

Zell Renaud, Ducret Thomas, Lefebvre Steven, Maaroufi Julien

Compte rendu S5.01



SAE 5.01

Sommaire

Sommaire.....	3
Introduction.....	5
Gestion de projet.....	6
Trello.....	6
Discord.....	6
GitHub.....	7
Docker.....	7
Intelligence Artificielle.....	8
Pourquoi Yolov8 ?.....	8
Le dataset.....	8
L'annotation des images.....	8
L'entraînement.....	10
Application Mobile.....	13
Environnement.....	13
Vue.....	14
Login.....	14
Inscription.....	15
Accueil.....	16
Déconnexion.....	16
Info.....	17
Receive Data.....	18
Caméra.....	19
Display Photo.....	20
History.....	22
API & Back-Office.....	23
Environnement.....	23
Hébergement.....	24
Base de données.....	25
Back-Office.....	26
Le design.....	26
Accessibilité et connexion.....	26
Back-office.....	27
Requête et utilisation de l'API.....	28
JWT Token.....	28
Vérification de l'utilisateur via JWT.....	30
Requêtes disponibles.....	31
Test de l'API via Postman.....	33
Difficultés rencontrées.....	36
Application Mobile : Flutter Vision.....	36
Application Mobile : Gestion des encadrements d'objets.....	36
Application Mobile : Precision du modele.....	37
De longues étapes.....	38

API : Identification d'un utilisateur via le token.....	38
API : Gestion de l'expiration des tokens.....	39
Améliorations Possibles.....	40
Application Mobile : Gestions des modèles IA.....	40
Application Mobile : Administrateur.....	40
API : Routes API.....	41

Introduction

Dans un monde où les technologies de l'intelligence artificielle et de la vision par ordinateur évoluent rapidement, leur intégration dans les appareils mobiles offre des opportunités fascinantes. Ces avancées permettent non seulement de résoudre des problématiques complexes mais aussi d'améliorer considérablement l'expérience utilisateur dans des domaines variés tels que la reconnaissance d'objets, la réalité augmentée ou encore la navigation autonome.

Ce rapport présente le développement d'une application mobile capable de reconnaître et de classer des objets en temps réel, en s'appuyant sur des techniques modernes de vision par ordinateur et d'apprentissage profond. Ce projet, réalisé dans le cadre de la SAE 501, combine plusieurs axes technologiques majeurs : développement mobile, déploiement de modèles d'intelligence artificielle, et conception d'interfaces utilisateurs intuitives.

Pour répondre aux exigences de performance et de précision, nous avons utilisé Flutter pour le développement de l'application et Android Studio comme IDE. Le modèle de reconnaissance d'objets repose sur YOLO v8, optimisé et entraîné avec l'aide de la plateforme RoboFlow pour garantir des résultats performants sur des images en temps réel.

La partie serveur et API a été conçue avec Symfony, permettant de gérer les interactions entre l'application et les bases de données. Des tests approfondis ont été menés à l'aide de Postman, et l'API a été déployée sur un serveur dédié acquis spécifiquement pour ce projet.

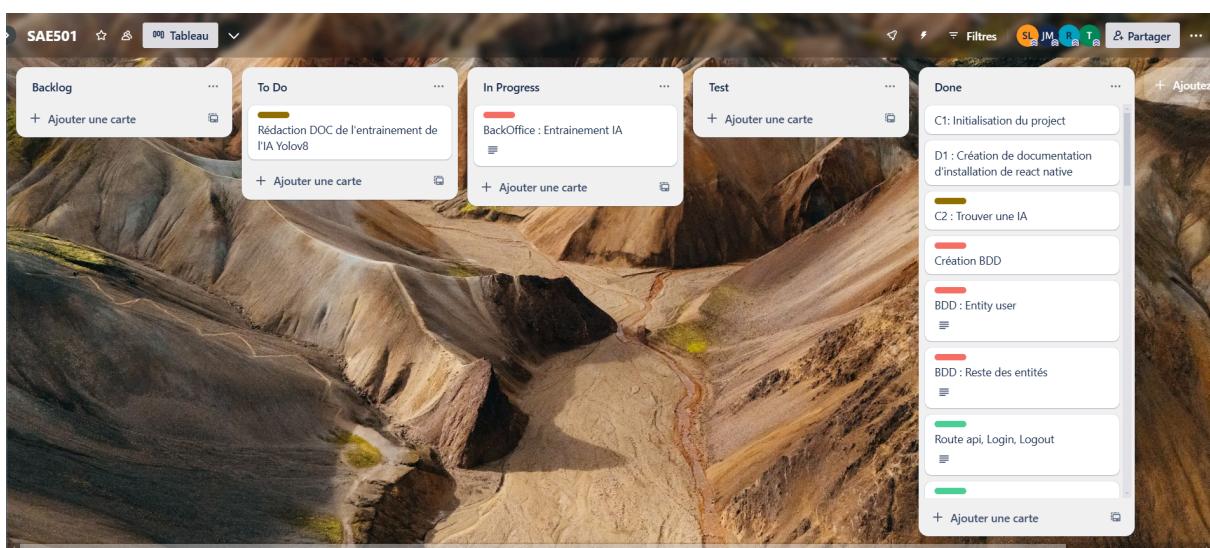
Ce document décrit en détail l'architecture technique de notre solution, les choix méthodologiques réalisés, les défis rencontrés, ainsi que les optimisations apportées pour assurer la fluidité et la performance de l'application. Enfin, une réflexion sur les perspectives d'évolution du projet et ses applications potentielles conclut ce rapport.

Gestion de projet

Afin de travailler efficacement, nous avons utilisé plusieurs applications de gestion de projet et d'outils collaboratifs.

Trello

Nous avons utilisé Trello une application web, pour organiser les tâches à réaliser sous forme de tickets et les trier en fonction de leur avancement. Ainsi, chacun pouvait s'attribuer une tâche et la déplacer selon son état d'avancement. Il suffisait donc aux membres de l'équipe de se connecter pour connaître les tâches restantes.



Discord

Nous avons également utilisé Discord, une application de discussion en ligne. Celle-ci permet de créer gratuitement des serveurs de discussion où différents canaux vocaux ou textuels peuvent être ajoutés.

Grâce à ce serveur, nous avons pu organiser et trier des ressources, des liens, des fichiers, des vidéos, et d'autres documents, permettant à chaque collaborateur de les consulter facilement.

GitHub

GitHub a joué un rôle central dans la gestion et le suivi de notre projet. Il nous a permis de collaborer efficacement en centralisant le code source, les ressources, et l'historique des modifications.

Ainsi, lors de modifications du code, les utilisateurs pouvaient accéder rapidement à la dernière version du projet, incluant l'API, l'application mobile, et leurs environnements respectifs.

Couplé avec Docker, la mise en place du projet pour chaque collaborateur a été rapide et efficace.

Docker

Docker permet la mise en place rapide d'un environnement de travail.

Il facilite la création de conteneurs à partir d'images disponibles en ligne ou générées localement à l'aide d'un fichier Docker. Ces conteneurs peuvent embarquer divers outils et applications, tels qu'un compilateur, une base de données, un framework, un serveur web, et bien plus encore.

En publiant les fichiers Docker dans un dépôt Git, toute personne disposant de Docker installé pourra créer facilement un environnement de travail identique. Cette solution élimine le besoin d'installer manuellement un serveur Apache, une base de données, ou un framework, et dispense même de la compilation du code pour exécuter l'application déployée.

Ainsi en utilisant Docker nous facilitons l'exportation du projet mais surtout la collaboration au sein de l'équipe.

Intelligence Artificielle

Pourquoi Yolov8 ?

Après recherche, nous nous sommes mis d'accord pour utiliser le modèle Yolo, il répond parfaitement à nos critères et est aussi très populaire, ce qui nous a encore plus renforcé dans ce choix étant donné qu'en plus de la documentation, nous aurions sûrement accès à d'autres ressources provenant d'autres utilisateurs. Des alternatives comme Faster R-CNN, EfficientDet, SSD existent mais viennent toujours au prix d'un aspect de la détection, soit la vitesse de détection, la consommation de ressources ou une faiblesse dans soit la détection en temps réel ou avec une image. Avec tout ça en tête, nous avons décidé de commencer notre projet avec Yolo comme algorithme pour entraîner un modèle d'IA avec notre futur jeu de données, puisqu'il est versatile et populaire.

Le dataset

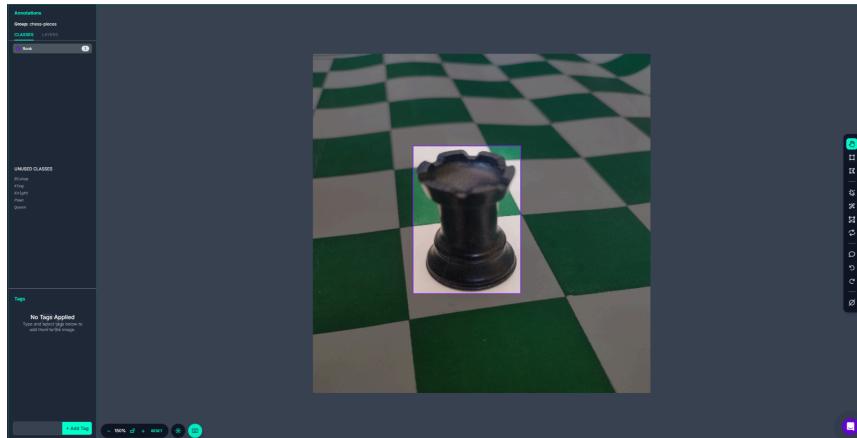
Après avoir trouvé notre algorithme d'entraînement, il nous fallait maintenant trouver un jeu de données qui tourne autour des pièces d'échecs. Pour ce faire, nous avons dans un premier temps utilisé le site kaggle pour trouver un dataset de 300 images, qu'il nous fallait maintenant annoter.

L'annotation des images

L'annotation des images, ça désigne le fait de marquer quels éléments se trouve où sur les images. Pour faire ça, nous avons hésité entre une librairie python appelée labellmg, ou Roboflow, une plateforme qui nous permet, entre autres, d'annoter nos images en répartissant les tâches. Bien que labellmg nous a paru bien comme premier choix étant donné que ça nous aurait permis de tout avoir dans notre IDE, nous avons fini par choisir Roboflow après avoir pesé le pour et le contre. Roboflow nous permettra au besoin de par exemple se partager la tâche qu'est l'annotation des images si dans le futur on doit annoter une quantité plus importante d'images.

On va tout d'abord créer autant de classes que d'objets que l'on veut reconnaître. Dans notre cas de jeu d'échec, on a décidé de reconnaître les pièces et leurs couleurs, ce qui nous fait 12 classes au total.

Voilà comment se passe l'annotation : on encadre ce que l'on veut que l'IA reconnaissse et on y assigne une classe, dans ce cas “black-rook” (pour une tour noir) :



```
4.jpg.rf.c349c1732a26b671a16b1085211a2da2.txt × ② data  
1 9 0.31875 0.60390625 0.05625 0.109375  
2 9 0.3515625 0.375 0.0390625 0.0859375  
3 9 0.453125 0.2640625 0.034375 0.078125  
4 9 0.31484375 0.30234375 0.040625 0.075  
5 9 0.36796875 0.2234375 0.0359375 0.0703125  
6 9 0.3828125 0.17890625 0.0328125 0.075  
7 9 0.33359375 0.25859375 0.0328125 0.0796875  
8 11 0.11796875 0.4984375 0.05625 0.10625  
9 11 0.31328125 0.1609375 0.0234375 0.06875  
10 8 0.30234375 0.19375 0.03125 0.0921875  
11 6 0.27890625 0.2328125 0.0234375 0.109375  
12 4 0.21015625 0.3015625 0.05 0.1671875  
13 2 0.25546875 0.240625 0.04375 0.1640625  
14 3 0.61484375 0.6875 0.05 0.0953125  
15 3 0.5390625 0.4890625 0.0421875 0.090625  
16 3 0.5609375 0.37421875 0.0265625 0.08125  
17 3 0.521875 0.30546875 0.0265625 0.071875  
18 3 0.534375 0.22578125 0.028125 0.0609375  
19 3 0.63359375 0.584375 0.046875 0.096875  
20 8 0.51875 0.63203125 0.05625 0.1296875  
21 8 0.65625 0.27890625 0.046875 0.1078125  
22 5 0.6078125 0.2328125 0.0390625 0.0859375  
23 5 0.7203125 0.6890625 0.059375 0.1296875  
24 6 0.725 0.5046875 0.0515625 0.1375  
25 6 0.71015625 0.3609375 0.046875 0.1109375  
26 2 0.72265625 0.4203125 0.0734375 0.159375v
```

Une fois toutes les annotations terminées, on peut depuis Roboflow transformer les données dans un format adapté à YoloV8. On peut voir sur la capture de gauche un fichier label lié à une image, sur chaque ligne on a : un numéro qui représente l'index de la classe de la pièce qu'on a entouré et les 4 autres chiffres sont sa position, il faut imaginer les 4 sommets du carré, qui dessine alors une boîte. On peut ensuite les télécharger dans un projet à part pour entraîner l'IA.

L'entraînement

Une fois que toutes nos images sont annotées, on peut les séparer en 3 dossiers. Un dossier Test, Train et un Valid. Contenant chacun deux sous-dossiers, un dossier image et un dossier label qui contient le fichier texte associé à chaque image (cf capture gauche).

```
from ultralytics import YOLO
model = YOLO("yolov8m.pt")
# Train the model on the COCO8 example dataset for 100 epochs
results = model.train(data="data.yaml", epochs=100, imgsz=640)
```

Le dossier Train contient 70% des images de notre dataset et il permet d'entraîner le modèle.

Le dossier Valid contient 20% des images et permet de voir comment l'IA réagit à des données qu'elle n'a jamais vues.

Et enfin le dossier Test contient 10% des images et il permet d'évaluer les performances après que l'entraînement soit fini.

Sur la capture ci-dessous, on peut y retrouver le code permettant de lancer un entraînement de notre IA avec notre dataset. Dans la 6eme ligne (les mots en rouge sont ce qu'on appelle des hyperparamètres dans ce domaine, ils sont configurés manuellement) on renseigne notre data.yaml qui lui donne le chemin de notre jeu de données. Le voici :

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 12
names: ['black-bishop', 'black-king', 'black-knight', 'black-pawn',
        'black-queen', 'black-rook', 'white-bishop', 'white-king',
        'white-knight', 'white-pawn', 'white-queen', 'white-rook']

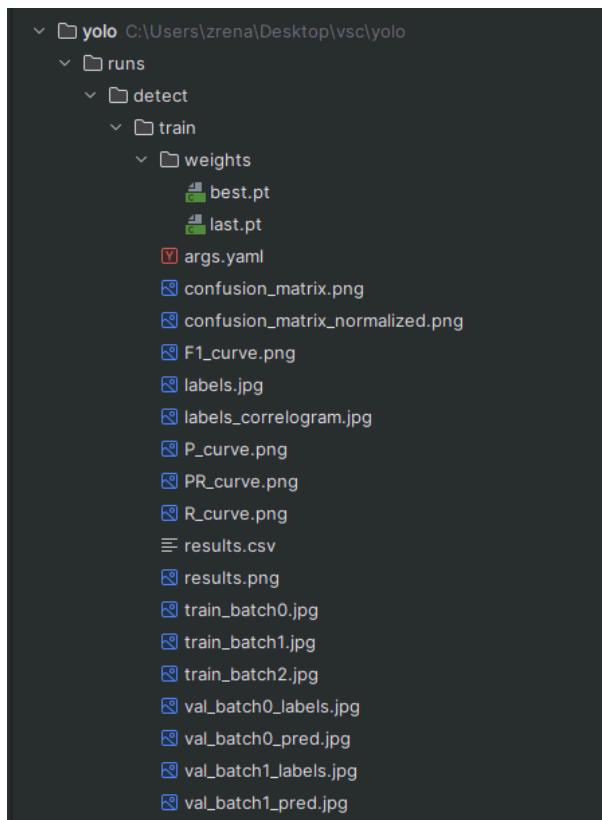
roboflow:
    workspace: university-of-the-philippines-yieax
    project: sp2-ym1iq
    version: 1
    license: CC BY 4.0
    url: https://universe.roboflow.com/university-of-the-philippines-yieax/sp2-ym1iq/dataset/1
```

L'hyperparamètre epoch représente le nombre de fois que le modèle sera entraîné sur notre dataset. Par exemple, si on renseigne 50 epochs, l'IA fera 50 itérations d'entraînement sur notre dataset. Il est recommandé d'augmenter le nombre d'epochs en fonction de la taille de notre dataset.

Entre 10 et 50 pour les petits datasets, 50 à 200 pour les moyens, et plus de 200 pour les plus importants. Il est bon de le savoir mais pas nécessaire, étant donné que maintenant il existe une fonctionnalité appelée ‘early stopping’ qui surveille le résultat de l’IA après chaque epoch et se charge d’arrêter l’entraînement dès lors qu’aucune amélioration significative n’est observée au cours des 100 derniers epochs(on a pris arbitrairement 100, on peut changer cette valeur et elle n’est d’ailleurs pas la meilleure).

Cela signifie que par exemple on pourrait renseigner 700 epochs, et que si aucune amélioration n’est observée après la 112ème epoch, cette fonctionnalité fera en sorte que l’entraînement s’arrête à l’epoch numéro 212.

A noter qu’il est quand même possible que le modèle sur apprenne même avec l’early stopping s’il est mal paramétré, c’est pourquoi il est recommandé d’être attentif aux résultats et de tester plusieurs valeurs d’epoch.



Sur cette capture, on observe les fichiers générés après l’entraînement de l’IA. Beaucoup de fichiers sont utiles pour l’analyse de données et voir comment s’est passé l’entraînement. Le fichier qui nous intéresse pour implémenter l’IA dans notre application mobile est le fichier best.pt. Il n’est pas compatible mobile comme tel, il faut alors le convertir en fichier .tflite avec ces lignes de code :

```
1 from ultralytics import YOLO  
2  
3 model = YOLO("best.pt")  
4  
5 model.export(format="tflite")
```

Avec ça, on peut maintenant utiliser notre IA entraînée sur notre propre jeu de données dans notre application mobile.

Application Mobile

L'application mobile a pour objectif principal d'utiliser un modèle d'IA entraîné pour détecter des objets en temps réel ou sur des photos. Elle propose également d'autres fonctionnalités, telles que la connexion et l'inscription des utilisateurs, le partage de photos avec les objets détectés, la consultation de l'historique des détections de l'utilisateur ainsi que de celles des autres utilisateurs, et enfin, le téléchargement du dernier modèle d'IA disponible sur le serveur.

Environnement

Pour cette application mobile, nous avions initialement choisi d'utiliser React Native. Cependant, en raison de la complexité du langage, nous avons décidé d'opter pour le framework Flutter, qui utilise le langage Dart.

Pour le développement avec Flutter, nous utilisons l'IDE Android Studio, spécialement conçu pour le développement d'applications Android. Cet IDE facilite le développement, la gestion des dépendances, ainsi que l'utilisation d'émulateurs. Ainsi, il est possible d'héberger directement un émulateur pour tester notre application ou de connecter un téléphone à notre machine afin de l'utiliser comme appareil de test.

Ainsi pour la majorité des tests nous utilisons cette émulateur :



Cela est directement visible dans l'IDE. Cependant, nous utiliserons également un téléphone pour tester la caméra et la détection d'objets avec l'IA.

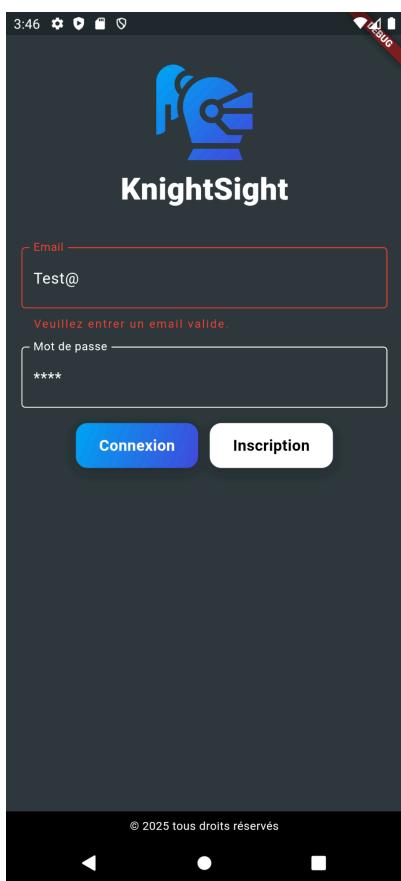
Pour ce faire, il suffit d'avoir un téléphone connecté au même réseau Wi-Fi que la machine de développement, de le configurer en mode développeur et d'activer le débogage sans fil. Ensuite, grâce à une association par QR code ou un code d'appairage, nous pourrons compiler et envoyer directement un fichier APK sur le téléphone afin de simuler l'application, comme illustré ici :



Vue

Login

La page de connexion (Login) est la première page visible de l'application mobile. Elle permet à un utilisateur de se connecter à son compte via un formulaire de connexion, de naviguer vers le formulaire d'inscription, et propose également une connexion automatique si un token de connexion est stocké sur le téléphone de l'utilisateur.



Si un token de connexion est présent, l'application vérifie sa validité avant d'afficher la vue. Pour ce faire, le token est envoyé à l'API, qui indique s'il est toujours valide. Si le token est valide, l'utilisateur est redirigé vers la page suivante. Dans le cas contraire, il reste sur la page de connexion.

Pour la vue de connexion (Login), nous avons opté pour un design simple et minimaliste, que l'on retrouvera de manière cohérente dans toute l'application.

Cette vue est composée d'un logo représentant un chevalier, suivi du titre de l'application, nommé "KnightSight", puis du contenu principal de la page.

Sur la page de connexion, nous trouvons les champs Email et Mot de passe, accompagnés des boutons Connexion et Inscription.

Chaque champ utilise des validateurs spécifiques : l'un pour les adresses email et l'autre pour les mots de passe. Ainsi, avant l'envoi des informations via le bouton Connexion, l'application vérifie leur format et informe l'utilisateur si celui-ci est incorrect.

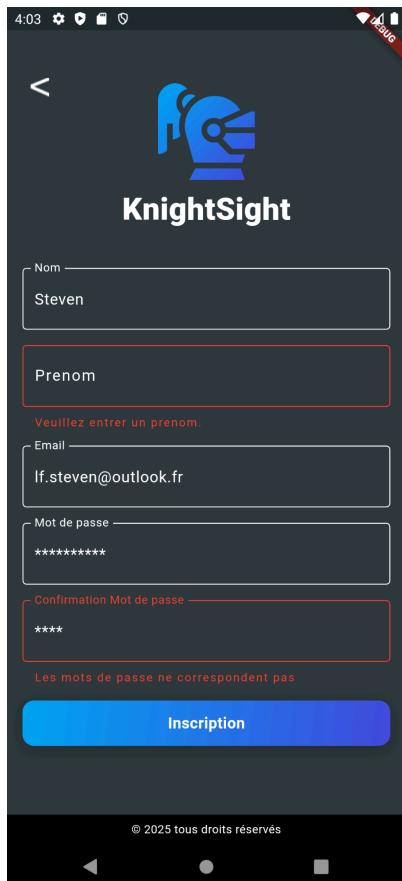
Si les informations sont correctes, elles sont envoyées à l'API, qui répond soit par une erreur de connexion, soit par un token de connexion.

En cas d'erreur, l'application attend de nouvelles informations. Si aucune erreur n'est détectée, elle stocke le token et redirige l'utilisateur vers la page suivante.

Par ailleurs, le bouton Inscription redirige l'utilisateur vers un autre formulaire, lui permettant de créer un compte pour accéder à l'application.

Inscription

La page d'inscription permet à l'utilisateur de se créer un compte dans l'application. Elle demande plusieurs informations qui sont utilisées par l'API et stockées dans sa base de données.



Sur la page d'inscription, une flèche apparaît pour la première fois en haut à gauche. Cette flèche permet de revenir à la page précédente et est utilisée sur plusieurs vues de l'application.

En ce qui concerne le contenu de la page, il y a plusieurs champs de saisie pour le prénom, le nom, l'email, le mot de passe et la confirmation du mot de passe. Des validateurs sont également appliqués à ces champs.

Les champs "Nom" et "Prénom" utilisent les mêmes validateurs, tandis que "Email" et "Mot de passe" réutilisent ceux de la page précédente.

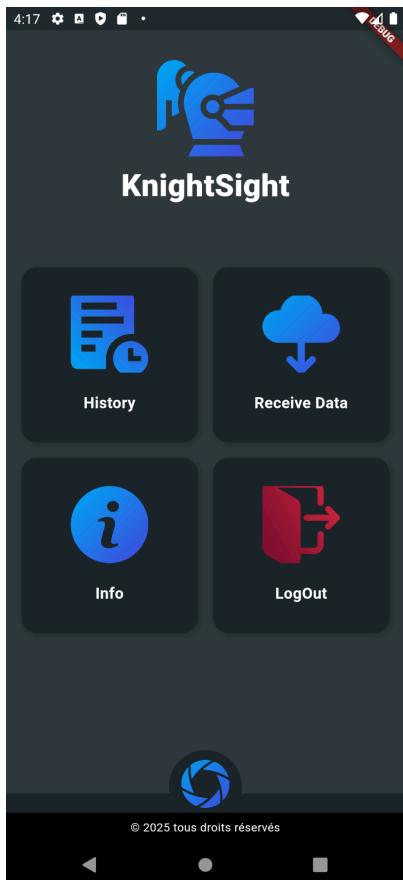
Le validateur de la confirmation du mot de passe vérifie que la valeur saisie dans le champ de confirmation est exactement identique à celle du champ "Mot de passe".

Quant au bouton "Inscription", il envoie toutes les informations à l'API. En cas de validation, l'utilisateur est redirigé vers la page de connexion. En cas d'erreur, aucun changement n'est effectué, à part l'affichage des messages de validation.

Sur cette page, il a été nécessaire de mettre en place une solution pour gérer l'apparition du clavier de l'utilisateur. En effet, lorsque le clavier apparaît, il peut cacher les champs de saisie situés en bas de l'écran. Pour résoudre ce problème, nous avons implémenté un redimensionnement dynamique de la page lors de l'ouverture du clavier, garantissant ainsi que les champs de saisie qui ont le focus soient toujours visibles.

Accueil

La page d'accueil est celle sur laquelle l'utilisateur est redirigé après sa connexion à l'application. À partir de cette page, il peut accéder aux fonctionnalités de l'application, à l'exception de la connexion et de l'inscription. De plus, il s'agit de la première page protégée par un token. Dès que l'utilisateur accède à la page, si celui-ci ne possède pas de token valide ou si celui-ci n'est pas validé par l'API, il est redirigé vers la page de connexion (Login).



Sur cette page, l'en-tête, comprenant le titre de l'application, reste visible. Le contenu principal présente quatre boutons permettant de déplacer l'utilisateur vers différentes pages :

- History
- Receive Data
- Info
- Logout

Nous trouvons également un bouton en bas de la page, juste au-dessus du pied de page, qui permet d'ouvrir la caméra du téléphone tout en utilisant le modèle d'IA (Yolov8) pour détecter des objets.

Déconnexion

Le bouton en rouge Logout dans la page Accueil est un bouton permettant à l'utilisateur de se déconnecter de l'application, il va simplement supprimer le token du stockage local du téléphone et déplacer l'utilisateur sur la page de Login.

Info

La page Info est une petite page dédiée à l'affichage de plusieurs informations sur l'application.



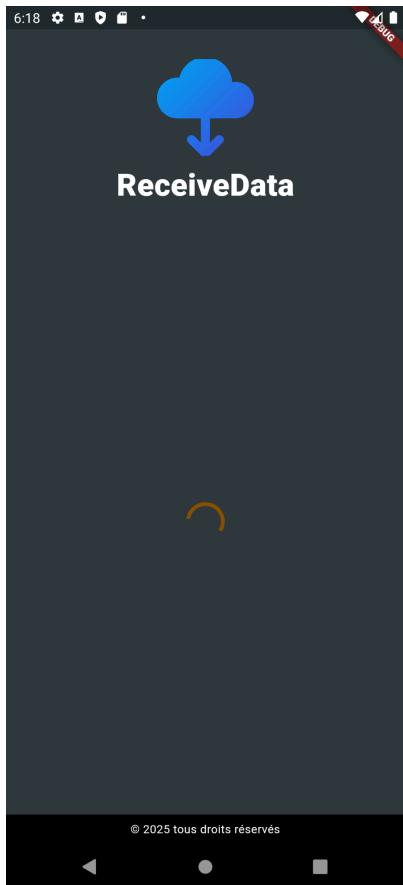
Elle comporte une flèche de retour permettant à l'utilisateur de revenir à la page d'accueil. Pour l'en-tête, l'image et le titre sont modifiés afin de correspondre au thème de la page.

Le contenu de la page présente une liste de paragraphes, chacun affiché sur un fond blanc, avec un titre et un texte descriptif.

En ce qui concerne le pied de page (footer), il contient un bouton permettant d'accéder à la caméra, ainsi qu'un bouton de déconnexion. Ce dernier fonctionne selon la même logique que celui de la page d'accueil.

Receive Data

La vue Receive Data permet à l'utilisateur de télécharger le dernier modèle d'IA disponible sur le serveur.



Elle commence par vérifier, via un appel à l'API, si une version plus récente que celle installée dans les fichiers de l'application est disponible sur le serveur.

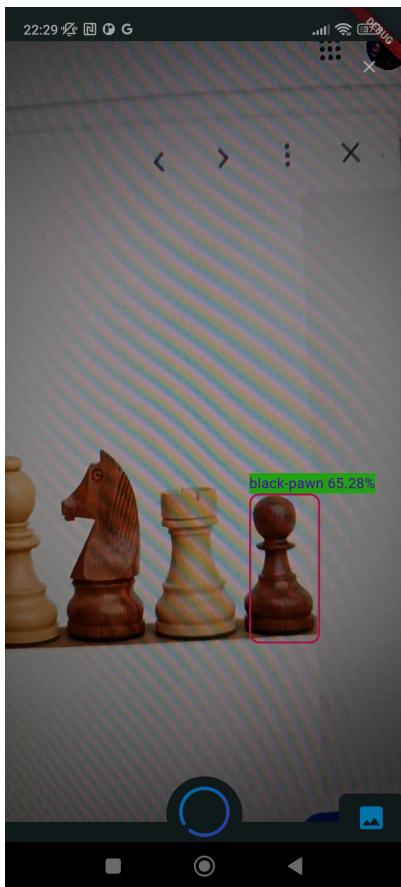
Si la version est à jour, l'utilisateur est redirigé vers la page d'accueil.

Dans le cas contraire, un nouveau modèle d'IA est téléchargé dans les fichiers de l'application, remplaçant le précédent. Le fichier stockant la version du modèle est mis à jour, et un appel à l'API est effectué pour recevoir la liste des objets pouvant être détectés. Ces derniers sont ensuite enregistrés dans un fichier label.txt.

Lorsque le téléchargement est terminé, l'utilisateur est redirigé vers la page d'accueil. En accédant par la suite à la page Caméra, il pourra utiliser le nouveau modèle avec les labels mis à jour.

Caméra

La caméra est accessible depuis plusieurs pages à l'aide du bouton situé au-dessus du pied de page, notamment sur la page d'accueil.



La page "Caméra" permet la détection d'objets via le flux vidéo. Elle utilise un modèle d'IA situé dans les dossiers locaux de l'application. Pour exploiter ce modèle, nous utilisons la dépendance flutter vision.

Cependant, cette dépendance ne permet que l'utilisation des modèles présents directement dans le dossier assets. Il a donc fallu forkler le dépôt Git et modifier directement la dépendance pour permettre le chargement et la fermeture des modèles YOLO situés dans les dossiers internes de l'application.

Sur la vue de la caméra, plusieurs éléments sont visibles :

- Le flux vidéo, qui occupe la majeure partie de l'écran.
- Une croix en haut à droite pour quitter la caméra.
- Un bouton circulaire permettant de prendre une photo.
- Un bouton situé en bas à droite.

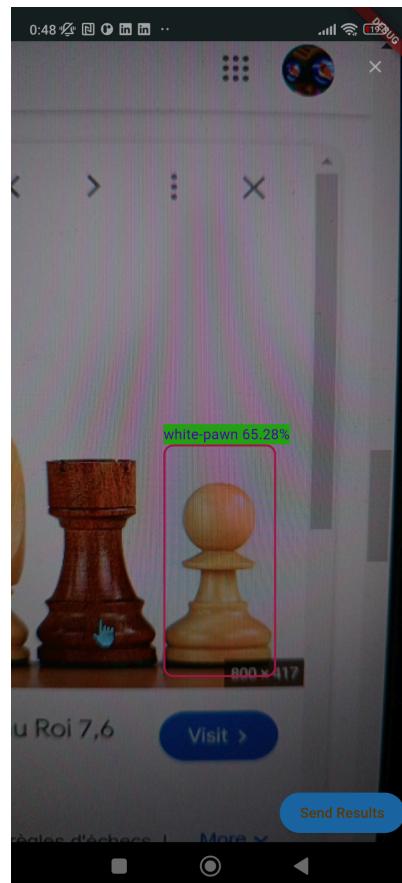
Le bouton en bas à droite permet de détecter des objets directement à partir de photos de la galerie. L'utilisateur peut simplement sélectionner une image depuis la galerie de son téléphone.

Enfin, sur l'écran de la caméra, des rectangles peuvent apparaître autour des objets détectés, avec le nom des labels et le pourcentage de certitude de reconnaissance affiché.

Les résultats des détections dans les photos sont affichés dans une nouvelle page appelée `Display_photo`.

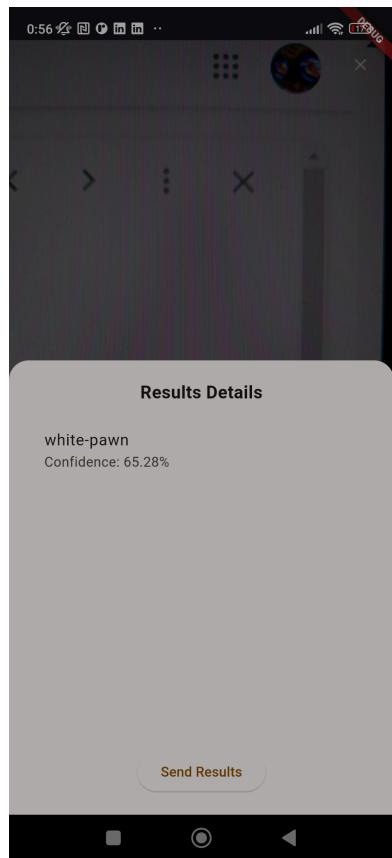
Display Photo

La vue Display Photo permet d'afficher des résultats de détections, à l'aide d'une image et les résultats de l'IA.



Dans cette vue, une croix en haut à droite permet de quitter la vue et de revenir à la précédente.

De plus, en cas de détection d'une ou plusieurs reconnaissances, un bouton permet d'ouvrir une fenêtre modale affichant les détails des détections.

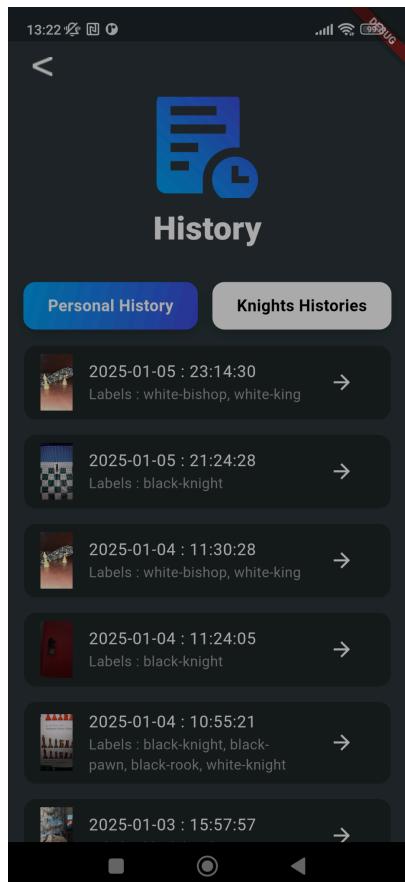


Dans cette fenêtre modale, l'utilisateur peut envoyer l'image ainsi que les encadrements à l'API.

Ces images pourront ensuite être utilisées pour l'historique des utilisateurs et pour l'entraînement d'une IA basée sur les résultats de la communauté.

History

La page “History” permet à un utilisateur de consulter son historique de reconnaissance d’objets ainsi que les dernières reconnaissances effectuées par la communauté. Il peut également visualiser ces reconnaissances en plein écran.



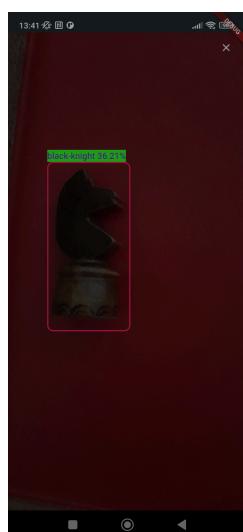
Sur cette vue, une flèche de retour et un en-tête personnalisé sont également disponibles.

Le contenu de la page comporte deux boutons. Le bouton de gauche permet de consulter l'historique de l'utilisateur. Chaque entrée de l'historique se compose d'une image, d'un titre accompagné d'une date et d'une heure, ainsi que d'une section de labels affichant les noms des objets détectés.

Chaque section inclut également une flèche permettant d'afficher l'image en grand format, accompagnée des encadrements correspondants.

Pour obtenir les informations des encadrements, l'application utilise le lien du fichier .txt renvoyé par l'API correspondant à l'image. Ce fichier permet de reconstruire les résultats fournis par le modèle API en respectant le même format.

Ainsi, nous pouvons réutiliser la vue `Display_photo`, qui est programmée pour fonctionner avec ce format. Toutefois, il a été nécessaire d'implémenter un booléen afin de désactiver le bouton `Send Result` lorsque la source de l'image provient de l'historique. Cela permet d'éviter les duplications d'images sur le serveur.



API & Back-Office

L'API développée dans ce projet joue un rôle essentiel en permettant à l'application mobile d'interagir avec les données stockées sur le serveur. Elle permet tout d'abord à l'application d'accéder aux informations partagées par les autres utilisateurs, facilitant ainsi l'échange de résultats d'analyses d'objets. Ensuite, elle gère le stockage des données collectées, telles que les images et les résultats des reconnaissances, et permet leur consultation ultérieure.

Enfin, elle garantit la sécurité de l'application en contrôlant l'accès aux données sensibles. Elle implémente des mécanismes de validation des utilisateurs et de protection des informations, assurant ainsi que seules les actions autorisées peuvent être effectuées et que les données restent protégées.

Ainsi, l'API est un élément clé pour l'échange de données, le stockage sécurisé et la gestion des droits d'accès, garantissant une expérience fluide et sécurisée pour l'utilisateur.

Environnement

Pour l'API, nous avons choisi le framework PHP Symfony. Symfony est un framework de développement web côté serveur, puissant et basé sur le langage PHP. Il est rapide à mettre en place et offre aux développeurs une large gamme de Bundles (dépendances) très utiles.

Grâce à sa vaste communauté et au soutien de celle de PHP, Symfony bénéficie d'une documentation riche et détaillée. Cela nous a permis d'accélérer considérablement le développement de l'API.

Pour la gestion de la base de données, Symfony intègre le bundle Doctrine, qui simplifie la gestion de la base via des commandes en ligne. Doctrine fournit également DBAL (Powerful DataBase Abstraction Layer), un outil performant qui traduit le code PHP en requêtes SQL, rendant le travail avec la base de données plus efficace et intuitif.

Comme mentionné précédemment, nous avons également utilisé Docker. Avec Docker, nous avons containerisé : Symfony/PHP, MySQL (pour la gestion de la base de données) et Nginx (serveur HTTP web). Ainsi, l'ensemble de l'environnement web de notre projet est dockerisé, ce qui permet de le rendre facilement exportable et modifiable.

Hébergement

Afin de permettre à l'application mobile de réaliser facilement des appels à l'API, nous avons choisi de l'héberger sur un serveur web, la rendant ainsi accessible sur Internet.

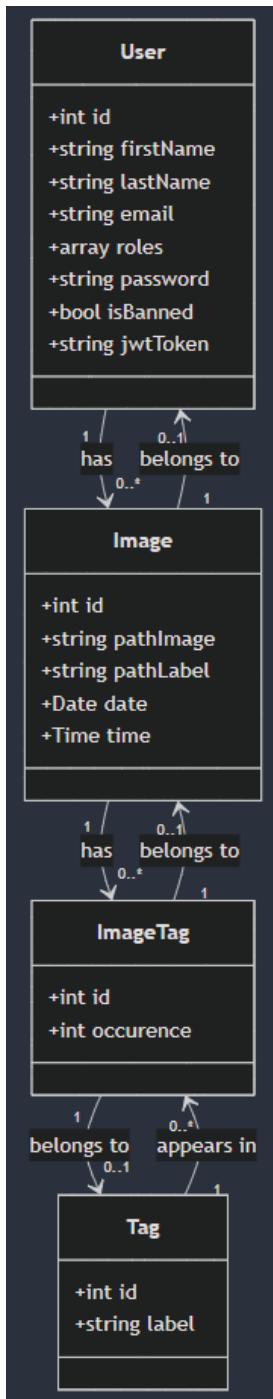
Initialement, nous avions envisagé d'utiliser une machine virtuelle fournie par l'IUT pour l'hébergement, mais nous avons préféré opter pour cette solution afin d'éviter la nécessité d'utiliser un VPN pour accéder à l'API.

Notre API est donc actuellement hébergée sur un VPS Linux de IONOS. Nous utilisons également WinSCP pour faciliter l'exportation et l'importation de fichiers avec le VPS. De plus, pour accéder facilement au terminal de ce VPS, nous utilisons une clé SSH, ce qui permet une connexion simple et sécurisée.

La mise en place de l'API sur ce serveur a été rapide. Il nous a suffi d'installer Docker et de lancer nos fichiers Docker. L'API est désormais disponible à l'adresse suivante : <http://212.227.57.57:8081/>.

Base de données

La conception de la base de données pour cette application repose sur des choix stratégiques visant à répondre aux besoins fonctionnels et techniques d'une API dédiée à l'analyse d'images via l'intelligence artificielle. Chaque entité, ainsi que ses relations, a été pensée pour maximiser l'efficacité, la maintenabilité et l'évolutivité de l'application.



L'entité user centralise les informations des utilisateurs, avec un email unique comme identifiant et des rôles personnalisables stockés sous forme de tableau pour le potentiel futur ajout de rôles. Le password est hashé avant d'être stocké avec la méthode BCrypt, il contient donc le salt dans le hachage. L'attribut isBanned permet de gérer les utilisateurs bloqués, et le champ jwtToken garantit une authentification sécurisée.

L'entité image représente les fichiers analysés, avec une relation Many-to-One liant chaque image à un utilisateur, permettant un suivi personnalisé. Les champs pathImage et pathLabel stockent les chemins des fichiers, et les attributs date et time permettent de suivre les moments de téléchargement, facilitant l'analyse et les statistiques.

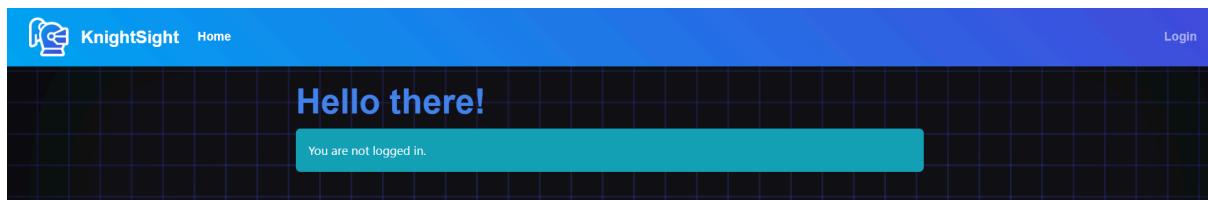
L'entité tag centralise les objets détectés dans les images sous forme de tags uniques, évitant les redondances. La table imageTag gère la relation Many-to-Many entre images et tags, avec un champ occurrence enrichissant les résultats IA. Cette structure permet une gestion flexible et évolutive des données liées aux objets détectés.

Les relations bidirectionnelles, comme entre user et image ou image et imageTag, permettent une navigation fluide. Les contraintes d'intégrité référentielle comme onDelete: CASCADE, assurent la cohérence des données et simplifient la suppression.

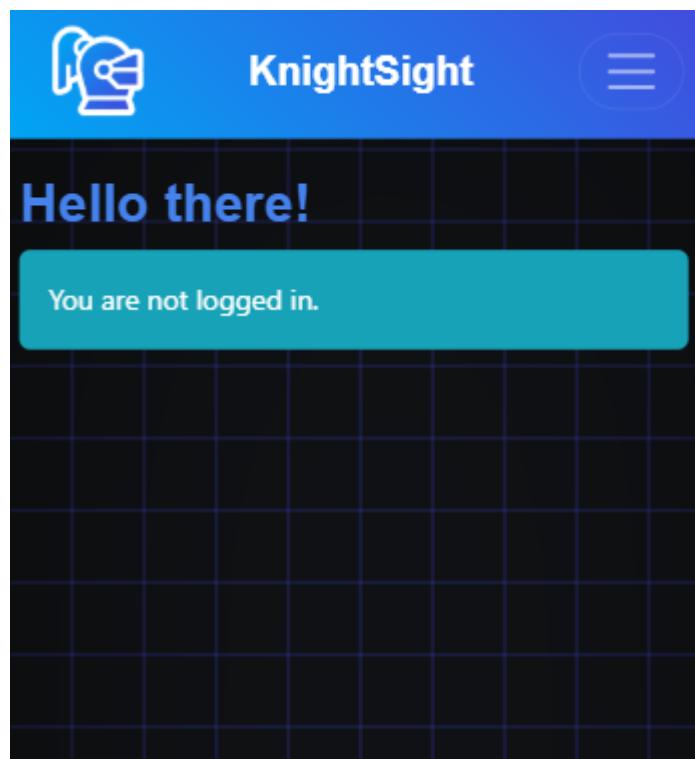
Back-Office

Le design

Le design de l'API a été soigneusement conçu pour refléter l'identité visuelle de l'application mobile. Les mêmes couleurs et styles graphiques ont été conservés afin de maintenir une cohérence visuelle et d'offrir une expérience utilisateur fluide entre les deux plateformes. Ce choix permet également d'éviter des changements inutiles, tout en renforçant l'identité de l'écosystème numérique.

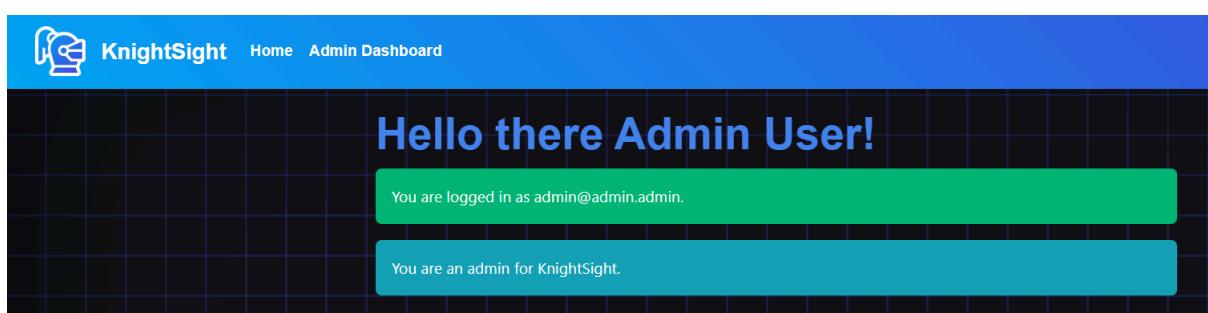


Accessibilité et connexion



Le site est entièrement accessible aussi bien sur PC que sur mobile, garantissant une navigation fluide et intuitive quel que soit le support utilisé. Cette compatibilité multiplateforme permet aux utilisateurs de consulter et d'interagir avec l'API sans contraintes techniques, rendant son utilisation pratique et rapide.

L'accès au back-office est sécurisé et limité aux administrateurs. La procédure débute par une page de connexion simple et efficace. Une fois authentifié, l'utilisateur peut se rediriger vers la partie administrateur, où il pourra accéder aux fonctionnalités du back-office.



Back-office

Pour la gestion du back-office, nous avons intégré EasyAdmin, un outil puissant et flexible permettant de créer rapidement des interfaces d'administration dans des projets Symfony. EasyAdmin offre une configuration simple et intuitive, permettant de définir et de personnaliser des pages pour gérer les données essentielles de l'application. Grâce à ce dernier, nous avons pu développer notre back-office promptement tout en conservant une interface ergonomique et efficace.

ID	First Name	Last Name	Roles	Banned	...
53	Admin	User	ROLE_ADMIN, ROLE_USER	NO	...
54	Lincoln	McDermott	ROLE_USER	NO	...
55	Chloe	Ritchie	ROLE_USER	NO	...
56	Otis	Quitzon	ROLE_USER	NO	...
57	Cleo	Goyette	ROLE_USER	NO	...
59	Jacky	Ritchie	ROLE_USER	NO	...
60	Kavon	Christiansen	ROLE_USER	NO	...
61	Mandy	Runolfsdottir	ROLE_USER	NO	...
63	Mabelle	Kilback	ROLE_USER	NO	...
65	Martina	Roob	ROLE_USER	NO	...
67	Britney	Rau	ROLE_USER	NO	...
69	Ethel	Nicolas	ROLE_USER	NO	...
70	Callie	Sipes	ROLE_USER	NO	...
71	Kale	Cormier	ROLE_USER	NO	...

Le back-office est organisé en quatre parties principales utilisant la méthode **CRUD** (Create Read Update Delete) :

- Utilisateurs** : Gestion des comptes des utilisateurs, avec la possibilité de bannir, visualiser, modifier, ajouter ou supprimer leurs profils.
- Tags** : Gestion des labels associés aux tags, par exemple black-bishop représente un fou noir.
- Images** : Gestion des images uploadées avec le chemin associé aux images, le label de l'image ainsi que la date de sa création.
- Image Tag** : Une section dédiée à la gestion des relations entre les images et leurs tags. Chaque ligne représente une association unique entre une image et un tag, accompagnée du nombre d'occurrences de ce tag dans l'image. Une image peut être liée à plusieurs tags, chacun apparaissant sur une ligne distincte.

EasyAdmin facilite l'ajout, la modification et la suppression de ces entités, tout en offrant une interface visuelle claire et structurée. Cette organisation nous garantit une gestion centralisée et efficace des ressources essentielles de l'application.

Requête et utilisation de l'API

L'API constitue le pont entre l'application mobile et le serveur, permettant un échange fluide et sécurisé des données. Pour exploiter pleinement ses fonctionnalités, il nous a été nécessaire de mettre en place des requêtes structurées et adaptées aux besoins de l'application. Cette section explore les différentes interactions entre l'application et l'API, en détaillant les types de requêtes utilisées, les données échangées, et la manière dont ces requêtes sont intégrées dans le flux global de l'application.

Les requêtes de l'API sont conçues pour répondre à plusieurs objectifs : récupérer les données des utilisateurs, stocker les résultats des analyses, et garantir la sécurité des échanges via des mécanismes d'authentification et de validation. Chaque étape du processus, qu'il s'agisse de l'envoi de données, de leur stockage ou de leur récupération, s'appuie sur des protocoles standardisés pour assurer une communication rapide et fiable.

Cette partie se concentre donc sur les différents aspects techniques liés à l'utilisation de l'API, incluant les types de requêtes HTTP mises en œuvre (GET, POST, etc.), les formats de données échangées (JSON), ainsi que l'intégration de ces interactions dans l'architecture de l'application mobile.

JWT Token

Dans le cadre de ce projet, la sécurité de l'API est gérée par le système d'authentification basé sur JSON Web Tokens (JWT), en utilisant le bundle Lexik JWT Authentication. Ce choix permet de garantir un accès sécurisé aux ressources tout en assurant une gestion fluide et efficace des sessions utilisateur.

Lexik JWT Authentication permet de gérer l'authentification des utilisateurs grâce à un mécanisme basé sur des tokens. Chaque utilisateur, après s'être identifié reçoit un token JWT signé qui contient des informations sur l'utilisateur pour valider son identité. Ce token est ensuite envoyé avec chaque requête à l'API, pour permettre une authentification sécurisée, les tokens JWT sont signés et ne peuvent pas être altérés, ce qui garantit l'intégrité des données envoyées.

Concernant l'intégration de Lexik JWT Authentication dans l'API, plusieurs étapes de préparation et de configuration ont été nécessaires pour garantir son bon fonctionnement. Ce bundle, utilisé pour gérer l'authentification des utilisateurs via des tokens JWT, repose sur la création et l'utilisation de clés cryptographiques, ainsi que sur la configuration précise de certains fichiers.

Pour sécuriser les tokens JWT, il est nécessaire de générer une paire de clés cryptographiques (privée et publique) au format PEM. Cette étape a été réalisée grâce à la commande suivante :

```
php bin/console lexik:jwt:generate-keypair
```

Cette commande génère deux fichiers :

- **private.pem** : pour signer les tokens JWT
- **public.pem** : pour vérifier la signature des tokens JWT

Ces fichiers sont stockés dans le dossier `config/jwt` du projet. Ils doivent être protégés et ne jamais être exposés publiquement, car leur compromis pourrait affecter la sécurité de l'authentification.

Une fois les fichiers PEM créés, leur emplacement et leur configuration sont spécifiés dans le fichier `config/packages/lexik_jwt_authentication.yaml` :

```
lexik_jwt_authentication:  
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'  
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'  
    pass_phrase: '%env(JWT_PASSPHRASE)%'  
    token_ttl: 259200 # 259200 seconds = 3 jours|
```

- `secret_key` et `public_key` définissent les chemins des fichiers PEM.
- `pass_phrase` est la phrase secrète utilisée lors de la création des clés.
- `token_ttl` spécifie la durée de validité du token en secondes (ici, 259200 secondes, soit 3 jours).

Une route dédiée a été mise en place pour permettre aux utilisateurs de s'authentifier et de récupérer leur token JWT. Cette route est exposée via le contrôleur suivant :

```
/*  
 * API to login, return an auth token if success  
 */  
#[Route(path: "/login_token", name:"api_login", methods: ["POST"])]  
4 references | 0 overrides  
public function loginToken(Request $request): JsonResponse  
[  
    // Get the JSON data from the request body  
    $data = json_decode(json: $request->getContent(), associative: true);  
  
    $email = $data['email'] ?? null;  
    $password = $data['password'] ?? null;  
    $user = $this->entityManager->getRepository(User::class)->findOneBy(['email' => $email]);
```

Lorsqu'un utilisateur fournit des identifiants valides, l'API retourne un token JWT signé, qu'il pourra utiliser pour accéder aux requêtes API sécurisées.

Pour protéger les ressources de l'API, certaines routes ont été sécurisées en nécessitant la présence d'un token JWT valide. Cette protection est configurée dans le fichier `config/packages/security.yaml` :

```
# API firewall (JWT Authentication)
api:
    pattern: ^/api/
    stateless: true
    jwt: ~
```

- Le firewall `api` protège toutes les routes commençant par `/api`
- L'option `stateless` indique que l'API ne maintient pas de sessions côté serveur, ce qui est l'idéal pour notre application mobile

Vérification de l'utilisateur via JWT

L'authentification via JWT repose sur la vérification des utilisateurs à chaque requête protégée, en utilisant le token transmis dans l'en-tête de la requête. Cette étape garantit que seules les requêtes provenant d'un utilisateur authentifié peuvent utiliser les informations du serveur.

Pour cela, lorsqu'une requête est reçue, l'API récupère le token JWT transmis dans l'en-tête et un processus de validation s'effectue. Voici les étapes incluses dans ce processus :

- Extraction du contenu de l'en-tête.
- Vérification de la présence du préfixe `Bearer`.
- Extraction du token lui-même (chaîne après `Bearer`).
- Validation du format et de l'intégrité du token.

```
$data = json_decode(json: $request->getContent(), associative: true);

// Validate JWT token
$authorizationHeader = $request->headers->get(key: 'Authorization');
if (!$authorizationHeader || strpos(haystack: $authorizationHeader, needle: 'Bearer ') !== 0) {
    throw new \InvalidArgumentException(message: 'Invalid or missing Authorization token.');
}

$jwtToken = substr(string: $authorizationHeader, offset: 7);

// Find user by JWT token
$user = $this->userRepository->findUserByJwtToken(token: $jwtToken);
if (!$user) {
    throw new \InvalidArgumentException(message: 'Invalid or expired JWT token.');
}
```

Une fois le token extrait, il est utilisé pour identifier l'utilisateur correspondant dans la base de données. Si aucun utilisateur ne correspond, ou si le token est invalide ou expiré, l'API renvoie une erreur.

Requêtes disponibles

Les requêtes API disponibles jouent divers rôles et offrent diverses fonctionnalités. Elles couvrent la gestion des utilisateurs, des images, des tags et du modèle de l'IA. Elles garantissent une interaction fluide et sécurisée entre l'application et le serveur.

Requêtes de Gestion des Images

Ces requêtes assurent l'enregistrement, la récupération et l'association des images avec leurs informations.

/api/upload-image (POST)

Cette requête permet aux utilisateurs d'envoyer des images au serveur. Le traitement se déroule comme suit :

- Vérification et validation du jeton JWT pour authentifier l'utilisateur.
- Décodage des données de l'image en base64.
- Enregistrement de l'image sur le serveur et extraction des étiquettes associées.
- Conversion des étiquettes en identifiants de tags existants ou création de nouveaux tags.
- Sauvegarde de l'image et des tags dans la base de données.

/api/images/latest-history/{nbr} (GET)

Récupère les dernières images traitées par un utilisateur, incluant les chemins des fichiers, les étiquettes associées et leurs données.

/api/image-tags/{imageId} (GET)

Retourne les tags associés à une image donnée, y compris leurs identifiants, labels et occurrences.

/api/image-labels/{imageId} (GET)

Fournit uniquement les noms (labels) des tags associés à une image spécifique.

Requêtes de Gestion des Modèles IA

Ces requêtes permettent la gestion des versions et la distribution des modèles d'apprentissage automatique.

/api/modele/version (GET)

Fournit la version actuelle du modèle IA en lisant un fichier texte stocké dans un répertoire serveur dédié.

/api/modele/download (GET)

Permet de télécharger le fichier du modèle IA (par exemple, un fichier YOLOv8 au format .tflite). Si le fichier est introuvable, une exception est levée.

Requêtes de Gestion des Utilisateurs

Ces requêtes traitent l'enregistrement, l'authentification et la gestion des historiques des utilisateurs.

/register (POST)

Enregistre un nouvel utilisateur avec un prénom, un nom, un email et un mot de passe sécurisé (hashé). Un rôle par défaut est attribué.

/api/token/validate (POST)

Valide un jeton JWT pour s'assurer qu'il est valide et retourne les informations de l'utilisateur associé.

/api/user/{id}/history (GET)

Accessible uniquement aux administrateurs, cette requête fournit l'historique des images traitées par un utilisateur spécifique.

/api/user/history (GET)

Retourne l'historique des images associées à l'utilisateur connecté, en fonction de son jeton JWT.

/api/user/info/{id} (GET)

Fournit des informations de base sur un utilisateur, telles que son prénom, son nom et son rôle.

Requêtes de Gestion des Tags

Ces requêtes facilitent la gestion des tags associés aux images.

/api/tags (GET)

Récupère tous les tags disponibles dans la base de données, incluant leurs identifiants et labels.

/api/tag/{id} (GET)

Fournit les détails spécifiques d'un tag identifié par son ID.

Fonctionnalités spécifiques

Réception d'image (/api/upload-image)

Cette requête est essentielle au bon fonctionnement de l'application, car elle permet d'ajouter des images à la banque de données. Elle a pour rôle de :

- Gérer l'intégration des images, une ressource clé pour l'application.
- Assurer la cohérence entre les données utilisateur et celles stockées en base de données, tout en garantissant la sécurité via la validation des jetons JWT.

Elle veille à ce que chaque image téléchargée soit correctement traitée, classée et mise à disposition pour une futur utilisation de manière sécurisée.

Envoi du modèle IA (/api/modele/download)

Cette requête est cruciale pour la mise à jour du modèle d'intelligence artificielle auprès des utilisateurs. Elle permet :

- D'offrir le modèle le plus récent, garantissant des performances optimales et des mises à jour régulières.
- De centraliser la gestion du modèle sur le serveur, offrant ainsi un contrôle total sur les versions distribuées.

Elle est donc indispensable pour garantir un accès fiable au moteur IA, contribuant à l'efficacité et aux performances du projet.

Test de l'API via Postman

Postman est un outil essentiel pour le développement, le test et le débogage des API. Dans notre projet, Postman a joué un rôle clé dans la vérification du bon fonctionnement de plusieurs requêtes API, y compris celles utilisant des tokens JWT pour l'authentification. Bien que Postman n'ait pas été utilisé pour tester l'envoi d'images (car cette tâche était gérée différemment), l'outil a été extrêmement utile pour toutes les autres requêtes du projet.

Suivi et Analyse des Requêtes et Réponses

Une fonctionnalité de Postman particulièrement utile est sa capacité à enregistrer et afficher l'historique des requêtes. Cela a facilité le débogage et l'analyse des interactions avec l'API.

- **Examen des Réponses** : En testant différentes requêtes, nous avons pu observer les réponses du serveur, analyser les codes d'état HTTP renvoyés (par exemple, 200 OK, 401 Unauthorized, 500 Internal Server Error) et nous assurer que l'API répondait correctement à chaque type de requête.
- **Tests de Cas Limites** : Nous avons utilisé Postman pour tester différents scénarios, comme l'envoi de requêtes mal formées ou l'absence de certaines données.

Postman a permis de simuler ces situations et d'observer les réponses du serveur dans des conditions variées.

The screenshot shows the Postman interface with a failed API request. The request URL is `http://212.227.57.57:8081/api/user/351/history`. The 'Authorization' tab is selected, showing 'Bearer Token' as the auth type and a placeholder token. A note explains that the authorization header will be automatically generated when the request is sent. The response status is '404 Not Found' with a duration of '224 ms' and a size of '302 B'. The response body is {"error": "User not found."}.

Test des Requêtes avec JWT pour Assurer la Sécurité

Postman a permis de tester la sécurité des requêtes et leur bon fonctionnement. Pour tester les requêtes avec un Token il faut donc

- **Ajouter le Token dans les En-têtes (Headers)** : Dans Postman, il est facile d'ajouter le token JWT à l'en-tête de la requête en utilisant la clé **Authorization** avec la valeur **Bearer <token_jwt>**. Cela nous a permis de tester les requêtes sécurisées, telles que celles nécessitant une authentification avant d'accéder à des données sensibles ou de télécharger des ressources.
- **Vérification des Scénarios d'Erreur d'Authentification** : Postman a également permis de tester des scénarios où un token invalide ou expiré est envoyé dans la requête. En observant les réponses du serveur, nous avons pu confirmer que des erreurs appropriées (par exemple, un code HTTP 401 "Unauthorized") étaient renvoyées lorsqu'un utilisateur non autorisé tentait d'effectuer des actions protégées.

GET http://localhost:8081/api/user/history

Params Authorization Headers Body Scripts Settings Cookies

Auth Type

Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE3MzU3NzMwMTUsImV4cCI6MTczNjAzMjlxNSwicm9sZXMiOlisiUk9MRV9BRE1JTlslJPTTEVFVNfUiJdLCJ1c2VybmfTzSI6ImFkbWluQGFkbWluLmFkbWluIn0.W_NHjjDQygGSFVv-2bXg5GYI1pd8VQJE2bsopiQRgiSoeWRckI7167q7a5ITBgrGHlkzivDrgti2h4wxmYbkw3zHCVFcRvlZZx2MWXVO85OBZTQGCcr9CgA7rY12kqXYFpWE39icy5uZWeG-
```

Difficultés rencontrées

Application Mobile : Flutter Vision

Lors du développement de l'application mobile, nous avons rencontré divers problèmes, notamment concernant l'implémentation de l'IA dans la caméra. Afin d'utiliser Yolo dans l'application, nous avons dû chercher une dépendance Flutter qui correspondait à nos attentes tout en étant compatible avec notre environnement.

Après plusieurs tests de dépendances et également de versions de Yolo, à savoir Yolov5, Yolov8 et Yolov11, nous avons choisi d'utiliser Yolov8 avec flutter_vision.

Dans un deuxième temps, flutter_vision ne permet l'utilisation que des fichiers se trouvant dans le dossier assets de l'application. Cette logique est gênante, car nous souhaitons stocker les modèles d'IA ailleurs, c'est-à-dire dans les dossiers locaux de l'application.

Ainsi, pour pallier ce problème, nous avons "forké" le repository de l'application et avons commit sur ce fork les modifications nécessaires. Vous pouvez les consulter sur ce repository public : <https://github.com/HamelenYakkington/SAE-501.git>

Ces modifications permettent de charger des fichiers directement depuis les fichiers locaux mais également permettent la fermeture du modèle IA tout en arrêtant toute les actions de celle-ci en cours.

Application Mobile : Gestion des encadrements d'objets

Lors de la détection d'objets par l'IA, nous affichons des rectangles autour des objets détectés, tout en affichant le nom de celui-ci et la confiance que l'IA a dans cette détection.

Le problème que nous avons rencontré est que l'affichage de ces rectangles était initialement modifié seulement dans le cas où une nouvelle détection avait été trouvée. Ainsi, il était possible pour un utilisateur de détecter un chevalier, de déplacer la caméra pour que l'encadrement soit sur autre chose, puis de prendre une photo. De cette manière, nous pouvions avoir des décalages dans les résultats ou même du "sabotage".

Pour résoudre ce problème, nous avons fait en sorte que, lorsque les résultats sont vides, les encadrements soient tout simplement supprimés. Cependant, en procédant ainsi, l'expérience utilisateur pouvait être largement impactée, car l'IA ne détectait pas forcément

le même objet à chaque image du flux vidéo, ce qui faisait que les encadrements clignotaient.

Nous avons donc dû trouver un juste milieu et actualiser l'affichage et les détections seulement toutes les 4 images du flux vidéo. Ainsi, les décalages possibles dans les résultats de l'IA seront moins importants, et lorsque aucun objet n'est détecté, les encadrements seront supprimés.

Application Mobile : Precision du modèle

En plus du problème de version de Yolo, nous avons aussi été confrontés à une incompréhension quant au manque de précision du modèle après avoir changé plusieurs fois de dataset en pensant que ça en était la cause. Le problème provenait en réalité du modèle de Yolov8. En effet il existe 5 modèles différents de Yolov8 :

Yolov8n(ano), Yolov8s(mall), Yolov8m(edium), Yolov8l(arge), Yolov8x(Extra Large).

La version YoloV8n étant prévue pour des appareils avec des ressources limitées, on s'est dit que c'était parfait pour nous étant donné qu'on développe une application mobile. Cette version est légère et rapide, mais peine quand il s'agit de précision. Et c'était exactement notre problème, nous obtenions aux alentours de 60% de confiance avec ce modèle. Le taux de confiance était si bas qu'il ne nous est pas venu à l'esprit que le problème serait le modèle mais plutôt le jeu de données. C'est donc à ce moment qu'on a perdu du temps à chercher un jeu de données qui nous permettrait d'avoir un meilleur taux de confiance sur notre détection de pièces d'échecs.

La solution était d'utiliser un autre modèle de YoloV8, nous nous sommes dirigés vers YoloV8m, une version plus précise et plus lourde. Nous avons observé un taux de confiance beaucoup plus élevé sans que l'application se ralentit de façon significative, nous avons donc réessayé avec un dataset beaucoup plus conséquent (presque 10 000 images) ce qui nous pouvait prendre plus de 10 minutes par itération avec une carte graphique de dernière série.

Depuis les résultats étaient beaucoup plus précis, cependant il y avait toujours des incohérences et le modèle n'arrivait pas à reconnaître la majeure partie des pièces, nous avons donc mieux inspecté le dataset ce qui nous a permis de constater qu'une petite partie des images contenait des erreurs par exemple des tags mal placés et cela nous avait échappé la première fois.

Nous avons donc refait un dataset plus petit mais mieux labélisé à partir de ce dernier et nous avons également testé le modèle YoloV8s, cependant, étant donné que notre dataset est réduit, il risque d'y avoir des imprécisions ou des pièces non labellisées lors de l'utilisation de l'application voir encore d'autres erreurs.

De longues étapes

On peut également noter que le procédé pour tester l'IA avec notre application reste long car il nécessite beaucoup de manipulations qui prennent un temps conséquent chacune et que nous ne pouvions pas tous faire en fonction de la tâche étant donné que nous nous sommes partagés les tâches.

Pour tester un nouveau modèle d'IA il nous faut d'abord avoir un bon dataset, dans notre cas il a fallu le refaire à partir d'un modèle déjà existant ce qui nous a pris beaucoup de temps, ensuite après l'acquisition d'images convenables, il faut les labelliser une à une. Une fois ceci fait, il faut exporter les images avec leurs labels et ensuite on peut donc entraîner notre IA.

Cependant l'entraînement d'IA consomme beaucoup de ressources et l'utilisation d'un GPU performant (dans notre cas nous avons utilisé une carte graphique de dernière série) afin de pouvoir accélérer le processus, après ceci nous avons donc notre fichier best.pt à convertir en fichier .tflite et c'est ensuite que nous pouvons avoir le modèle sur notre application mobile.

Il faut ensuite synchroniser le modèle d'IA avec l'API et ensuite on peut enfin tester le modèle à l'aide de la caméra, à ce stade si un problème ralentissant l'application est compliqué à identifier, il pourrait venir de l'appareil utilisé, du modèle d'IA ou encore de l'application.

Tout ce procédé est donc laborieux étant donné que plus l'on descend plus l'on a passé de temps et si le dataset est défectueux, il faut alors tout recommencer en espérant avoir identifié le problème correctement afin d'avoir un modèle correct. Le fait que nous n'avons jamais entraîné d'IA est également un facteur contraignant étant donné que nous ne pouvons pas très bien analyser nos problèmes liés à ce dernier.

API : Identification d'un utilisateur via le token

L'une des principales difficultés que nous avons rencontrées concernait l'identification d'un utilisateur à partir de son token JWT en utilisant la librairie LexikJWTAuthenticationBundle. Lors des tests initiaux, nous avons eu certains problèmes concernant l'identification d'un utilisateur via le token.

Pour résoudre ce problème, nous avons décidé d'enregistrer le token directement dans la base de données, associé à chaque utilisateur. Cette approche nous a permis de disposer d'une référence claire pour vérifier l'authenticité du token et de maintenir une meilleure traçabilité. Et permettre de vérifier le statut de l'utilisateur si celui-ci a été banni ou s'il existe encore.

API : Gestion de l'expiration des tokens

La gestion de l'expiration des tokens a également posé des défis. Par défaut, un token expiré devait automatiquement être rejeté par le système, mais cela n'était pas toujours le cas en raison d'erreurs dans la configuration initiale du bundle. Ces erreurs ont entraîné des situations où des tokens expirés étaient encore acceptés, créant ainsi des failles potentielles dans la sécurité de l'application.

Pour résoudre ce problème, nous avons ajusté les paramètres d'expiration dans la configuration de LexikJWTAuthenticationBundle.

Améliorations Possibles

Application Mobile : Gestions des modèles IA

Actuellement, la gestion des téléchargements des modèles IA est plutôt simple en termes de complexité. Afin de l'améliorer, nous pouvons envisager de nombreuses solutions. Par exemple, plutôt que d'effacer les anciennes versions, nous pourrions les sauvegarder, permettant ainsi à l'utilisateur de sélectionner l'IA qu'il souhaite utiliser via une vue "Options".

Notre fonctionnalité "Receive Data" pourrait également permettre de choisir la version d'IA que l'utilisateur souhaite télécharger.

D'un autre côté, la détection sur flux vidéo et sur image utilise le même modèle d'IA, mais il est aussi possible de permettre à l'utilisateur de choisir un modèle plus léger pour la caméra et un modèle plus lourd pour les images. Ainsi, la flexibilité des détections et la précision pourraient être améliorées.

Actuellement, si un téléphone est utilisé pour plusieurs comptes, ces comptes doivent obligatoirement utiliser le même modèle d'IA. Nous pourrions aussi implémenter une logique permettant de dissocier les préférences des utilisateurs en matière d'IA dans l'application, ce qui est largement réalisable avec l'idée de la vue "Options".

Nous n'avons pas pu implémenter une telle idée car nous y avons pensé trop tard, et une si importante modification du code faite trop rapidement aurait simplement pu détruire le projet. Il aurait fallu modifier l'api en plus de l'application mobile et également de la base de données.

Application Mobile : Administrateur

Actuellement, le compte administrateur a pour intérêt la possibilité de consulter le backoffice et d'utiliser les CRUDs de celui-ci. Cependant, il n'a pas d'actions réservées dans l'application. Nous aurions effectivement pu permettre à celui-ci d'interagir plus activement avec l'application que les autres utilisateurs.

Par exemple, en lui permettant de voir les informations des utilisateurs ayant envoyé des photos, ou encore de supprimer des photos, de modifier les labels des objets détectés, et même de bannir directement un utilisateur sans passer par le backoffice.

Nous avons préféré éviter cette logique, car le sujet de notre projet n'implique pas l'implémentation d'une telle logique. De plus, il aurait fallu sécuriser davantage l'application et l'API qu'elles ne le sont déjà, afin d'empêcher toute action malveillante.

API : Routes API

Les possibilités d'évolution des routes API sont étroitement liées aux besoins de l'application, permettant des ajustements en fonction des exigences spécifiques. Par exemple, dans le cas d'un administrateur, il est envisageable de restreindre ou d'accorder l'accès à certaines routes uniquement aux utilisateurs disposant du rôle approprié.

Par ailleurs, une meilleure gestion et protection du token JWT pourraient être envisagées, notamment en évitant le stockage du token dans la base de données tout en maintenant un accès efficace à l'utilisateur correspondant.

Enfin, des améliorations pourraient inclure l'ajout d'informations supplémentaires aux routes existantes pour enrichir les fonctionnalités ou répondre à de nouveaux besoins fonctionnels.