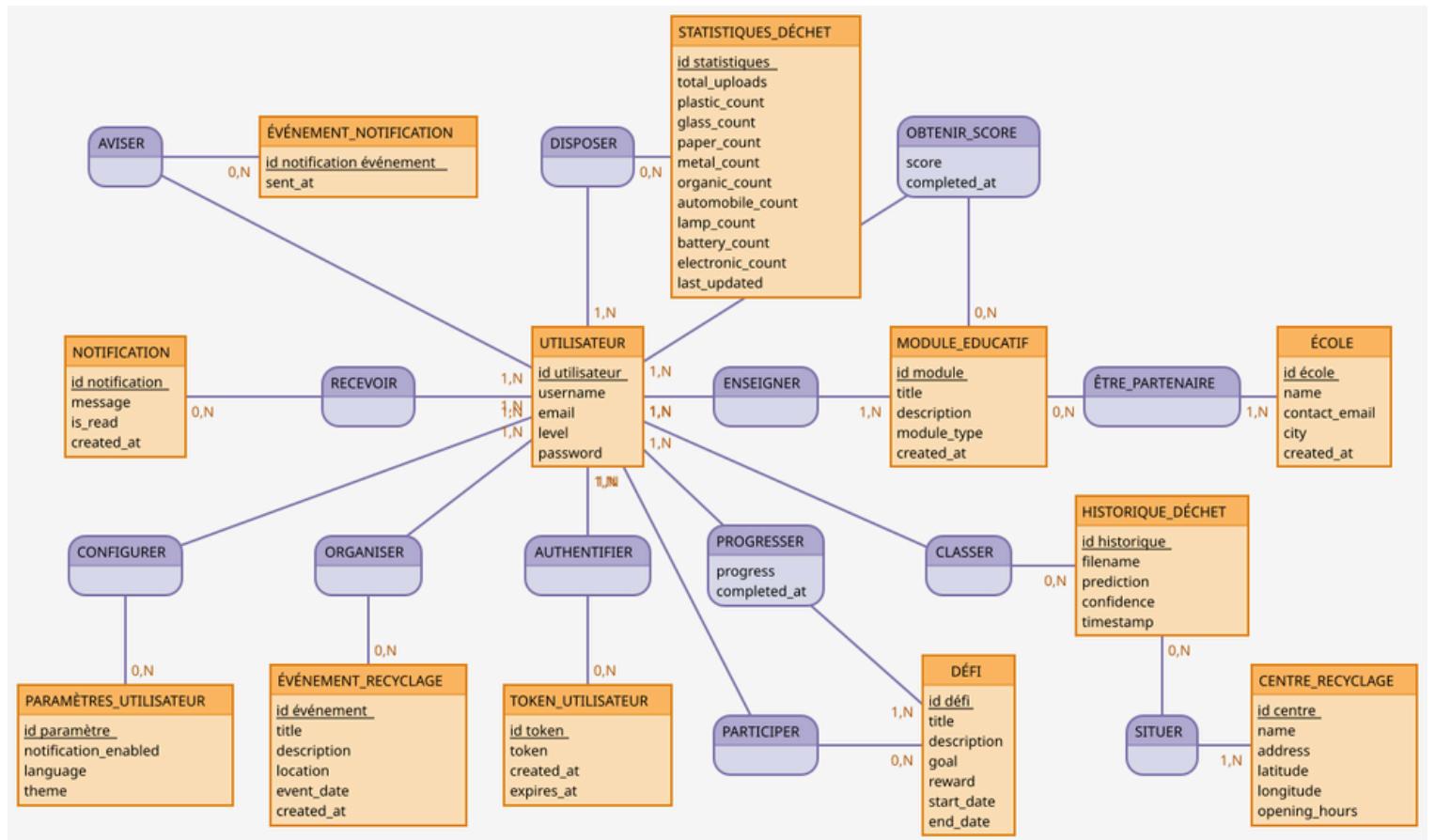


Figure: Modèle conceptuel de donnée

• Diagramme d'architecture logicielle

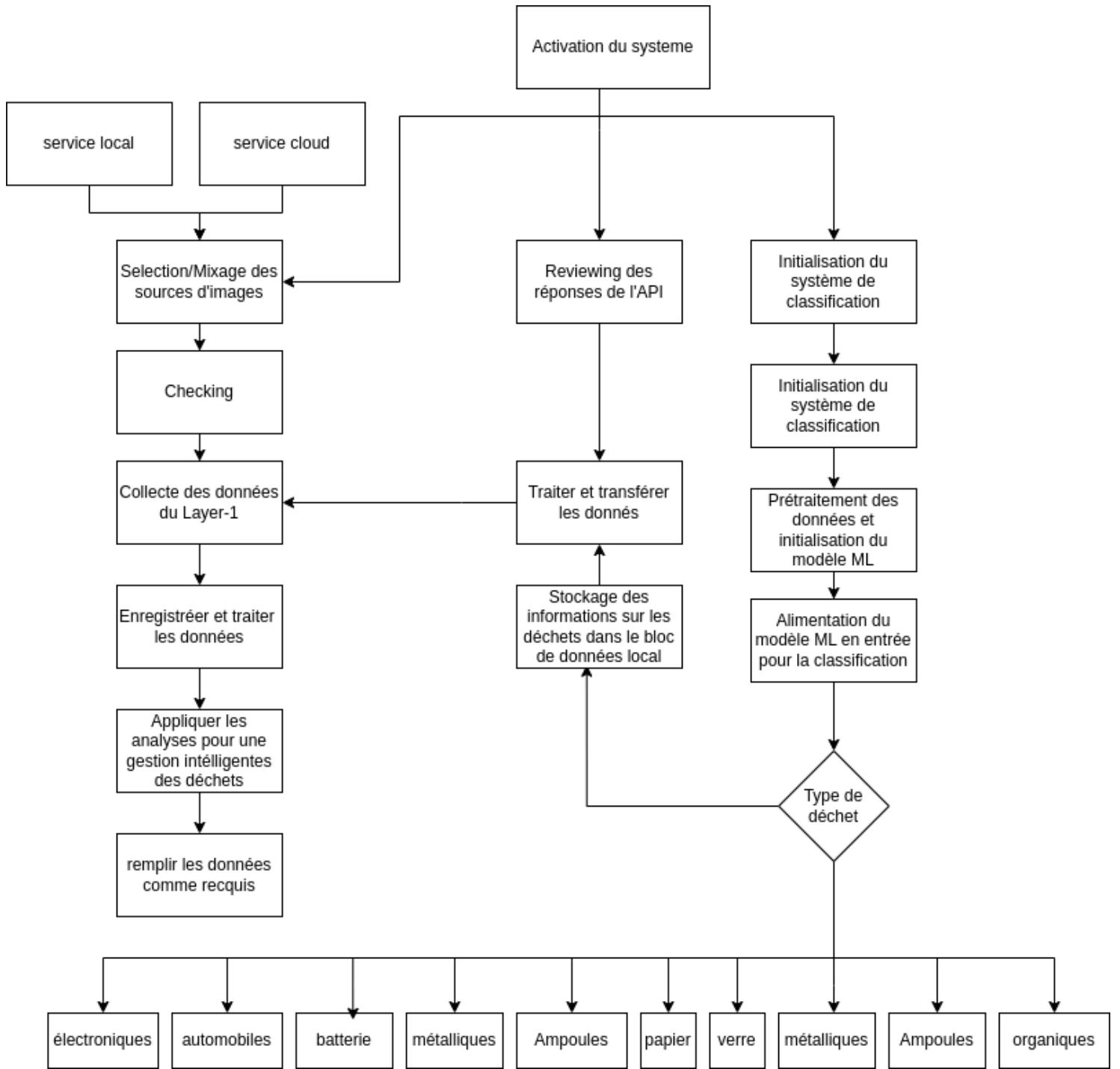


Figure: Diagramme d'architecture logicielle

- Diagramme UML de déploiement utilisé (draw.io)

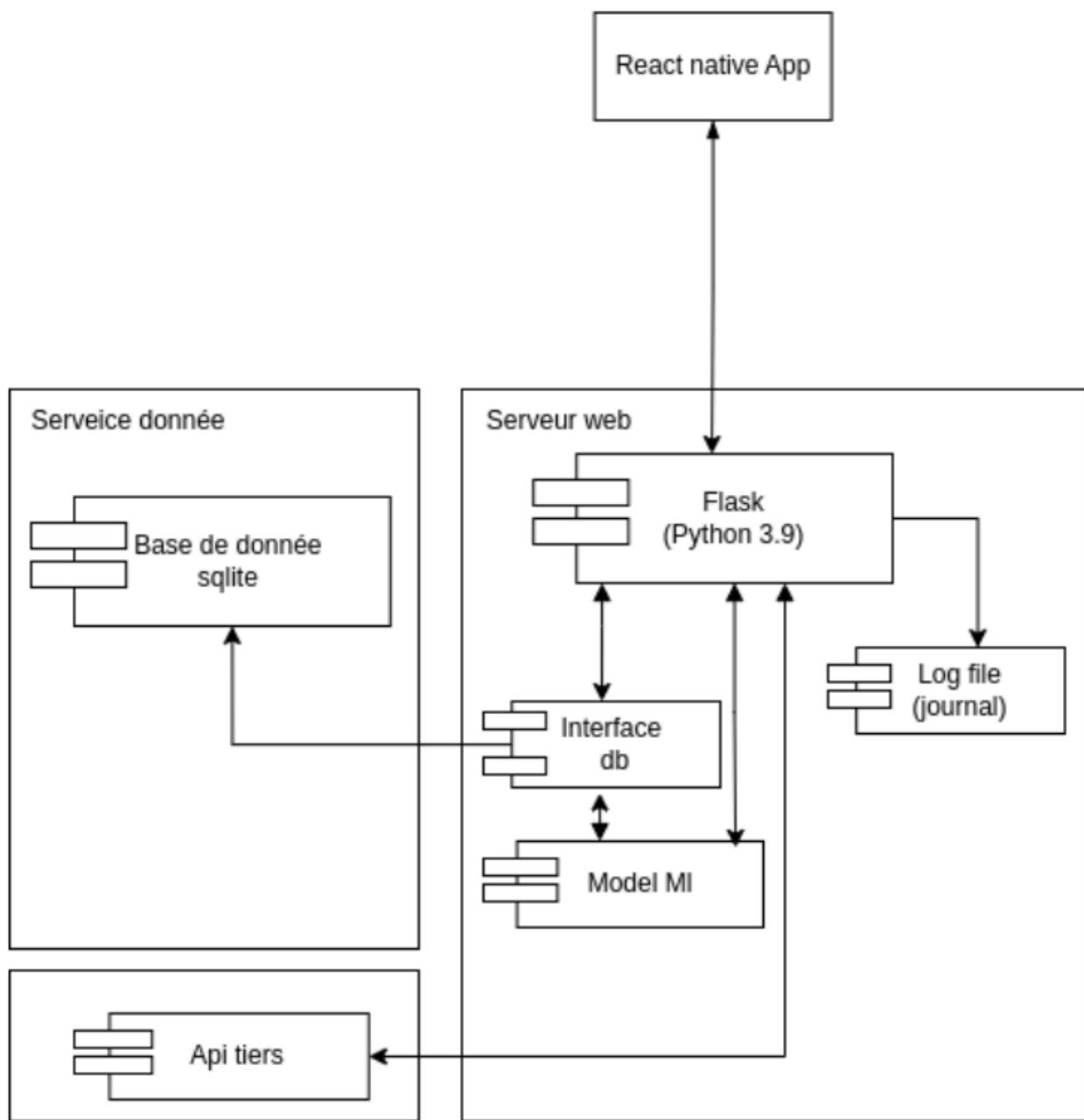


Figure: Diagramme de déploiement

b. Difficultés rencontrées et les solutions apportées:

L'un des principaux défis rencontrés concernait la **faible précision du modèle** lors de la classification des déchets, notamment des erreurs fréquentes entre plastique et métal. Pour résoudre ce problème, nous avons optimisé le modèle en utilisant des algorithmes CNN plus avancés, augmenté le dataset avec des images issues d'open datasets et appliqué des techniques d'augmentation telles que la rotation, le recadrage et l'ajout de filtres.

Nous avons également rencontré des **erreurs "no such table"** lors des requêtes en raison d'une mauvaise initialisation de la base de données. La solution a consisté à créer une commande spécifique "flask init-db" avec "app.cli.command()" et à ajouter "db.create_all()" dans le fichier principal, en intégrant des vérifications automatiques avant chaque lancement.

Un autre problème concernait la **mauvaise gestion des tokens d'authentification**, entraînant leur expiration prématurée. Nous avons ajusté la durée de vie des tokens directement dans la configuration pour éviter ce souci. De plus, l'**absence de configuration CORS** bloquait les appels API. Ce problème a été corrigé en intégrant "CORS(app)" via le module "flask-cors" et en définissant précisément les origines autorisées.

La gestion des **fichiers volumineux** posait également problème, entraînant des erreurs de timeout. Pour y remédier, nous avons ajouté des configurations spécifiques comme "MAX_CONTENT_LENGTH" et mis en place un traitement asynchrone pour gérer le téléchargement et le traitement des images plus efficacement.

Enfin, **le déploiement sur O2switch** a été particulièrement complexe en raison d'erreurs liées à l'environnement Python et à des incompatibilités de versions, de problèmes dans la gestion des dépendances via "requirements.txt" et de difficultés à configurer le fichier ".htaccess" pour rediriger vers l'application Flask. Pour surmonter ces obstacles, nous avons créé un environnement virtuel Python avec "virtualenv", installé les dépendances via "pip install -r requirements.txt", ajouté un fichier "passenger_wsgi.py" définissant l'entrée du point Flask, et configuré ".htaccess" comme suit :

```
PassengerEnabled    on    PassengerAppRoot    /home/user/visionrec
PassengerPython    /home/user/visionrec/venv/bin/python3    DirectoryIndex
passenger_wsgi.py
```

Enfin, les logs Apache ("~/logs/log.log") ont été utilisés pour identifier et corriger les éventuelles erreurs restantes.

Ce processus d'optimisation a permis d'améliorer la robustesse et les performances de notre application tout en facilitant son déploiement et son utilisation en production.

c. Interface utilisateur

Présentation de l'interface

Pour permettre au frontend de comprendre l'utilisation des endpoint , J'ai crée une documentation automatique avec swagger qui me permet de décrire chaque routes et les paramètres requis pour requêter .

The screenshot shows a Swagger UI interface with the title "spec : Documentation de l'api vision rec". At the top right are buttons for "Show/Hide", "List Operations", "Expand Operations", and "Raw". Below the title is a list of API operations grouped by resource:

- Auth:**
 - POST /auth/login
 - POST /auth/register
 - POST /auth/logout
- Waste:**
 - POST /waste/classify
- Schools:**
 - GET /schools/history
 - POST /schools/create
 - GET /schools
 - GET /schools/{id}
 - PUT /schools/{id}/update
 - DELETE /schools/{id}/delete
- Challenges:**
 - POST /challenges/create
 - GET /challenges
 - GET /challenges/{id}
 - PUT /challenges/{id}/update
 - DELETE /challenges/{id}/delete

[BASE URL: http://127.0.0.1:5000/api/spec/_resource_list.json , API VERSION: 1.0]

Image: Documentation 1

swagger http://127.0.0.1:5000/api/spec/_/resource_list.json api_key Explore

spec : Documentation de l'api vision rec

Show/Hide | List Operations | Expand Operations | Raw

POST /auth/login

POST /auth/register

Implementation Notes

Inscrit un nouvel utilisateur

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	(required)	Données d'inscription	body	object

Parameter content type: application/json ▾

Error Status Codes

HTTP Status Code	Reason
201	Inscription réussie
400	Erreur lors de l'inscription

Try it out!

POST /auth/logout

POST /waste/classify

GET /waste/history

Image: Documentation 2

GET /schools/{id}

PUT /schools/{id}/update

Implementation Notes
Mettre à jour les informations d'une école

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	<input type="text"/>		path	int
school_id	(required)	ID de l'école à mettre à jour	path	integer
body	(required)	Nouvelles informations de l'école	body	object

Parameter content type:

Error Status Codes

HTTP Status Code	Reason
200	École mise à jour avec succès
400	Erreur lors de la mise à jour de l'école

DELETE /schools/{id}/delete

Implementation Notes
Supprimer une école

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	<input type="text"/>		path	int
school_id	(required)	ID de l'école à supprimer	path	integer

Error Status Codes

HTTP Status Code	Reason

Image: Documentation 3

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	<pre>{ "username": "john_gdceoexv", "email": "johnqi@exadmeplex.com", "password": "password123" }</pre>	Données de connexion	body	object

Parameter content type: application/json ▾

Error Status Codes

HTTP Status Code	Reason
200	Connexion réussie
400	Erreur de connexion

[Try it out!](#) [Hide Response](#)

Request URL

```
http://adidome.com:80/visionrec/auth/login
```

Response Body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhCI6MTczOTUwNjg4MiwanRpIjoiNjA4YTRkMT",
  "success": true,
  "message": "Connexion réussie",
  "user_id": 2
}
```

Response Code

```
200
```

Response Headers

```
{
  "Access-Control-Allow-Origin": "http://adidome.com",
  "Connection": "keep-alive",
  "Content-Length": "515",
  "Content-Type": "application/json",
  "Date": "Fri, 14 Feb 2025 04:21:22 GMT",
  "Server": "o2switch-PowerBoost-v3",
  "Status": "200 OK"
}
```

POST /auth/register

Image: Documentation 4

B. Conception détaillée

Dans cette partie nous allons fournir une compréhension approfondie des éléments spécifiques et techniques de notre travail

a) Architecture Logicielle Détailée: MVC (Modèle-Vue-Contrôleur)

L'application Vision Rec est conçue selon le modèle MVC :

- Modèle (Model) : Gestion de la logique métier avec SQLAlchemy (classes User, Challenge, WasteHistory, School, etc.).
- Vue (View) : API REST avec Flask et documentation avec Flask-Restful-Swagger.
- Contrôleur (Controller) : Gestion des routes, des opérations CRUD et des réponses JSON.

b) Structure du Projet

```
.
├── app
│   ├── db.py
│   └── errors
│       └── handlers.py
│           └── __pycache__
│               └── handlers.cpython-312.pyc
│   ├── __init__.py
│   └── models
│       ├── challenge.py
│       ├── school.py
│       ├── user.py
│       └── waste.py
└── routes
    ├── auth
    │   ├── auth_routes.py
    │   └── utils.py
    └── __init__.py
        ├── challenge_routes.py
        ├── main_routes.py
        ├── school_routes.py
        └── utils.py
    └── static
        └── utils
            ├── geolocation.py
            └── image_processing.py
    └── app.db
    └── config.py
    └── instance
        └── app.db
    └── logs
        └── app.log
    └── models
        └── waste_classifier_model.h5
    └── paparapa.py
    └── passenger_wsgi.py
    └── requirements.all.txt
    └── requirements.txt
    └── run.py
    └── swagger_doc.py
    └── tmp
        └── restart.txt
```

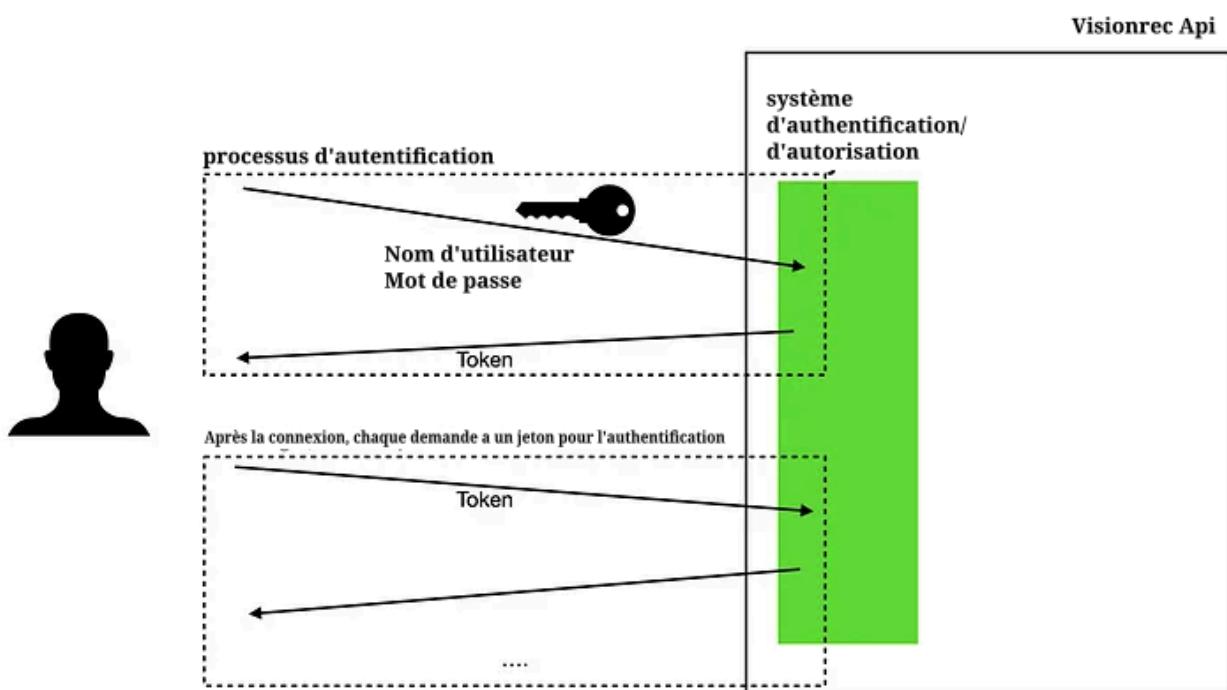
Image: Structure du projet

c) Modules Clés

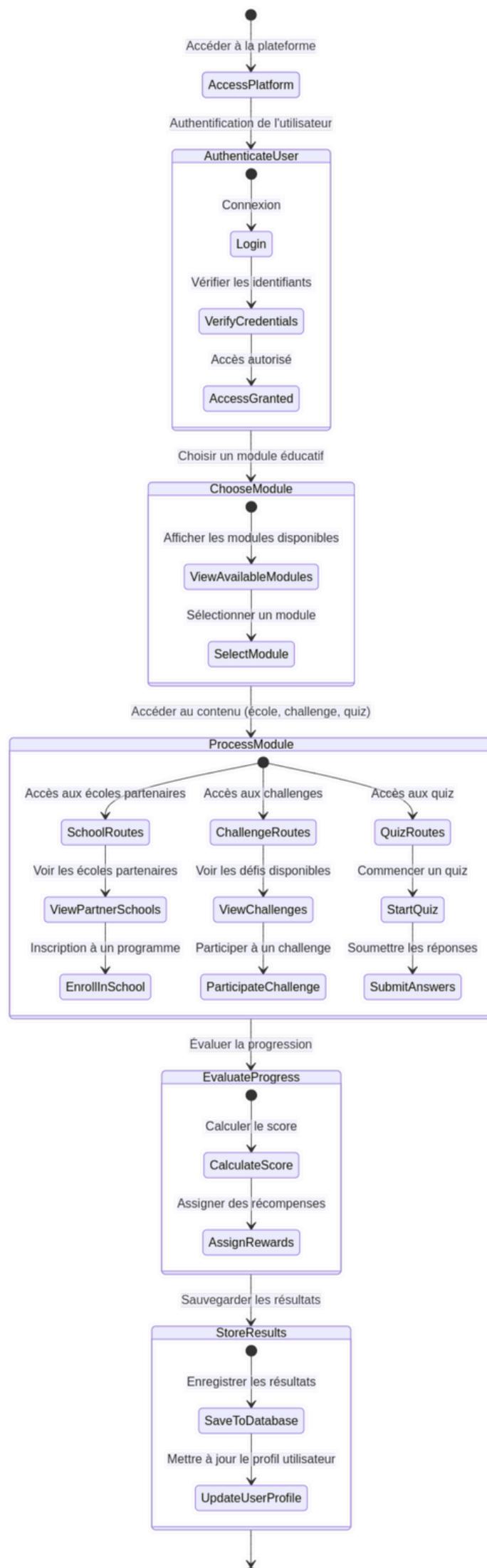
- Authentification : Gestion des tokens JWT avec flask-jwt-extended.
 - Formation Continue : Routes pour les écoles, challenges et quiz.
 - Analyse des Déchets : Modèles entraînés Tensorflow et visualisation/traitement des images avec OpenCV
 - Temps réel : nous utiliserons Socket io pour diffuser en temps réel les résultats de la classification
- Intégration Swagger

Chaque route utilise “@swagger.operation()” pour générer la documentation interactive.

d) Illustration par des diagrammes d'architecture détaillés.



Fogure: Gestion des tokens JWT



e) Conception des Composants :

Description détaillée de chaque composant du système

Le système repose sur une interaction dynamique entre plusieurs entités pour gérer efficacement la classification des déchets, le suivi des statistiques, la sensibilisation des utilisateurs et leur engagement dans des défis et événements liés au recyclage.

Tout d'abord, un utilisateur peut s'authentifier dans le système via un token d'authentification stocké dans la table TOKEN_UTILISATEUR. Une fois connecté, il peut classer des déchets en téléchargeant une image qui sera analysée et enregistrée dans HISTORIQUE_DÉCHET, associée à lui via la relation CLASSER. À chaque classification, les statistiques de tri sont mises à jour dans STATISTIQUES_DÉCHET, permettant de suivre l'évolution de ses contributions.

L'utilisateur peut également recevoir des notifications via NOTIFICATION, qui l'informent d'événements importants, comme des mises à jour sur ses performances ou des rappels de défis en cours. Il peut participer à des défis (DÉFI) ayant un objectif spécifique (ex. classifier 10 plastiques), et son progression est suivie dans PROGRESSER.

Pour renforcer l'éducation au tri, le système intègre des modules éducatifs (MODULE_EDUCATIF), proposés par des écoles partenaires (ÉCOLE). Les utilisateurs peuvent suivre ces modules et obtenir un score (OBTENIR_SCORE) en fonction de leurs performances, notamment dans des quiz ou vidéos interactives.

En parallèle, les événements de recyclage (ÉVÉNEMENT_RECYCLAGE) sont organisés, et les utilisateurs peuvent y être informés grâce aux ÉVÉNEMENT_NOTIFICATION. Ces événements visent à favoriser l'engagement communautaire en proposant des actions locales.

Enfin, les centres de recyclage (CENTRE_RECYCLAGE) sont référencés dans le système, permettant aux utilisateurs d'identifier les points de dépôt les plus proches et de mieux comprendre où diriger leurs déchets selon leur nature.

Le modèle

Pour mettre en place un modèle nous avons importé les bibliothèques nécessaires, notamment pour le prétraitement des images, l'entraînement du modèle et l'évaluation des performances.

```
1 # Importation des dépendances nécessaires
2 import warnings # Pour gérer les avertissements
3 import tensorflow as tf # Pour le deep learning
4 # Pour le prétraitement des images
5 from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
6 # Modèle pré-entraîné MobileNetV2
7 from tensorflow.keras.applications import MobileNetV2
8 from tensorflow.keras.preprocessing import image # Pour manipuler les images
9 # Pour charger un modèle sauvegardé
10 from tensorflow.keras.models import load_model
11 # Couches du modèle
12 from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization
13 from tensorflow.keras.models import Model # Pour créer un modèle
14 from tensorflow.keras.optimizers import Adam # Optimiseur Adam
15 # Callbacks pour l'entraînement
16 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
17
18 # Importation des modules de scikit-learn pour l'évaluation
19 from sklearn.metrics import classification_report, confusion_matrix
20
```

Ensuite, nous avons chargé les images depuis un dataset organisé en sous-dossiers représentant chaque classe de déchets.

```
35 # Définition des répertoires de base pour le dataset
36 BASE_DIR = r"_____ " # Chemin vers le dataset
37 train_dir = os.path.join(BASE_DIR, 'train') # Répertoire d'entraînement
38 test_dir = os.path.join(BASE_DIR, 'test') # Répertoire de test
39
40 # Visualisation des données d'entraînement
41 # Chemin vers le dossier d'entraînement
42 train_dir = r"_____ \train"
43 class_names = os.listdir(train_dir) # Liste des classes (dossiers)
44
```

Ces images seront augmentées artificiellement grâce à 'ImageDataGenerator' afin de rendre le modèle plus robuste aux variations des données réelles.

```

45
46 # Affichage du nombre d'images par classe et visualisation de quelques images
47 for class_name in class_names:
48     class_dir = os.path.join(train_dir, class_name)
49     class_images = glob.glob(os.path.join(
50         class_dir, '*.jpg')) # Liste des images .jpg
51     print(
52         f"Nombre d'images d'entraînement dans la classe '{class_name}': {len(class_images)}")
53
54 # Affichage de 5 images aléatoires de la classe
55 if len(class_images) > 0:
56     plt.figure(figsize=(10, 5))
57     for i in range(5):
58         plt.subplot(1, 5, i+1)
59         # Choix aléatoire d'une image
60         img_path = np.random.choice(class_images)
61         # Chargement et redimensionnement
62         img = load_img(img_path, target_size=(224, 224))
63         # Conversion en tableau et normalisation
64         img_array = img_to_array(img) / 255.0
65         plt.imshow(img_array)
66         plt.axis('off')
67         plt.title(class_name)
68     plt.show()
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 # Augmentation des données
101 train_datagen = ImageDataGenerator([
102     rescale=1.0/255, # Normalisation des pixels
103     rotation_range=30, # Rotation aléatoire jusqu'à 30 degrés
104     width_shift_range=0.2, # Décalage horizontal jusqu'à 20%
105     height_shift_range=0.2, # Décalage vertical jusqu'à 20%
106     shear_range=0.2, # Cisaillement jusqu'à 20%
107     zoom_range=0.2, # Zoom jusqu'à 20%
108     horizontal_flip=True, # Retournement horizontal
109     fill_mode='nearest' # Remplissage des pixels manquants
110 ]
111
112 # Générateur de données de test (sans augmentation)
113 test_datagen = ImageDataGenerator(rescale=1.0/255) # Normalisation des pixels
114
115 # Chargement des données d'entraînement
116 train_generator = train_datagen.flow_from_directory(
117     # Chemin des données d'entraînement
118     directory=os.path.join(BASE_DIR, 'train'),
119     target_size=IMG_SIZE, # Redimensionnement des images
120     batch_size=32, # Taille du batch
121     class_mode='categorical' # Classification catégorielle
122 )
123
124 # Chargement des données de test
125 test_generator = test_datagen.flow_from_directory(
126     directory=os.path.join(BASE_DIR, 'test'), # Chemin des données de test
127     target_size=IMG_SIZE, # Redimensionnement des images
128     batch_size=32, # Taille du batch
129     class_mode='categorical', # Classification catégorielle
130     shuffle=False # Pas de mélange pour l'évaluation
131 )
132
133 # Affichage des étiquettes de classe
134 # Dictionnaire des classes et leurs indices
135 class_labels = train_generator.class_indices
136 print("Étiquettes de classe et valeurs encodées :")
137 print(class_labels)
138

```

Le modèle est construit à partir de MobileNetV2, un modèle pré-entraîné sur ImageNet, auquel on ajoute plusieurs couches personnalisées ('Dropout', 'Dense', 'BatchNormalization', etc.) pour affiner l'apprentissage sur le dataset spécifique des déchets. Les couches de MobileNetV2 sont gelées pour ne pas modifier ses poids initiaux, ce qui permet d'accélérer l'entraînement et de conserver les connaissances du modèle de base.

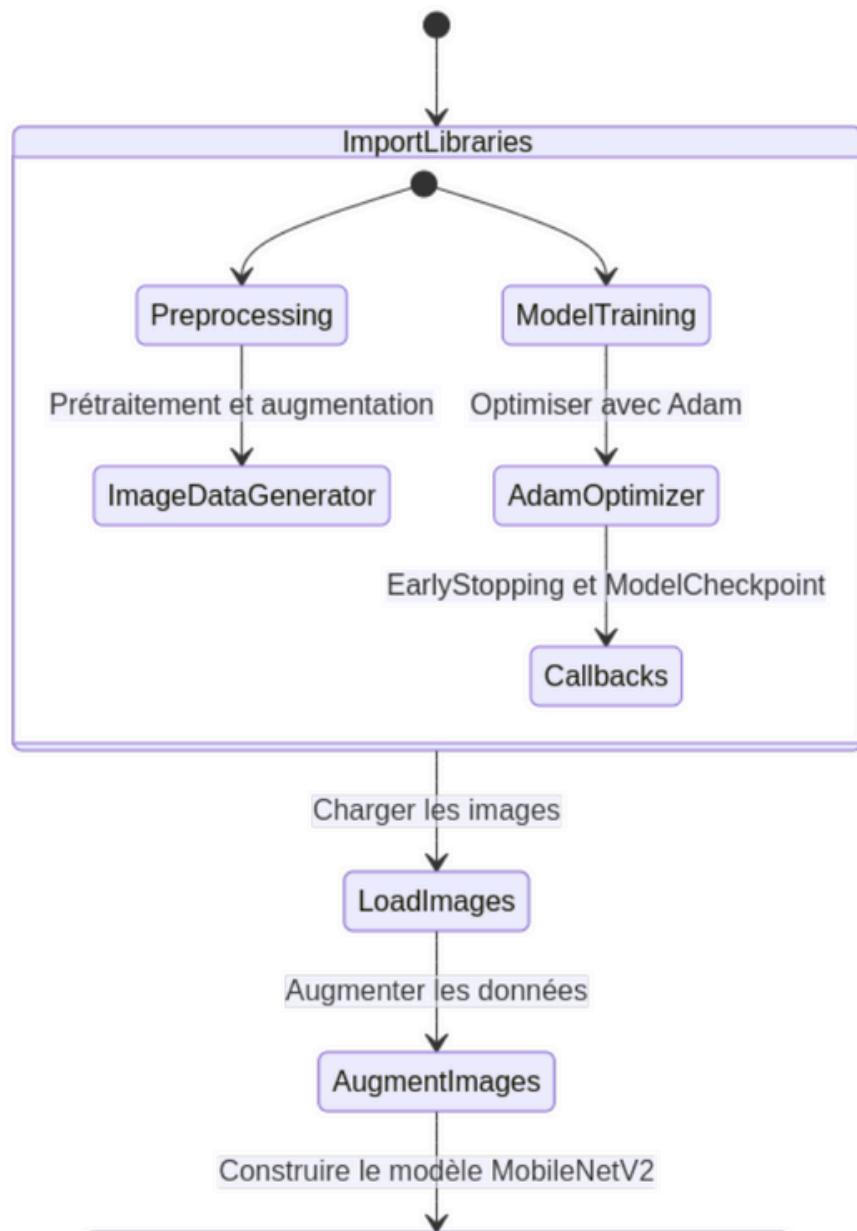
L'entraînement du modèle est optimisé avec 'Adam' et surveillé à l'aide de callbacks ('EarlyStopping', 'ModelCheckpoint') pour stopper l'entraînement en cas de surapprentissage et sauvegarder les meilleurs poids.

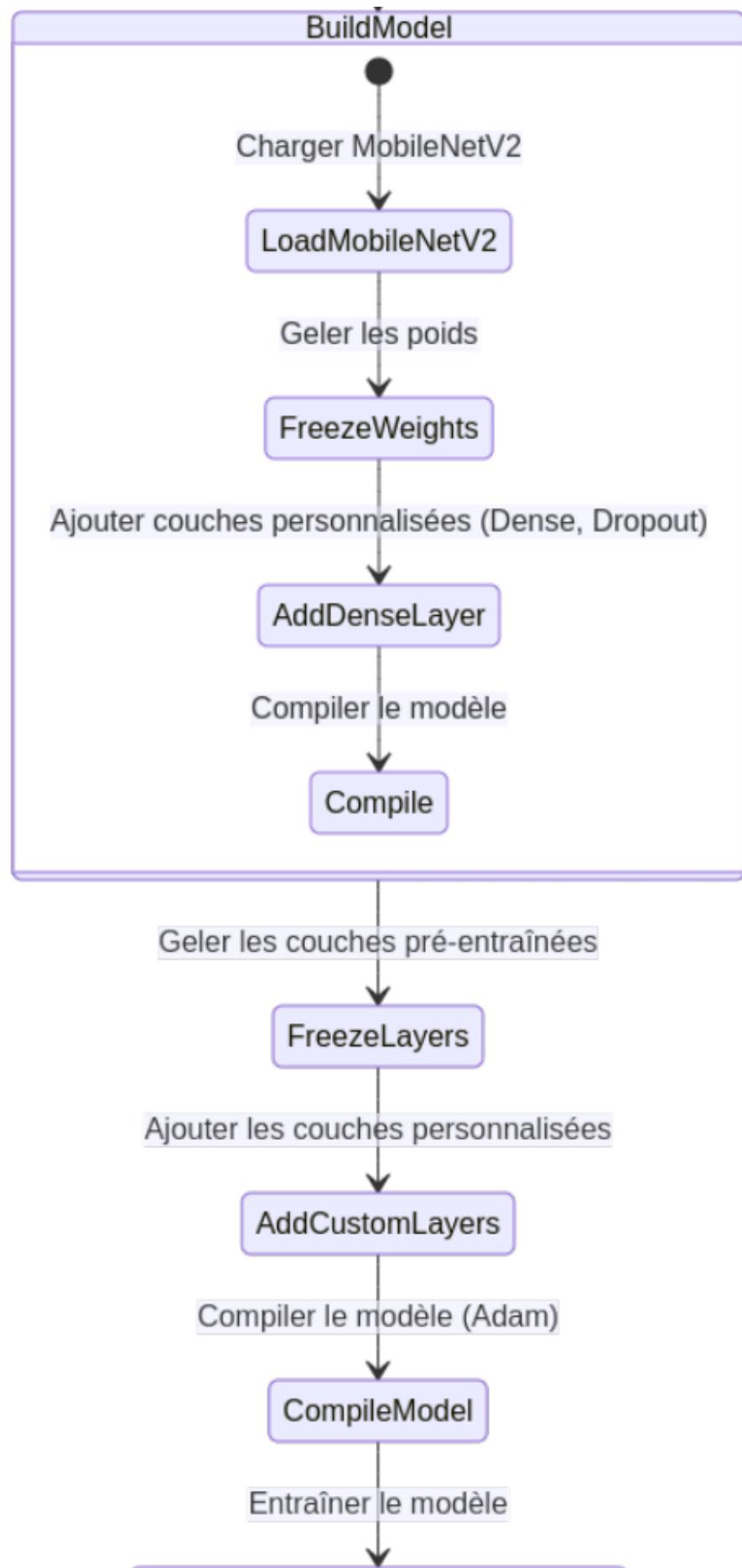
```
152  
153 # Ajout de couches personnalisées  
154 model = Sequential()  
155 model.add(base_model)  
156 model.add(Dropout(0.2))  
157 model.add(Flatten())  
158 model.add(BatchNormalization())  
159 model.add(Dense(512, activation="relu", kernel_initializer='he_uniform'))  
160 model.add(BatchNormalization())  
161 model.add(Dropout(0.2))  
162 model.add(Dense(256, activation="relu", kernel_initializer='he_uniform'))  
163 model.add(BatchNormalization())  
164 model.add(Dropout(0.2))  
165 model.add(Dense(128, activation="relu", kernel_initializer='he_uniform'))  
166 model.add(Dropout(0.2))  
167 model.add(Dense(9, activation="softmax")) # Couche de sortie pour 9 classes  
168  
169 # Compilation du modèle  
170 model.compile(optimizer=Adam(learning_rate=0.0001),  
171 | | | | loss='categorical_crossentropy', metrics=['accuracy'])  
172
```

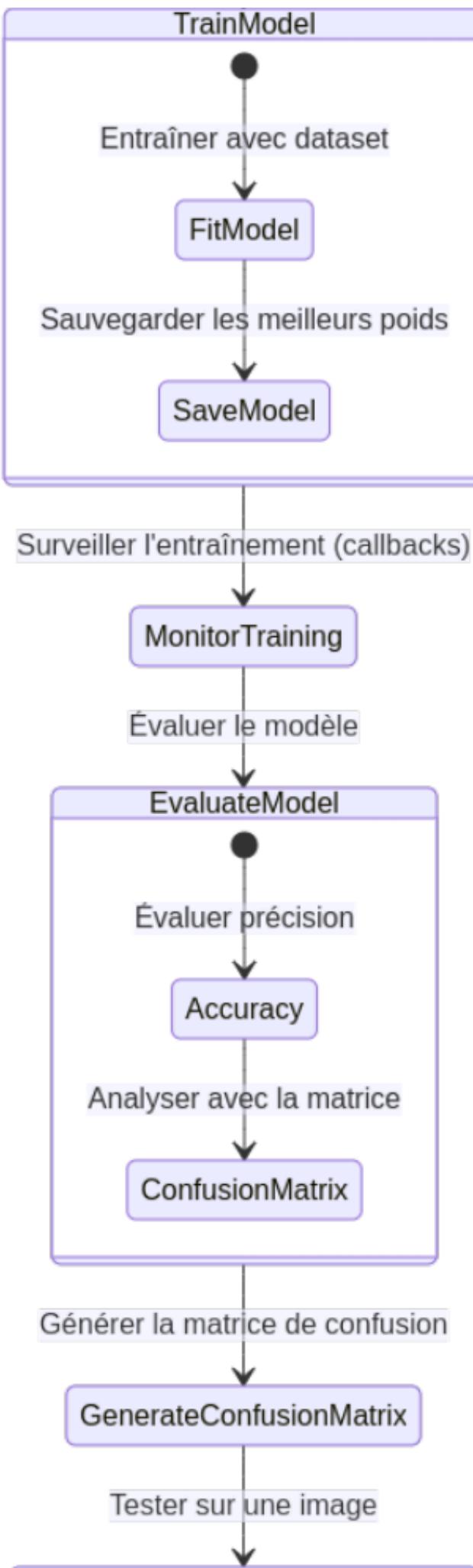
Une fois entraîné, le modèle est évalué sur les données de test avec des métriques comme l'accuracy et une matrice de confusion, générée avec 'seaborn', qui permet d'analyser les erreurs de classification.

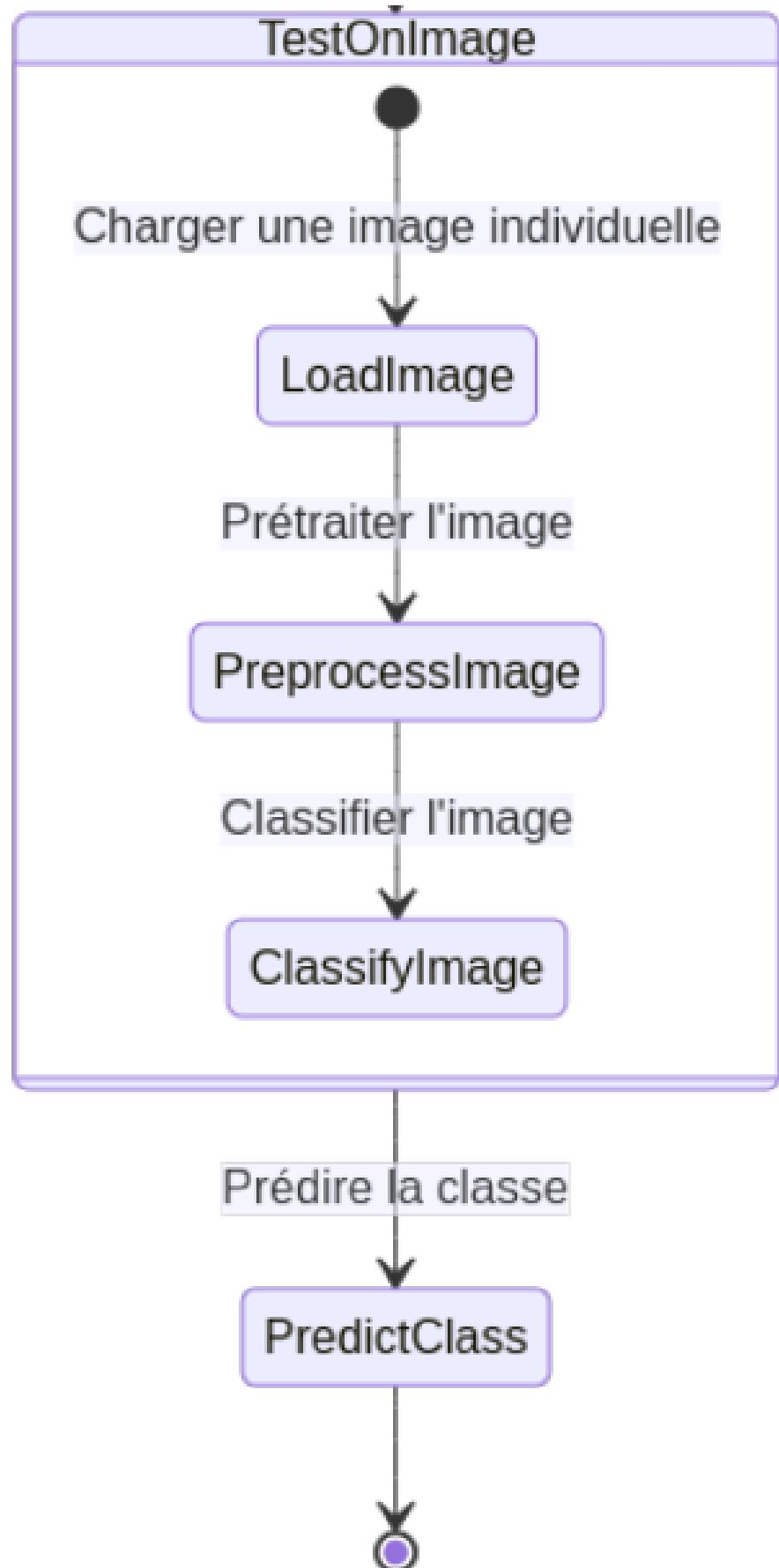
Enfin, le modèle est testé sur une image individuelle : il charge l'image, la prétraite en la redimensionnant et en la normalisant, puis effectue une prédiction pour afficher la classe la plus probable. L'ensemble du processus permet donc de prendre une image en entrée, d'extraire ses caractéristiques via MobileNetV2 et d'attribuer une classe correspondante grâce aux couches personnalisées, rendant l'application utilisable pour le tri des déchets.

voici une suite de diagrammes qui illustre le fonctionnement de notre model:









f) Conclusion

Le projet Vision Rec repose sur une architecture logicielle basée sur le modèle MVC, où le modèle gère la logique métier avec SQLAlchemy, la vue est une API REST sous Flask avec documentation Swagger, et le contrôleur gère les routes et les réponses JSON. Il inclut des modules clés tels que l'authentification avec JWT, la gestion des challenges, l'analyse des déchets via un modèle de deep learning (MobileNetV2 avec TensorFlow et Keras), et le traitement des images avec OpenCV. L'application intègre également des fonctionnalités de diffusion en temps réel via Socket.IO, des notifications pour les utilisateurs, des modules éducatifs fournis par des écoles partenaires, et le suivi des progrès dans des défis de recyclage. Le système met à jour les statistiques des utilisateurs, propose des événements de recyclage, et référence des centres de recyclage pour améliorer la gestion des déchets, tout en garantissant une expérience fluide et sécurisée.

Discussion sur l'impact de ces détails sur le projet global.

Un impact significatif sur son efficacité, sa robustesse et son évolutivité, tout en garantissant une expérience utilisateur fluide et engageante. L'architecture MVC, en séparant clairement la logique métier, l'interface utilisateur et le contrôle des actions, permet une gestion modulaire et flexible du système. Cela facilite l'ajout de nouvelles fonctionnalités, comme l'intégration de nouveaux types de défis ou la mise à jour des modèles de classification, sans perturber les autres parties de l'application.

L'intégration de l'authentification JWT assure une sécurité optimale pour la gestion des utilisateurs, tout en permettant une authentification sans état, ce qui simplifie la scalabilité du projet. Les modules éducatifs et les notifications renforcent l'aspect éducatif et communautaire du projet, en favorisant l'engagement des utilisateurs dans des actions concrètes pour le recyclage.

De plus, l'utilisation de MobileNetV2 et des techniques de deep learning permet d'assurer une précision élevée dans la classification des déchets, tout en étant optimisé pour des performances rapides, cruciales pour une application mobile ou web.

Le système de gestion des événements et des statistiques, couplé à un modèle d'apprentissage supervisé et des visualisations interactives, permet non seulement de suivre les performances des utilisateurs mais aussi de proposer des actions en temps réel, maximisant ainsi l'impact du projet sur la communauté. Enfin, l'intégration des fonctionnalités de temps réel via Socket.IO et l'évaluation des utilisateurs via des métriques précises assurent que l'application reste pertinente et réactive face aux besoins des utilisateurs, tout en créant une plateforme participative et motivante. En somme, la conception détaillée de Vision Rec met en place un système flexible, performant, sécurisé, et hautement interactif qui favorise l'adhésion des utilisateurs et améliore activement la gestion des déchets dans un cadre communautaire.

h) Tests unitaires et validation :

Stratégie de tests

nous avons utilisé le module unittest pour vérifier le comportement de petites unités de code (comme une fonction ou une méthode) de manière isolée.

```
import unittest
from app.routes.auth.utils import validate_login, validate_registration
from app.models.user import User
from werkzeug.security import generate_password_hash
from unittest.mock import patch, MagicMock

class TestAuthUtils(unittest.TestCase):

    @patch('app.routes.auth.utils.User')
    def test_validate_login_success(self, mock_user):
        # Simuler un utilisateur existant
        mock_user_instance = MagicMock()
        mock_user_instance.password = generate_password_hash('password123')
        mock_user.query.filter_by().first.return_value = mock_user_instance

        user, error = validate_login('test@example.com', 'password123')
        self.assertIsNotNone(user)
        self.assertIsNone(error)
```

Ici par exemple nous testons la fonction validate_login utilisée pour authentifier les utilisateurs

```
(env) juste@me10:~/appython/recycy$ python -m unittest discover tests/
-----
Ran 1 test in 0.107s
OK
(env) juste@me10:~/appython/recycy$ 
```

Le résultat est positif et indique qu'un seul test a été exécuté avec succès.

```
class TestAuthUtils(unittest.TestCase):
    @patch('app.routes.auth.utils.User')
    def test_validate_login_success(self, mock_user):
        # Simuler un utilisateur existant
        mock_user_instance = MagicMock()
        mock_user_instance.password = generate_password_hash('password123')
        mock_user.query.filter_by().first.return_value = mock_user_instance

        user, error = validate_login('test@example.com', 'password123')
        self.assertIsNotNone(user)
        self.assertIsNone(error)

    @patch('app.routes.auth.utils.User')
    def test_validate_login_invalid_credentials(self, mock_user):
        # Simuler un utilisateur inexistant
        mock_user.query.filter_by().first.return_value = None

        user, error = validate_login('test@example.com', 'password123')
        self.assertIsNone(user)
        self.assertEqual(error, "Email ou mot de passe incorrect")
```

```
(env) juste@me10:~/appython/recycy$ python -m unittest discover tests
...
Ran 2 tests in 0.110s
OK
○ (env) juste@me10:~/appython/recycy$
```

Les deux tests sont passés avec succès

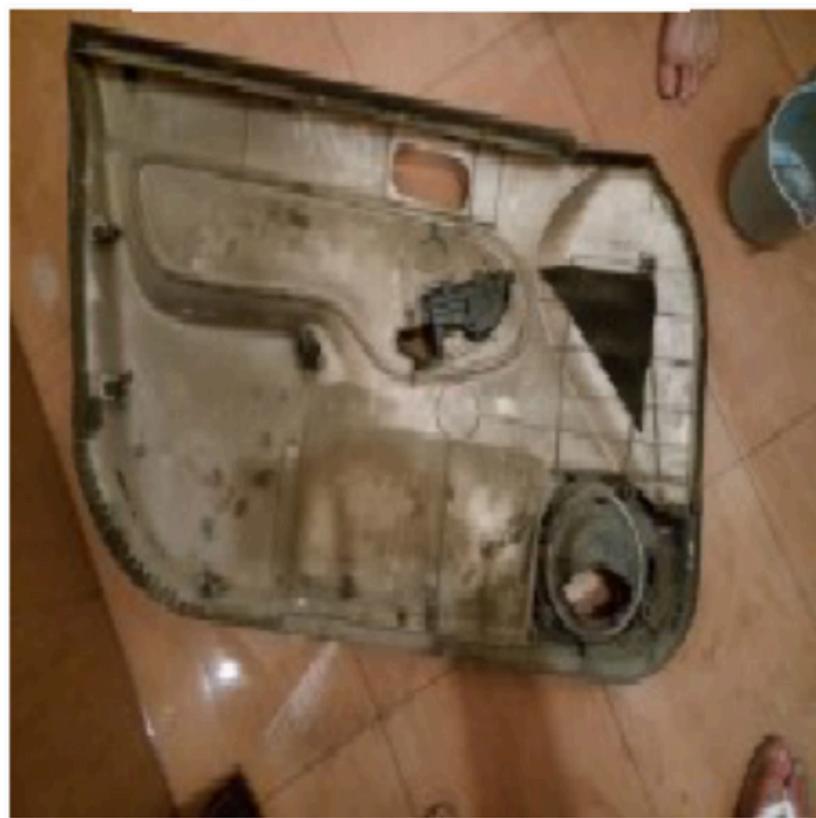
```
217 # Prédiction sur une nouvelle image
218 model = load_model('mobilenetv2_waste_classification.h5')
219 IMG_SIZE = (224, 224)
220 # Chemin de l'image à prédire
221 img_path = r"_____ .jpg"
222
223 # Prétraitement de l'image
224 img = image.load_img(img_path, target_size=IMG_SIZE)
225 img_array = image.img_to_array(img)
226 img_array = np.expand_dims(img_array, axis=0)
227 img_array = img_array / 255.0
228
229 # Prédiction
230 predictions = model.predict(img_array)
231 predicted_class_index = np.argmax(predictions, axis=1)[0]
232 class_labels = sorted(os.listdir(train_dir)) # Étiquettes des classes
233 predicted_class = class_labels[predicted_class_index]
234
235 # Affichage de la prédiction
236 print(f"Classe prédite : {predicted_class}")
237 plt.imshow(img)
238 plt.title(f"Prédiction : {predicted_class}")
239 plt.axis('off')
240 plt.show()
```

Ce test permet de vérifier la capacité du modèle MobileNetV2 à classifier correctement une image de déchet en la comparant aux classes disponibles dans le dataset d'entraînement. L'image est d'abord chargée et prétraitée (redimensionnée à 224x224 pixels, convertie en tableau de valeurs normalisées entre 0 et 1). Ensuite, elle est passée au modèle pour prédiction, et la classe avec la plus grande probabilité est sélectionnée grâce à argmax(). Enfin, la classe prédite est affichée avec l'image correspondante pour une validation visuelle. Ce test permet d'évaluer la précision du modèle sur des images réelles et de détecter d'éventuels cas de mauvaise classification, ce qui peut aider à affiner l'entraînement si nécessaire.

Classe prédicta: Papier



Classe prédicta: Automobile



Certains bugs ont été identifiés et résolus au cours du développement de l'application. Tout d'abord, un problème d'authentification avec les tokens JWT a été détecté : certaines requêtes échouaient malgré un token valide, en raison d'une configuration incorrecte de la durée de validité ou d'un mécanisme de renouvellement défaillant. nous l'avons résolu en ajustant la durée d'expiration des tokens et en améliorant la gestion des en-têtes de requête. Ensuite, un problème de classification d'images a été observé, où le modèle produisait des prédictions erronées pour certaines catégories de déchets. Pour y remédier, le modèle a été réentraîné avec un dataset plus diversifié et les hyperparamètres ont été optimisés pour augmenter la précision. Un autre bug concernait la latence dans l'envoi des notifications, particulièrement lors d'événements à grande échelle. Ce problème a été résolu en implémentant des queues et en améliorant la gestion des événements en temps réel avec Socket.IO. Enfin, un problème d'affichage des résultats des défis a été identifié, où certains utilisateurs ne voyaient pas leur progression mise à jour. La logique de mise à jour des défis a été corrigée pour garantir un rafraîchissement en temps réel des résultats. Ces résolutions ont permis d'améliorer la stabilité et l'expérience utilisateur de l'application.

i) Tests d'intégration (collectif)

Pour tester l'intégration de notre système, nous avons mis en place plusieurs stratégies afin de valider la communication entre les différents composants. Initialement, nous avons rencontré des difficultés pour héberger le serveur sur un environnement distant, empêchant ainsi mon binôme, Jeahusan, de tester l'API depuis son application React Native. Pour contourner ce problème, nous nous sommes retrouvés à la bibliothèque où j'ai connecté mon PC à Internet via un câble Ethernet, puis partagé ma connexion en local avec Ubuntu nmcli. Cela lui a permis d'accéder à mon serveur Flask et de tester l'API en conditions réelles. En complément, j'ai effectué des tests via Postman pour vérifier les endpoints individuellement, et **Swagger** a facilité l'envoi de requêtes directement depuis la documentation interactive. Ces tests nous ont permis d'identifier et de résoudre d'éventuels problèmes d'intégration, garantissant ainsi le bon fonctionnement de l'ensemble du système.

j) Discussion

Retour sur les choix effectués

Le choix de Flask, TensorFlow et OpenCV pour le développement du backend de l'application s'est avéré pertinent au regard des besoins du projet. Flask a permis de concevoir une API légère et rapide, facilitant l'interaction entre le front-end et le back-end. Son architecture simple a simplifié l'implémentation et l'intégration avec les autres technologies.

L'adoption de TensorFlow s'est imposée naturellement pour la classification des déchets. Grâce à ses capacités avancées en apprentissage profond, il a permis d'entraîner un modèle efficace et d'obtenir des prédictions précises sur les images téléchargées par les utilisateurs. Son support pour les réseaux neuronaux convolutifs (CNN) a été un atout majeur pour le projet.

Enfin, OpenCV a été essentiel pour la préparation des images avant leur soumission au modèle d'apprentissage. Ses fonctionnalités avancées ont facilité le redimensionnement, la correction des couleurs et la suppression du bruit, améliorant ainsi la qualité des données en entrée.

Limitations du logiciel actuel.

Malgré ces choix techniques pertinents, certaines limitations persistent et pourraient impacter l'efficacité et l'expérience utilisateur du système. Tout d'abord, en termes de performance, le modèle de classification d'images peut rencontrer des latences significatives lorsqu'il est exécuté sur un serveur dépourvu de GPU optimisé. Ces délais de traitement peuvent ralentir le temps de réponse, affectant ainsi la fluidité de l'expérience utilisateur. Ensuite, concernant la précision, bien que les résultats de classification soient globalement satisfaisants, des erreurs persistent, notamment dans la distinction entre des matériaux similaires, comme le plastique et le métal. Ces inexactitudes peuvent nuire à la fiabilité du système.

Par ailleurs, la scalabilité constitue une autre limite : Flask, bien que léger et facile à utiliser, n'est pas optimisé pour gérer des montées en charge importantes. Une migration vers un framework plus robuste, comme FastAPI, ou un déploiement avec des outils comme Gunicorn et Nginx pourrait être envisagée pour mieux gérer les requêtes simultanées et améliorer la stabilité du système. Enfin, la gestion des fichiers volumineux pose également des défis. Bien qu'OpenCV et Flask soient efficaces pour traiter des images, des fichiers de grande taille peuvent entraîner des erreurs de timeout ou une surcharge des ressources. L'intégration d'outils de traitement asynchrone, tels que Celery, pourrait permettre de mieux gérer ces cas et d'optimiser l'utilisation des ressources serveur. Ces limitations, bien que non bloquantes, méritent d'être prises en compte pour garantir une expérience utilisateur optimale et une scalabilité future du système.

Comparaison avec d'autres solutions existantes

Technologie	Avantages	Inconvénients
Django + TensorFlow	Framework robuste, gestion avancée de la sécurité et des bases de données	Plus lourd que Flask, configuration plus complexe
FastAPI + PyTorch	Excellentes performances, support natif de l'asynchronisme, alternative plus rapide à Flask	Moins de documentation et de support communautaire par rapport à Flask
Node.js + TensorFlow.js	Écosystème riche, idéal pour des applications temps réel	Moins performant pour le traitement d'images complexes
Flask + TensorFlow Lite	Optimisé pour les appareils mobiles et IoT	Moins performant pour des modèles complexes nécessitant de gros calculs

Conclusion et Perspectives

Bilan du projet

Le projet a permis de développer une application innovante de reconnaissance d'images pour le tri des déchets, répondant ainsi à un besoin crucial de gestion environnementale. Grâce à l'utilisation de technologies modernes telles que React Native, Tailwind CSS, TypeScript, et Axios, l'application offre une interface fluide et réactive, tout en assurant une robustesse et une flexibilité accrues. L'intégration de TensorFlow.js et OpenCV pour le traitement des images en temps réel a permis de fournir des résultats précis et fiables.

Axes d'amélioration et fonctionnalités futures

Optimisation des performances : Améliorer les performances de l'application sur les appareils bas de gamme en optimisant les algorithmes de traitement d'image et en offrant des options de configuration adaptées.

Gestion du stockage local : Mettre en place une stratégie de cache plus efficace pour réduire la consommation de bande passante et améliorer les performances globales de l'application.

Fonctionnalités supplémentaires : Ajouter des fonctionnalités telles que des notifications push pour rappeler aux utilisateurs de trier leurs déchets, ou des rapports analytiques pour les entreprises de recyclage afin de suivre et optimiser leurs processus de tri.

Annexes:

Code source : <https://github.com/Juste-medis/visionrec>

Bibliographie, sitographie et références

Sources, articles, livres, et IA : Pour la réalisation de ce projet, diverses sources académiques, articles, livres, et ressources en intelligence artificielle ont été consultés afin de garantir une compréhension approfondie des concepts et des technologies utilisées.

Outils et technologies utilisés : Décrire les outils, frameworks, et technologies utilisés dans le développement de l'application, tels que React Native, Tailwind CSS, TypeScript, Axios, TensorFlow.js, et OpenCV.

En conclusion, ce projet a non seulement permis de développer une solution technique innovante pour le tri des déchets, mais il a également mis en lumière l'importance de la collaboration et de l'intégration continue dans le développement logiciel. Les perspectives d'amélioration ouvrent la voie à des fonctionnalités encore plus avancées et à une meilleure accessibilité pour tous les utilisateurs.

