Avianity

Une application web full stack MERN pour la documentation et l'étiquetage collaboratif d'images d'oiseaux en Suisse

Travail de Bachelor

Sara Faggella

Août 2024

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

Remerciements

Je tiens à remercier chaleureusement Prof. Dr. Jacques Pasquier-Rocha et Quentin Nater de m'avoir permis de réaliser ce travail passionnant.

Un grand merci également à ma famille et amis qui m'ont soutenue tout au long de l'été. Un merci tout particulier à mes parents qui m'ont grandement aidée pour la relecture et les corrections de ce rapport.

Résumé

Ce rapport a été rédigé dans le contexte d'un travail de Bachelor en Informatique à l'Université de Fribourg. Une application web avec le full-stack MERN a été implémentée pour ce projet.

Il s'agit d'une plateforme sur laquelle les utilisateurs peuvent collaborer entre eux pour la création et la gestion d'une banque d'images d'oiseaux de Suisse. L'objectif derrière ce projet serait de pouvoir construire, à terme, un dataset fiable de photos d'oiseaux de Suisse.

Mots-clés : Développement web, Dataset, Taxonomie, Reconnaissance automatique, Crowdsourcing, Fine-grained image recognition

Table des matières

1. Introduction **2**

1.1. Motivation 2
1.2. Objectifs 3
1.3. Structure du rapport 3

2. Contexte **5**

2.1. Problématique 5
2.1.1. L'observation des oiseaux 5
2.1.2. Les applications d'auto-ID 6
2.2. État-de-l'art 7
2.2.1. Fine-Grained Image Analysis (FGIA) 7
2.2.2. La construction de dataset 8
2.2.3. Quantité ou qualité 9
2.2.4. Réflexions 10

3. Cadre **12**

3.1. Description formelle 12
3.2. Fonctionnalités retenues 12
3.3. Algorithme de consensus 13
3.3.1. Weighted Majority Voting 13
3.3.2. Notation 14
3.3.3. Mise en situation 15
3.3.4. Exploration 16
3.3.5. Critères de décision 19

4. Avianity : Point de Vue Utilisateur **21**

4.1. Pages d'accueil 21
4.2. Gestion du compte utilisateur 23
4.2.1. Cas : Enregistrer un nouveau compte 23
4.2.2. Cas : Connexion à son compte 25

4.2.3. Autres 25
4.3. Gestion de la banque d'images 26
4.3.1. Cas : Déposer une nouvelle photo 26
4.3.2. Cas : Consulter la banque d'images 28
4.3.3. Cas : Générer un dataset 30
4.3.4. Cas : Tentative d'identification 31
4.3.5. Cas : Signaler une photo 32

5. Avianity : Point de Vue Développeur **34**
5.1. Vue d'ensemble 34
5.1.1. Stack technologique 34
5.1.2. Architecture 36
5.2. Base de données 37
5.3. Serveur 40
5.3.1. API et endpoints 40
5.3.2. Structure 42
5.4. Client 47
5.4.1. React et DOM 47
5.4.2. Structure 47
5.4.3. Routing des pages 49
5.4.4. Contextes React 50

6. Conclusion **54**
6.1. Résultats 54
6.2. Perspectives futures 54
6.3. Mot de Fin 55

A. Acronymes récurrents **56**

B. License de la Documentation **57**

C. Code Source et Site du Projet **58**

Bibliographie **59**

Liste des figures

2.1. Exemple d'une classification générique vs. aux fins détails [59] 8

3.1. Évolutions des degrés de confiance sur un cas simple 15
3.2. Évolution des probabilités pour un cas simple 16
3.3. Évolution des probabilités pour deux espèces qui s'alternent 17
3.4. Évolution des probabilités pour trois espèces qui s'alternent 18

4.1. Page d'accueil (déconnecté) 22
4.2. Page d'accueil (connecté) 22
4.3. Page à propos du site 23
4.4. Enregistrer un compte : formulaire d'inscription 24
4.5. Enregistrer un compte : confirmation par mail 24
4.6. Enregistrer un compte : message de bienvenue 24
4.7. Connexion à son compte : formulaire de connexion 25
4.8. Formulaire pour le changement de mot-de-passe 26
4.9. Page utilisateur et demande de changement de statut de membre 26
4.10. Dépot de nouvelle photo : import de l'image 27
4.11. Dépot de nouvelle photo : formulaire 27
4.12. Dépot de nouvelle photo : page de remerciement 28
4.13. Consulter la banque d'images : la galerie 29
4.14. Consulter la banque d'images : les détails pour une photo 29
4.15. Consulter la banque d'images : galerie vide 30
4.16. Consulter la banque d'images : filtre "classée" activé 30
4.17. Consulter la banque d'images : générer un dataset 31
4.18. Tentative d'identification : formulaire 32
4.19. Tentative d'identification : remerciement 32
4.20. Signaler une photo : critères de validité 33
4.21. Signaler une photo : remerciement 33

5.1. L'architecture de l'application 37

5.2. Un schéma ERD représentant les trois entités principales de l'application 38
5.3. Les requêtes possibles pour notre application 41
5.4. La structure des fichiers côté serveur. 42
5.5. La structure des fichiers côté client. 48
5.6. Le flow de navigation pour notre GUI. 50

Liste des codes source

5.1. La création d'un schéma Mongoose pour les documents photo. 39
5.2. La création d'un modèle Mongoose pour les documents photo. 39
5.3. Initialisation du serveur. 42
5.4. Exemple de routing avec l'application Express. 43
5.5. Exemple d'endpoint avec le routeur Express. 44
5.6. Exemple de requête faite par le client. 45
5.7. Exemple de méthode controller. 45
5.8. Calcul de la probabilité approximée qu'une espèce soit correcte. 46
5.9. Initialisation du client. 48
5.10. Le routing des pages côté client. 49
5.11. Exemple de configuration d'un context React. 50
5.12. Utilisation du context React. 51
5.13. Mise à jour du contexte React. 52

1
Introduction

1.1. Motivation **2**
1.2. Objectifs **3**
1.3. Structure du rapport **3**

1.1. Motivation

Les applications de reconnaissance automatique (ou applications d'auto-ID) sont apparues au cours de ces dernières années et ne font que de gagner en popularité depuis. Grâce à ces applications, les utilisateurs peuvent faire identifier le sujet de leur photo ou enregistrement audio en seulement quelques clics. Certaines de ces applications comme PlantID [61] ou Merlin Bird ID [18] se spécialisent dans la reconnaissance des espèces végétales ou animales, et c'est à partir de ces dernières qu'est né mon intérêt pour le sujet de la reconnaissance automatique.

C'est un concept qui peut avoir des conséquences fort positives, notamment dans l'idée de sensibiliser le grand public à la cause environnementale et la conservation des espèces locales. [48] [46]

Aussi beau que cela puisse paraître, plusieurs recherches ont évalué la performance de ces applications pour la reconnaissance des espèces naturelles, et malheureusement, il est difficile pour l'instant de les considérer fiables. [50] [49] [57] [39] [56]

En l'état actuel, il est possible que la popularité grandissante de ces applications et leur performance insatisfaisante puissent engendrer certains problèmes, par exemple dans le cadre de projets participatifs pour le recensement de la faune ou de la flore.

Dans le cadre de ce travail, des recherches ont eu lieu au sujet de la reconnaissance automatique pour les sujets aux détails fins, c'est-à-dire, la capacité d'une machine à reconnaître à quelle sous-catégorie un sujet appartient (p. ex. reconnaître l'espèce d'un oiseau). [59]

À l'issue de mes recherches, j'en ai conclu qu'il vaudrait la peine de s'attarder davantage sur le processus de construction des datasets dans le domaine, de manière à obtenir une plus grande quantité de données fiables.

À travers ce projet, je souhaiterais explorer une méthode de construction de dataset qui puisse engendrer des données de qualité. J'ai choisi comme thématique la collecte de photos d'oiseaux de suisse, car c'est un sujet qui m'intéresse tout particulièrement.

1.2. Objectifs

J'aimerais pouvoir créer une plateforme où la création, la gestion et l'annotation des données sont toutes réalisées de manière collaborative par des passionnés d'ornithologie.

Je souhaite que les utilisateurs de la plateforme puissent importer leurs propres photos et qu'ils puissent collaborer ensemble pour l'annotation des photos, de telle manière à déterminer l'espèce la plus probable, en partant du principe que la décision d'un groupe a de plus grandes chances d'être fiable par rapport à l'avis d'un seul individu. [52]

En ce qui concerne mes objectifs d'apprentissage, j'aimerais avant tout améliorer ma compréhension du développement web. Je suis partie de l'hypothèse initiale que c'est en appliquant la théorie que je vais être davantage en mesure de la maîtriser. Un autre objectif d'apprentissage est de pouvoir apprendre des langages pour le développement web et acquérir des outils qui me seront peut être à nouveau utiles à l'avenir.

1.3. Structure du rapport

Chapitre 1 : Introduction

Ce chapitre donne un bref aperçu des enjeux et objectifs de ce projet de Bachelor, puis, la structure du rapport est présentée.

Chapitre 2 : Contexte

Ce chapitre commence par établir un lien entre les enjeux de la surveillance des oiseaux et les applications de reconnaissance automatique. Ensuite, nous présenterons une courte introduction au sujet de la reconnaissance automatique pour les sujets aux détails fins tels que les oiseaux. Finalement, nous discuterons des enjeux et pistes d'amélioration en ce qui concerne la création de dataset.

Chapitre 3 : Cadre

Ce chapitre décrit les choix qui ont été essentiels pour la conception du projet. Nous commencerons par une brève description du projet, des fonctionnalités que j'ai souhaité implémenter, puis nous aborderons les décisions algorithmiques qui ont dû être faites pour pouvoir implémenter un système de crowdsourcing.

Chapitre 4 : Implémentation, point de vue utilisateur

Ce chapitre présente l'application finale d'un point de vue de l'utilisateur final. Un lien vers le site web sera fourni. Ensuite, nous verrons les différentes fonctionnalités de l'application avec l'aide de captures d'écran.

Chapitre 5 : Implémentation, point de vue développeur

Ce chapitre présente comment l'application a été implémentée. Nous décrirons quels outils et architecture ont été choisis, puis nous entrerons dans les détails pour chaque composante de l'architecture, à savoir, la base de données, le serveur web et l'interface utilisateur.

Chapitre 6 : Conclusion

Nous terminerons ce rapport avec quelques réflexions sur l'implémentation suivi de la conclusion. Nous commencerons par discuter du résultat de l'implémentation, des voies d'améliorations et des suggestions pour de futures travaux.

2
Contexte

2.1. Problématique **5**
2.1.1. L'observation des oiseaux 5
2.1.2. Les applications d'auto-ID 6
2.2. État-de-l'art **7**
2.2.1. Fine-Grained Image Analysis (FGIA) 7
2.2.2. La construction de dataset 8
2.2.3. Quantité ou qualité 9
2.2.4. Réflexions 10

Dans l'idée de mieux comprendre le chemin de pensée qui m'a mené à la réalisation de ce projet, ce chapitre aborde les enjeux de la surveillance des oiseaux et le rôle du public dans la récolte de données. Quelques clés de compréhension seront également fournies pour comprendre l'état de la recherche dans le domaine de la reconnaissance automatique des espèces vivantes.

2.1. Problématique

2.1.1. L'observation des oiseaux

L'observation des oiseaux est une tâche de la plus haute importance : dans l'ensemble, les oiseaux sont omniprésents sur le territoire suisse. Le moindre changement dans l'environnement va donc se refléter très rapidement sur la quantité et le type d'oiseaux qui fréquentent les lieux.

C'est pourquoi la distribution des populations aviaires est un excellent indicateur de la santé de la biodiversité. Il est important de monitorer cette dernière pour les efforts de conservation à la fois de l'avifaune et de l'ensemble de l'écosystème. [51] [36]

Cela dit, il est très compliqué de déterminer efficacement la répartition des oiseaux. Habituellement, la marche à suivre serait de mobiliser des professionnels pour réaliser des enquêtes de terrain, mais il n'y a pas suffisamment de temps et de main d'œuvre pour couvrir l'entièreté des territoires et des espèces sur lesquels on aurait besoin d'informations.

La science participative est donc une alternative essentielle pour étendre notre champ de connaissances : il s'agit de faire appel à la contribution de citoyens bénévoles pour compter les oiseaux d'un lieu spécifique ou de recenser des observations. [13] [6]

L'aide des bénévoles recèle un fort potentiel pour contribuer à la surveillance des oiseaux : avec suffisamment de participants, on serait en mesure d'étendre le terrain de nos recherches et d'augmenter la quantité d'oiseaux observés, nous permettant ainsi de compléter nos savoirs et peut être même de déceler des phénomènes inattendus.

Ce genre de projet est aussi grandement bénéfique pour porter l'attention du public sur le sujet des oiseaux et de la conservation de l'environnement. [7] [1]

Cela dit, la science participative possède un grand défaut lorsqu'il es question de taxonomie : pour que les données récoltées soient utilisables, elles doivent assurer un certain taux de fiabilité. Cela nécessite que les participants soient en mesure d'identifier correctement les espèces. Cependant, la tâche d'identification est parfois très compliquée, même pour un expert. Il faudrait donc que l'ensemble des participants emploie une méthodologie reconnue pour contribuer correctement aux efforts de recensement. [53]

Or, plus le nombre de participants augmente pour un projet donné, plus il devient fastidieux de garantir que ce critère soit rempli.

Pour un projet in situ, il est courant que les participants soient formés au préalable pour employer certaines méthodologies et également pour éviter de perturber les espèces observées.

Mais dans l'idée de toucher un plus grand public, de plus en plus de projets ont lieu en ligne. Bien que le principe soit très prometteur, il n'est pas méthode courante d'imposer un protocole d'identification pour ce genre de projet. Il devient ainsi bien compliqué d'assurer la justesse des données obtenues.

2.1.2. Les applications d'auto-ID

Parallèlement, nous avons assisté ces dernières années à l'émergence de logiciels de reconnaissance automatique, couramment appelés des applications d'auto-ID. Parmi elles, nous avons de plus en plus d'applications qui se spécialisent dans la reconnaissance des espèces végétales ou animales.

Le concept connaît un grand succès auprès de tous types de public, et l'utilisation de telles applications ne fait qu'augmenter. Il en va de même dans le cadre de projets de science participative. [48] Il est même spécifié par certaines applications que leur objectif est de permettre à tous le monde de participer aux efforts de recensement grâce à la reconnaissance automatique [61] [16]

Le problème est que ces applications ne peuvent pas encore être considérées comme fiables. Selon un nombre grandissant de recherches [50] [49] [57], le taux d'erreur de ces applications ne descend pas vraiment en dessous des 10% et la marge d'erreurs augmente considérablement lorsque l'espèce photographiée est rare ou endémique. [39] [56]

Cela n'est pas vraiment embêtant lorsque l'application est utilisée dans un contexte de loisir (outre les risques d'ingérer des plantes toxiques par exemple), mais on pourrait considérer que l'usage de ces applications devient problématique lorsque l'utilisateur est étudiant en botanique par exemple et cherche à se former, ou lorsque l'on contribue à un projet de science participative et que les prédictions de l'application ne sont pas vérifiées. [38]

Indéniablement, le concept de reconnaissance automatique pourrait devenir très important pour sensibiliser le grand public à la conservation de l'environnement et pour contribuer aux efforts de recensement. Cela dit, les applications déjà accessibles au public présentent des problèmes de performance et aussi de manque de clarté quant aux limites de la technologie.

Pour en revenir à la problématique de la surveillance des oiseaux et du besoin de données supplémentaires, on pourrait avancer que la contribution des applications d'auto-ID dans les projets de science participative pourrait exacerber le manque de fiabilité des données récoltées.

Il est bien probable que l'engouement pour ces applications ne fera que grandir dans les années à venir. Il vaut donc grandement la peine, à mon avis, de chercher à améliorer la performance des modèles de prédiction pour l'identification des oiseaux. Ainsi, à termes, la contribution du grand public deviendra peut être d'une importance capitale pour la surveillance et la conservation des oiseaux.

Voyons à présent ce qui se fait dans le domaine de l'apprentissage automatique, afin de mieux comprendre les causes des performances insatisfaisantes des applications d'auto-ID actuelles.

2.2. État-de-l'art

2.2.1. Fine-Grained Image Analysis (FGIA)

Dans le domaine de la vision par ordinateur (computer vision), la reconnaissance d'image aux détails fins (Fine-Grained Image Recognition) décrit la capacité à reconnaître la sous-catégorie dans laquelle l'objet sur l'image se trouve, par exemple, identifier la race d'un chat ou le modèle d'une voiture (défintion reprise de Wei et col. (2022) [59]). Les sujets aux détails fins, tels qu'ils sont communément définis, inclus tout objet dont la variabilité inter-classe est relativement faible et la variabilité intra-classe est haute. [59] [42] Par exemple dans le cas des oiseaux, cela veut dire que certaines espèces peuvent grandement se ressembler (p.ex. les mésanges) et parfois au sein d'une même espèce les différences morphologiques sont frappantes (p.ex. les cas de dimorphisme sexuel [12]).

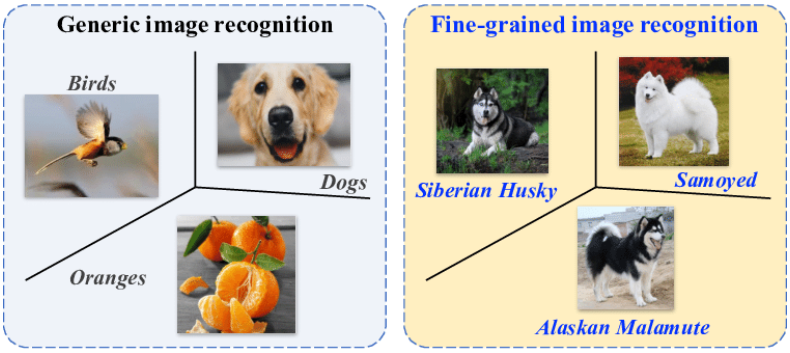


Figure 2.1. – Exemple d'une classification générique vs. aux fins détails [59]

Dans le domaine de l'analyse d'image aux détails fins (FGIA), on cherche à permettre à la machine de réaliser la tâche d'identification décrite ci-dessus.

Ce domaine de recherche est particulièrement actif depuis plus de 10 ans maintenant, ceci parce qu'il est particulièrement difficile de construire des modèles efficaces pour relever un tel défi. D'une part, le modèle de prédiction doit être en mesure, pour chaque sous-catégorie d'un objet, de pouvoir reconnaître quels sont les détails-clés pour déterminer cette sous-catégorie parmi d'autres. Mais aussi, comme pour la reconnaissance d'objet de manière générale, les prédictions du modèle doivent être résistantes aux variations de l'agencement spatial, la qualité ou autre distorsion de l'image. [59]

Ainsi le problème n'est pas simple à résoudre, mais qu'en est-il des pistes d'amélioration ? La plupart des recherches dans le domaine se sont concentrées sur l'élaboration de modèles de prédiction adaptés aux enjeux de la détection et du traitement des détails fins. [59]

À mon sens, il y a un autre point de discussion qui mériterait un regard plus poussé : il s'agit des datasets que l'on utilise aujourd'hui comme points de référence.

2.2.2. La construction de dataset

Conformément aux avancées dans le domaine du deep learning, on estime en FGIA que augmenter la quantité des données va naturellement entraîner des meilleures performances pour l'identification des espèces (notamment grâce à une meilleure représentativité par sous-catégorie).

Dans cette perspective, pour les sujets aux détails fins, on utilise aussi les méthodes de collectes et d'annotations des données qui sont de pratiques communes en deep learning.

- En termes de collecte, il s'agit souvent d'automatiser la récolte des données en écumant le web et enregistrant toutes données pertinentes (techniques qui s'appelle le web crawling and scraping [11])
- Pour l'annotation, on a beaucoup recours au crowdsourcing, qui se décrit par la délégation d'une tâche à des individus externes. Dans le contexte de la construction de dataset, on utilise souvent des plateformes comme Amazon Mechanical Turk (AMT [2]) où l'on emploie un grand nombre d'individus pour réaliser efficacement la tâche d'annotation.

Or, de telles méthodes de construction ne paraissent pas particulièrement adaptées au domaine de l'analyse des détails fins. Pour citer quelques problèmes appliqués au sujet des datasets d'oiseaux ([59] [55]) :

- Tout d'abord, la collecte de données en fouillant le web, comme il est décrit plus haut, ne fait que répercuter le problème de la représentativité inégale des espèces plus rares ou propre à des régions spécifiques.
- La plupart des images que l'on peut observer dans les grands datasets d'oiseaux (voir [4] [8]) sont trop peu représentatives d'un cas d'utilisation typique de la reconnaissance automatique : les images qu'un utilisateur peut prendre sont souvent éloignées du sujet, un peu floue, avec plusieurs oiseaux,.. Comme les données récoltées par le web scraping sont principalement cadrées et nettes, il en résulte que les modèles de prédiction performent sur des bonnes données et se cassent la figure sur les photos des utilisateurs.
- Le problème le plus marquant est au sujet des annotations : la tâche d'associer la photo à une espèce est souvent déléguée à des travailleurs qui ne sont en aucun cas spécialisés dans l'identification d'oiseaux. Ainsi, il en résulte une grande quantité de données annotées mais il est ensuite difficile d'estimer la qualité de ces annotations.

Reste à savoir si ces défauts péjorent ou non la performance des modèles de reconnaissance. La qualité des données a-t-elle vraiment autant d'impact, lorsque le dataset compte parfois plus de 50'000 datapoints ?

2.2.3. Quantité ou qualité

Évidemment, lorsque cela est possible, tout le monde s'accorderait à dire qu'il est important de faire attention à la qualité des données pour l'entraînement des modèles de prédiction. Après tout, l'expression "garbage in garbage out" ne s'est pas popularisée toute seule.

Dans le domaine de la FGIA, il est surprenant de constater le peu de discussions autour de la construction de dataset. Il me semble donc pertinent d'explorer l'hypothèse selon laquelle améliorer la qualité des données d'entraînement (en les rendant plus fiables et mieux adaptées) ne pourrait pas engendrer des améliorations dans la performance des modèles de prédiction. [55] [59]

Malheureusement, comme les techniques modernes de deep learning nécessitent des quantités astronomiques de données, il est bien difficile d'imaginer une méthode de construction de dataset qui puisse permettre un suivi de la qualité. Il semblerait qu'il y ait donc un choix à faire entre qualité et quantité.

Une recherche sur le sujet m'a particulièrement inspiré pour la conception de ce travail de Bachelor : Van Horn et collaborateurs [55] proposent des méthodes de construction de dataset de manière à produire des données riches en classes et de bonne qualité.

Leur approche repose sur le crowdsourcing pour la collecte et l'annotation, mais une stratégie inédite a été de mobiliser une combinaison d'experts en ornithologie, de citoyens passionnés et des turkers. La collecte et les tâches complexes d'annotation étaient gérées par les experts et amateurs, les tâches qui ne nécessitaient pas forcément de pré-requis

ont été déléguées aux turkers. De cette collaboration est né un dataset particulièrement large (48'000 images pour 500 catégories) pour dire que les données produites ont pu être vérifiées par des experts du sujet.

Malgré ces résultats réjouissants, on se rend compte qu'un nombre réduit de collaborateurs fournit moins de données en plus de temps. Néanmoins certains avantages importants sont à souligner :

- D'une part, ce sont des passionnés d'ornithologie qui ont fourni leurs propres photos pour ce projet. Ce qui fait que les données collectées sont davantage alignées avec l'application prévue, c'est-à-dire que les photos ont été prises dans un contexte plus réaliste.
- D'autre part, en collaboration avec la communauté ornithologique, on s'assure ainsi que les recherches sont dirigées dans la bonne direction, autant dans le domaine de l'apprentissage automatique que dans le domaine ornithologique. La bonne communication est à mon avis primordiale entre les deux milieux de recherche.
- Il y a encore un point qui à mon avis fait toute la différence : Van Horn et col. [55] ont réalisé toute une batterie de tests sur la corruption des datasets, pour évaluer l'impact des annotations erronées sur la performance des modèles de prédiction. Ils ont découvert que la performance était particulièrement affectée par le taux de corruption du set de test : le set de test représente souvent 10-20% des données totales, et ce sont les données utilisées pour tester la performance du modèle pour déterminer si le réseau devrait adapter ou non ses paramètre et continuer d'être entraîné.

  D'après ces chercheurs [55], le moindre taux de corruption du set de test lors de l'entraînement engendre une baisse significative de la performance des modèles utilisés. Cela pourrait mettre en lumière l'importance d'un set de test de qualité par rapport par exemple aux 80-90% des données d'entraînement et de validation.

Cette partie de la recherche m'a paru particulièrement intéressante. Nous pourrions envisager par exemple de fournir au modèle un set de test qui a été produit avec l'aide de passionnés d'ornithologie, puis nous pourrions envisager d'avoir recours à d'autres datasets ou de ne pas assurer un suivi aussi rigoureux pour le reste des données dont on a besoin. Cela pourrait peut être nous permettre de concilier, à un certain niveau, le besoin de qualité et de quantité pour des meilleures performances.

2.2.4. Réflexions

Dans le cadre de ce projet, je pars de l'hypothèse qu'améliorer la qualité pourrait engendrer une meilleure performance des modèles de prédiction pour la FGIA. J'aimerais donc explorer une méthode de création de dataset qui ferait appel au crowdsourcing pour la collecte et l'annotation des données.

En suivant la réflexion de Van Horn et collaborateurs [55], j'aimerais que le projet cible davantage les membres de la communauté ornithologique, afin de pouvoir s'assurer que

les utilisateurs de ce site soient tous munis de connaissances suffisantes pour l'observation et l'identification des oiseaux.

L'élaboration de ce projet est ancrée dans l'idée que la construction de dataset peut à la fois se faire en s'assurant de la qualité dans les données récoltées, et en apportant également quelque chose d'utile à la recherche en ornithologie.

Je pense aussi que de localiser la récolte des données sur une région spécifique pourrait permettre de concentrer les efforts de la communauté sur les espèces en danger et/ou propres à une région. L'esprit de communauté aura certainement un effet sur la volonté des individus à contribuer à la collecte.

Finalement, rendre les données plus fiables et plus locales pourrait contribuer autant aux avancées en FGIA qu'aux connaissances ornithologiques.

Voilà quelques-unes des réflexions qui m'ont mené à choisir mon projet de recherche.

3
Cadre

3.1. Description formelle **12**
3.2. Fonctionnalités retenues **12**
3.3. Algorithme de consensus **13**
3.3.1. Weighted Majority Voting 13
3.3.2. Notation 14
3.3.3. Mise en situation 15
3.3.4. Exploration 16
3.3.5. Critères de décision 19

Dans ce chapitre, nous allons explorer quelques décisions qui ont dû être prises afin de poser le cadre du projet. Nous commencerons par présenter une description détaillée du projet ainsi que les fonctionnalités que j'ai choisi d'intégrer. Ensuite, nous examinerons les choix algorithmiques qui ont guidé le développement.

3.1. Description formelle

J'aimerais créer une application web destinée aux amateurs et experts en ornithologie, sur laquelle ils pourront collaborer ensemble pour la création et la gestion d'une banque d'images d'oiseaux de Suisse.

Ceci dans le but d'obtenir, à terme, un dataset conséquent et fiable d'images d'oiseaux de Suisse.

3.2. Fonctionnalités retenues

Afin de mener à bien ce projet, j'ai déterminé qu'il faudra implémenter en priorité les fonctionnalités suivantes :

- Un système de gestion des comptes utilisateurs
- Un système de gestion collaborative de la banque d'images. Pour ce faire, les utilisateurs doivent pouvoir :

- importer leurs propres photos d'oiseaux
- consulter la banque d'images
- signaler les photos qu'ils jugent inadéquates
- choisir une photo et proposer une tentative d'identification pour l'espèce qui y figure.

Pour l'annotation des données, j'aimerais tirer parti du principe de la sagesse des foules (wisdom of the crowd) : il s'agit d'une hypothèse selon laquelle l'opinion d'un groupe est plus fiable et moins volatile que l'avis d'un seul individu. [52]

Afin d'obtenir des étiquettes plus fiables, j'aimerais faire en sorte que l'on obtienne plusieurs tentatives d'identification par photo, et de cette manière on pourra évaluer quelles espèces reviennent le plus fréquemment dans les votes du groupe. On pourra ainsi émettre une décision lorsque le résultat est suffisamment décisif.

Pour ce faire, il faudra choisir et implémenter un algorithme de consensus pour que les tentatives d'identification des utilisateurs puissent aboutir un jour à une décision.

3.3. Algorithme de consensus

3.3.1. Weighted Majority Voting

A l'issue de mes recherches sur le sujet, j'ai choisi d'implémenter l'algorithme du "Weighted majority voting" (WMV), ou "vote pondéré de la majorité" pour la prise de décision lorsqu'il y a le vote de plusieurs individus à prendre en compte.

Dans les grandes lignes, l'algorithme WMV nécessite à ce que l'on puisse quantifier la "confiance" que l'on peut avoir pour la tentative d'identification d'un participant. Pour une photo donnée, à chaque espèce proposée, on calcule la somme de tous les degrés de confiance qui la concerne, et pour émettre une décision, on choisit l'espèce qui a la plus grande somme.

C'est une des méthodes de "label aggregation" (regroupement d'étiquettes) les plus simples, et pourtant, sa performance a tendance à être évaluée favorablement selon les cas d'utilisation. [47] [62]

Afin de déterminer la "confiance" que l'on peut avoir pour une tentative d'identifiation, j'ai décidé de structurer le processus ainsi : lorsqu'un utilisateur émet une tentative d'identification, il doit choisir l'espèce qu'il souhaite parmi une liste fermée des espèces d'oiseaux de Suisse. L'utilisateur doit également fournir un taux de certitude jusqu'à cent afin de représenter à quel point il est persuadé de sa propre réponse. De plus, à chaque création de compte utilisateur, on y attribue un "statut de membre" qui est représenté par une certaine pondération. Les experts en ornithologie peuvent, à travers l'application, déposer une demande de vérification de leur statut, et lorsque cette demande est approuvée, l'utilisateur se verra attribué un statut d'expert avec une plus forte pondération.

Le produit du degré de certitude ainsi avec la pondération du statut de membre représentera le poids d'une tentative d'identification pour l'algorithme du WMV. [60]

3.3.2. Notation

Voyons à présent une notation formelle de l'algorithme, en introduisant quelques formulations mathématiques :

* Soit S l'ensemble des espèces disponibles

$$S = \{\ s_i \texttt{ tel que } i = 1..|S|\ \} \tag{3.1}$$

* Soit U l'ensemble des utilisateurs

$$U = \{\ u_j \texttt{ tel que } j = 1..|U|\ \} \tag{3.2}$$

* Soit P l'ensemble des photos dans la base de données

$$P = \{\ p_k \texttt{ tel que } k = 1..|P|\ \} \tag{3.3}$$

* Soit $w_{i,j,k}$ le poids que l'utilisateur u_j attribue à l'association de l'espèce s_i à la photo p_k. En d'autres termes, $w_{i,j,k}$ désigne le degré de certitude que l'utilisateur u_j à choisi lors de la tentative d'identification de la photo p_k.

* Soit f_j l'indice de fiabilité de l'utilisateur u_j (soit, la pondération du statut de membre).

* Nous pondérons la tentative d'identification avec l'indice de fiabilité f_j :

$$w'_{i,j,k} = w_{i,j,k} \ * \ f_j \tag{3.4}$$

  S'il n'existe pas de tentative d'identification pour la photo p_k de la part de l'utilisateur u_j, la pondération est simplement fixé à zéro.

* Pour vérifier le degré de confiance qu'un label s_i corresponde à la photo p_k, nous faisons la somme de tous les degrés de certitude en faveur du label s_i. Nous désignons cette somme par $C_{i,k}$:

$$C_{i,k} = \sum_j \ w'_{i,j,k} \tag{3.5}$$

* Ensuite, la décision de quelle espèce/label correspond le mieux à la photo p_k se définit par $D^{WMV}(p_k)$, on la décrit comme telle :

$$\begin{gathered} D^{WMV}(p_k) = s_{i_{max}} \texttt{ tel que} \\ C_{i_{max},k} = \underset{i}{Max}(C_{i,k}) \end{gathered} \tag{3.6}$$

3.3.3. Mise en situation

Maintenant, nous avons un algorithme pour émettre une décision à n'importe quel moment quant à l'étiquette de p_k. Mais si par exemple il n'y a que quelques utilisateurs qui ont identifié la photo, on peut difficilement considérer que la photo soit classée et que son étiquette soit fiable. Afin de récolter suffisamment de prédictions pour que le résultat du WMV soit fiable, il faudra déterminer des critères pour la prise de décision. Pour ce faire, il serait bien d'en apprendre un peu plus sur la manière dont le WMV fait évoluer ses valeurs lorsque le nombre de tentatives d'identification augmente.

Commençons par présenter l'exemple d'une photo p_k qui reçoit un certain nombre de tentatives d'identification. Sur cette photo figure un oiseau noir et les utilisateurs sont plutôt d'avis que c'est un merle. Une personne détermine que c'est un corbeau et il met un degré de certitude relativement haut. Supposons pour la simplicité que tous ces utilisateurs sont des membres ordinaires et l'indice de fiabilité est négigeable. Voyons alors la liste des tentatives d'identification dans l'ordre d'apparition.

Ordre	Espèce	Certitude
1	merle	80
2	corbeau	80
3	merle	50
4	merle	40
5	merle	90
6	merle	90

(3.7)

Figure 3.1 montre comment la somme des degrés de confiance pour ces deux espèces évolue avec le WMV :

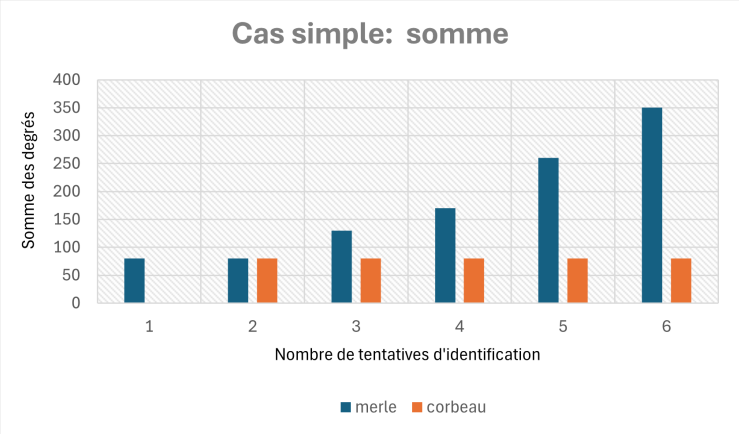


Figure 3.1. – Évolutions des degrés de confiance sur un cas simple

Nous pouvons remarquer que l'écart d'une espèce à l'autre se démarque déjà à la quatrième identification, bien que le degré de confiance de cette dernière n'est pas très haut (40% de certitude).

Nous pourrions envisager de mettre en place un critère que lorsqu'une espèce donnée est particulièrement évidente pour les utilisateurs, et que la somme correspondante se

démarque suffisamment, alors une décision peut être prise pour classifier la photo avec cette espèce.

D'un autre côté, il faut se demander si un comité de 4 à 6 personne peut être considéré comme suffisant pour émettre une bonne décision. Après tout, la photo pourrait être assez ambigüe et d'autres utilisateurs peuvent venir après et déterminer que la bonne étiquette devrait être autre chose.

Il vaut donc la peine d'explorer d'autres situations pour déterminer des critères suffisamment fiables. Avant cela, j'ai trouvé qu'il serait davantage intéressant de transformer l'ensemble de données en une distribution des proportions plutôt qu'une somme de votes. D'une part parce que les valeurs seraient plus intuitives et interprétables si l'échelle allait plutôt de 0 à 1. D'autre part aussi, car il sera plus facile pour nous de déterminer un seuil avec un ensemble de nombre fermé.

L'ensemble des données sera donc normalisé par la proportion que $C_{i,k}$ occupe (pour la photo p_k et l'espèce s_i) par rapport au reste des tentatives d'identification qui concernent d'autres espèces.

$$C^n_{i,k} = \frac{C_{i,k}}{\sum_{x \in S} C_{x,k}} \tag{3.8}$$

Vous pouvez aperçevoir comment cette transformation change la distribution des valeurs sur la figure 3.2

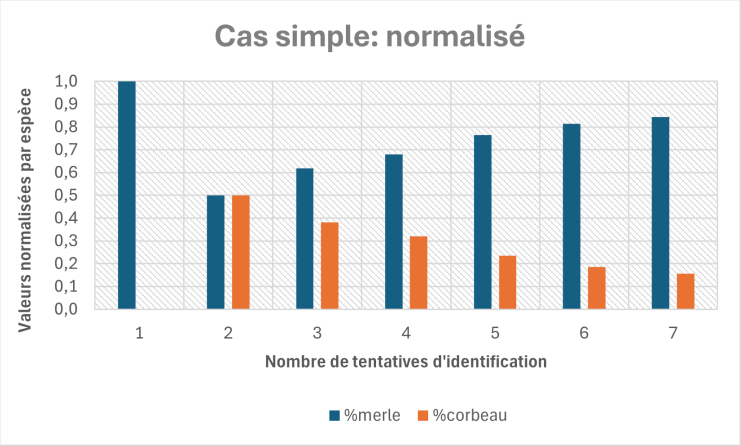
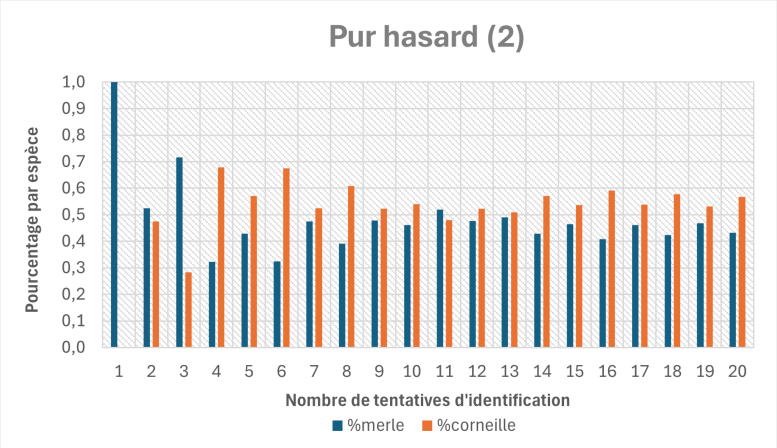


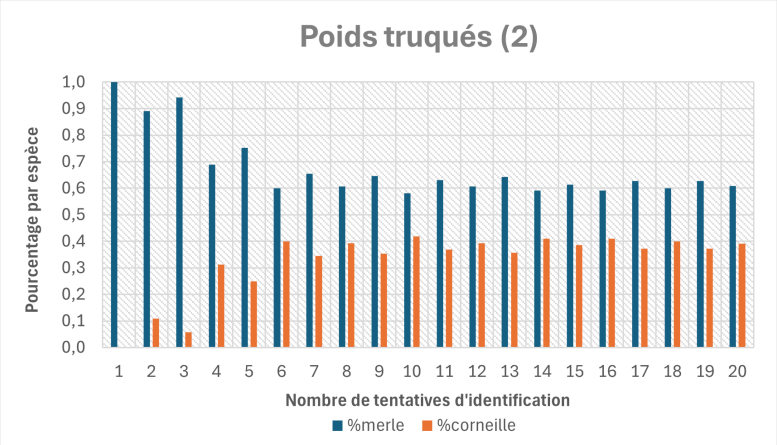
Figure 3.2. – Évolution des probabilités pour un cas simple

3.3.4. Exploration

Afin de trouver des bons critères pour la prise de décision, voyons quelques situations supplémentaires représentées avec notre nouvelle échelle de valeurs :

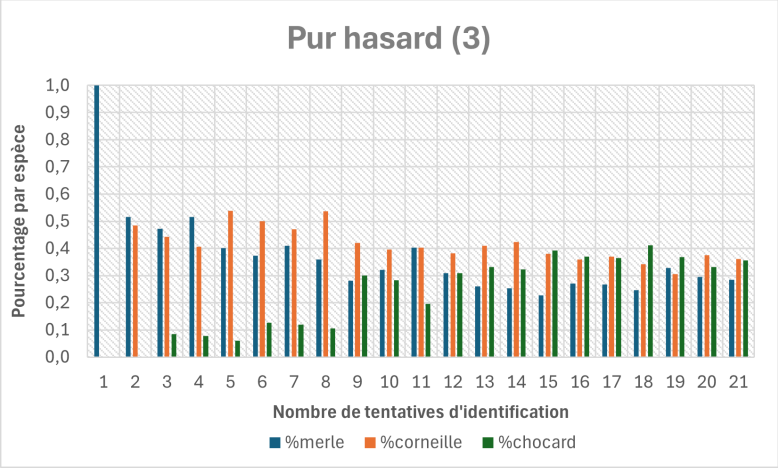


(a) Valeurs générées au hasard

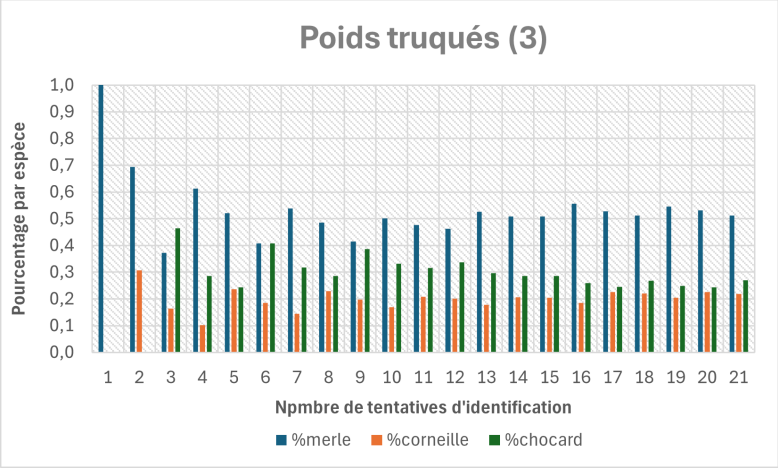


(b) Valeurs truquées en faveur d'une espèce

Figure 3.3. – Évolution des probabilités pour deux espèces qui s'alternent



(a) Valeurs générées au hasard



(b) Valeurs truquées en faveur d'une espèce

Figure 3.4. – Évolution des probabilités pour trois espèces qui s'alternent

Figure 3.3 représente deux situations où se sont alternés, pour 10 itérations, consécutivement un vote pour le merle noir et un vote pour la corneille. Sur la figure de gauche, le degré de confiance a été déterminé au hasard de 1 à 100. Sur la figure de droite, les votes en faveur du merle ont été "truqués" afin de n'obtenir que des valeurs allant de 50 à 100, afin de simuler une tendance générale à reconnaître davantage le merle plutôt qu'une autre espèce.

Après avoir généré les valeurs plusieurs fois, je remarque tout d'abord qu'au bout de quelques tentatives d'identification, il n'y a plus de variations trop extrême dans la distributions des votes. Les premières itérations sont plus volatiles par comparaison : par exemple, dans la figure 3.3(a), l'écart entre les deux valeur s'éleve à 0,350 à partir de la sixième tentative d'identification, et à la septième, l'écart redescend à 0,053. Il est normal que les valeurs ne changent pas autant brusquement lorsque nous avons plus de tentatives d'identification et le numerateur et dénominateur ne font que prendre en valeur tout les deux.

L'écart de valeur est assez distinct et constant dans la situation truquée. En ce qui concerne la question de déterminer lorsqu'une espèce se démarque clairement de l'autre, mon idée de base était de prendre une décision lorsque l'espèce avec la plus grande valeur dépasse un certain seuil. Cela dit, nous avons sur la figure 3.4 les mêmes situations que nos graphiques sur figure 3.3 mais cette fois-ci pour trois espèces. Je me suis fait la remarque que les proportions "maximales" pourrait atteindre des valeurs relativement différentes selon si il y a peu d'espèces proposées ou si ce chiffre tend vers trois, quatre ou plus encore. Je ne pense donc pas qu'il soit avisé de prendre une décision en fonction d'un seuil fixe, sachant que le seuil des valeurs dépendrait du nombre d'espèce entre lesquelles les utilisateurs hésitent. D'une autre part, lorsque les utilisateurs sont vraiment mitigés sur une photo, il ne devrait pas être simple d'atteindre le seuil de prise de décision.

3.3.5. Critères de décision

Après davantage de réfléxion, j'ai déterminé plusieurs critères pour la prise de décision.

1. Pour les cas où l'espèce sur une photo est tant reconnaissable qu'une tendance se forme dans les votes des utilisateurs : dans l'intention de déceler ces tendances, comparée par exemple à une situation où les votes sont mitigés, je trouve qu'il serait intéressant d'observer l'écart qui se forme entre la réponse la plus favorisée en comparaison aux autres.

   Ainsi, mon premier critère sera l'écart des valeurs entre la réponse la plus grande, comparée à la seconde plus grande. Après une batterie de test j'ai fixé le seuil à 0.2.

   Les simulations avec des valeurs au pur hasard n'arrivent guère à ce genre d'écart lorsqu'il y a suffisamment de tentatives d'identification. Or, dans les premières itérations les valeurs sont volatiles et il y a bien des chances qu'une situation mitigée atteigne ce critère-ci. Sans compter la première tentative d'identification qui sera

toujours égale à 1. Pour remédier à cela, nous avons d'autres critères.

2. Afin de laisser suffisamment de temps pour que les tendances se forment, j'ai décidé de ne pas permettre la prise de décision avant un certain nombre de tentatives d'identification.

   Selon le résultat des tests, je serai encline à penser que nous devrions permettre une décision à partir du dixième vote environ. Cela dit, ce critère est entièrement dépendant du nombre d'utilisateurs qui seraient actifs à priori. Dans tous les cas, la mise en place de ces critères est relativement arbitraire et peut être changée rapidement. Ainsi dans le cadre de ce projet je trouve que de permettre la décision à partir du dixième vote est adéquat.

3. Finalement, pour les situations où la photo n'est pas très facile à identifier et que les réponses entre utilisateurs sont partagées, il se peut que les deux critères ci-dessus ne peuvent jamais être atteints. En effet, même au cours de réelles discussions de groupe, il y a bien des chances pour que le résultat soit mitigé lorsque plusieurs étiquettes paraissent viables.

   Il faut donc être en mesure de prévoir ce genre de cas, et je suis d'avis qu'il vaut mieux classer les photos aux résultats mitigés lorsque le sujet a "suffisamment mijoté", car parmi toutes les espèces d'oiseaux, il y en a naturellement qui se ressemblent. Rejeter les photos compliquées reviendrait en partie à rejeter les espèces dont l'identification est plus subtile à appréhender.

   Selon une étude [58], dans un contexte de crowdsourcing, augmenter la quantité de votes améliore la performance du groupe jusqu'à environ 20 contributeurs. Ainsi, nous pourrions tenter de fixer notre limite de vote à ce chiffre-là.

Outre ces trois points-ci, j'estime qu'un critère idéal de décision serait de garder une trace de l'évolution des valeurs dans le temps. Nous pourrions alors, par exemple, essayer de détecter lorsque les valeurs commencent à stagner vers un point donné. Nous pourrions également envisager d'explorer d'autres algorithmes de consensus, cela dit, de plus amples recherches sur le sujet seraient alors nécessaires. Je garde ce genre de réflexions pour des perspectives d'avenir si ce projet venait à être repris dans le futur.

Pour résumer, les trois critères de prise de décision pour ce projet sont les suivants :

1. Il doit y avoir un minimum de 10 tentatives d'identification faite pour une même photo.
2. Il doit y avoir un écart d'au minimum 0.2 (20% de différence) entre la valeur normalisée la plus haute et la seconde plus haute.
3. Le nombre maximum de tentatives d'identification est 20. À partir de ce nombre, la décision sera prise en faveur de la plus haute valeur (normalisée).

4

Avianity : Point de Vue Utilisateur

4.1. Pages d'accueil **21**
4.2. Gestion du compte utilisateur **23**
4.2.1. Cas : Enregistrer un nouveau compte 23
4.2.2. Cas : Connexion à son compte 25
4.2.3. Autres 25
4.3. Gestion de la banque d'images **26**
4.3.1. Cas : Déposer une nouvelle photo 26
4.3.2. Cas : Consulter la banque d'images 28
4.3.3. Cas : Générer un dataset 30
4.3.4. Cas : Tentative d'identification 31
4.3.5. Cas : Signaler une photo 32

Nous allons maintenant vous présenter l'implémentation finale du point de vue d'un utilisateur. Chacune des fonctionnalités principales de l'application sera mise en lumière à travers un cas d'utilisation. Des captures d'écran seront notre support visuel pour illustrer les explications.

Veuillez vous référez à l'annexe C pour le lien vers l'application. Ce lien deviendra peut être obsolète à l'avenir.

4.1. Pages d'accueil

Tout d'abord, les figures 4.1 et 4.2 montrent la page d'accueil. Elle ne comporte pas les mêmes composants selon si l'utilisateur est authentifié ou non.

Si l'utilisateur n'est pas authentifié, la page affichera un texte de bienvenue et les deux options d'authentification seront affichées dans la barre de navigation. Lorsqu'on est authentifié, la barre de navigation affiche à la place un icône qui mène à la page utilisateur ainsi qu'un bouton de déconnexion.

L'utilisateur doit être connecté pour avoir accès aux deux options principales de l'application : "import" nous envoie vers la fonctionnalité de dépôt de photos, et "explore"

nous envoie vers la galerie de photos, dans laquelle nous pourrons consulter les photos et interagir avec.

Autres points notables sur la barre de navigation : deux options sont toujours présentes, il s'agit d'un lien "home" qui renvoie à la page d'accueil et un lien "about us" qui mène vers une page pour en apprendre plus sur le projet (figure 4.3). Lorsque nous quittons la page d'accueil, le logo et le titre de l'application seront présents en haut à gauche de la page et cliquer dessus nous renvoie à la page d'accueil.

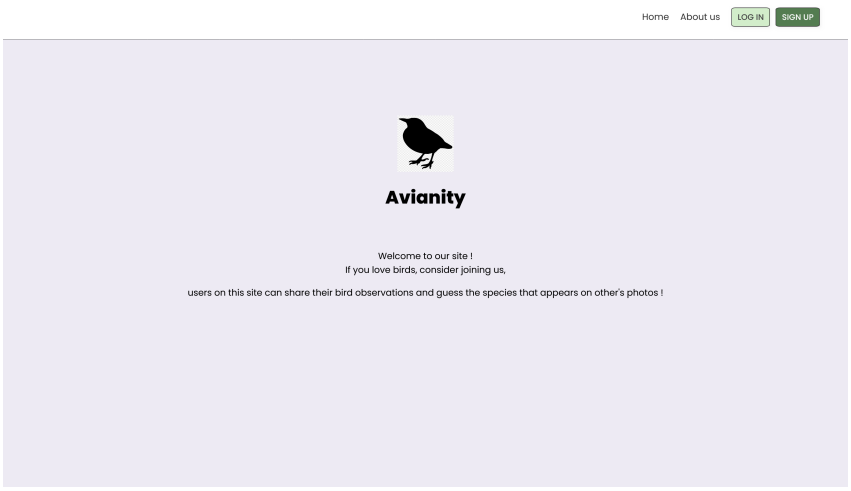


Figure 4.1. – Page d'accueil (déconnecté)

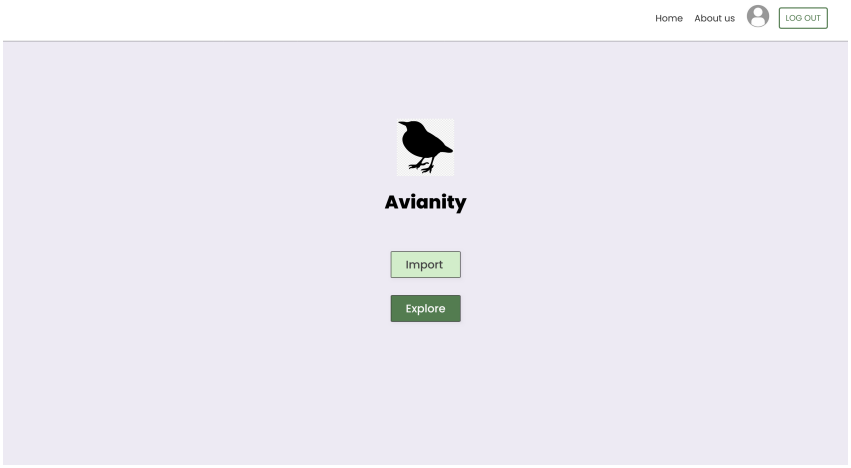


Figure 4.2. – Page d'accueil (connecté)

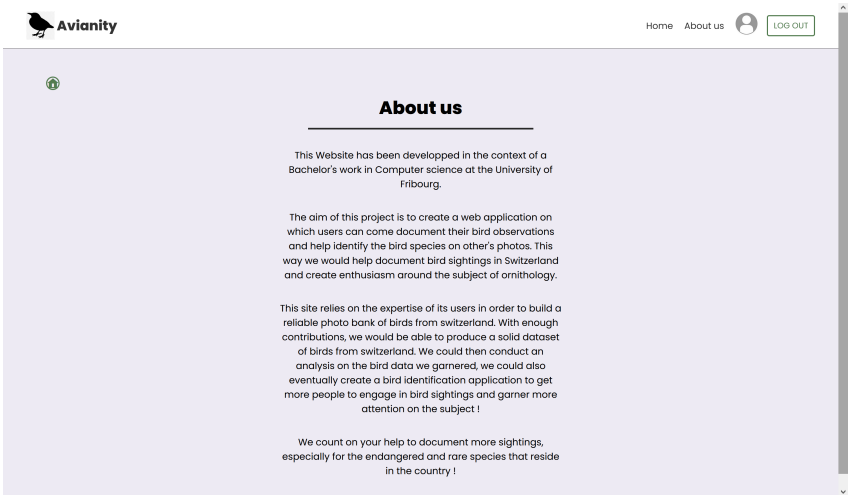


Figure 4.3. – Page à propos du site

4.2. Gestion du compte utilisateur

4.2.1. Cas : Enregistrer un nouveau compte

Lorsque l'utilisateur veut créer un nouveau compte, il sera redirigé vers une page d'inscription (figure 4.4). Sur cette page figure un formulaire d'inscription ainsi qu'un lien vers la page de connexion, dans le cas où l'utilisateur s'est trompé de bouton et souhaitait plutôt se connecter à un compte existant. Le formulaire d'inscription comporte un champ pour entrer son adresse email, le nom d'utilisateur et le mot-de-passe qu'il souhaite définir pour son nouveau compte.

Il est requis de l'utilisateur qu'il soumette une adresse email valide (et qui n'existe pas encore dans la base de donnée) ainsi qu'un mot-de-passe suffisamment sûr (tel qu'il est précisé au bas du formulaire). Dans le cas échéant, un message d'alerte sera affiché en bas du formulaire, et l'utilisateur ne pourra pas continuer le processus d'enregistrement tant que l'alerte ne sera pas levée.

Lorsque l'utilisateur aura entré des informations valides, il sera dirigé vers la page de confirmation de l'adresse email (figure 4.5). À cet instant, un code à six chiffres a été envoyé à l'adresse indiquée. Une fois que l'utilisateur aura correctement recopié ce code, la création du compte peut enfin se finaliser. L'utilisateur reçoit un message du navigateur pour l'informer du succès de l'opération, puis il est redirigé vers la page d'accueil qui maintenant affiche les fonctionnalités du site (figure 4.6).

À l'instant où l'utilisateur est authentifié, un token de porteur apparaîtra dans son stockage local (on peut vérifier cela dans la fenêtre des outils de développeur), ce qui permettra au client d'envoyer des requêtes au serveur sur des routes protégées.

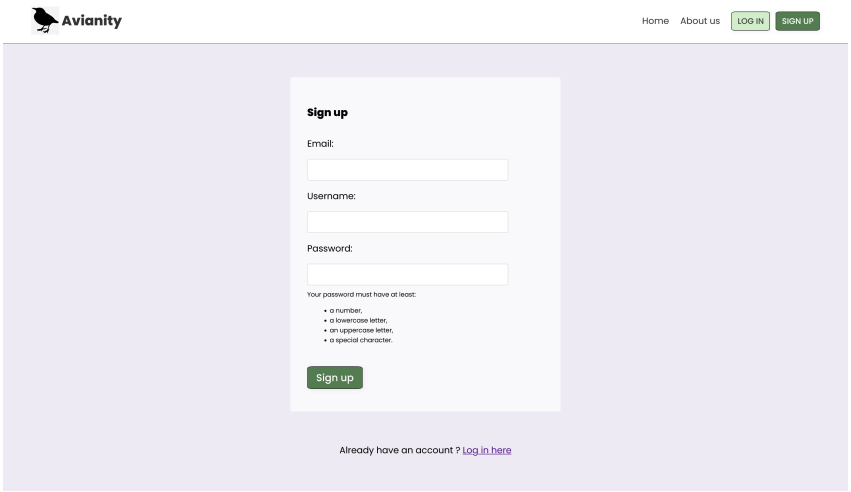


Figure 4.4. – Enregistrer un compte : formulaire d'inscription

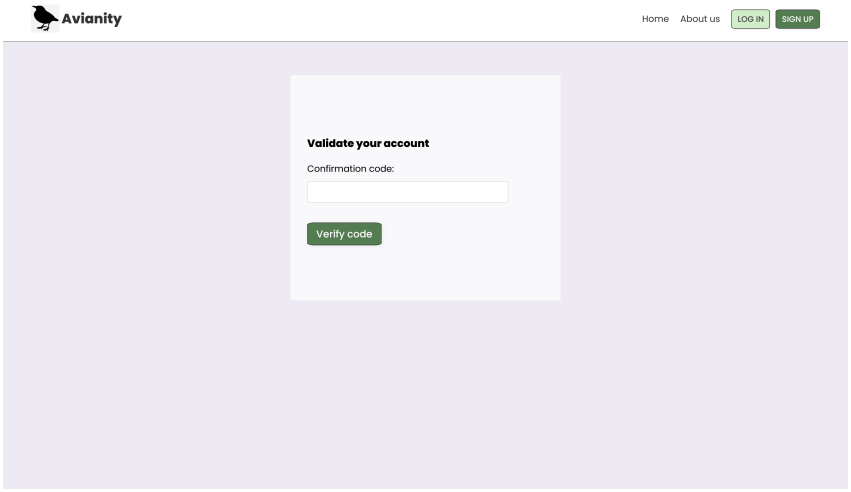


Figure 4.5. – Enregistrer un compte : confirmation par mail

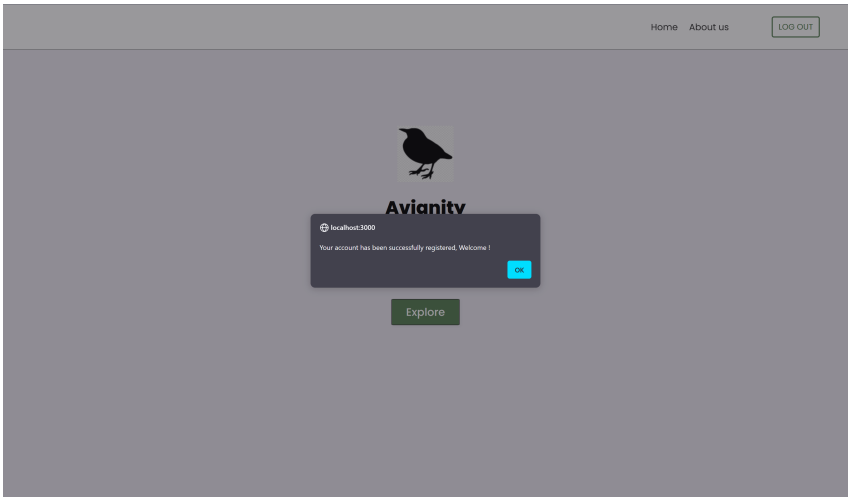


Figure 4.6. – Enregistrer un compte : message de bienvenue

4.2.2. Cas : Connexion à son compte

Lorsqu'un utilisateur veut se connecter, il sera redirigé vers une page de connexion (figure 4.7). De nouveau, cette page comporte un formulaire et un lien vers l'autre option d'authentification. Le formulaire demande à ce que l'on fournisse notre adresse email et mot-de-passe afin de se faire correctement authentifier. Si l'on se trompe pour l'adresse email ou le mot-de-passe, une alerte nous l'indiquera (nous vérifions d'abord la validité et l'existence de l'adresse email, puis nous regardons si le mot-de-passe coïncide).

Dans le cas où l'utilisateur a oublié son mot-de-passe, le lien "forgot password ?" au fond du formulaire le redirigera vers la page de récupération du mot-de-passe où il sera en mesure de changer ce dernier.

Si l'utilisateur entre des informations correctes, il sera promptement authentifié et un token de porteur apparaîtra dans son stockage local.

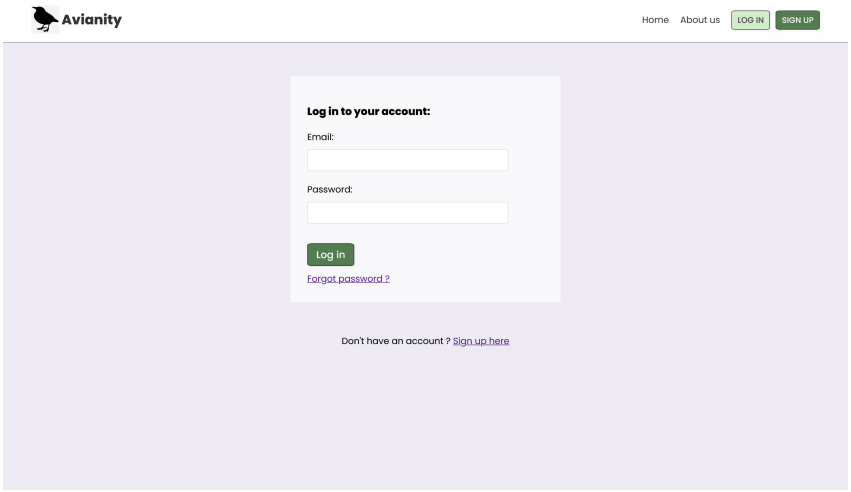


Figure 4.7. – Connexion à son compte : formulaire de connexion

4.2.3. Autres

Il reste deux fonctionnalités concernant la gestion des comptes. À commencer par la possibilité de changer son mot-de-passe (figure 4.8), que l'on peut accéder depuis la page utilisateur (voir figure 4.9), ou la page de connexion si l'on a oublié le mot-de-passe et que l'on ne peut plus se connecter. Dans les deux cas, l'adresse email de l'utilisateur est requise afin que l'on puisse lui envoyer un code de confirmation par email. Lorsque l'utilisateur inscrit son nouveau mot-de-passe et le code de confirmation, les informations de l'utilisateur sont mises à jour et un nouveau token de porteur est renvoyé par sécurité.

La dernière fonctionnalité concerne le changement de membre de l'utilisateur. Comme mentionné dans le chapitre précédent, l'utilisateur peut faire une demande auprès de la modération pour faire vérifier son statut d'expert dans le domaine de l'ornithologie.

L'option de changement de statut est accessible depuis la page utilisateur (figure 4.9). Après avoir appuyé sur "Ask to change status", il s'affiche alors un nouveau champ pour

entrer son nom entier (pour que l'administrateur puisse vérifier l'identité de la personne manuellement). Lorsque l'utilisateur confirme, un email avec des informations sur l'utilisateur est envoyé à l'adresse du site. Lorsque l'email est envoyé, un message s'affiche pour informer l'utilisateur que la requête a bien été envoyée, puis il est libre de rester sur cette page ou retourner à l'accueil.

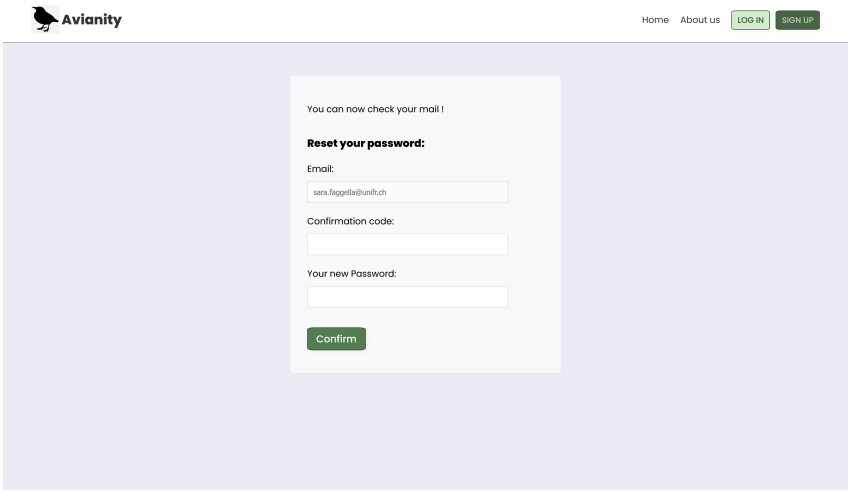


Figure 4.8. – Formulaire pour le changement de mot-de-passe

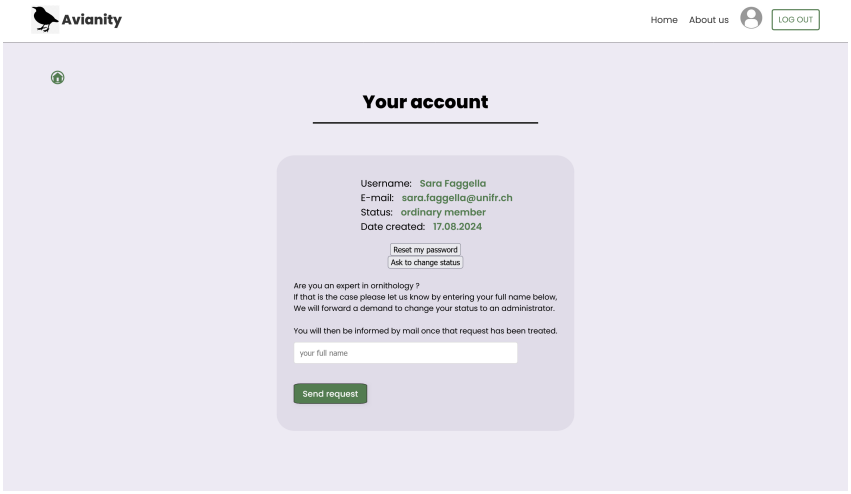


Figure 4.9. – Page utilisateur et demande de changement de statut de membre

4.3. Gestion de la banque d'images

4.3.1. Cas : Déposer une nouvelle photo

Il y a une photo d'oiseau que l'utilisateur aimerait partager sur le site. Depuis la page d'accueil, il clique sur l'option "Import" (figure 4.2). Il est redirigé sur une page où s'affichera un formulaire avec des directives et un bouton pour déposer un fichier photo (figure 4.10). La photo choisie s'affichera sur la gauche de l'écran, de cette manière, il peut véri-

fier que c'est bien celle qu'il veut importer et qu'elle correspond aux conditions du site.

Lorsque l'utilisateur valide la sélection, il appuie sur "confirm" et il sera redirigé sur une nouvelle page. Là un autre formulaire apparaît : l'utilisateur doit fournir la date et le lieu de l'observation, puis, on lui demande d'émettre une tentative d'identification avec degré de certitude, dont la valuer exacte s'affiche à la droite du champ (figure 4.11).

Lorsque l'utilisateur valide le formulaire, les informations sont envoyées au serveur. Une nouvelle photo et une nouvelle tentative d'identification seront ainsi enregistrées dans la base de données. L'utilisateur est renvoyé à une page de remerciement et il est libre de retourner à la page d'accueil (figure 4.12).

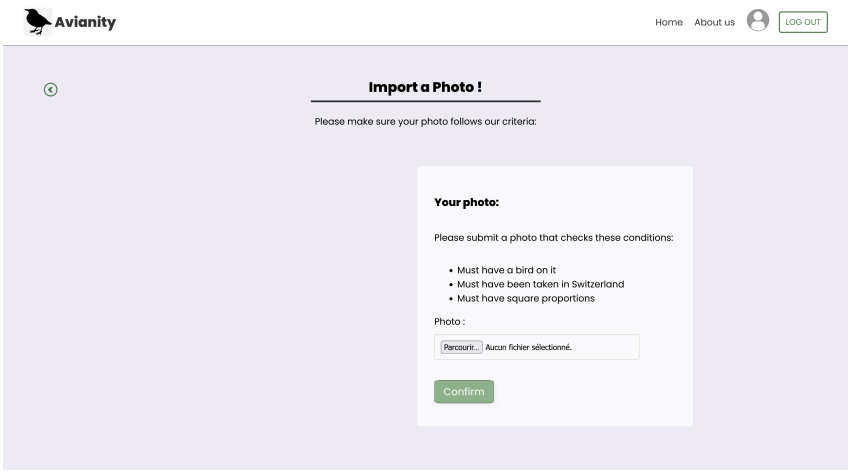


Figure 4.10. – Dépot de nouvelle photo : import de l'image

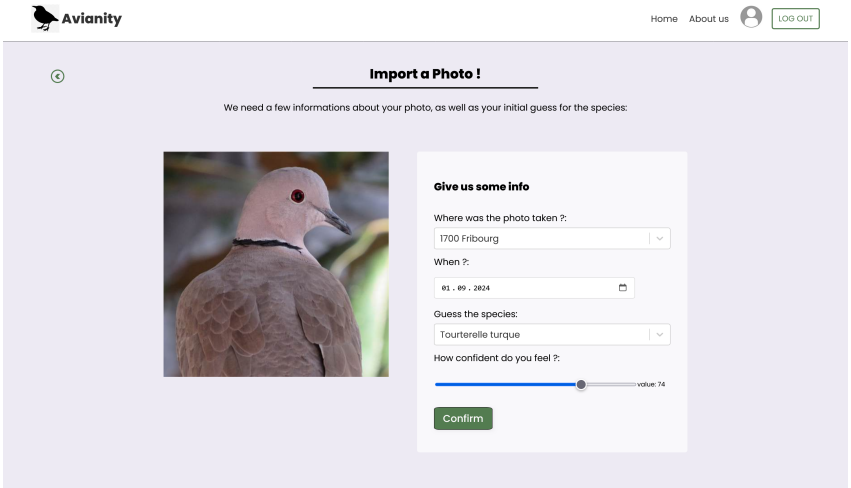


Figure 4.11. – Dépot de nouvelle photo : formulaire

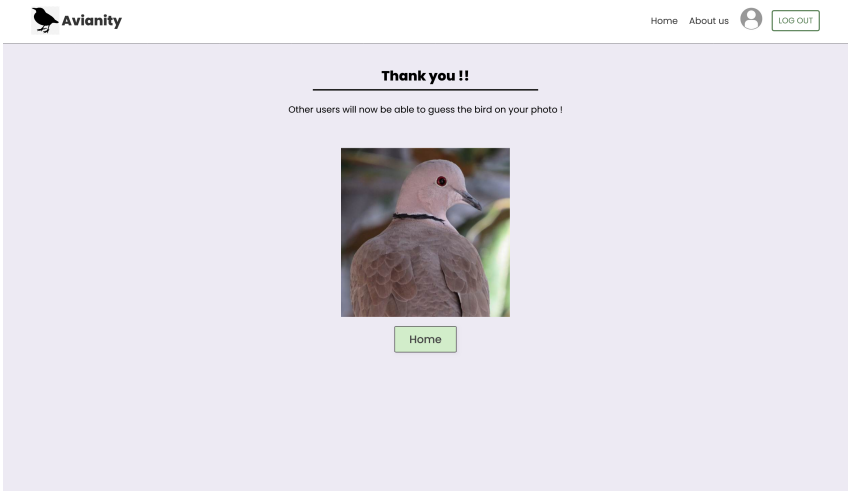


Figure 4.12. – Dépot de nouvelle photo : page de remerciement

4.3.2. Cas : Consulter la banque d'images

L'utilisateur aimerait consulter toutes les photos qui ont été déposées dans la banque d'images, il clique alors sur "explore". Il est ainsi redirigé vers une page où se trouve, au centre, une liste déroulante avec toutes les photos de la banque d'images (figure 4.13).

Lorsque l'utilisateur souhaite obtenir les détails d'une photo, il est amené à cliquer sur la photo souhaitée et presser le bouton "details". Là, une nouvelle page s'affiche avec la photo agrandie et une liste d'informations "générales" (figure 4.14). On peut s'informer du lieu et de la date de l'observation, du nom de l'utilisateur à qui appartient l'observation, ainsi que d'une liste des trois (ou moins) espèces associées à l'image qui ont les meilleurs scores à l'agorithme de consensus (voir section 3.3). Ces scores sont affichés sous la forme de probabilités.

Sur le bas de cette page se trouvent deux options : à gauche, un bouton pour signaler la photo comme inadéquate ou non identifiable, et à droite, un bouton pour soumettre une tentative d'identification. Pour une photo donnée, l'utilisateur ne peut la signaler ou l'identifier qu'une seule fois.

Retournons à la page où s'affiche la banque d'images (figure 4.13). En dessus de la galerie, nous pouvons constater la présence de deux filtres qui servent à affiner la sélection de photos. Le premier déroulant sert à filtrer les images selon si nous voulons voir toutes les photos, uniquement celles qui sont classées ou uniquement celles qui ne sont pas classées (figure 4.15 et 4.16).

Une photo est considérée classée lorsque suffisamment d'utilisateurs ont tenté d'identifier cette photo et qu'ils sont parvenus à un consensus (voir section 3.3). Le deuxième déroulant sert à filtrer les photos selon l'espèce potentielle qui y figure. Lorsqu'une espèce du déroulant est selectionnée, par exemple le "merle noir", chaque photo de la banque d'images est inspectée et celle-ci est selectionnée uniquement si l'espèce choisie fait partie

des trois espèces avec le plus haut score de l'algorithme de consensus.

Pour en revenir au filtre qui concerne le statut de l'image, si le filtre est activé et que l'on souhaite voir uniquement les photos classées, alors un troisième bouton apparaît en dessous de la galerie. Celui-ci sert à générer un dataset basé sur la sélection active de photos.

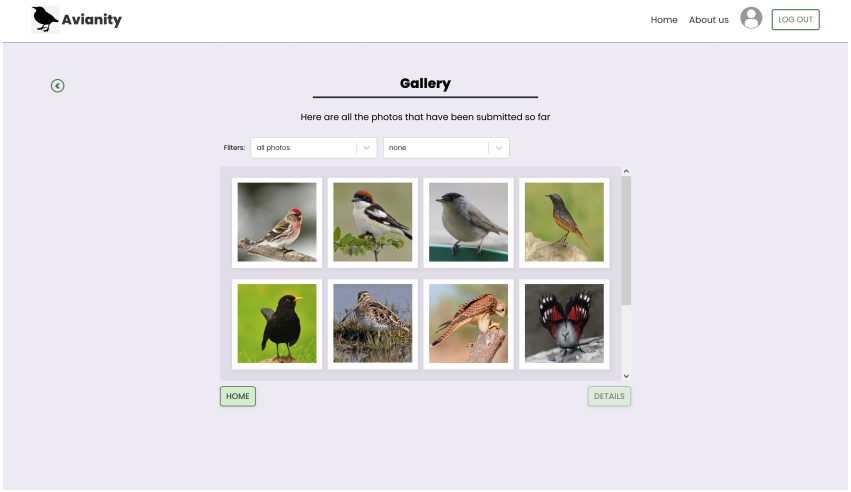


Figure 4.13. – Consulter la banque d'images : la galerie

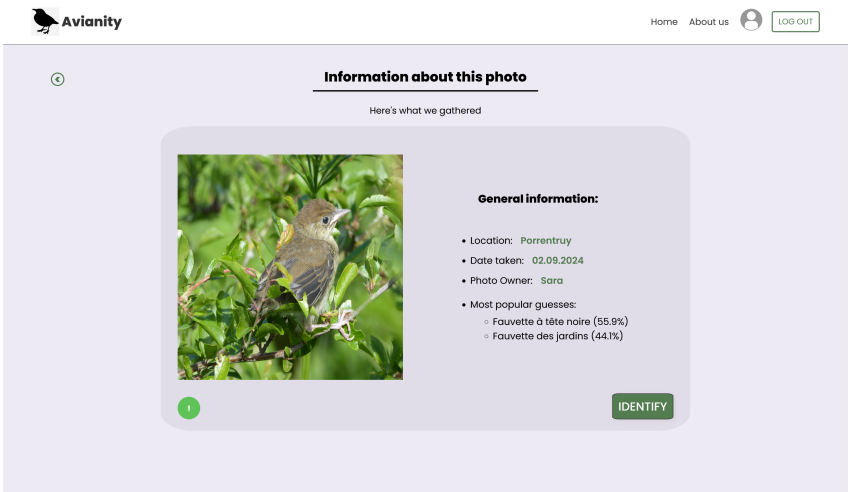


Figure 4.14. – Consulter la banque d'images : les détails pour une photo

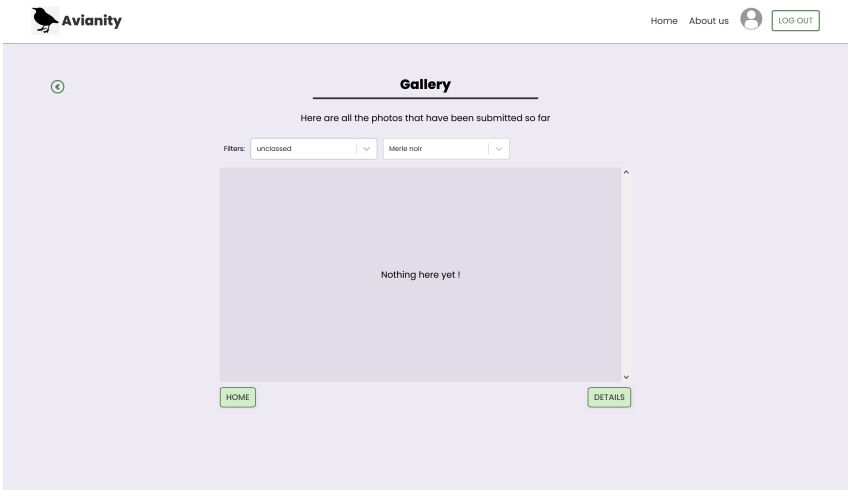


Figure 4.15. – Consulter la banque d'images : galerie vide

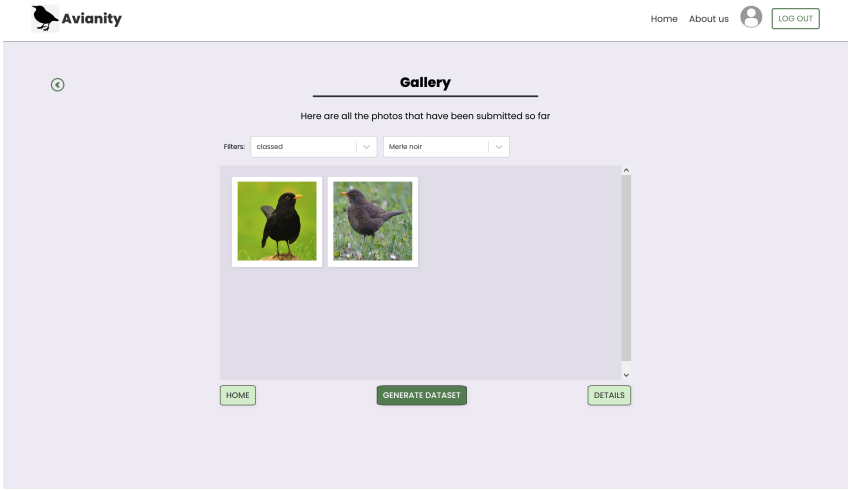


Figure 4.16. – Consulter la banque d'images : filtre "classée" activé

4.3.3. Cas : Générer un dataset

Lorsque l'utilisateur veut générer un dataset des photos classées, il doit avoir filtré la galerie pour n'avoir à l'écran que les photos classées, alors un bouton "Generate dataset" apparaîtra en dessous de la galerie (figure 4.17).

Pour qu'une photo soit classée, il nécessite que 10 utilisateurs au moins aient soumis une tentative d'identification et que l'écart entre le plus grand pourcentage et le second plus grand ait dépassé les 20%. Si l'écart reste plus bas que 20%, alors une décision ne sera prise qu'à partir de la vingtième tentative.

En appuyant sur le bouton "Generate dataset", quatre nouveaux éléments s'afficheront : au centre, une zone de texte non-modifiable où apparaîtra le dataset, et en dessous se trouve un bouton “Download images” pour télécharger le fichier zip de toutes les images sélectionées. À droite de cela se trouve un bouton pour copier le jeu de donnée dans le

presse-papier du système, et à gauche se trouve un déroulant pour choisir le format du dataset. L'utilisateur peut choisir le format JSON et CSV selon ses préférences. S'il le souhaite, il peut également trier les photos par espèce pour obtenir une sélection plus affinée.

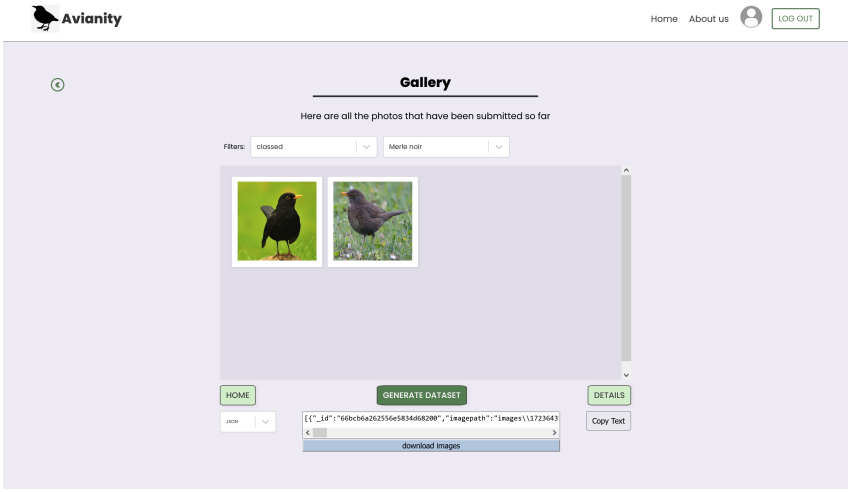


Figure 4.17. – Consulter la banque d'images : générer un dataset

4.3.4. Cas : Tentative d'identification

Revenons à la page où s'affiche les détails d'une photo sélectionnée (figure 4.14). L'utilisateur veut soumettre une tentative d'identification, il appuie donc sur le bouton "Identify". Nous sommes alors redirigés sur une nouvelle page, où l'on retrouve la photo à gauche et un formulaire à droite, pour émettre une prédiction de l'espèce (figure 4.18).

L'utilisateur est maintenant libre de choisir une espèce dans le déroulé et indiquer le degré de certitude qui lui semble approprié. Lorsque l'utilisateur a rempli le formulaire, il peut appuyer sur le bouton de confirmation. Les informations sont alors envoyées au serveur pour qu'il puisse ajouter une nouvelle tentative d'identification à la base de données. L'utilisateur est renvoyé à la page où s'affiche la galerie. Un message de remerciement apparaît alors en haut de la page (figure 4.19). L'utilisateur est maintenant libre de retourner à la page d'accueil ou de continuer à consulter la galerie.

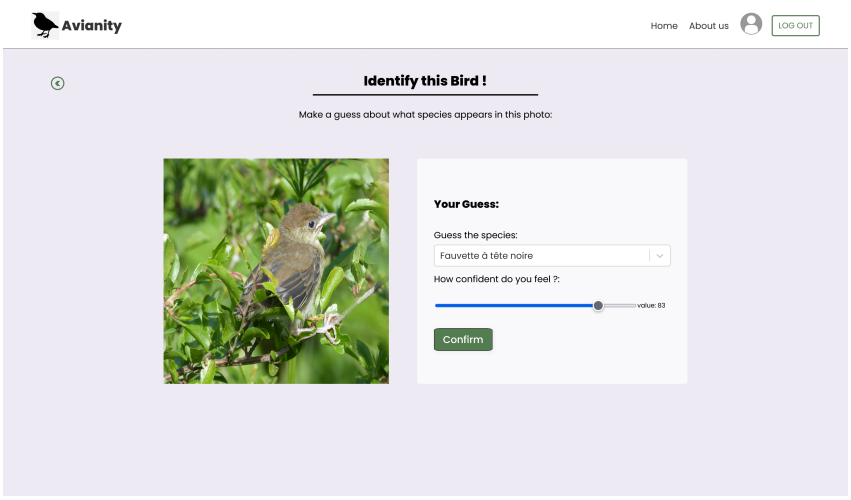


Figure 4.18. – Tentative d'identification : formulaire

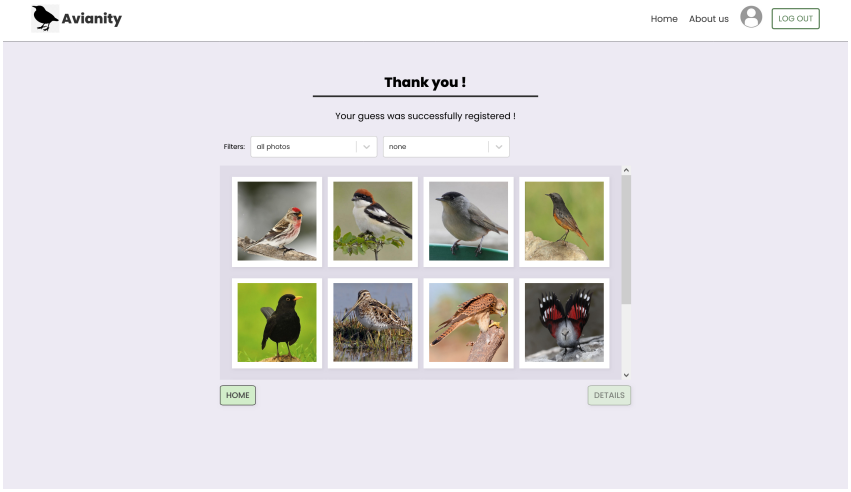


Figure 4.19. – Tentative d'identification : remerciement

4.3.5. Cas : Signaler une photo

L'utilisateur consulte la galerie et il a trouvé une photo qui lui paraît bizarre. Il fait afficher les détails de la photo et clique sur le bouton rond qui contient un point d'exclamation (figure 4.14). Il est alors redirigé vers une autre page où s'affiche la photo à gauche et une liste de conditions à droite, que la photo doit remplir pour être considérée adéquate (figure 4.20).

Après considération, l'utilisateur décide que la photo ne remplit pas les conditions et confirme sa signalisation. Le client va alors vérifier le nombre de fois que cette photo a été signalée.

Si le nombre de signalisations dépasse un certain seuil (fixé à trois par exemple), le client envoie une requête au serveur pour supprimer la photo de la base de données, ainsi que

les tentatives d'identification qui concerne cette dernière.

Si le seuil n'est pas encore dépassé, le client envoie une requête au serveur pour ajouter à la liste des signalements l'identificateur de l'utilisateur. De telle manière nous pourrons savoir que cet utilisateur a déjà signalé la photo et ne peut plus le refaire.

Lorsque la requête à été traitée, l'utilisateur est redirigé sur la page où se trouve la galerie et un message de remerciement s'affiche, dépendant de si la photo a été supprimée ou non (figure 4.21).

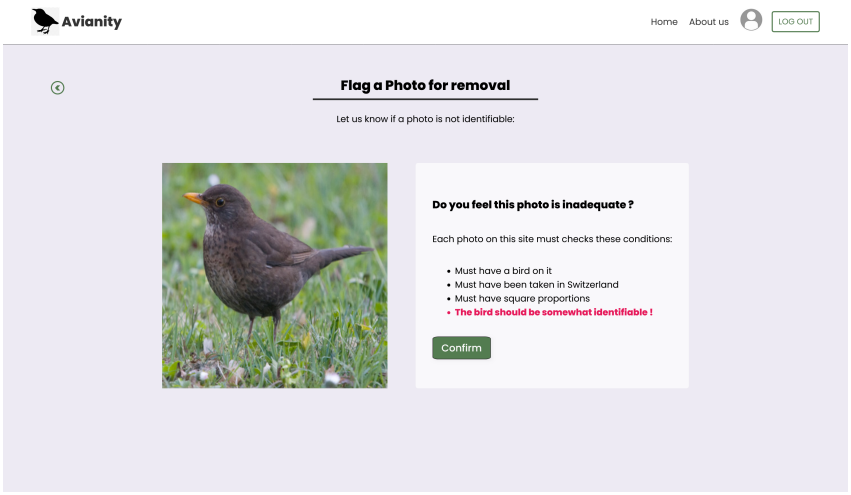


Figure 4.20. – Signaler une photo : critères de validité

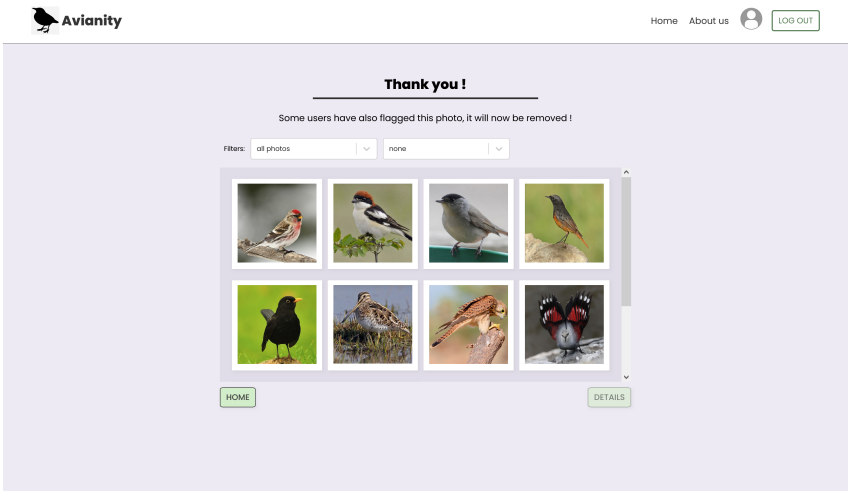


Figure 4.21. – Signaler une photo : remerciement

5

Avianity : Point de Vue Développeur

5.1. Vue d'ensemble 34
5.1.1. Stack technologique 34
5.1.2. Architecture 36
5.2. Base de données 37
5.3. Serveur 40
5.3.1. API et endpoints 40
5.3.2. Structure 42
5.4. Client 47
5.4.1. React et DOM 47
5.4.2. Structure 47
5.4.3. Routing des pages 49
5.4.4. Contextes React 50

Ce chapitre va couvrir la manière dont les fonctionnalités désirées ont été implémentées : nous commençons par une brève présentation des outils qui m'ont aidée au long du développement et du déploiement de l'application. Ensuite, nous parlerons de l'architecture globale et comment les éléments interagissent entre eux. Finalement, nous entrerons en matière sur chaque élément de l'architecture (avant tout la base de données, serveur et client) et à chaque partie, une attention particulière sera portée sur les quatres technologies principales qui assurent le bon développement de l'application (à savoir : MongoDB, Express, React et Node.js).

Veuillez vous référez à l'annexe C pour un accès à la page du projet sur Github.

5.1. Vue d'ensemble

5.1.1. Stack technologique

Ce projet a été conçu en full-stack, c'est-à-dire que l'implémentation doit couvrir toutes les couches du développement web (à savoir la base de données, le serveur et le client). [35] Pour se faire, j'ai décidé de travailler avec le full-stack prédéfini MERN [19] : il s'agit d'un stack technologique. C'est un ensemble de technologies qui interagissent ensemble pour offrir un environnement propice au développement web.

MERN et les langages employés

En l'occurrence, les technologies du stack MERN se retrouvent dans l'acronyme :

- MongoDB est une base de données NoSQL,
- Express est un framework pour mettre en place le serveur,
- React est une librairie pour faciliter le développement du client,
- Node.js est un environnement d'exécution JavaScript.

Nous verrons les détails de chaque technologie dans les sections suivantes.

Il y a plusieurs avantages à choisir un stack comme MERN pour ce projet : comme c'est un stack populaire, la compatibilité entre les technologies est relativement assurée et les ressources sont abondantes. D'autant plus, avec Express et Node.js qui ont tout deux recours à Javascript, cela implique qu'il n'y a besoin que de trois langages pour ce projet, à savoir HTML, CSS et Javascript. Ce sont les trois langages les plus importants pour le développement de l'interface utilisateur [1] :

- HTML, ou HyperText Markup Language, est un langage de Markup qui sert à donner forme aux informations présentées à l'utilisateur. Ces informations sont représentées dans le code sous forme de tags imbriqués les uns dans les autres ;
- CSS, ou Cascading Style Sheets, est employé sur les tags HTML pour y appliquer un style particulier (changer la couleur, adapter les proportions..) ;
- JavaScript est considéré comme le langage pilier du développement web. Alors que HTML et CSS servent à contrôler la présentation des informations, JavaScript sert à implémenter l'élément d'interactivité avec les utilisateurs, ainsi que de contrôler le comportement des applications web.

Logiciels

Au cours du développement et du déploiement, j'ai eu recours à quelques logiciels pour faciliter certaines parties du processus :

- MongoDB Atlas est un service d'hébergement cloud pour les bases de données Mongo. Le cloud sur lequel j'ai choisi de déployer ma base de données appartient à Amazon Web Services.
- Postman est une plateforme en ligne pour la gestion des APIs. J'ai avant tout utilisé Postman pour tester l'accès à mes endpoints et vérifier si la requête est correctement traitée.
- Git et Github m'ont servi pour mettre en ligne mon implémentation finale.
- Render est un service cloud gratuit pour héberger les applications web. Je l'ai utilisé pour tester le déploiement de mon projet.

[1] Pour de bons tutoriels sur les technologies utilisées, vous pouvez vous référez au programme en ligne Odin project [22] et/ou à l'excellent tutoriel sur le stack MERN de Net Ninja [20].

- Finalement, Cloudinary est un service cloud d'hébergement pour divers types de fichier. Je l'ai utilisé pour stocker mes fichiers photos au cours du déploiement, de telle sorte à maintenir les fichiers volumineux sur une plateforme flexible.

Librairies additionnelles

Pour conclure le stack technologie, j'ai encore utilisé quelques librairies pour enrichir certaines fonctionnalités. Pour n'en citer que quelques unes :

- Mongoose est un wrapper pour MongoDB, elle facilite les manipulations des données depuis le serveur.
- Cors est une librairie qui, dans notre cas, assure l'envoi et la réception des requêtes entre client et serveur (lorsque ces derniers ne partagent pas la même origine/adresse).
- Dotenv nous laisse définir des variables globales côté serveur.
- Json Web Tokens (JWT) m'a permis d'implémenter un système relativement simple d'authentification et de protection des routes : elle permet de générer un token unique pour chaque utilisateur avec l'identifiant de la personne et une chaîne secrète stockée avec Dotenv. Ensuite ce token est envoyé au client lorsque l'utilisateur est correctement authentifié. De telle manière lorsque le client envoie une requête vers le serveur, le token est intégré au header (voir un exemple dans le listing 5.6), nous n'avons ensuite qu'à vérifier que le token soit valide avant de traiter la requête.

5.1.2. Architecture

Maintenant que nous avons passé en revue les diverses technologies qui constituent l'écosystème de l'application, voyons comment nous avons structuré cette dernière.

J'ai choisi pour ce projet une architecture à trois niveaux : ce type d'architecture sépare clairement trois différentes composantes : le niveau présentation où se trouve le client/interface utilisateur, le niveau application où les requêtes du client sont traitées, puis le niveau données, où les documents sont stockés et récupérés.

Chaque niveau de l'architecture est séparé l'un de l'autre d'un point de vue logique et aussi physique, ce qui veut dire que le client, serveur et base de données tournent tous sur un serveur qui leur est propre et ils peuvent donc être hébergés sur différentes plateformes. Cela veut aussi dire que leur implémentation doit être séparée l'une de l'autre avec chacune sa propre logique. La communication entre les niveaux se fait donc sous la forme de requêtes. Vous trouverez un aperçu de l'architecture sur la figure 5.1.

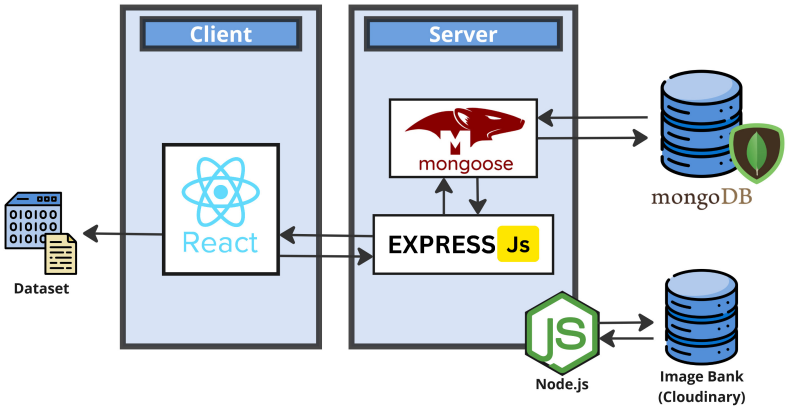


Figure 5.1. – L'architecture de l'application

Dans le cas de la base de données, l'utilisation de MongoDB Atlas m'assure d'emblée une base de données fonctionnelle et hébergée sur le cloud.

Cela dit, j'utilise aussi une autre entité de stockage pour ce projet, afin de pouvoir entreposer les photos des utilisateurs sans avoir à surcharger MongoDB avec des fichiers potentiellement volumineux. Lors du développement, j'attribue cette charge au système de fichiers du côté serveur. Cela est réalisé avec la librairie Multer pour Node.js, qui nous permet d'accéder au système de fichiers et traiter les requêtes contenant des images. Cela étant dit, le service cloud "CLoudinary" a été utilisé lors de la phase de déploiement, d'une part pour anticiper l'import de nombreux fichiers photos, et d'autre part, cela nous assure un certain niveau de sécurité si le déploiement de notre application venait à être interrompu, ainsi les photos qui ont été déposées jusqu'à maintenant résident sur une autre plateforme et il n'y a donc pas de risque de les avoir perdues. [3] [10]

Du côté serveur, le framework Express nous permet de recevoir et traiter les requêtes du client, et comme j'ai choisi d'utiliser Mongoose pour faciliter la manipulation des données, c'est au rôle de Mongoose d'envoyer les requêtes à MongoDB pour récupérer ou modifier les documents.

Côté client, nous avons React qui facilite l'implémentation du routing des pages. En naviguant l'interface client, les utilisateurs déclencheront des requêtes vers le serveur pour obtenir ou déposer des informations vers la base de données. Lorsque la base de données contient suffisamment de photos classées, le client doit pouvoir offrir à l'utilisateur la possibilité de télécharger les données sous forme de dataset.

Maintenant que nous avons un aperçu de l'architecture, nous pouvons entrer dans les détails de l'implémentation. Nous allons présenter chacune des trois composantes principales de l'application, à savoir la base données, le serveur et le client.

5.2. Base de données

MongoDB est notre type de base de données pour cette application. C'est une base de données noSQL, c'est-à-dire que les données ne sont pas stockées sous forme de table avec

un élément par ligne et un attribut par colonne, tel que nous le ferions dans une base de données relationnelle. À la place, MongoDB stocke chaque élément sous format JSON au sein de ce qu'on appelle un document. Plusieurs documents peuvent être regroupés à l'intérieur d'une collection. Il n'ont pas pour autant besoin de partager les mêmes attributs, car MongoDB ne requiert pas de structure uniformisée pour une même collection. Ceci permet à la structure des données d'être modifiée au fil du temps.

Comme il a été discuté dans les sections précédentes, la structure des données a été réalisée avec la librairie Mongoose. C'est au travers de Mongoose que je définis mes schémas. Ainsi à chaque création ou modification de document, Mongoose vérifie que la forme du document respecte le schéma qui a été imposé pour ce type d'objet. Mongoose utilise les termes de schémas et modèles, ceux-ci sont équivalents à l'emploi habituel des termes : un schéma Mongoose définit quels attributs est-ce que l'élément doit posséder. Puis, un modèle Mongoose peut être défini comme le constructeur pour créer un document. Il doit être associé à un schéma et tout document qui sera ensuite créé à l'aide du modèle sera soumis aux limites imposées par le schéma.

Sur la figure 5.2, un ERD (Entity-Relationship Diagram, notation de Chen) a été défini pour déterminer quelle sera la relation entre les trois entités principales pour notre application, à savoir, les utilisateurs ("User"), les photos ("Photo"), et les tentatives d'identification ("Guess").

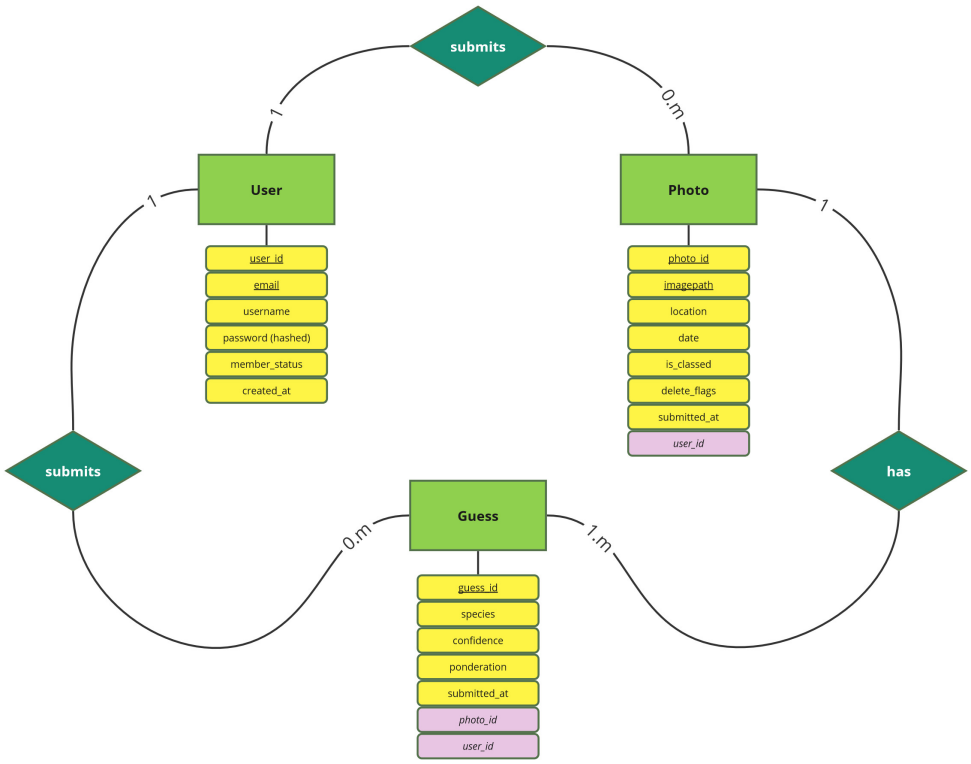


Figure 5.2. – Un schéma ERD représentant les trois entités principales de l'application

Les foreign keys sont représentées en rose dans la figure 5.2. Un utilisateur peut soumettre autant de photos qu'il le souhaite, mais il ne peut émettre qu'une tentative d'identification par photo. Dans les schémas Mongoose, nous avons plusieurs options pour représenter les

foreigns keys. Personnellement, j'ai choisi de rajouter en attribut l'identifiant des objets à référencer, puis, lorsque nous aurons besoin par exemple de récupérer les tentatives d'identification qui concernent une photo spécifique, nous pouvons filtrer la recherche de manière à ce que l'attribut `photo_id` soit équivalent à l'identifiant de la photo. Vous pouvez observer sur le listing 5.1 comment le schéma pour les documents photo a été implémenté.

```
1  const mongoose = require('mongoose')
2  const Schema = mongoose.Schema
3
4  // Mongoose schema for the photo document
5  const photoSchema = new Schema(
6      {
7          imagepath: { type: String, required:true },
8
9          location: { type: String, required: true },
10
11         date: { type: String, required: true },
12
13         is_classed: { type: Boolean, default: false },
14
15         delete_flags: [{ type: String }],
16
17         user_id: { type: String, required: true }
18
19     }, { timestamps: true } )
```

Listing 5.1 – La création d'un schéma Mongoose pour les documents photo.

Chacun de nos schémas sont stockés sur un fichier javascript qui leur est propre. Une fois que le schéma d'un élément a été défini, il reste encore à y associer un modèle Mongoose ainsi que le nom de la collection qui stockera les instances du modèle dans la base de données. La commande qui figure dans l'extrait 5.2 nous montre comment créér un modèle Mongoose. `'Photo'` est le nom de la collection et en second argument nous donnons la référence pour le schéma Photo que nous avons décrit dans l'extrait 5.1. Nous exportons le modèle afin de l'utiliser plus tard dans des méthodes dédiées au traitement des requêtes (voir un exemple dans le listing 5.7).

```
1  // Export the Mongoose Photo schema as a model
2  module.exports = Mongoose.model('Photo', photoSchema)
```

Listing 5.2 – La création d'un modèle Mongoose pour les documents photo.

Une fois que les modèles sont définis, il n'y a plus qu'à les utiliser : nous faisons appel alors à une autre fonctionnalité de Mongoose, c'est celle de faciliter les requêtes à la base de données pour créer, supprimer ou modifier les documents. Nous verrons dans un instant comment les requêtes sont traitées par le serveur et quand est-ce que Mongoose intervient pour la manipulation des données.

5.3. Serveur

Comme nous l'avons présenté, le niveau application, ou backend, doit contenir tout ce qui concerne le traitement des requêtes.

Pour ce faire, les étapes du routing peuvent se répartir ainsi :

- Premièrement, le client devrait avoir à disposition la liste des requêtes qu'il peut faire, il doit pouvoir connaître l'adresse de ces requêtes (les endpoints) ainsi que d'éventuelles informations complémentaires (si la requête a besoin d'un corps, de headers,..). Ce que l'on décrit-là s'appelle l'API, où l'Interface de Programmation d'Application, et désigne plus généralement toutes les informations et méthodes qui permettent la communication entre deux logiciels.

- Maintenant que le client peut envoyer des requêtes, il faut être en mesure d'intercepter ces dernières du côté serveur. Il faut que l'application écoute continuellement sur une adresse donnée, et lorsqu'il reçoit une requête, il doit chercher si l'adresse spécifiée par la requête mène à un endpoint qui existe.

- Lors du processus de routing, le routeur exécute du middleware s'il y a lieu de le faire (par exemple, nous pouvons exécuter du code pour s'assurer de la protection des routes/ vérifier la validité du token porteur).

- Si le routeur a trouvé le bon endpoint, il confie le traitement de la requête à la méthode qui lui est attribuée.

5.3.1. API et endpoints

Nous allons d'abord présenter brièvement l'API pour ce serveur. Nous utilisons le style d'architecture REST pour notre api, qui nous fournit un format standardisé pour les types de requêtes que l'on peut faire. Les types de requêtes couvrent tout ce dont on a besoin : GET (récupérer), PUT/PATCH (modifier), POST (créer) et DELETE (supprimer). Sur cette base et avec les schémas que nous avons créés pour notre base de données, nous pouvons définir les endpoints qui constitueront l'interface de communication entre le client et le serveur.

Veuillez vous réferrez à la figure 5.3 pour les endpoints les plus importants pour ce projet, excluant sept requêtes qui concernent les utilisateurs et deux requêtes pour respectivement récupérer une liste fermée des espèces d'oiseaux de Suisse et une liste fermée des communes de Suisse avec code postal, ceci dans un soucis de clarté. Nous associons à chaque endpoint une description de la requête ainsi qu'une description de la réponse attendue.

GET	/api/photo/all
Description	Récupère toutes les photos selon les filtres précisés
Réponse	Retourne la sélection

GET	/api/photo/:id
Description	Récupère un document photo avec l'ID
Réponse	Retourne le document

PATCH	/api/photo/:id
Description	Met à jour les données modifiables du document
Réponse	Retourne le document modifié

DELETE	/api/photo/:id
Description	Supprime un document et le fichier associé
Réponse	Retourne le document supprimé

POST	/api/photo
Description	Créé un nouveau document photo
Réponse	Retourne le nouveau document

(a) Pour la gestion des photo

GET	/api/guess/all
Description	Récupère toutes les tentatives d'ID selon les filtres précisés
Réponse	Retourne la sélection

GET	/api/guess/likely/:id
Description	Récupère les 3 meilleurs scores de confiance pour une photo
Réponse	Retourne 3 paires score-espèce

DELETE	/api/guess/:id
Description	Supprime toutes les tentatives d'ID qui concerne une photo
Réponse	Retourne ce qui a été supprimé

POST	/api/guess
Description	Créé une nouvelle tentative d'identification
Réponse	Retourne le nouveau document

(b) Pour la gestion des tentatives d'identification

Figure 5.3. – Les requêtes possibles pour notre application

5.3.2. Structure

En ce qui concerne l'implémentation du serveur, nous allons voir en quoi le framework Express facilite ce processus. Vous pouvez vous référez à la figure 5.4 pour la structure des fichiers du serveur.

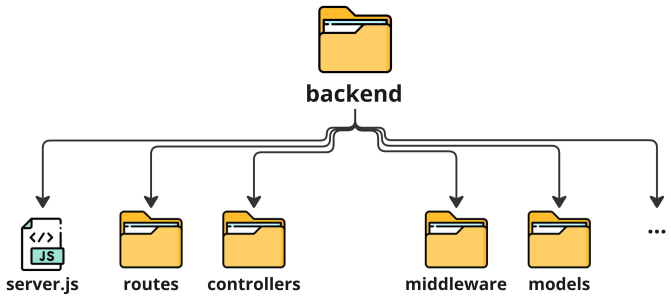


Figure 5.4. – La structure des fichiers côté serveur.

Server.js est le fichier qui est exécuté au démarrage du serveur. L'initialisation d'Express est la première chose à faire. Ensuite, nous avons trois tâches à accomplir :

- Nous devons établir une connexion à la base de données à l'aide d'une chaîne de connexion fournie par MongoDB Atlas. Cette chaîne est importante et doit rester confidentielle, elle est donc stockée dans les variables d'environnement avec Dotenv.

  L'extrait de code 5.3 nous montre comment la connexion à la base de données est réalisée à l'aide de `mongoose.connect`. Le premier argument demande notre chaîne de connexion `MONGO_URI` que l'on accède avec Dotenv, et le deuxième argument est facultatif et désigne le nom de la base de données sur laquelle on aimerait stocker nos collections.

- Une fois que la connexion est assurée avec la base de données, nous pouvons commencer à écouter à l'adresse du serveur dans le cas où le client nous envoie des requêtes.

  Pour faire cela, à la ligne 4 de l'extrait 5.3 nous initialisons Express et stockons l'instance dans la variable `app`. Une fois que nous nous sommes connecté sans erreur à Mongo, nous éxecutons `app.listen` afin que l'application Express puisse intercepter les requêtes et naviguer vers le endpoint approprié. La commande requiert un numéro de port en premier argument, suivi d'une fonction de rappel (facultative) qui s'éxécute lorsque l'application se met à écouter pour d'éventuelles requêtes.

```
1 // Server.js file
2 require('Dotenv').config()
3 const express = require('express')
4 const app = express()
5 const mongoose = require('mongoose')
6 // route imports
7 ..
8 // general middleware
```

```
9  ..
10 // route handlers
11 ..
12
13 // Database connection, then listening for requests
14 console.log("Server starting...")
15 mongoose.connect( process.env.MONGO_URI, { dbName: 'tb_mern' })
16     .then( () => {
17
18         app.listen( portNumber = process.env.PORT || 4000, () => {
19             console.log('\nconnected to the database and listening on port ' +
                   portNumber + "\n waiting for requests...\n\n")
20         })
21
22     })
23     .catch( (err) => { console.log(err) })
```

Listing 5.3 – Initialisation du serveur.

- Finalement, pour que les requêtes entrantes puissent être inspectées, il faut mettre en place le routeur et les middleware. Afin de rendre mon code plus modulaire, `Server.js` se charge uniquement d'exécuter les middlewares globaux, puis il redirige le traitement de la requête au fichier "route" approprié. Pour faire cela, l'extrait de code 5.4 nous montre comment l'application Express fait pour gérer les requêtes qu'elle reçoit. la commande `app.use` peut préciser une adresse relative ou non : si ce n'est pas le cas, alors `app.use` fait éxécuter la fonction qui lui est associée pour toutes les requêtes interceptées. Si elle contient une adresse en argument, alors la fonction n'est exécutée uniquement si l'adresse de destination de la requête contient le même bout d'adresse.

```
1     const photoRoutes = require('./routes/photoRoutes')
2
3     // example global middleware
4     app.use( (req, res, next) => {
5         console.log('\nrequest PATH: ' + req.path +
6         '\nrequest METHOD: ' + req.method)
7         res.set('Access-Control-Allow-Origin', process.env.DEV_PROXY)
8         next()
9     })
10
11    // example route handler
12    app.use('/api/photo', photoRoutes)
```

Listing 5.4 – Exemple de routing avec l'application Express.

Par exemple, le middleware sur la ligne 4 de l'extrait 5.4 concerne toutes les requêtes entrantes, et doit reçevoir trois variables en argument : l'une représente la requête avec toutes les informations données par le client, la seconde représente la réponse, et la troisième concerne les middlewares (tous ce qui n'est pas endpoint), et c'est une fonction qui doit être éxécutée à la fin du middleware afin de pouvoir continuer de traiter la requête jusqu'à ce qu'elle arrive au bon endpoint. Lorsque la requête a fini d'être traitée, on attache à l'objet réponse un statut et un éventuel objet et cette réponse sera ensuite rendue à l'application Express afin d'envoyer le résultat au client.

Le middleware décrit dans notre extrait ne sert qu'à signaler sur le terminal qu'une requête a été reçue, puis un header est rajouté à la requête pour préciser que cette dernière est sans danger si elle provient de l'adresse spécifiée avec `DEV_PROXY`.

Une fois que tous les middlewares concernés sont éxécutés, nous regardons si l'adresse de destination correspond à l'une des routes de l'API. Si par exemple la requête désigne que l'on veut récupérer toutes les photos et l'adresse de destination est correcte, dans l'extrait 5.4 la fonction `photoRoutes` serait alors éxécutée ce qui nous mène dans le fichier route correspondant.

Extrait 5.5 montre un example de comment les endpoints sont définis dans les fichiers routes. Afin de continuer le processus de routing, on initialise `express.Router`, et, en outre, on définit plus loin les différents endpoints qui peuvent être utilisés par les clients autorisés. Lorsque le routeur a trouvé le bon endpoint, il exécute la méthode qui servira à traiter la requête. Dans le cas de l'extrait 5.5, si nous avions une requête dont l'adresse de destination est `/api/photo/all`, alors la fonction `getPhotos` du fichier `photoControllers` va être éxécutée et on lui passe automatiquement en argument l'objet de requête et de réponse.

```
1  // photoRoutes.js
2  const express = require('express')
3  const router = express.Router()
4
5  // controller methods
6  const {
7      getPhotos,
8      ..
9  } = require('../controllers/photoControllers')
10 ..
11 // Middleware for route protection
12 const requireAuth = require('../middleware/requireAuth')
13 router.use(requireAuth)
14
15 // Protected routes
16 router.get('/all', getPhotos)
17 ..
```

Listing 5.5 – Exemple d'endpoint avec le routeur Express.

Toujours dans une idée de modularité, la logique de routing est réservée aux fichiers routes, et la logique de traitement de la requête est stockée dans des fichiers "controllers" dans le dossier du même nom.

Au sein de ces méthodes controllers se trouvent tout ce qui concerne la manipulation des données aux travers de Mongoose. Lorsque nécessaire, des méthodes sont importées d'un dossier "utils" ou "algorithms" qui contiennent des bouts de code qui ne sont pas vraiment dans la logique du traitement de la requête. Comme par exemple l'envoi d'email, la gestion des fichiers photos, où encore, la gestion de l'algorithme de consensus.

À ce propos, voyons un exemple de méthode controller. Par exemple, un utilisateur vient de soumettre une nouvelle tentative d'identification. Le client récupère les informations

nécessaires à la création d'un nouveau document "Guess", puis il envoie une requête au serveur pour enregistrer un nouveau document dans la base de données. Ensuite, nous devons vérifier les résultats de l'algorithme de consensus, voir si l'ajout d'une tentative d'identification n'aurait pas engendré la classification de l'image : le client envoie une requête au serveur pour obtenir les trois meilleurs scores au WMV (voir section 3.3), donc, comme le montre l'extrait 5.6, on envoie une requête vers `/api/guess/likely/` en y ajoutant l'identifiant de la photo au lieu de `:id` (voir figure 5.3).

```
const response = await fetch('/api/guess/likely/' + photo._id, {
    method: "GET",
    headers: { 'Authorization': `Bearer ${ user.token }`
    }
})
```

Listing 5.6 – Exemple de requête faite par le client.

Notez comme l'adresse de destination n'est pas entière, ceci car un proxy est employé par le client afin que nous n'avons pas à écrire l'adresse absolue à chaque requête, de telle manière, lorsque l'adresse qui héberge le serveur change, nous n'avons qu'à modifier cette partie de l'adresse qui est stockée dans un fichier de configuration. Si l'adresse de destination est correcte et que le serveur écoute sur cette même adresse, la requête sera alors interceptée et le routeur naviguera jusqu'à l'endpoint `/api/guess/likely/:id`. Le traitement de la requête sera alors délégué à la méthode qui lui est associée, dans ce cas, `getBestScores` dans le fichier controller dédié aux documents "Guess". Nous pouvons voir dans l'extrait 5.7 comment la méthode est structurée.

```
// guessControllers
const Guess = require('../models/guess')
const getLikelySpecies = require('./algorithms/getLikelySpecies')
..
const getBestScores = async ( req, res ) => {
    console.log('Guess: GET best scores start')
    try {
        const { id:photo_id } = req.params

        const guesses = await Guess.find({ photo_id: photo_id})
        const response = getLikelySpecies( guesses )

        if (!response) {
            throw Error( 'GET best scores: algorithm did not succeed' )
        }

        // should return 2 lists of corresponding species-confidence pairs with the
            highest scores
        console.log('Guess: GET best scores stop: successful')
        res.status(200).json( {species: response.species, confidence: response.conf} )

    } catch (error) {
        console.error('Guess: GET best scores: an error occured: ' + error.message)
        res.status(400).json({ error: error.message })
    }
}
```

Listing 5.7 – Exemple de méthode controller.

Dans la méthode, nous récupérons l'identifiant de la photo depuis les paramètres de la requête (les informations importantes que l'on insère dans l'adresse de destination), puis on appèle une méthode statique Mongoose pour obtenir une liste de tous les documents "Guess" qui contiennent l'identifiant de la photo (ligne 10). Nous déléguons ensuite la partie algorithmique du processus à une méthode `getLikelySpecies` qui retourne normalement 3 pairs pour une espèce avec un score associé à l'algorithme du WMV.

Nous n'allons pas entrer dans tous les détails du processus, mais en somme, nous récupérons en premier temps la liste de toutes les espèces différentes qui ont été proposées pour cette photo. Ensuite, pour chacune de ces espèces, nous faisons la somme des degrés de confiance pondérés et nous normalisons la valeur obtenue (puis nous retournons uniquement les trois meilleurs scores).

Par rapport à ce qui a été présenté dans la section 3.3, nous avons une methode `computeCertitude` dans un script dédié qui se charge de faire le calcul de la somme des degrés de confiance pour une espèce s_i divisée par la somme de tous les degrés de confiance pour chaque espèce (qui concerne la photo p_k), tel que le calcul de $C^n_{i,k}$ le décrit (voir le calcul 3.8). L'extrait de code 5.8 nous montre comment a été implémenté le calcul de $C^n_{i,k}$ pour l'espèce "species" et la liste "guesses" (qui ne concernent qu'une seule photo).

```
1  // computing the WMV score that some species corresponds to the photo
2      let numerator = 0
3      let denominator = 0
4
5      guesses.forEach( (elem) => {
6
7          const cert = (elem.certitude * elem.ponderation)
8
9          if (elem.species === species){
10             numerator += cert
11             denominator += cert
12         } else {
13             denominator += cert
14         }
15     })
16
17     let conf = numerator / denominator
18
19     if (conf == 0){ conf = 0.000001 }
20     return conf
```

Listing 5.8 – Calcul de la probabilité approximée qu'une espèce soit correcte.

5.4. Client

Ensuite nous avons l'implémentation de l'interface utilisateur.

L'interface utilisateur doit pouvoir permettre à l'utilisateur de naviguer l'application web et utiliser les fonctionnalités que nous avons décrites dans la section 3.2. Idéalement, nous devrions faire en sorte que l'interface soit intuitif et simple d'utilisation, j'ai donc décidé de segmenter l'application en un certain nombre de pages, comme vous l'aurez peut être deviné avec les captures d'écran de l'application finale (voir le chapitre 4).

Nous allons voir dans les sections suivantes comment React nous aide à établir le lien d'une page à l'autre.

5.4.1. React et DOM

Tout d'abord, il serait nécessaire de présenter brièvement un concept important de la programmation web : DOM, ou Document Object Model, est une manière de représenter les éléments HTML d'une page sous la forme d'un arbre de composants. Par exemple, la barre de navigation pour notre projet est constituée de plusieurs conteneurs pour pouvoir disposer les options de navigation d'une certaine manière plus tard avec CSS. Cela engendre un DOM dont la source désigne la barre de navigation (le tag HTML étant "nav") qui contient plusieurs conteneurs (le tag HTML étant "div") qui imbriquent à leur tour d'autres éléments.

La représentation des éléments avec DOM nous rend la tâche plus intuitive lorsque l'utilisateur interagit avec le site et que nous devons changer les éléments, il n'y a donc qu'à modifier la composition d'un arbre.

React est une librairie qui simplifie la modularisation des différents éléments qui composent une page. Nous pouvons définir un élément dans un script qui lui est propre, puis, à l'intérieur d'un composant, nous n'avons qu'à importer cet élément pour l'intégrer au DOM de la manière souhaitée.

Une autre particularité de React est la manière dont les données peuvent persister à travers les pages. Nous en discuterons un peu plus tard.

5.4.2. Structure

À propos de la structure : il est bon de noter que le frontend a été instancié avec l'outil "create-react-app" pour la ligne de commande, qui créé et configure automatiquement notre application React. La structure du dossier est donc en partie influencée par create-react-app, vous pouvez vous référer à la figure 5.5 pour les dossiers et fichiers les plus pertinents.

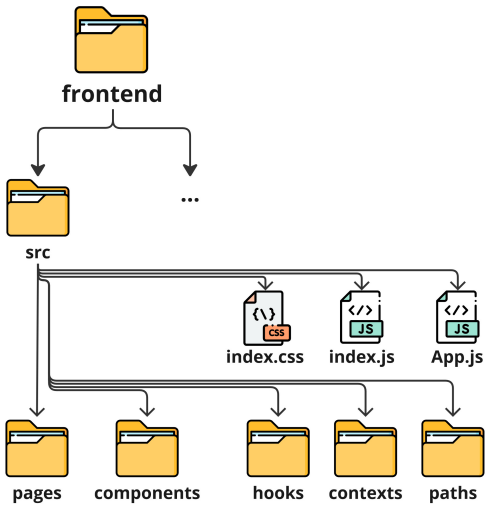


Figure 5.5. – La structure des fichiers côté client.

Index.js est le fichier exécuté au démarrage de l'application. Nous pouvons voir comment il se construit dans le listing 5.9. Il lui est associé le fichier index.css dans lequel la grande partie de la stylisation de l'interface a lieu. C'est dans Index.js que create-react-app a défini l'élément source de notre arbre des composants. Ainsi, le système de routing des pages va venir s'installer à l'intérieur de cet élément source, et pour la lisibilité du code, create-react-app nous laisse définir le système de routing sur un autre script dénommé App.js.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import './index.css';
4  import App from './App';
5  //import { disableReactDevTools } from '@fvilers/disable-react-devtools'
6
7  // global contexts
8  import { AuthContextProvider } from './contexts/AuthContext';
9  //disableReactDevTools()
10
11 const root = ReactDOM.createRoot(document.getElementById('root'));
12 root.render(
13   <React.StrictMode>
14
15     <AuthContextProvider>
16         <App />
17     </AuthContextProvider>
18
19   </React.StrictMode>
20 );
```

Listing 5.9 – Initialisation du client.

5.4.3. Routing des pages

Le routing des pages avec React peut être assuré par la librairie “react-router-dom”. Nous démarrons le routeur pour nos pages dans App.js comme on le montre le listing 5.10, il est nécessaire ensuite de définir une liste des routes qui peuvent être prises par l'utilisateur en navigant l'application.

```
1  // hooks
2  import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
3  import { useAuthContext } from './hooks/contexthooks/useAuthContext'
4
5  // layouts (to wrap context providers as page layouts)
6  ..
7
8  // pages
9  import Home from './pages/Home'
10 ..
11
12 // components
13 import Navbar from './components/Navbar'
14
15 function App() {
16     const { user } = useAuthContext()
17
18     return (
19         <div className="App">
20             <BrowserRouter>
21
22                 <Navbar />
23                 <div className='pages'>
24                     <Routes>
25
26                         <Route
27                           path="/"
28                           element={ <Home /> }
29                         />
30
31                         // other routes
32                         ..
33
34                     </Routes>
35                 </div>
36             </BrowserRouter>
37         </div>
38     );
39 }
40
41 export default App;
```

Listing 5.10 – Le routing des pages côté client.

La barre de navigation est définie en-dessus de toutes les routes (ligne 22), afin que l'on puisse l'atteindre à tout moment. Ensuite nous avons la liste de routes : pour chacune d'entre elles, nous y associons une adresse URL (path) ainsi que d'un composant qui sera affiché pour cette route (element). J'ai stocké ces composants dans le dossier “pages”.

Figure 5.6 illustre la manière dont j'ai organisé le routing des pages. Pour ce qui est de la gestion de la banque d'images, nous avons ce que je décrirais comme deux routes principales : l'option pour importer une photo, et l'option pour consulter la banque d'images, à partir de laquelle nous pouvons voir les détails de chaque photo et décider si l'on veut identifier ou signaler l'image.

Sur la figure 5.6, les cadres qui regroupent certaines pages désignent les contextes React que j'ai décidé de mettre en place.

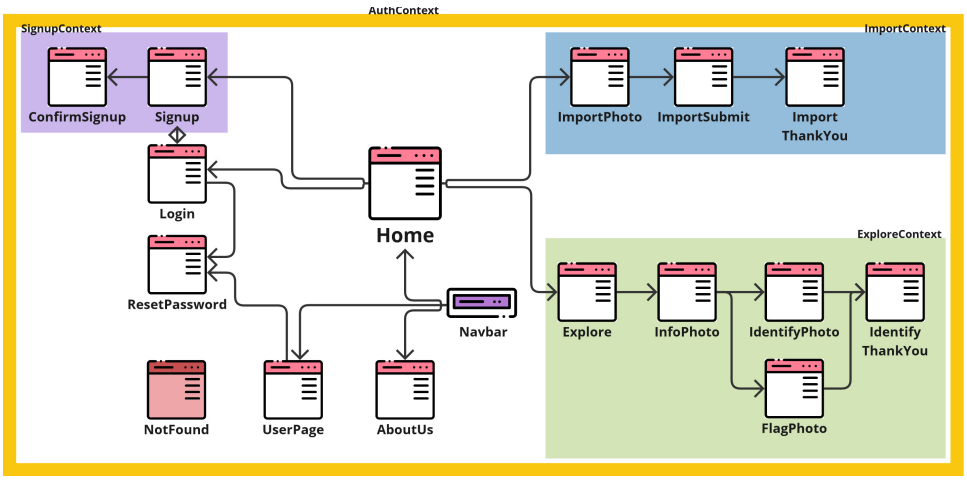


Figure 5.6. – Le flow de navigation pour notre GUI.

5.4.4. Contextes React

Les contextes sont une autre particularité de React : c'est un mécanisme pour faire perpétuer certaines informations à travers différentes pages. Un contexte React définit le nom des variables qui vont retenir de l'information (contenu dans "l'état" du context), et un "fournisseur de contexte" (context provider) va venir envelopper l'élément DOM pour lequel on souhaiterait rendre l'information disponible.

Pour prendre exemple sur le contexte global sur la figure 5.6, le contexte d'authentification (`AuthContext`) permet de garder une trace de si l'utilisateur est connecté ou non. Nous pouvons voir comment ce contexte a été implémenté avec l'extrait 5.11. Un contexte React s'initialise avec la fonction `createContext`, et il est nécessaire ensuite de créér une fonction "context provider" comme `AuthContextProvider`, qui prend en argument l'élément DOM qui souhaite avoir accès aux informations du contexte, et il enveloppe ensuite cet élément DOM avec un provider qui lui fournit l'état du contexte. En l'occurence, l'état du contexte ne maintient qu'une seule variable nommée `user`.

```
1 // AuthContext.js
2 import { createContext, useReducer, useEffect } from 'react'
3
4 export const AuthContext = createContext()
5
6 export const authReducer = ( state, action ) => {
7     switch (action.type) {
```

```
        case 'LOGIN':
            return { user: action.payload }

        case 'LOGOUT':
            return { user: null }

        default:
            return state
        }
    }

export const AuthContextProvider = ( { children } ) => {
    const [ state, dispatch ] = useReducer( authReducer, { user: null } )

    useEffect( () => {
        const user = JSON.parse(localStorage.getItem('user'))
        if ( user ){ dispatch({ type: 'LOGIN', payload: user }) }
    }, [] )

    return (
        <AuthContext.Provider value = { { ...state, dispatch } }>
            { children }
        </AuthContext.Provider>
    )
}
```

Listing 5.11 – Exemple de configuration d'un context React.

Tel qu'on le montre dans l'extrait 5.12, pour vérifier si l'utilisateur est connecté sur la page d'accueil par exemple, il suffit d'appeler une fonction `useContext(AuthContext)` et celle-ci retournera la valeur du contexte. Dans notre cas il retourne un objet utilisateur si l'utilisateur est connecté, et sinon il nous retourne un object vide.

```
import { useContext } from 'react'
import { AuthContext } from '../../contexts/AuthContext'

export Home = () => {
    const { user } = useContext( AuthContext )
    ..
    return (
        // if user is connected, show this and this
        { user && .. }

        // if user is not connected, show this
        { !user && .. }
    )
}
```

Listing 5.12 – Utilisation du context React.

Pour mettre à jour le contexte lorsqu'un utilisateur est authentifié ou lorsqu'il se déconnecte, Nous devons implémenter une méthode à notre contexte, tel qu'il a été fait dans l'extrait 5.11 avec `authReducer` : j'utilise la fonction React "useReducer" pour maintenir l'état du contexte, et lorsque la fonction est appelée, elle reçoit un type d'action et un éventuel objet en argument, et en fonction du type d'action, il va mettre à jour l'état

en conséquence. Dans notre exemple, appeler `'LOGIN'` va mettre à jour l'état pour qu'il contienne l'objet utilisateur, et `'LOGOUT'` va s'assurer que l'état soit maintenant vide.

Comme présenté dans l'extrait 5.13, Pour utiliser cette fonction à l'intérieur de la page de connexion, nous faisons en sorte que notre provider, en plus de l'objet utilisateur, nous retourne aussi une fonction `dispatch` (dans le cas de cet extrait de code, une fonction utilitaire/hook `useAuthContext` se charge d'appeler useContext avec un peu de gestion des erreurs). La fonction `dispatch` demande à ce que l'on lui précise un type d'action et éventuellement un objet, et `dispatch` va ensuite se charger d'appeler le reducer à l'intérieur du contexte afin de mettre à jour l'état.

```
1  import { useAuthContext } from '../../hooks/contexthooks/useAuthContext'
2
3  export const useLogin = () => {
4      const { dispatch } = useAuthContext()
5      ..
6      // fetch request to '/api/user/login'
7      // returns response object
8      const json = response.json()
9
10     if(response.ok){
11         // save the user to local storage
12         localStorage.setItem('user', JSON.stringify(json))
13
14         // update auth context
15         dispatch({
16             type: 'LOGIN',
17             payload: json
18         })
19         ..
20     }
21 }
```

Listing 5.13 – Mise à jour du contexte React.

Une fois que le context a été correctement implémenté, il reste encore à envelopper l'arbre des composants avec le "context provider" de `AuthContext`. Nous avons fait ça dans `Index.js` (voir extrait 5.9), en important `AuthContextProvider` et en englobant toute l'application avec.

Le contexte est maintenant fonctionnel, on peut maintenant appeler `useAuthContext` (ou `useContext(AuthContext)`) à n'importe quelle page afin de récupérer l'objet `user`.

J'utilise trois autres contextes React de manière similaire à travers l'application, mais ceux-ci ne sont utilisables uniquement lorsqu'on se trouve dans l'une des pages qui sont contenues dedans. Cela nous permet de maintenir des informations seulement là où nous en avons besoin. Par exemple, lorsqu'on consulte la banque d'images et qu'une photo est selectionnée, nous pouvons alors choisir de signaler ou identifer et les informations concernant la photo selectionnée resteront disponibles à travers les différentes pages, du moment que celles-ci sont contenues dans le contexte.

La présentation de l'implémentation est maintenant terminée. Nous avons fait le tour de tous les aspects importants de mon application. En espérant avoir été complète dans mes explications, de manière à favoriser la compréhension générale du fonctionnement des applications web, et également pour permettre une meilleure navigation des implémentations avec le stack MERN.

6
Conclusion

6.1. Résultats

À l'issu de ce projet, nous pouvons dire que tous les objectifs ont été atteints. L'application finale est fonctionnelle, elle permet une gestion des utilisateurs sûre et les requêtes vers le serveur sont protégées. Les utilisateurs connectés peuvent importer leurs photos et ils peuvent consulter celles des autres dans la galerie. Ils peuvent filtrer la sélection d'images, générer un dataset, ainsi que télécharger les photos. Finalement, les utilisateurs ont la possibilité de gérer le contenu de la banque d'images, en signalant les photos inadéquates, ils peuvent également soumettre une tentative d'identification, ce qui mettra à jour l'algorithme de consensus.

En somme, je pense que toutes les fonctionnalités sont opérationnelles et les objectifs sont atteints, autant en terme d'implémentation qu'en terme des apprentissages.

6.2. Perspectives futures

Si ce site venait à être mis à disposition, je proposerais tout d'abord quelques voies d'amélioration : tout d'abord, il serait bien de faire quelques tests supplémentaires au sujet de l'algorithme de consensus. Comme nous l'avons mentionné à la section 3.3, par exemple, il y aurait des critères plus intéressants pour déterminer quand une photo peut être classée. Nous pourrions trouver un moyen de vérifier que les tendances de votes se sont stabilisées par exemple. L'exploration d'autres choix pour l'algorithme de consensus serait également pertinente.

Ensuite, pour que les données récoltées soient aussi intéressantes que possible, il faudrait envisager de demander aux utilisateur de nous fournir davantage d'informations, comme par exemple, les contours de l'oiseau sur l'image, ou délimiter les différentes parties anatomiques qui sont visibles sur la photo. Ce sont des données très précieuses pour améliorer la performance des modèles de reconnaissance. [40]

Finalement, il serait bien d'apporter quelques améliorations à l'implémentation : Nous pourrions envisager l'ajout d'un rôle administrateur à l'intérieur du site, pour garder un oeil sur la banque d'images et pour traiter les demandes de changement de statut. Il y

a une marge d'amélioration pour le traitement des erreurs, et nous pourrions également envisager l'ajout de tests afin de faciliter le debugging des fonctionnalités.

Il serait important aussi de pouvoir laisser les utilisateurs choisir si ils souhaitent ou non que leurs photos soient utilisées à des fins scientifiques, ou si ils souhaitent uniquement partager leurs découvertes avec d'autres passionnés.

6.3. Mot de Fin

Voilà ce rapport terminé. J'espère avoir éveillé l'intérêt des lecteurs sur la problématique de la reconnaissance automatique et de la construction de dataset.

Pour ma part, j'ai hâte de voir l'évolution du domaine de la FGIA. Qui sait, peut être qu'à termes, ce sera grâce à une meilleure qualité des données que le domaine se laissera enfin pousser des ailes.

A

Acronymes récurrents

API	Application Programming Interface
CSS	Cascading Style Sheet
CSV	Comma-Separated Values
DOM	Document Object Model
ERD	Entity-Relationship Diagram
FGIA	Fine-Grained Image Classification
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
MERN	MongoDB ; Express ; React ; Node.js
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
REST	Representational state transfer
SQL	Structured Query Language

B

License de la Documentation

Copyright (c) 2024 Sara Faggella.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation ; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

The GNU Free Documentation Licence can be read from [41].

C

Code Source et Site du Projet

Vous pouvez consulter le code source à cette adresse Github :
`https://github.com/FaggellS/Bachelor-Avianity`

L'application web a été déployée le 06.09.24 à l'adresse suivante :
`https://bachelor-avianity.onrender.com/`

Bibliographie

[1] *Action « Oiseaux de nos jardins » : 142'000 oiseaux dénombrés - BirdLife Suisse.* https://www.birdlife.ch/fr/content/aoj-bilan24 (dernière consultation le Août 25, 2024). 6

[2] *Amazon Mechanical Turk.* https://www.mturk.com/ (dernière consultation le Août 25, 2024). 8

[3] *Best way to store and serve images with node and mongodb - Reddit.* https://shorturl.at/QfBHI (dernière consultation le Août 26, 2024). 37

[4] *Birds 525 species - Kaggle.* https://www.kaggle.com/datasets/gpiosenka/100-bird-species/data?select=birds.csv (dernière consultation le Août 25, 2024). 9

[5] *Blob - MDN Web Docs.* https://developer.mozilla.org/en-US/docs/Web/API/Blob (dernière consultation le Août 29, 2024).

[6] *Challenges in Bird Population Monitoring - Bird Lens.* https://tinyurl.com/2f6et25y (dernière consultation le Août 26, 2024). 6

[7] *Citizen science : importance and benefits. European Commission.* https://shorturl.at/9gX9n (dernière consultation le Août 25, 2024). 6

[8] *CUB-200-2011 - Paperswithcode.* https://tinyurl.com/yy2xkhup (dernière consultation le Août 25, 2024). 9

[9] *Deploy a Full Stack App - React, Node.js, Express, Mongo | MERN Tutorial - Youtube.* https://www.youtube.com/watch?v=l134cBAJCuc (dernière consultation le Août 29, 2024).

[10] *Developer Community - MongoDB.* https://www.mongodb.com/community/forums/t/images-and-metadata-on-mongodb/261703 (dernière consultation le Août 26, 2024). 37

[11] *Difference between Web Scraping and Web Crawling - GeeksForGeeks.* https://www.geeksforgeeks.org/difference-between-web-scraping-and-web-crawling/ (dernière consultation le Août 25, 2024). 8

[12] *Dimorphisme sexuel - Wikipedia.* https://fr.wikipedia.org/wiki/Dimorphisme_sexuel (dernière consultation le Août 28, 2024). 7

[13] *Entretien - Le rôle capital de la surveillance des oiseaux - European Environnment Agency.* https://shorturl.at/JMwfQ (dernière consultation le Août 25, 2024). 6

[14] *Express.js Request Object - JavaTPoint.* https://shorturl.at/fielo (dernière consultation le Août 29, 2024).

[15] *FlatIcon.* https://www.flaticon.com/ (dernière consultation le Août 29, 2024).

[16] *Flora Incognita.* https://eu-citizen.science/project/380 (dernière consultation le Août 25, 2024). 6

[17] *How to format date in ReactJS.* https://shorturl.at/K3SKF (dernière consultation le Août 29, 2024).

[18] *Merlin Bird ID.* https://merlin.allaboutbirds.org/the-story/ (dernière consultation le Août 25, 2024). 2

[19] *MERN Stack Explained - MongoDB.* https://www.mongodb.com/resources/languages/mern-stack (dernière consultation le Août 26, 2024). 34

[20] *MERN Stack Tutorial 1 - What is the MERN Stack ? - Youtube.* https://rb.gy/96mud6 (dernière consultation le Août 29, 2024). 35

[21] *NPM Documentation.* https://www.npmjs.com/ (dernière consultation le Août 29, 2024).

[22] *The Odin Project.* https://www.theodinproject.com/paths (dernière consultation le Août 25, 2024). 35

[23] *Ornitho.* https://www.ornitho.ch/ (dernière consultation le Août 29, 2024).

[24] *Papa Parse Documentation.* https://www.papaparse.com/ (dernière consultation le Août 29, 2024).

[25] *Photo Gallery Manager Project - Github.* https://rb.gy/as8kk7 (dernière consultation le Août 29, 2024).

[26] *Proxying API Requests in Development - Create React App.* https://rb.gy/a7m6jf (dernière consultation le Août 29, 2024).

[27] *React CORS Guide : What It Is and How to Enable It.* https://www.stackhawk.com/blog/react-cors-guide-what-it-is-and-how-to-enable-it/ (dernière consultation le Août 29, 2024).

[28] *React Proxy | Easiest Fix to CORS Errors - Youtube.* https://www.youtube.com/watch?app=desktop&v=N4yUiQiTvwU (dernière consultation le Août 29, 2024).

[29] *React Router Documentation.* https://reactrouter.com/en/main (dernière consultation le Août 29, 2024).

[30] *React TextArea - Handle the Value Change Event.* https://js.devexpress.com/React/Documentation/20_2/Guide/UI_Components/TextArea/Handle_the_Value_Change_Event/ (dernière consultation le Août 29, 2024).

[31] *Stack Overflow.* https://stackoverflow.com/ (dernière consultation le Août 29, 2024).

[32] *Students Projects - Software Engineering UNIFR.* https://www.unifr.ch/inf/softeng/en/research/students.html (dernière consultation le Août 29, 2024).

[33] *Swiss Bird Index SBI - VogelWarte.* https://www.vogelwarte.ch/modx/en/projects/population-trends/sbi-state/ (dernière consultation le Août 29, 2024).

[34] *VogelWarte.* https://www.vogelwarte.ch/fr/ (dernière consultation le Août 29, 2024).

[35] *What is a Full Stack ? - W3Schools.* https://www.w3schools.com/whatis/whatis_fullstack.asp (dernière consultation le Août 26, 2024). 34

[36] *Why Monitor Birds ? - National Park Services.* https://www.nps.gov/articles/000/why-monitor-birds.htm (dernière consultation le Août 25, 2024). 5

[37] Bais A. *Best Practices for Managing Mongoose Schemas in Node.js* . https://rb.gy/dcinfv (dernière consultation le Août 29, 2024).

[38] M. Escobar I. M. Sagarra G. N. Lugilde D. S. Romero J. Estrella M. Berjano R., L. Tirado J. *Mind your app : Could plant ID applications lead to an increase in extinction risk ?* Phytotaxa, Août 2023. https://shorturl.at/vdfg0 (dernière consultation le Août 25, 2024). 7

[39] L. Bacon K. Campbell N., Peacock J. *A repeatable scoring system for assessing Smartphone applications ability to identify herbaceous plants.* PLOS ONE, Avril 2023. https://shorturl.at/RvTOZ (dernière consultation le Août 28, 2024). 2, 6

[40] Fei-Fei L. Deng J., Krause J. *Fine-Grained Crowdsourcing for Fine-Grained Recognition.* CVPR 2013, Juin 2013. https://shorturl.at/1VsFG (dernière consultation le Août 29, 2024). 54

[41] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (dernière consultation le Août 29, 2024).

[42] E. Hattab H. *Awesome Fine-Grained Image Classification - Github.* https://github.com/arkel23/AFGIC (dernière consultation le Août 28, 2024). 7

[43] S. Trueblood J. Hasan E., Duhaime E. *Boosting wisdom of the crowd for medical image annotation using training performance and task features.* Cognitive Research : Principles and Implications, Mai 2024. https://cognitiveresearchjournal.springeropen.com/articles/10.1186/s41235-024-00558-6 (dernière consultation le Août 29, 2024).

[44] Khachariya J. *How make zip of images and download it using React JS in 3 Easy steps.* https://rb.gy/2f6kgt (dernière consultation le Août 29, 2024).

[45] Qianmu L. Jian W. Xindong W. Jing Z., S. Victor S. *Consensus algorithms for biased labeling in crowdsourcing.* Information Sciences, Mars 2017. https://tinyurl.com/4v8uu9xn (dernière consultation le Août 29, 2024).

[46] Schader M. *eBird in Action : Building capacity to maximize habitat impact for grassland and savanna bird conservation.* Land Trust Bird Conservation Initiative, Mars 2014. https://tinyurl.com/mu3ydkcp (dernière consultation le Août 25, 2024). 2

[47] V. Luxburg U. H. Franz V. Meyen S., M. B. Sigg D. *Group decisions based on confidence weighted majority voting.* Cognitive Research : Principles and Implications, Mars 2021. https://cognitiveresearchjournal.springeropen.com/articles/10.1186/s41235-021-00279-0 (dernière consultation le Août 29, 2024). 13

[48] Bonnet P. and al. *How citizen scientists contribute to monitor protected areas thanks to automatic plant identification tools.* Ecological Solutions and Evidence, Septembre 2020. https://besjournals.onlinelibrary.wiley.com/doi/10.1002/2688-8319.12023 (dernière consultation le Août 25, 2024). 2, 6

[49] Brianna M. Casario Pamela C. Zipse C. Grabosky J. Ryan J., Schmidt. *An Analysis of the Accuracy of Photo-Based Plant Identification Applications on Fifty-Five Tree Species.* Arboriculture, Urban Forestry, Janvier 2022. https://auf.isa-arbor.com/content/48/1/27 (dernière consultation le Août 25, 2024). 2, 6

[50] Gaudin S. *Reconnaître automatiquement les essences forestières avec son téléphone : rêve ou réalité ? Test de l'application Pl@ntNet.* Revue forestière française, Septembre 2023. https://revueforestierefrancaise.agroparistech.fr/article/view/7687 (dernière consultation le Août 28, 2024). 2, 6

[51] Zeeman S. *Why are Bird Surveys so important ?* Février 2023. https://shorturl.at/7isDY (dernière consultation le Août 25, 2024). 5

[52] Mysore A. Goel S. Simoiu C., Sumanth C. *Studying the "Wisdom of Crowds" at Scale.* AAAI Conference on Human Computation, Crowdsourcing, Octobre 2019. https://web.stanford.edu/~csimoiu/doc/wisdom-of-crowds.pdf (dernière consultation le Août 27, 2024). 3, 13

[53] T. Garnett S. Garrard G. O'Connor J. L. Oliver J. Robinson C. Tulloch A. A. Fuller R. Steven R., Barnes M. *Aligning citizen science with best practice : Threatened species conservation in Australia.* Conservation Science and Practice, Août 2019. https://conbio.onlinelibrary.wiley.com/doi/10.1111/csp2.100 (dernière consultation le Août 28, 2024). 6

[54] Singh V. *Simplifying Data Input in Express : An Overview of req.params, req.query, and req.body.* https://tinyurl.com/259drzfv (dernière consultation le Août 29, 2024).

[55] Farrell R. Haber S. Barry J. Ipeirotis P. Perona P. Belongie S. V. Horn G., Branson S. *Building a bird recognition app and large scale dataset with citizen scientists : The fine print in fine-grained dataset collection.* IEEE 2015, Juin 2015. https://openaccess.thecvf.com/content_cvpr_2015/papers/Horn_Building_a_Bird_2015_CVPR_paper.pdf (dernière consultation le Août 28, 2024). 9, 10

[56] Perona P. V. Horn G. *The Devil is in the Tails : Fine-grained Classification in the Wild.* arXiv, Septembre 2017. https://arxiv.org/abs/1709.01450 (dernière consultation le Août 25, 2024). 2, 6

[57] Song Y. Cui Y. Sun C. Shepard A. Adam H. Perona P. Belongie S. V. Horn G., M. Aodha O. *The iNaturalist Species Classification and Detection Dataset.* CVPR 2018, 2018. https://arxiv.org/abs/1707.06642 (dernière consultation le Août 25, 2024). 2, 6

[58] Collmar D. Walter V., Kölle M. *Measuring the Wisdom of the Crowd : How Many is Enough ?* PFG, Avril 2022. https://link.springer.com/article/10.1007/s41064-022-00202-2 (dernière consultation le Août 27, 2024). 20

[59] Mac Aodha O. Wu J. Peng Y. Tang J. Yang J. Belongie S. Wei X. S., Song Y. Z. *Fine-Grained Image Analysis with Deep Learning : A Survey.* IEEE, Décembre 2022. https://arxiv.org/pdf/2111.06119 (dernière consultation le Août 28, 2024). vi, 2, 7, 8, 9

[60] Wu T. Bergsma J. Movellan J. Whitehill J., Ruvolo P. *Whose Vote Should Count More : Optimal Integration of Labels from Labelers of Unknown Expertise.* University of California, San Diego, Janvier 2009. https://papers.nips.cc/paper_files/paper/2009/file/f899139df5e1059396431415e770c6dd-Paper.pdf (dernière consultation le Août 27, 2024). 14

[61] Seeland M. Mäder P. Wäldchen J., Rzanny M. *Automated plant species identification - Trends and future directions.* PLoS Comput Biol., Avril 2018. https://www.ncbi.

nlm.nih.gov/pmc/articles/PMC5886388 (dernière consultation le Août 28, 2024). 2, 6

[62] Shen D. Yu X. Yue D., Yu G. *A Weighted Aggregation Rule in Crowdsourcing Systems for High Result Accuracy*. IEEE 2014, Août 2014. https://ieeexplore.ieee.org/document/6945699 (dernière consultation le Août 29, 2024). 13