Informaticien/-ne CFC
Travail pratique individuel 2025 (TPI)

ICT Berufsbildung
Formation professionnelle
Freiburg-Fribourg

Modèle de rapport v8

Nom du candidat : Loïc Jaquier

Candidat Nº151259

BirdBox Tracker

Sommaire

1 Résumé du rapport du TPI

1.1 Situation de départ

Une société d'ornithologie possède des nichoirs a oiseaux qui sont répertoriés sur un fichier Excel ou un bloc note. Cette méthode de référencement n'est pas optimale car elle ne garantit pas l'intégrité et la traçabilité des données. Les données ne sont pas centralisées ce qui peut entrainer des redondances d'informations. La société peut tout de même exercer son activité de cette manière mais cela peut facilement créer une confusion au niveau de la gestion des informations des nichoirs.

1.2 Mise en œuvre

Pour répondre à cette problématique j'ai réalisé l'implémentation d'une application web client-serveur, soumise au contraintes de la structure MVC. Elle utilisera les technologies HTML et JavaScript pour l'affichage et le fonctionnement du coté frontend. Le backend quant à lui utilisera le langage PHP pour le traitement des requêtes et MySQL pour le stockage des informations dans une base de données. L'API de géolocalisation Google Maps est utilisé depuis le client afin d'afficher une carte et des marker et une librairie de génération de codes QR est téléchargée sur le serveur. Le développement de cette application suit les phases utilisées tout au long de la formation à l'EMF qui sont l'analyse, la conception, l'implémentation et finalement les tests du programme réalisé.

1.3 Résultats

L'application finale comporte deux interfaces distinctes. La première est destinée aux administrateurs pour la gestion des informations des nichoirs a oiseaux. Et la deuxième, accessibles depuis les code QR généré par l'administrateur, sert à envoyer un commentaire ou une observation par rapport au nichoir. Les observations peuvent finalement être visualisé et gérée par les administrateurs. Cette application permet de répondre à la problématique exposée dans le premier paragraphe.

2 Les grandes lignes du projet

2.1 Analyse de la situation initiale

Une société d'ornithologie gère et entretiens des nichoirs à oiseaux. Les nichoirs sont répertoriés dans un carnet ou un fichier Excel mais il peut arriver que certains soient parfois oubliés ou que des informations manquent. Cette méthode n'est pas efficace car elle ne respecte pas de procédure stricte d'enregistrement des informations, l'intégrité des données n'est donc pas respectée.

2.2 Analyse de l'état désiré

L'objectif final de ce travail serait de proposer une application WEB pour la gestion des nichoirs avec laquelle la société pourrait enregistrer tous ceux qu'elle possède déjà et par la suite les nouveaux qu'elle pourrait acquérir. De ce fait, les informations seraient strictement enregistrées, centralisées et facilement accessible par les membres.

De plus, les membres auront la possibilité de générer des codes QR pour chacun des nichoirs. Ces codes QR serviront au marcheur pour accéder à une page de commentaire dédié au nichoir. Les membres pourront ensuite visualiser et prendre en compte ces commentaires. Cela peut les aider pour l'entretiens ou des informations par exemple.

2.3 Cahier des charges / exigences du système

Dans le cadre de ce projet TPI, j'ai été mandaté pour réaliser l'application de gestion de nichoir à oiseaux « BirdBox Tracker ». Cette application s'adresse aux sociétés d'ornithologie. Elle a pour but de remplacer les méthodes standard de référencement du matériel (bloc-notes, fichier Excel, …) en proposant une solution centralisée avec un référencement plus stricte des données. Ceci ayant pour but d'éviter de perdre du matériel ou d'utiliser des données qui ne sont plus à jour.

L'application comportera deux interfaces principales.

- La première interface servira aux Administrateurs ou aux membres de la société directement pour la gestion des nichoirs. Elle comportera une multitude de fonctionnalités, comme la visualisation des nichoirs sur une carte, la création et mise à jour de nichoir l'ajout de photo pour les nichoirs, la génération de codes QR ou encore la gestion d'observations pour chacun des nichoirs.

- La seconde interface sera la page accessible par les marcheurs ou les passant en scannant le code QR que les administrateurs auront généré et installé sur le nichoir. Ils pourront visualiser l'espèce d'oiseaux présente dans le nichoir et auront la possibilité de laisser une observation pour ce dernier.

Ce sera une application WEB avec la structure client-serveur qui utilisera principalement les technologies suivantes :

- PHP
- HTML
- JavaScript

- MySQL
- API : Google Maps
- librairie : PHP-QRCode

L'architecture de l'application devra respecter le modèle MVC.

Le projet est organisé selon les phases standard apprises à l'EMF, analyse, conception, implémentation / réalisation, tests et documentation. Une planification du projet ainsi qu'un journal de travail seront également utilisés afin de suivre l'avancement du projet.

2.4 Organisation du projet

Méthode d'organisation :

Afin de maintenir une organisation et une gestion du temps adéquate, je vais mettre en place un planning et un journal de travail.

Le planning en cascade ou waterfall en anglais nous permet de représenter les taches à réaliser tout au long de la durée du mandat. Les taches sont organisées dans l'ordre chronologique et séparées selon les phases d'analyse, de conception, de réalisation et de test.

Le journal de travail nous permet quant à lui de référencer les taches effectuées au cours de chacune des journées avec une brève conclusion. On y voit également les différentes réunions avec le supérieur ou les experts ainsi que les problèmes rencontrés et solutions trouvées.

Ces documents vont me permettre de suivre et comparer tout au long du projet l'avancement des différentes tâches avec ce qui était initialement prévu.

Aperçu d'un extrait du planning (voir en annexe pour plus de détails et meilleur lisibilité) :

On peut visualiser su ce planning, les différentes taches ainsi que leur avancement su l'axe temporel sur la droite. On commence en haut à gauche avec la première tâche au commencement de la période et on termine en bas à droite du planning avec la dernière tâches tout à la fin de la période.

Certaines taches tel que la documentations ou le maintiens de documents d'organisation se font tout au long de la réalisation du projet.

Projet : BirdBox Tracker
Jaquier Loïc — 151259
(Griser les périodes non-comprises dans la durée du TPI - idem jc

Tâches	Etat (En cours / Terminé)
Analyse	Terminé
Lecture du cahier des charges et pris en main du prjet	Terminé
Maquettes	Terminé
Diagramme de séquence	Terminé
Diagramme d'activité	Terminé
Diagramme Use Case	Terminé
Shéma ER	Terminé
Documentation d'analyse	Terminé
Conception	Terminé
Modèle relationnel DB	Terminé
Diagramme de classe client	Terminé
Diagramme de class serveur	Terminé
Diagramme de séquence 1	Terminé
Diagramme de séquence 2	Terminé
Planification / conception des test fonctionnels	Terminé
Implémentation	Terminé
Inscription et authentification utilisateur	Terminé
Affichage des nichoirs, de leurs information et de la carte	Terminé
Edition des informations des nichoirs	Terminé
Ajout gestion des images par nichoir	Terminé
Génération de codes QR	Terminé
Interface de saisi des observations	Terminé
Visualisation et gestion des observations par nichoir	Terminé
Implémentation du style	Terminé
Tests	En cours
Tests intermédiaires	En cours
Protocol de test final	En cours
Documentation	En cours
Documentation de projet	En cours
Document d'organisation (Journal et planning)	En cours

Jour > Lundi 19 mai 25 | Mardi 20 mai 25 | Mercredi 21 mai 25 | Jeudi
heures > 1 2 3 4 5 6 7 8

Zone d'avancement >> étirer sur la droite pour montrer l'avancement du projet

Première visite des experts (11h15)

Figure 1 Extrait de planning

Aperçu d'un extrait du journal (voir en annexe pour plus de détails et meilleur lisibilité) :

Voici maintenant l'aperçu du journal de travail effectué. On y voit les taches effectuées pendant les jour de travail ainsi qu'un bref paragraphe de réflexion personnel sur la journée. Le tempos de chaque tâche est indiqué sur la droite de celle-ci et le temps total de travail est calculé à la fin du journal.

Projet : BirdBox Tracker

Jaquier Loïc 151259

Date	Travail effectué	Temps [H.h]
19.05.25	Lecture du cahier des charges	0.5
	Création du planning et du journal	1.5
	Shémas d'activité	2.0
	Schéma use case	1.0
	Réunion avec supérieur	1.0
	Maquettes	0.5
	Shéma de séquence	1.5
	Réflexion personnelle ↓↓↓ Total >	8.0
	Stresse du début. Avancement a jour selon le planning	
20.05.25	Documentation d'analyse / conception	2.0
	Diagramme de classe	2.5
	Diagramme de séquence d'intercation	2.5
	Conception et planification des tests	0.5
	Discussion avec le supérieur	0.5
	Réflexion personnelle ↓↓↓ Total >	8.0
	Avancement a jour selon le planning. Attention a bien documenter au fur et a mesure.	
21.05.25	Implémentation : Inscription et authentification	3.5
	Implémentation : Affichage des nichoirs sur la carte	3.0
	Documentation	1.0
	Réunion avec le supérieur	0.5
	Réflexion personnelle ↓↓↓ Total >	8.0
	Content de commencer l'implémentation. Implémentation efficace grace au test technologiques de la semaine préparatoire.	
22.05.25	Implémentation : affichage des nichoirs sur la carte	3.0
	Préparation a la première visite	0.5
	Documentation	2.5
	Première visite des experts (décision : possibilité de changer la TM selon les critères)	1.5
	Discussion avec le supérieur	0.5
	Réflexion personnelle ↓↓↓ Total >	8.0
	Léger stress pour la première visite. La première visite ma globalement mis en confiance pour la suite du projet	

Interaction supérieur
Interaction expperts
Prolème
Résolution

Figure 2 Extrait de journal

On pourra également visualiser dans le journal les problèmes rencontrés ainsi que les visites avec le supérieur ou les experts. Le recensement des problèmes rencontrés peut servir à justifier un retard pris au niveau du planning.

La documentation contient l'ensemble des explications de la réalisation du mandat dans sa globalité. Il a été convenu avec les experts lors de la première visite (le 20.05.2025) que les chapitres de la documentations pouvaient être adaptés en fonction du projet.

Acteurs principaux du projet :

- **Formateur en entreprise** : Frédéric Hertling
- **Supérieur professionnel** : Daniel Bütschi
- **Expert principal** : Alexis Clément
- **Expert secondaire** : Hans Wilhelm Schwendimann
- **Candidat** : Loïc Frédéric Jaquier

Méthodes de sauvegarde :

Les documents et fichiers sont sauvegardés en local sur l'ordinateur de l'école, sur OneDrive et sur GitHub. Pour tous les fichiers, il est donc possible d'obtenir un historique des versions grâce à

OneDrive et GitHub mais j'ai également réalisé une gestion des versions manuelles en créant un dossier par jour de travail qui contiens les fichiers dans l'état de la date mentionnée.

Répertoire local avec synchronisation OneDrive :

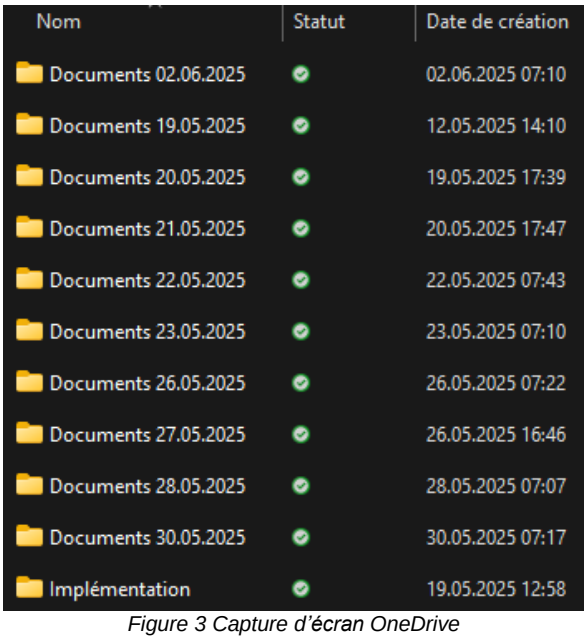


Figure 3 Capture d'écran OneDrive

Répertoire GitHub :

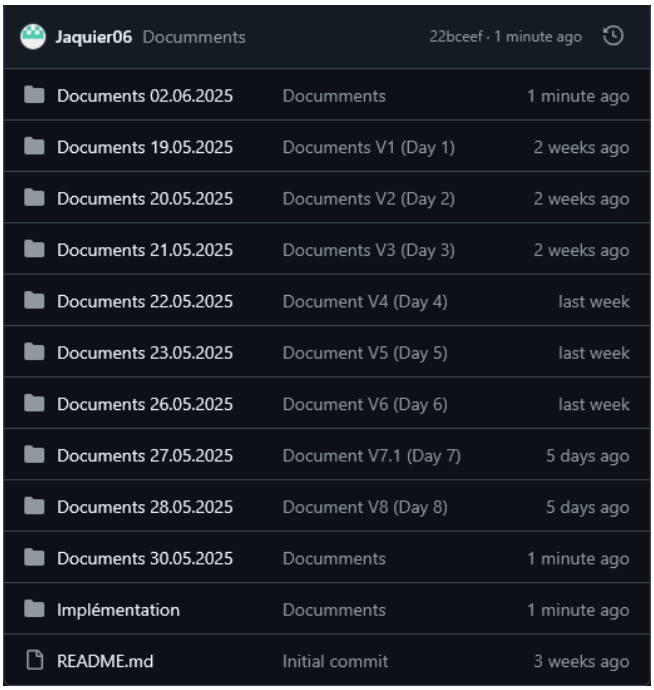


Figure 4 Capture d'écran GitHub

3 Analyse préliminaire

3.1 Objectifs du système

3.1.1 Analyse de l'état actuel

Voici une représentation du processus initial de référencement des nichoirs. On peut constater qu'il n'est pas strict et qu'il peut facilement devenir problématique.

Potentiels problèmes avec ce processus :

- Perte d'informations (plusieurs supports)
- Données non mises à jour
- Informations partiellement ou non répertoriées
- Méthodologies variant selon les différents membres

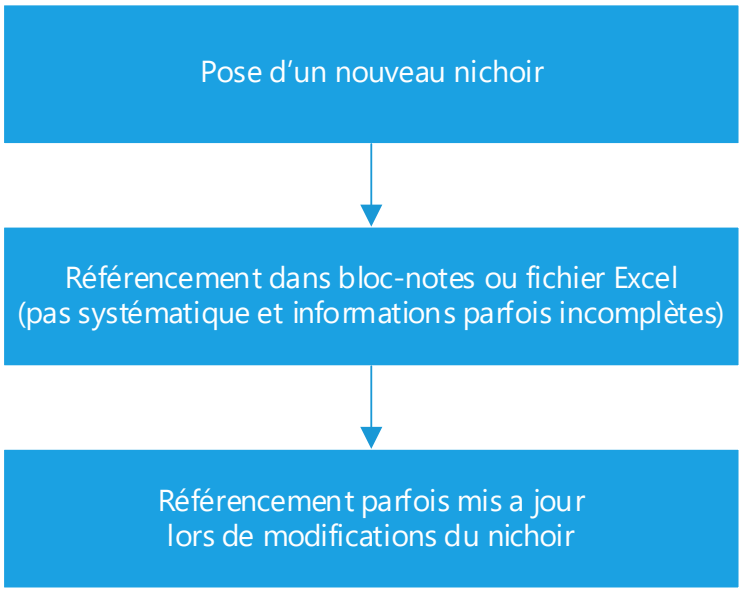


Figure 5 Représentation du processus initial

3.1.2 Analyse de l'état désiré

Voici ma proposition pour un nouveau processus de gestion des nichoirs :

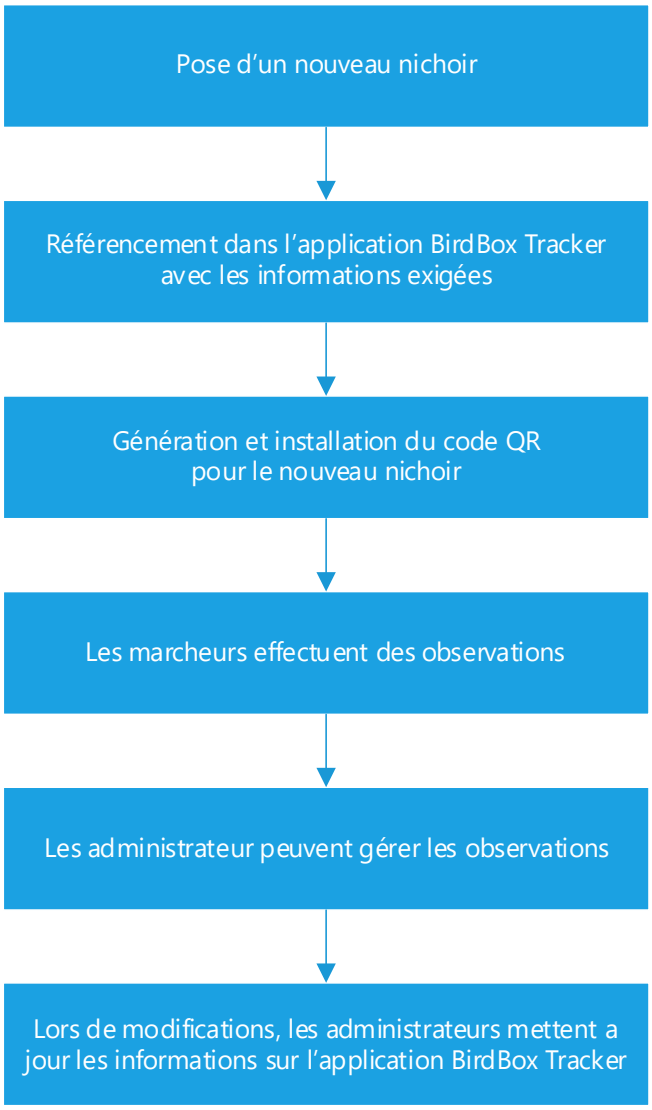


Figure 6 Représentation du processus final

Ce nouveau processus est plus complet et rigoureux que le processus initial. Cela permet de garantir que toutes les informations restent correctes et à jour. Si tout le monde respecte cette marche à suivre, on peut garantir une meilleure gestion des données.

Cela offre également d'autres fonctionnalités tel que les observations qui peuvent être laissées par les marcheurs. Les administrateurs peuvent ensuite en prendre compte pour des futur modifications, réparations ou autres interventions sur les nichoirs.

Aperçu de l'état final de l'application Birdbox Tracker :

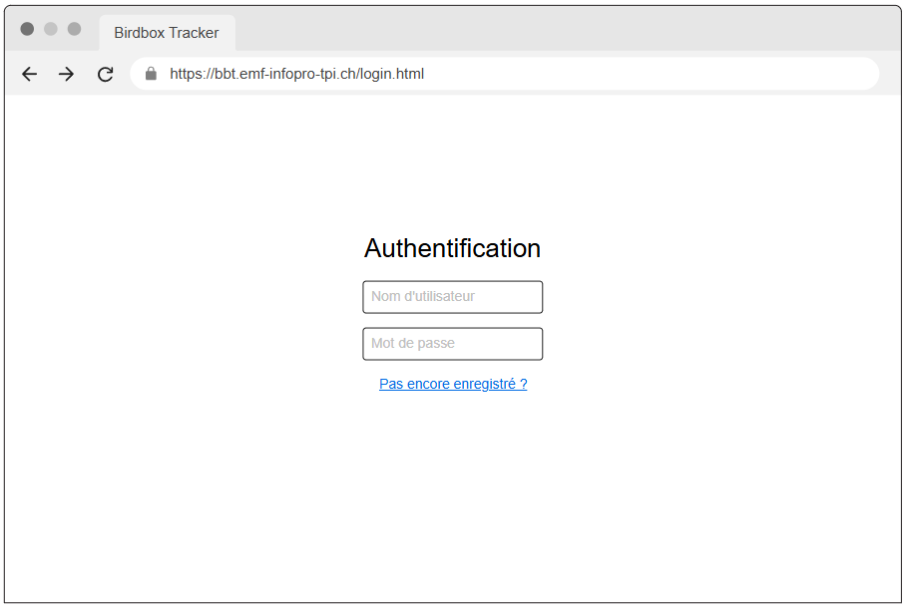


Figure 7 Maquette page d'authentification

Première page affichée lors du chargement de l'application.

L'utilisateur peut compléter les champs afin de se connecter avec son nom d'utilisateur ainsi que son mot de passe.

Il peut également accéder à la page d'inscription s'il n'est pas déjà enregistré dans l'application.

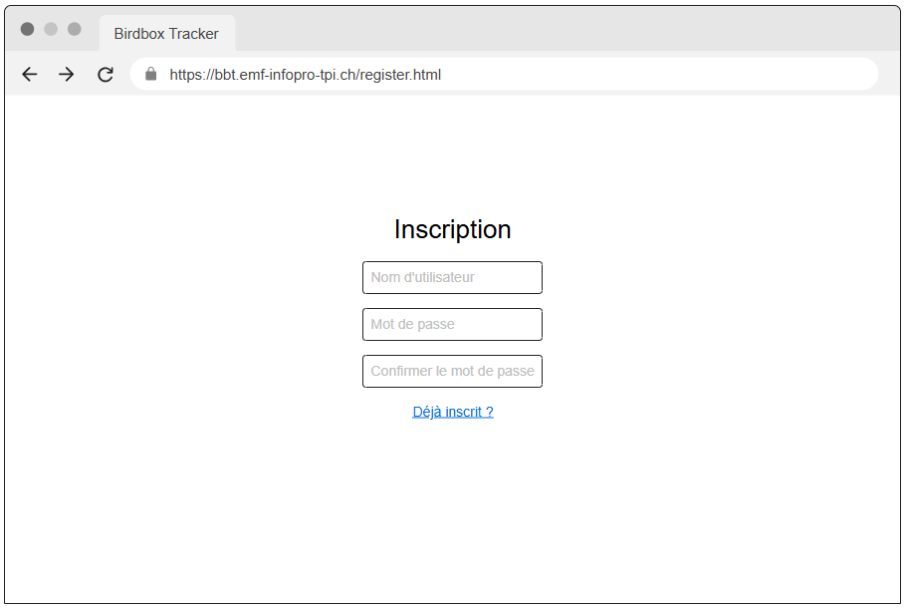


Figure 8 Maquette page d'inscription

Ceci est la page d'inscription qui permet de s'inscrire en tant que nouvel utilisateur si on ne possède pas déjà un compte.

Cette page est accessible depuis la page de connexion.

Il est bien entendu possible de revenir sur la page de connexion si l'utilisateur possède déjà un compte.

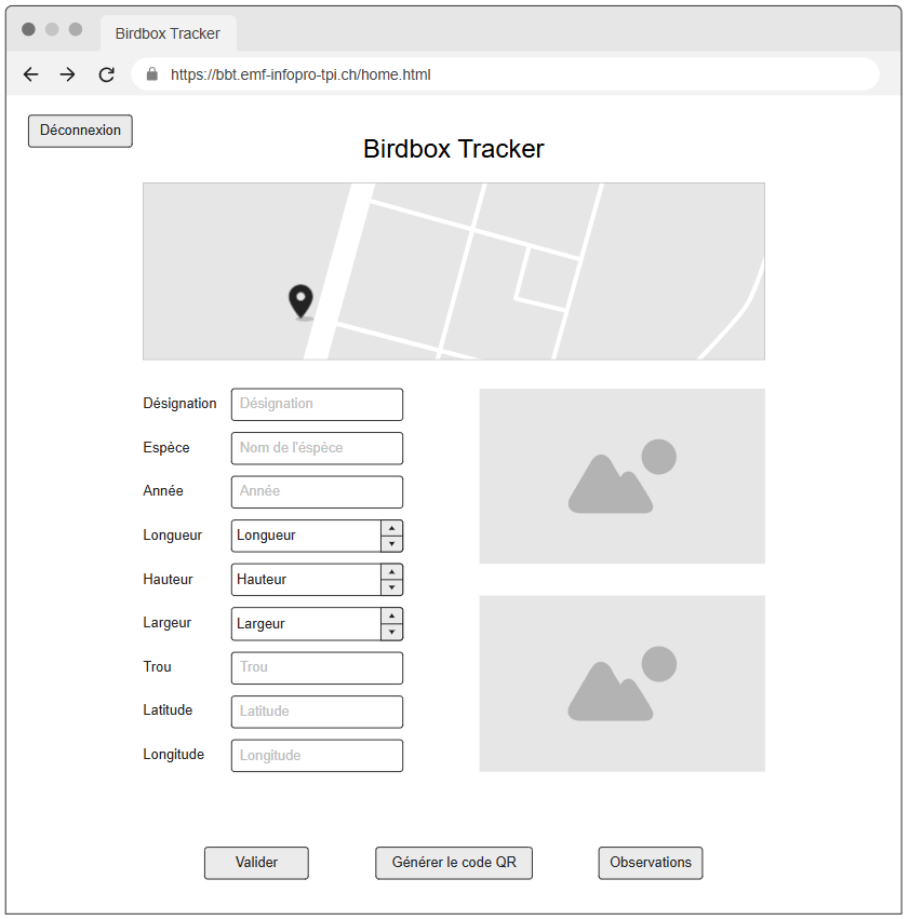


Figure 9 Maquette page de gestion des nichoirs

Ceci est la page principale de l'application, elle permet de gérer les nichoirs enregistrés.

Plusieurs options s'offrent à l'utilisateur :

- Visualisation des nichoirs sur la carte
- Gestion des nichoirs (Création et modification)
- Génération du code QR
- Accès aux observations du nichoir sélectionné
- Déconnexion de l'application

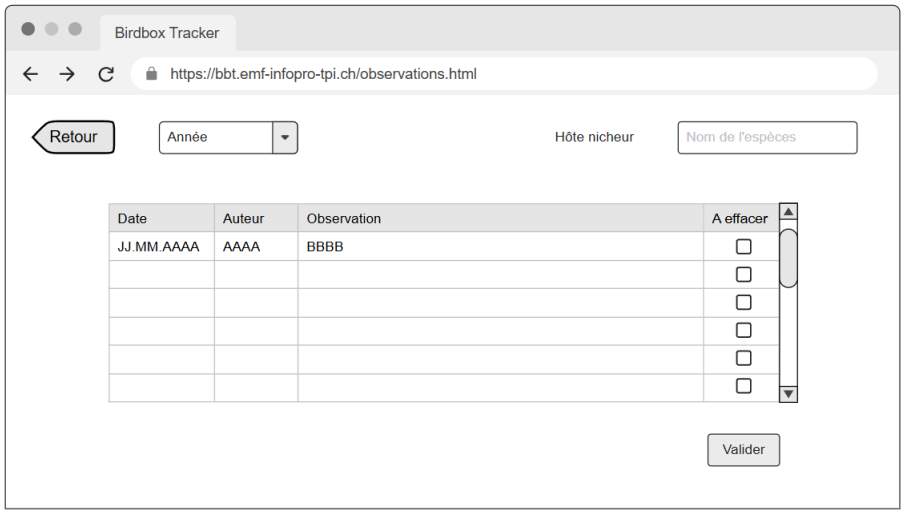


Figure 10 Maquette page de gestion des observation

Voici la page de gestion des observations qui permet d'afficher et de gérer toutes les observations effectuées par les promeneurs sur le nichoir sélectionné précédemment.

Les observations sont triées par année et peuvent être sélectionnées pour finalement les supprimer.

Une fois les modifications effectuées par l'utilisateur, il peut revenir sur la page précédente.

Figure 11 Page d'envoi des observations

Voici finalement la page qui permettra au promeneur d'envoyer les observations sur les nichoirs.

Cette page est accessible uniquement via les codes QR générés depuis la page principale et affichera les informations de l'espèce en fonction du nichoir.

Les promeneurs pourront donc entrer leur nom ainsi que leur commentaire et valider pour envoyer l'observation.

Les promeneurs n'ont pas accès au reste de l'application.

3.1.3 Objectifs

Voici le diagramme présentant le contexte global de mon application WEB. On y voit clairement les éléments interne et externe au projet.

Nous avons donc une application cliente (HTML, JS) qui va effectuer des requêtes sur une application backend PHP.

Le backend va rediriger les requêtes vers le service nécessaire en fonction de la requête passée par le client. On utilise une base de données MySQL pour stocker les données de l'application, l'API de Google Maps pour afficher des repères sur une carte et une librairie de génération de codes QR.

La librairie de génération des codes QR n'est pas directement intégrée à notre application mais elle est téléchargée sur le serveur. C'est pourquoi on la voit sur le schéma à l'interne de notre projet et non comme une API externe.

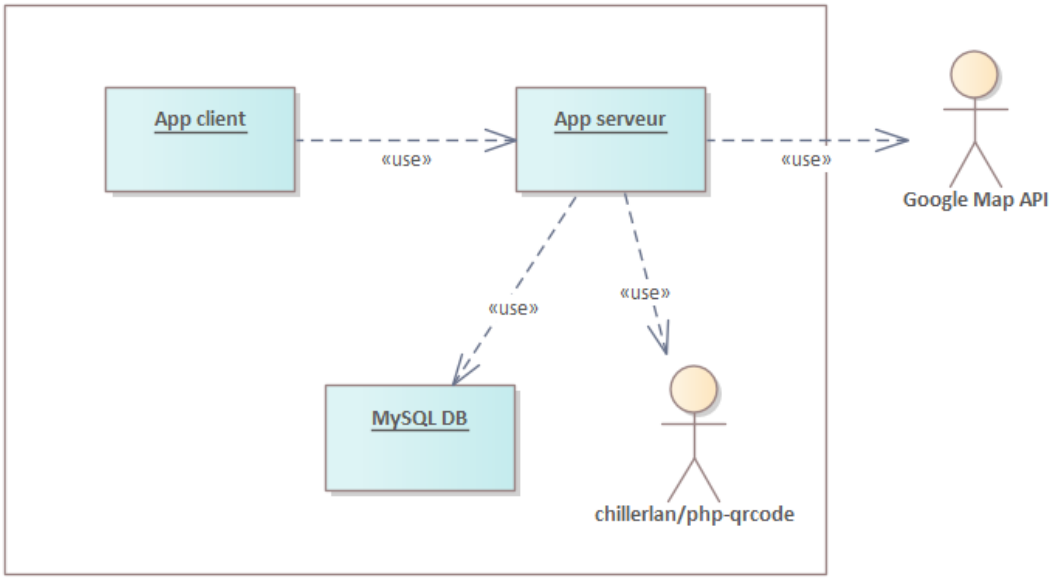


Figure 12 Diagramme de contexte

Voici également le schémas ER qui nous servira pour la conception et l'implémentation de notre base de données dans laquelle nous stockerons toutes les donnes de notre application.

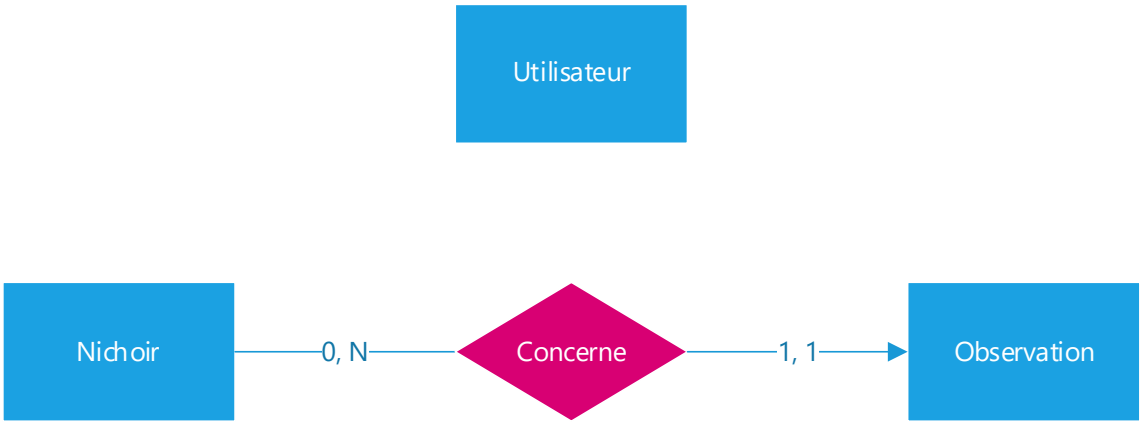


Figure 13 Diagramme entité-relation de la DB

On y voit les trois entités principales de notre programme (Utilisateur, Nichoir, Observation).

On peut constater que l'entité Utilisateur n'est pas lié aux autres. C'est explicable car elle nous servira uniquement pour la connexion sur notre application.

On voit ensuite le lien entre l'entité Nichoir et l'entité Observation. On peut lire la relation de la manière suivante : « Une observation concerne uniquement un seul nichoir tandis qu'un nichoir peut être concerné par une ou plusieurs observations. »

Liste des objectifs :

- Accès à une interface de gestion administrateur
  - Enregistrement, connexion et déconnexion utilisateur
  - Visualisation des nichoirs sur une carte
  - Visualisation des informations de chaque nichoir dans des champs de la page
  - Modification des informations des nichoirs dans les champs
  - Modification des coordonnées d'un nichoir en le déplaçant sur la carte
  - Ajout d'un nouveau nichoir en cliquent sur la carte
  - Visualisation de l'image du nichoir sélectionnée (si présente)
  - Ajout ou remplacement de l'image du nichoir sélectionné
  - Génération du code QR pour accéder la page d'observation du nichoir
  - Téléchargement du code QR
  - Affichage des observations du nichoir sélectionné
  - Sélectionner les observations puis valider pour supprimer
- Accès à une interface simple d'envoi d'observations pour les utilisateurs
  - Visualiser l'espèce du nichoir
  - Envoyer une observation après avoir complété les champs obligatoires (nom et observation)

3.1.4 Processus de l'application

Fonctionnalités globales de l'application :

Voici le schéma de cas d'utilisation de notre application, il représente toutes les actions possibles de l'application, du côté client et serveur. On visualise clairement quel acteur peut effectuer l'action et quels services sont appelés par ces dernières.

On y voit l'invité qui a la possibilité de se connecter ou s'enregistrer sur l'application. Une fois cette première action effectuée, il devient administrateur, ce qui lui offre les possibilités suivantes :

- Gestion des nichoirs
- Gestion des observations
- Génération du code QR
- Quitter l'application

Et on voit finalement l'utilisateur qui a pour seul action possible, l'envoie d'observation.

On voit également que toutes les actions citées précédemment interagissent avec la base de données mise à part la déconnexion qui nécessite uniquement le gestionnaire de session du côté du serveur ainsi que la génération des codes QR qui utilise la librairie de génération du coté serveur également.

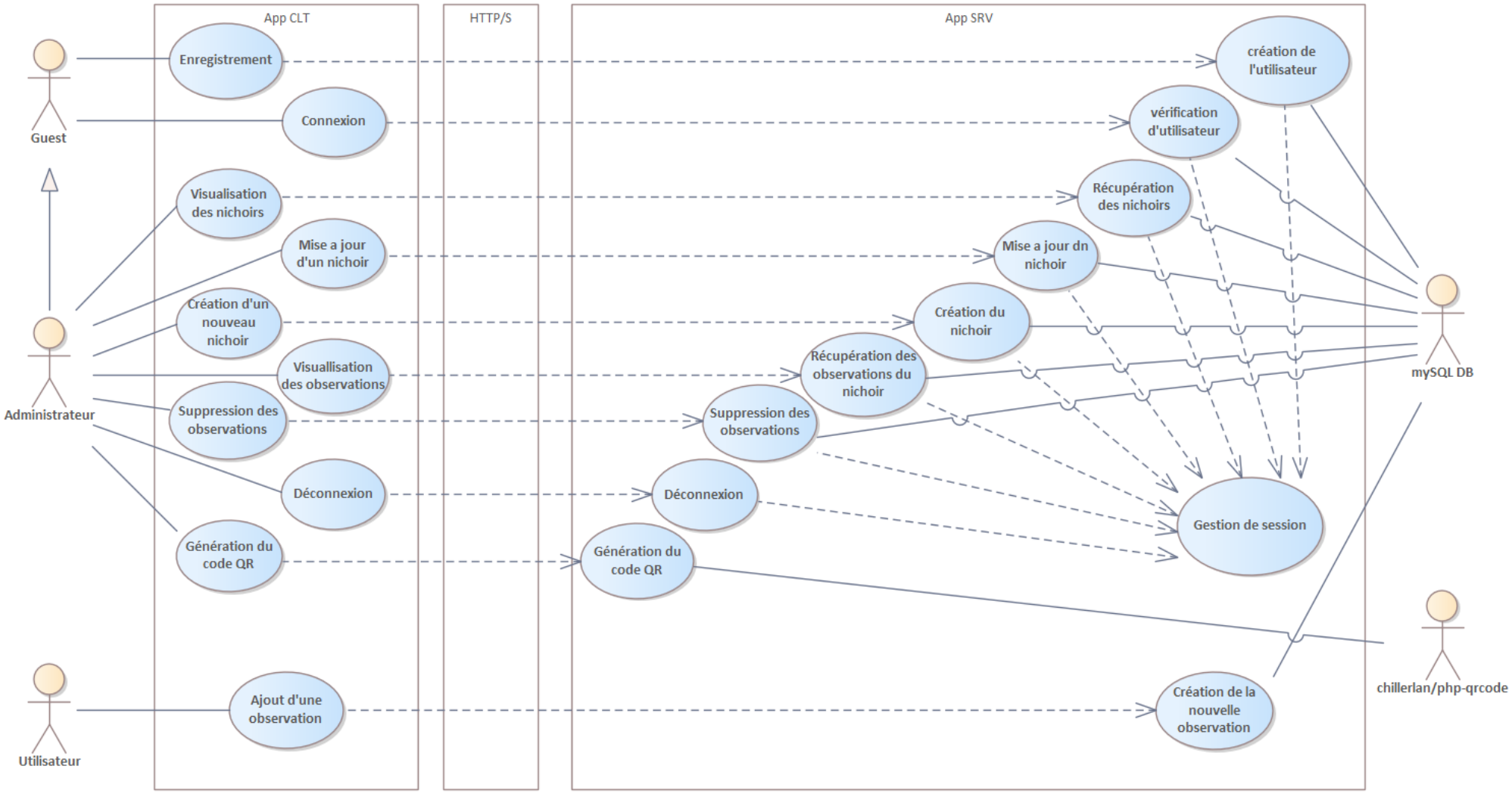


Figure 14 Diagramme de cas d'utilisation

Voici deux exemples de processus qui seront présent dans l'application :

Processus lors de l'ajout d'une observation par un marcheur :

Le marcheur commence par scanner le code QR du nichoir, il arrive ensuite sur une page ou il a la possibilité de remplir le formulaire d'observation avec son nom ainsi que son commentaire. Une fois fini, il le valide et le processus d'envoi de l'observation est lancé sur l'application client.

L'application vérifie tout d'abord que l'utilisateur a bien rempli les champs obligatoires. Elle envoie ensuite l'observation au backend. Si l'observation bien pu être enregistrée, un message de confirmation s'affiche. Sinon, un message d'erreur est renvoyé.

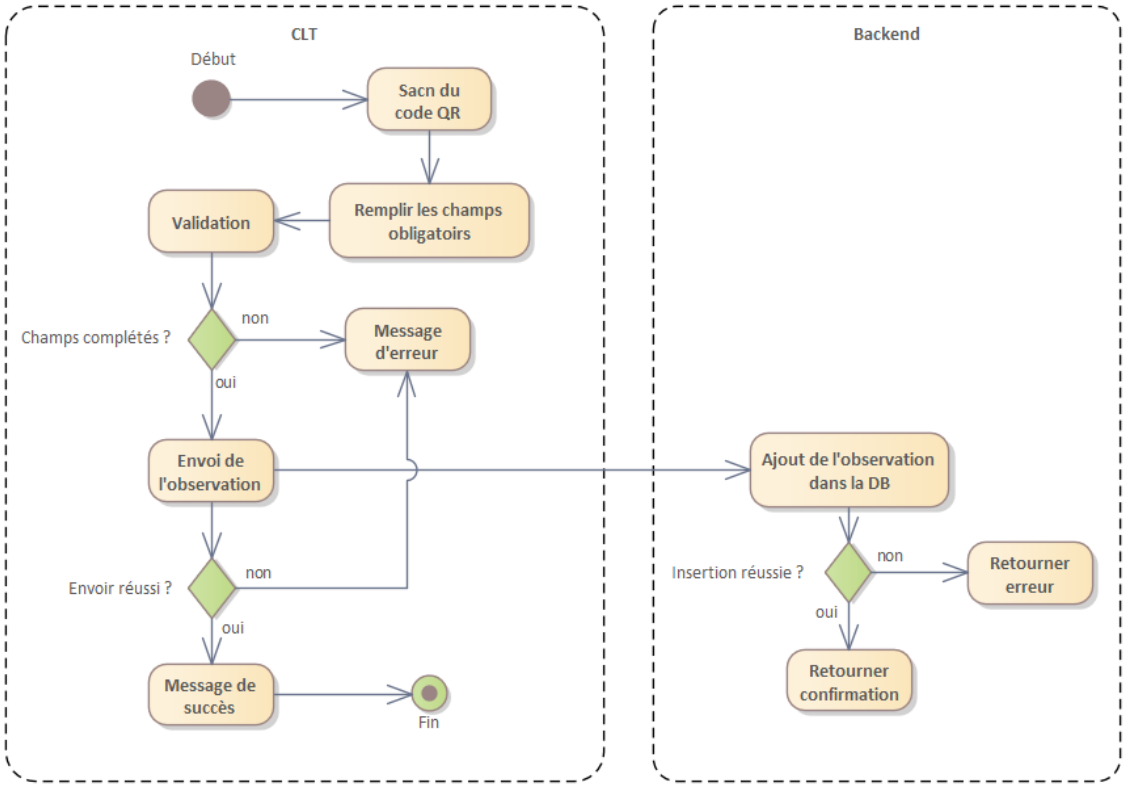


Figure 15 Diagramme d'activité (ajout d'une observation)

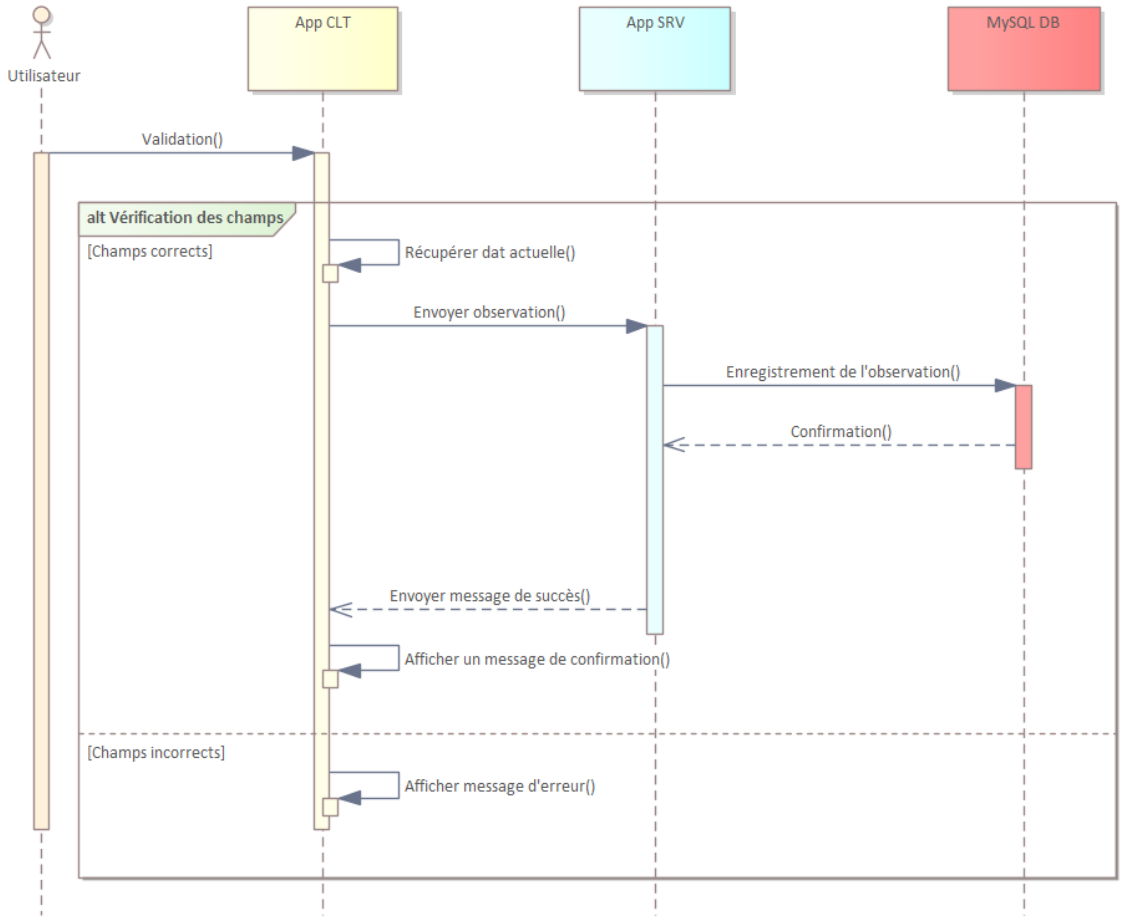


Figure 16 Diagramme de séquence (ajout d'une observation)

Processus lors de l'ajout ou de la modification d'un nichoir par l'administrateur :

Ce processus correspond à deux des cas d'utilisation présentés dans le diagramme Use Cas :

- Mettre à jour un nichoir
- Créer un nouveau nichoir

Avant le début de ce processus, l'administrateur aura soit sélectionné un nichoir et modifié ses informations pour une mise à jour, soit entré les informations pour la création d'un nouveau nichoir.

Lorsque l'administrateur valide l'action, le processus vérifie donc du coté client que toutes les informations ont correctement été entrées dans les champs de la page HTML. On vérifie ensuite s'il s'agit de la création d'un nouveau nichoir ou la modification d'un nichoir existant et on envoie au backend l'action correspondante avec les informations nécessaires à l'ajout ou à la mise à jour.

Du coté backend, avant d'effectuer toute requête dans la DB, on vérifie bien que la session est valide, que ce soit bien un utilisateur authentifié qui réalise l'action. On retourne ensuite au client un message d'erreur ou de succès en fonction du déroulement de l'interaction avec la DB et de la vérification de la session.

Le client affiche finalement un message de confirmation si tout s'est déroulé comme prévu. Sinon il rend un message d'erreur.

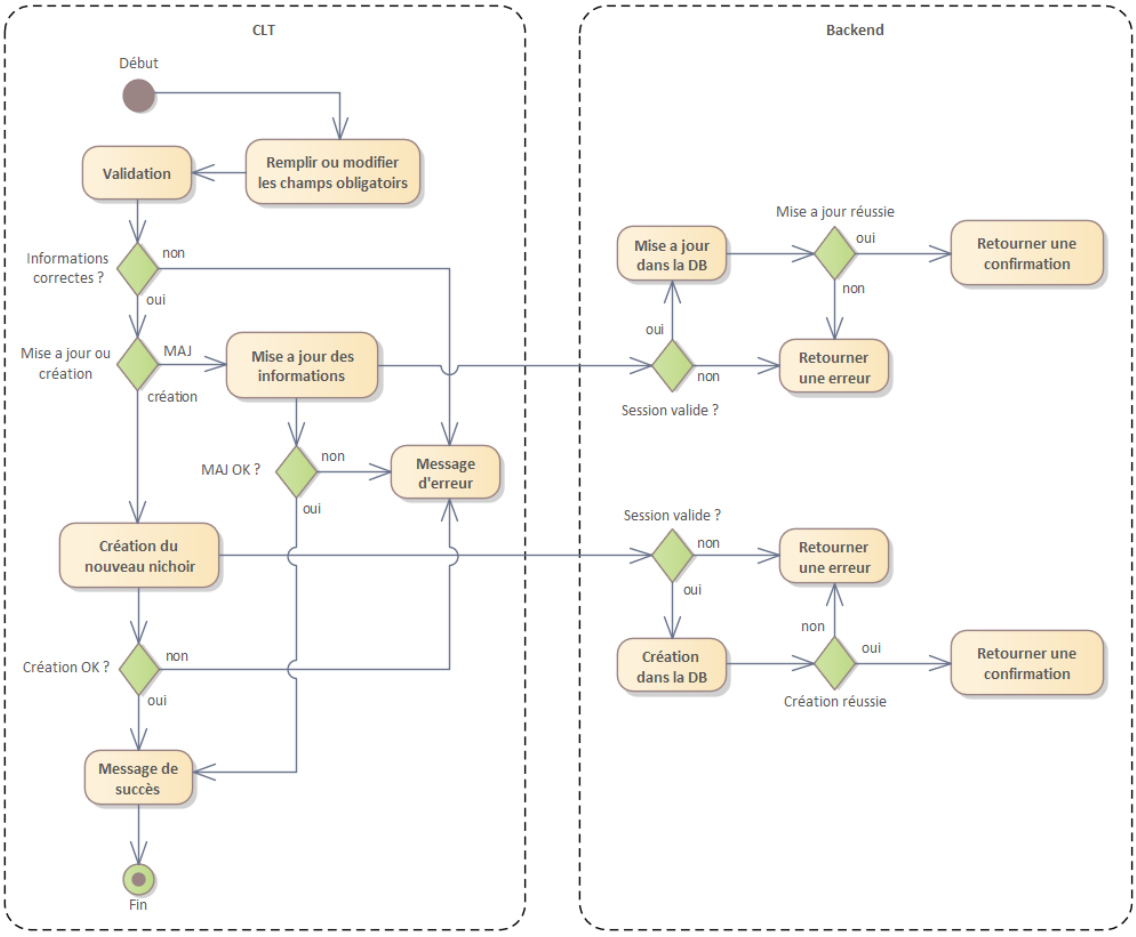


Figure 17 Diagramme d'activité (création / mise à jour d'un nichoir)

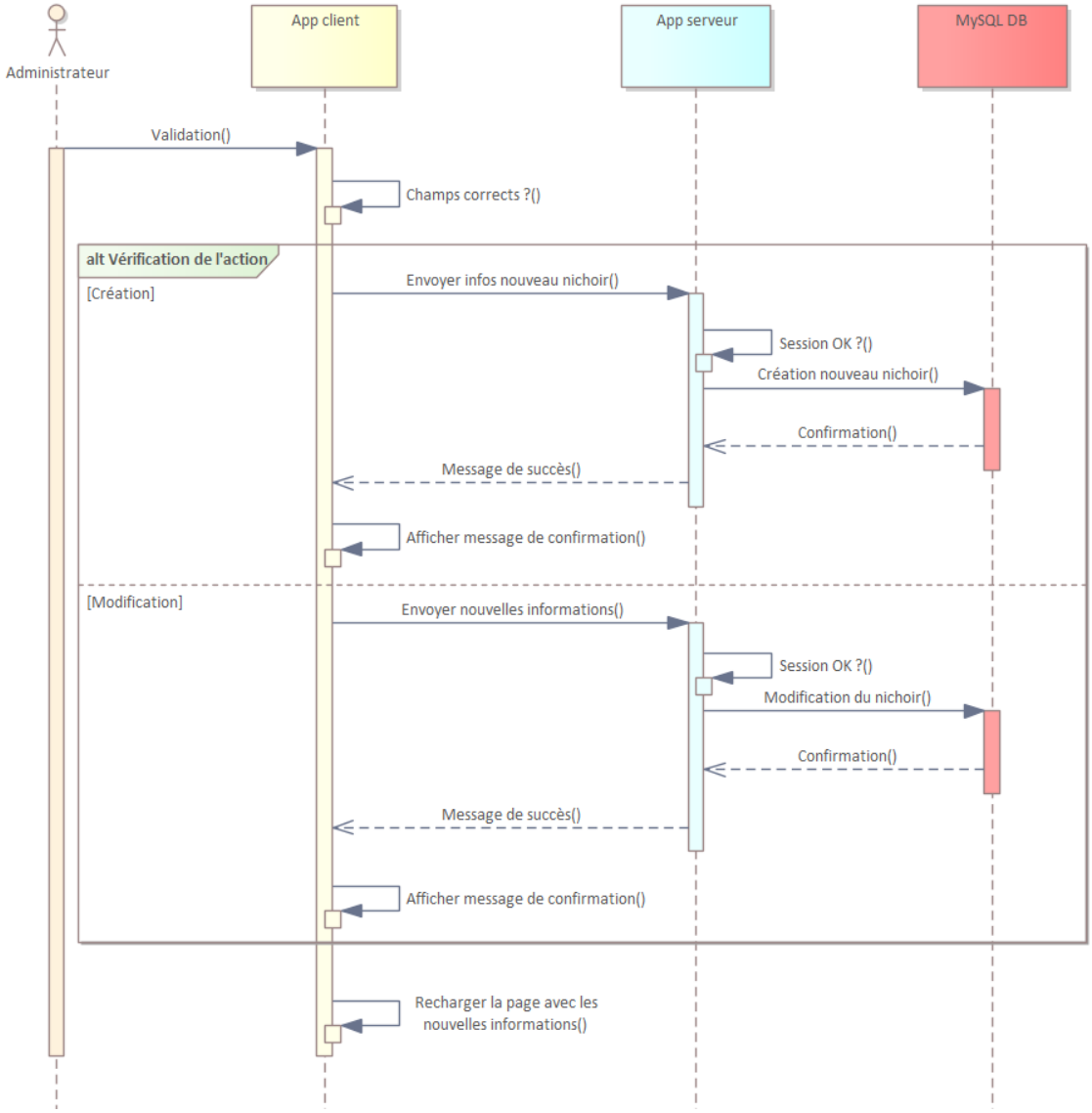


Figure 18 Diagramme de séquence (création / mise à jour d'un nichoir)

3.2 Variantes

3.2.1 Variante 1 (Génération du code QR)

Possibilité 1 :

Génération des codes QR du coté Frontend. Cette solution ne nécessite pas l'utilisation de dépendances externe ce qui rend l'implémentation plus simple.

Possibilité 2 :

Génération des codes QR du coté Backend. Pour cette solution, nous pouvons utiliser la librairie suggérée dans le cahier des charges (PHP-QRCode). Nous devons donc télécharger cette librairie dans notre projet pour pouvoir l'utiliser.

Choix et justification :

Bien que cela complique légèrement l'implémentation, j'implémenterai mon application avec la génération des codes QR du coté backend. Cela nous permet de bien faire la différence entre le frontend qui sert uniquement d'affichage et le backend qui est la partie métier de notre application.

3.2.2 Variante 2 (Utilisation des Marker)

Possibilité 1 :

Utilisation de « Marker » Google Maps pour afficher les points sur la carte. Cette solution fonctionne correctement et possède toutes les fonctionnalités nécessaires pour implémenter une application en respectant le cahier des charges. Malheureusement, lorsque j'ai réalisé les tests technologiques pour cette fonctionnalité, j'ai constaté que les « Marker » Google Maps seraient prochainement obsolètes.

Possibilité 2 :

Utilisation de « AdvancedElementMarker » Google Maps pour afficher les points sur la carte. Cette solution utilise une fonctionnalité a jour de l'API Google Map. Son principal inconvénient est qu'elle ne possède pas l'option « drag-and-drop ». Cette option est requise dans le cahier des charges.

Choix et justification :

Dans un premier temps, j'ai choisi de garder les « Marker » simples afin de suivre les attentes du cahier des charges. Il faudrait néanmoins prévoir une modification de cette fonctionnalité pour garantir le bon fonctionnement de l'application sur le long-terme.

3.2.3 Variante 3 (Gestion du code QR)

Possibilité 1 :

La première solution qui est proposé dans le cahier des charges est de générer et télécharger le code QR (sans l'afficher) pour le nichoir lorsque l'utilisateur appuie sur le bouton de génération.

Possibilité 2 :

La deuxième solution (proposée par moi-même) est de générer et afficher automatiquement le code QR lorsqu'on affiche les informations du nichoir et de remplacer le bouton de génération par un bouton de téléchargement. L'utilisateur peut donc ensuite télécharger le code QR en utilisant ce bouton.

Choix et justification :

J'ai finalement choisi la deuxième solution car elle nous permet de visualiser directement le code QR. Nous avons donc la possibilité de le tester directement sur la page web avant de le télécharger.

Je trouve également ce système plus logique et intuitif que la solution proposée dans le cahier des charges.

3.2.4 Variante 4 (Choix de l'API de géolocalisation)

Possibilité 1 :

Utiliser l'API Google Maps suggérée dans le cahier des charges.

Possibilité 2 :

Utilisation de l'API OpenStreetMap pour afficher la carte ainsi que les points sur celle-ci.

Choix et justification :

Premièrement, comme l'API Google Maps était initialement suggérée dans le cahier des charges, cela a déjà orienté mon choix vers cette solution.

Deuxièmement, l'API google Maps est très répandue, elle est très bien documentée et compatible avec la plupart des navigateurs et plateformes.

Finalement, elle offre une grande variété de fonctionnalités comme des cartes interactives, le calcul d'itinéraire et autre. Nous n'avons pour l'instant pas besoins de tant de fonctionnalités, mais si on imagine que notre application va évoluer, il est intéressant d'utiliser une solution avec une grande marge d'évolution.

3.2.5 Variante 5 (Page d'envoi d'observation)

Possibilité 1 :

Récupérer l'espèce du nichoir dans les paramètres du code QR pour l'afficher sur la page d'envoi d'observation.

Possibilité 2 :

Charger les informations du nichoirs grâce à une requête vers le backend en passant la pk du nichoir scanné en paramètre.

Afficher ensuite l'espèce récupérée dans les attributs du nichoir sur la page.

Choix et justification :

J'ai choisi d'utiliser la seconde solution pour une raison purement logique. Si on inclut le nom de l'espèce dans le lien du code QR, lorsque le nichoir sera modifié, il faudra à nouveau télécharger, imprimer et placer le code QR sur le nichoir pour le maintenir à jour.

En revanche, la PK du nichoir ne sera jamais modifiée. Donc si on effectue une requête avec cette dernière pour récupérer l'espèce dans la DB. On est sûr d'obtenir les dernières informations à jour lors du chargement de la page.

3.3 Analyse de risque

Le principal risque en cas de non-aboutissement de ce projet, c'est de ne pas pouvoir proposer la solution de gestions des nichoirs à la société d'ornithologie. Cela n'a pas de grave impacte directement car la méthode initiale (expliquée au point 2.1) continuera d'être utilisée. Mais cette première solution limitera tout de même la société et l'empêchera d'évoluer convenablement au cours du temps.

3.4 Sécurité de l'information et protection des données

- **Injection**

  Les injections SQL, JavaScript et HTML doivent être traitées et contré pour éviter toutes modifications non désirées dans la base de données, dans le fonctionnement ou sur l'affichage.

- **Gestion des accès**

  Une gestion des sessions doit être mise en place afin de vérifier si les actions que l'utilisateur tente d'effectuer sont autorisée ou non.

- **Protection des données sensibles**

  Les donnes sensibles de l'applications doivent être le moins visible possible. Que ce soit lors de la saisie, du transfert de données ou du stockage.

4 Concept

4.1 Architecture du système

Voici le diagramme de représentation des différentes classes de l'application client de mon projet. La structure du projet respecte le modèle MVC.

On peut constater que nous avons toutes nos vues définies précédemment dans les maquettes. Pour chacune des vues HTML, nous avons un controller JS et les appels vers le backend sont tous effectués dans un même fichier worker JS.

Il y a également les fichiers bean qui nous permettrons de transférer des objets au travers de notre application plutôt que des données directement.

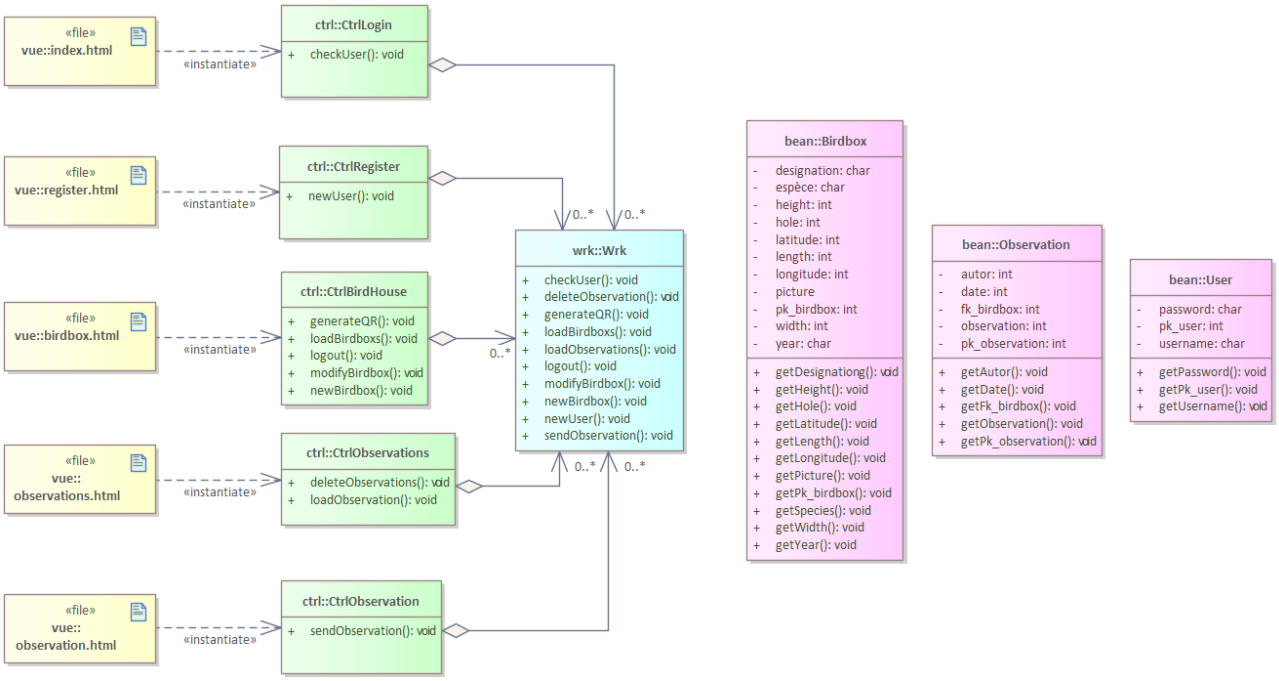


Figure 19 Diagramme de classe client

Voici le diagramme de class pour la partie serveur de notre application cette-fois.

On y voit le script PHP qui se charge de récupérer les requêtes HTTP transmises par le client. Les informations reçues sont par la suite transmises au controller puis au worker de chacun des bean en fonction du type de données concerné (Birdbox, User ou Observation). Les requêtes sur la DB sont ensuite exécutées depuis chacun des worker à l'aide de la classe WrkDB.

On peut également constater une classe CtrlQR qui nous permet d'effectuer les requêtes sur la librairie de génération de codes QR.

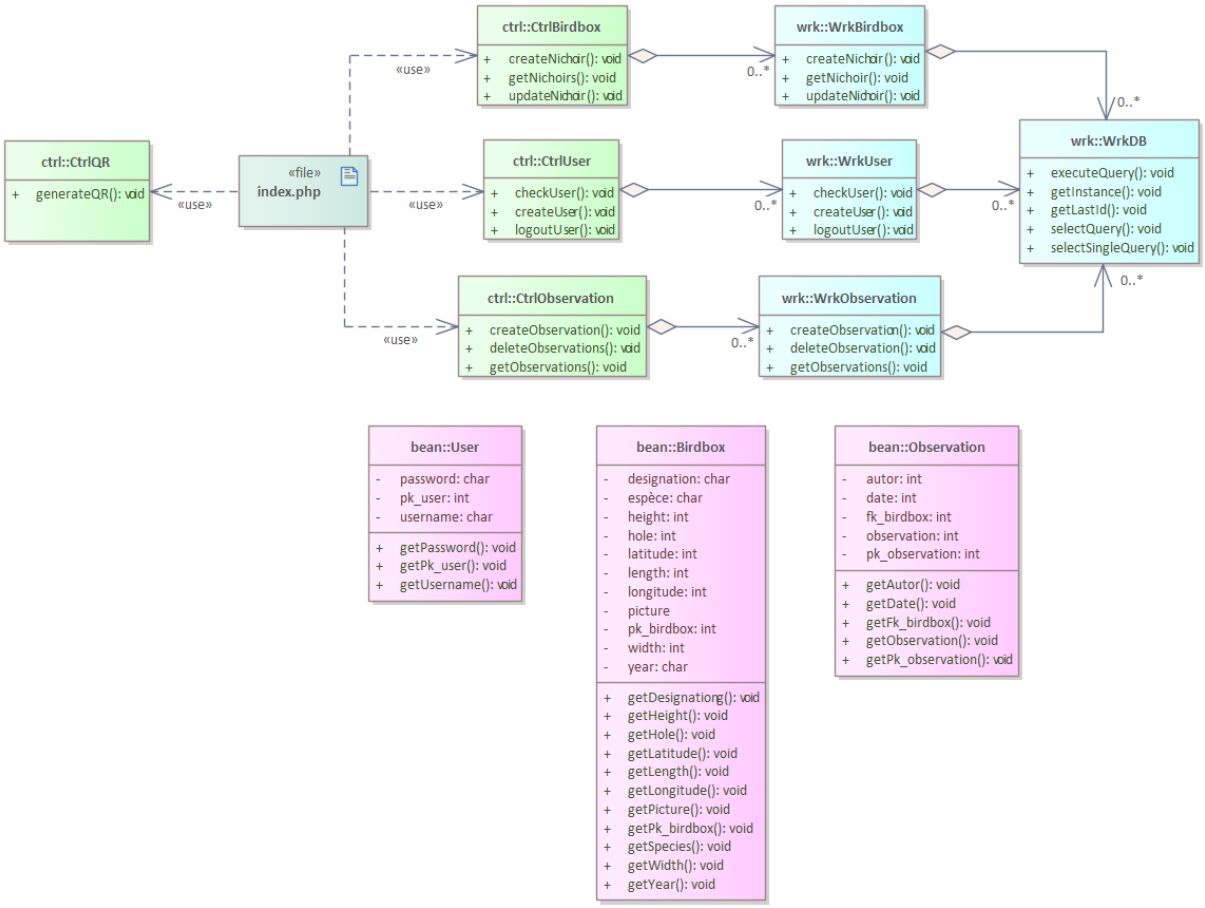


Figure 20 Diagramme de classe serveur

Voici la liste des différentes entités que nous aurons besoin pour implémenter notre application :

Nichoir	Observation	Utilisateur
• Désignation • Espèce • Année • Latitude • Longitude • Longueur • Largeur • Hauteur • Tour • Image	• Date • Auteur • Observation	• Nom d'utilisateur • Mot de passe

Et voici finalement le schéma relationnel de la base de données basé sur le schéma ER présenté dans l'analyse. En comparaison avec la liste ci-dessus, on ajoute le référencement des entités grâce au clés primaire et secondaires.

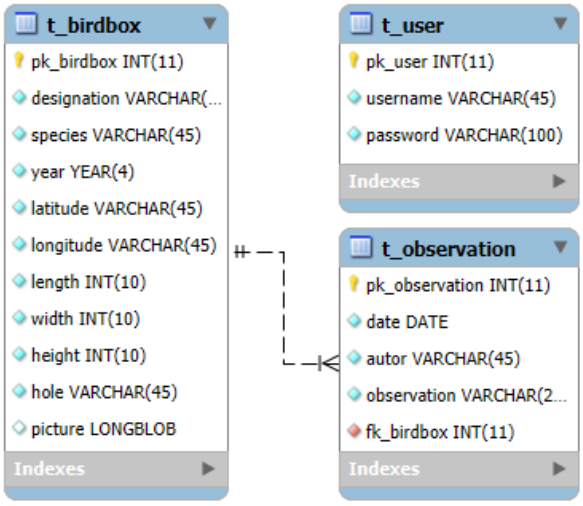


Figure 21 Schéma relationnel de la DB

Ce schéma respecte des conventions de nommage pour les tables, les clés primaires et les clés étrangères.

On voit tous les champs ainsi que leur type pour chacune des tables. Ils stockeront toutes les données nécessaires au fonctionnement de notre application.

On y voit la liaison entre la table « t_birdbox » et la table « t_observations » grâce à la clé étrangère « fk_birdbox » qui se situe dans la table des observations.

4.2 Conception des processus défini dans l'analyse (2.1.4)

Voici les schémas de séquences d'interactions des deux processus définis plus tôt dans l'analyse de ce projet.

On y voit la communication entre nos différents acteurs (client, serveur et base de données), mais cette fois de manière bien plus détaillée.

On peut visualiser les instances de classes ainsi que les méthodes qui sont appelées sur ces dernières. On voit également clairement les vérifications effectuées lors des différentes interactions entre les parties du programme.

Explication du diagramme de séquence d'interaction (création / mise à jour d'un nichoir) :

Une seule action est effectuée par l'utilisateur, la validation. On vérifie ensuite si les champs sont bien complétés. Dans le cas échéant, on retourne un message d'erreur. On vérifie ensuite quelle action l'utilisateur souhaite effectuer : une modification ou un ajout. Pour les deux actions, le processus est très similaire.

On envoie la requête correspondante vers le serveur qui va s'occuper de vérifier si la session est valide. Si elle ne l'est pas, on retourne une erreur. Si elle l'est, on peut envoyer la requête SQL d'ajout ou de modification vers là-bas de données.

On retourne ensuite vers le client avec le résultat de la requête. En fonction du résultat, le client affiche un message de confirmation ou d'erreur.

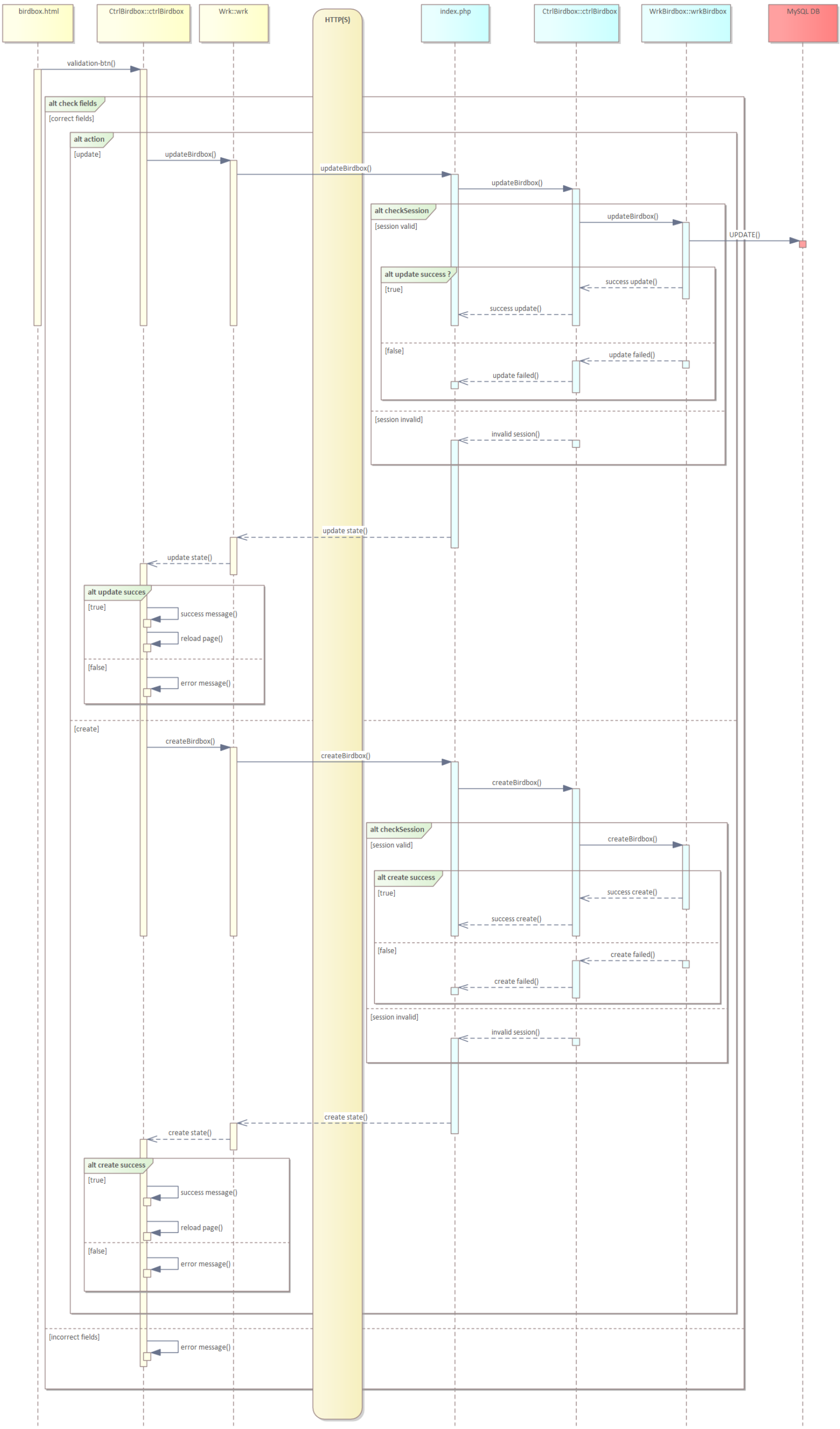


Figure 22 Diagramme de séquence d'interaction (création / mise à jour d'un nichoir

Explication du diagramme de séquence d'interaction (ajout d'une observation) :

Lorsque l'utilisateur valide l'envoi de l'observation, on vérifie si les champs obligatoires sont complétés. Sinon on affiche une erreur.

Ensuite on transmet la requête au serveur qui va à son tour effectuer la requête SQL d'insertion dans la base de données.

On retourne ensuite le résultat au client qui affiche un message de confirmation si tout s'est bien passé. Sinon, il affiche à nouveau un message d'erreur.

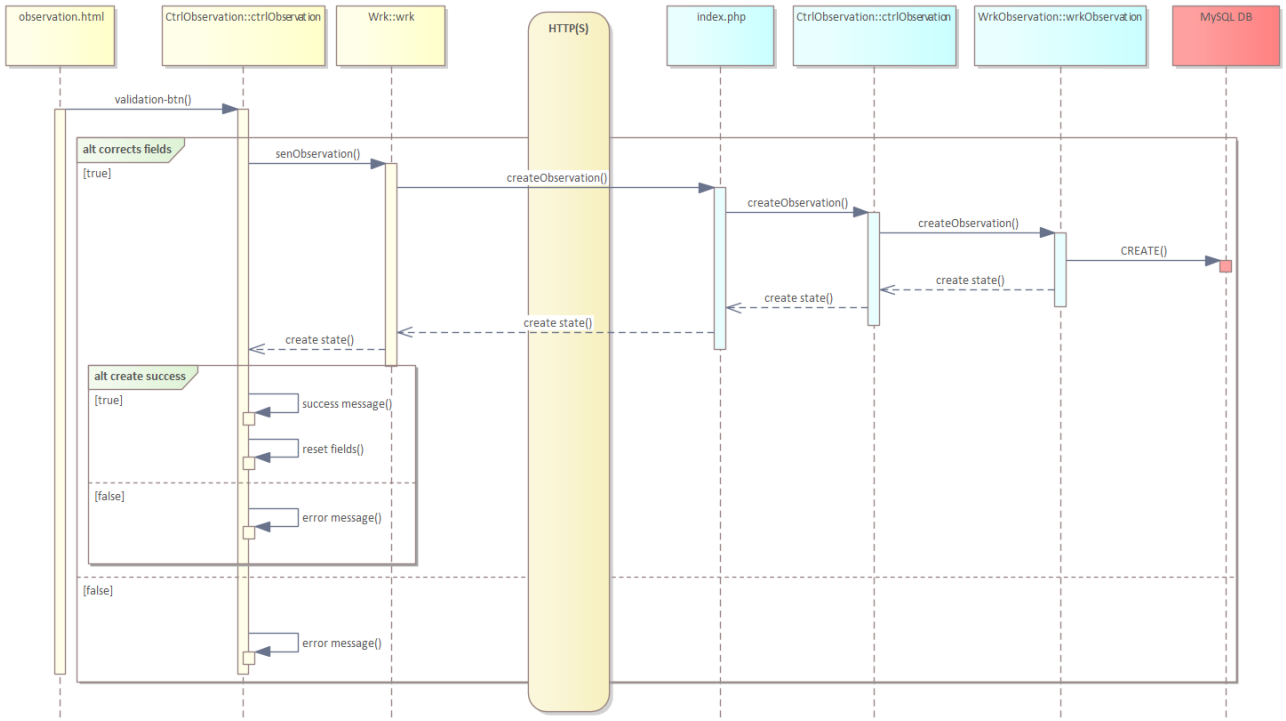


Figure 23 Diagramme de séquence d'interaction (ajout d'une observation)

4.3 Plan d'intégration des systèmes

Liste des systèmes et des technologies :

- Client (HTML / JavaScript)
- Serveur (PHP)
- Base de données (MySQL)
- API Google Maps (importé)
- librairie chillerlan/php-qrcode (téléchargé dans les fichiers du projet)

Liste des communications entre les systèmes :

Système 1	Système 2	Méthode de communication
Client	Serveur	Requêtes HTTPS
Serveur	Base de données	Requêtes SQL
Client	Google Map	Import de l'API puis de ces librairies
Serveur	QR code	Téléchargement de la librairie sur le serveur et utilisation de ses méthodes

4.4 Concept d'implémentation

L'implémentation débutera par la mise en place de la base de données selon les modèles établis durant la conception.

L'application sera ensuite implémentée fonctionnalité par fonctionnalité selon la liste suivante :

1. Inscription et authentification utilisateur
2. Affichage de la carte, des nichoirs et de leurs informations
3. Edition des informations des nichoirs
4. Ajout et modification des images par nichoir
5. Génération de codes QR
6. Interface de saisi des observations
7. Visualisation et gestion des observations par nichoir
8. Implémentation du style pour toute l'application

Chaque fonctionnalité sera ajoutée du coté client et serveur en temps voulu selon l'ordre chronologique ci-dessus.

Des tests fonctionnels continus sont réalisés tout au long de l'implémentation pour valider les fonctionnalités en cours de développement. Un test fonctionnelle final est également prévu à la fin de l'implémentation pour déceler les derniers problèmes et documenter le bon fonctionnement de l'application.

L'implémentation du style global se fait à la fin afin de garantir au minimum le fonctionnement complet de l'application.

4.5 Concept de tests

Voici un concept de test Blackbox pour l'application cliente de mon projet. C'est-à-dire que l'on va réaliser les tests du point de vue de l'utilisateur qui ne connais ni la structure, ni le fonctionnement de l'application.

Les tests abordent les fonctionnalités principales de l'application avec tous les cas de figures importants. Voir la réalisation des tests sur l'application au point 6 du rapport.

Le navigateur Google Chrome sera utilisé pour la réalisation de ces tests (plus de détails sur la version au point 6.2).

Voici la liste des actions qui ne sont pas testées dans ce protocole :

- Tests lorsque le backend est inaccessible
- Test avec différents navigateurs
- Injection dans tous les champs de l'application (uniquement login et la page de gestion des nichoirs)

Nr.	Objet testé	Description du test	Attente
1.01	Page d'inscription	Inscription avec les champs vides	Message d'erreur
1.02	Page d'inscription	Inscription avec un utilisateur existant	Message d'erreur
1.03	Page d'inscription	Inscription avec un nouvel utilisateur mais deux mots de passe différents	Message d'erreur
1.04	Page d'inscription	Inscription avec un nouvel utilisateur mais un mot de passe de moins de 8 caractères	Message d'erreur
1.05	Page d'inscription	Inscription avec un nouvel utilisateur et des champs corrects	Affichage de la page de gestion des nichoirs
2.01	Page d'authentification	Connexion avec les champs vides	Message d'erreur
2.02	Page d'authentification	Connexion avec un utilisateur inexistant	Message d'erreur
2.03	Page d'authentification	Connexion avec un utilisateur existant mais mauvais mot de passe	Message d'erreur
2.04	Page d'authentification	Connexion avec des identifiants corrects	Affichage de la page de gestion des nichoirs
3.01	Page de gestion des nichoirs	Chargement de la page lorsque l'utilisateur se connecte	Chargement de la carte Google Maps avec les points des nichoirs enregistrés
3.02	Page de gestion des nichoirs	Déconnexion de l'application	Retour sur la page d'authentification
3.03	Page de gestion des nichoirs	Sélection d'un point d'un nichoir sur la Map	Affichage des informations dans les champs de la page
3.04	Page de gestion des nichoirs	Validation avec champs incomplets après avoir créé un nouveau point sur la carte (nouveau nichoir)	Message d'erreur
3.05	Page de gestion des nichoirs	Validation avec champs incomplets après avoir cliqué sur un nichoir sur la carte (mise à jour du nichoir)	Message d'erreur
3.06	Page de gestion des nichoirs	Validation avec champs complets après avoir créé un nouveau point sur la carte (nouveau nichoir)	Message de confirmation
3.07	Page de gestion des nichoirs	Validation avec champs complets après avoir cliqué sur un point sur la carte (mise à jour du nichoir)	Message de confirmation
3.08	Page de gestion des nichoirs	Déplacement d'un nichoir sur la carte	Demande de confirmation du nouvel emplacement
3.09	Page de gestion des nichoirs	Insertion d'une image	Affichage de l'image
3.10	Page de gestion des nichoirs	Télécharger le code QR sans avoir sélectionné de nichoir	Message d'erreur
3.11	Page de gestion des nichoirs	Télécharger le code QR après avoir sélectionné un nichoir	Téléchargement du code QR
3.12	Page de gestion des nichoirs	Accès aux observations sans avoir sélectionné un nichoir	Message d'erreur
3.13	Page de gestion des nichoirs	Accès aux observations après avoir sélectionné un nichoir	Affichage de la page des observations du nichoir
4.01	Gestion des observations	Valider la suppression sans avoir sélectionné d'observations	Message d'erreur
4.02	Gestion des observations	Valider la suppression après avoir sélectionné des observations	Message de confirmation

5.01	Code QR	Scan du code QR d'un nichoir	Chargement de la page d'envoi d'observation avec les informations concernant l'espèce du nichoir
6.01	Ajout d'observation	Validation de la nouvelle observation avec des champs vides	Message d'erreur
6.02	Ajout d'observation	Validation de la nouvelle observation avec des champs corrects	Message de confirmation
7.01	Gestion des accès	Chargement de la page de gestion des nichoirs sans être connecté	Redirection sur la page d'authentification
7.02	Gestion des accès	Chargement de la page de gestion des observations sans être connecté	Redirection sur la page d'authentification
8.01	Sécurité	Injection HTML/JS (page de gestion des nichoirs) Validation de la mise à jour du nichoirs avec une injection de script JS dans des balises HTML dans un des champs du nichoir	Affichage de l'injection sans modification du comportement de l'application
8.03	Sécurité	Injection SQL (page de login) Tentative de connexion avec une injection SQL a la place du nom d'utilisateur	Message d'erreur

4.6 Moyens nécessaires

Voici la liste des Applications ou sites utilisés pour la réalisation de ce projet :

Plateforme	Utilité	(configuration)
Word	Rédaction du rapport de TPI	-
Excel	Planning et journal	-
Teams / Outlook	Communication	-
OneDrive / GitHub	Stockage des fichiers	-
Moqups	Maquettes	-
Enterprise Architect / Visio	Création des schémas	-
VS Code	Editeur de code	-
Google Cloud console	Gestion de l'API Google Map	Google Cloud (connexion avec compte google)
CPanel	Gestion de l'hébergement	Accès : CPanel Utilisateur : bbt Mot de passe : (sur demande au SUP)
Win SCP	Gestion des fichiers	Accès : bbt.emf-infopro-tpi.ch (port : 22) Utilisateur : bbt Mot de passe : (sur demande au SUP)
MySQL Workbench	Gestion de la base de données	Accès : bbt.emf-infopro-tpi.ch (port : 3306) Utilisateur : bbt_user Mot de passe : (sur demande au SUP)

5 Réalisation

5.1 Structure finale de l'application

Voici la représentation des structures de la partie client et serveur de l'application. On peut constater que la structure finale respecte globalement celle initialement proposée dans la conception.

Mis à part une légère variation au niveau des beans du client (voir conclusion d'implémentation pt.5.7) et l'apparition du dossier « ressources » qui contiens toutes les images nécessaire à l'affichage du client.

Structure du client

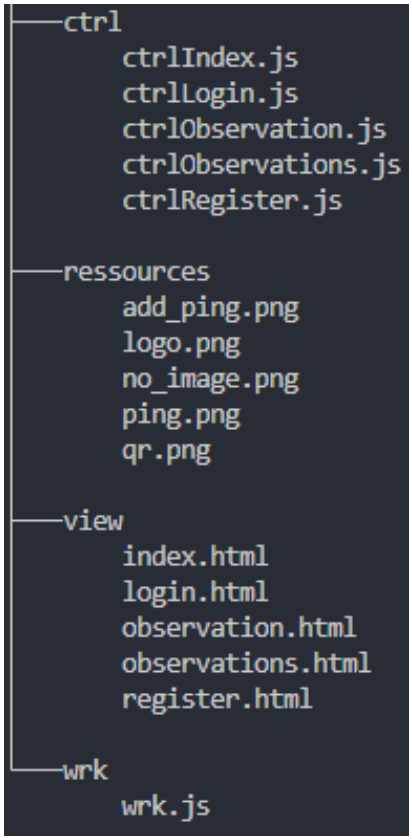


Figure 24 Structure du client

Structure du serveur

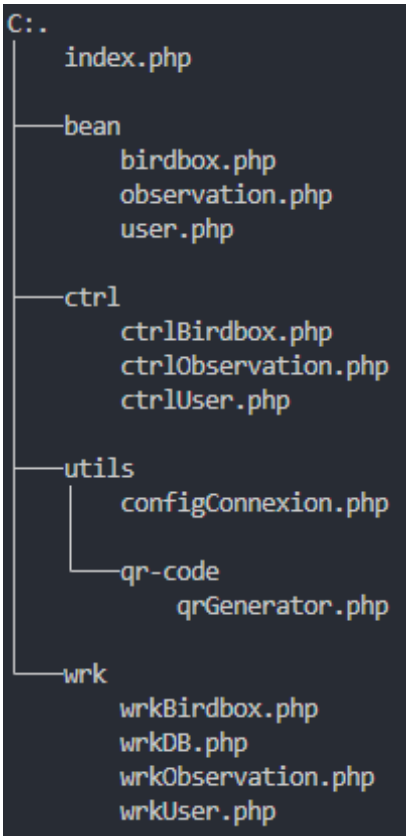


Figure 25 Structure du serveur

5.2 Problème(s) rencontré(s)

5.2.1 Chargement et affichage des images

5.2.1.1 Problème

La fonctionnalité d'upload d'image sur mon application ne fonctionnait pas parfaitement. Les images étaient parfois correctement affichées et parfois non. Ceci n'avait pas l'air d'être directement lié à leur taille.

5.2.1.2 Explication

Les images sont converties en Base64 avant d'être envoyées vers le serveur pour finalement être stockées dans un champ BLOB de la DB.

Le type BLOB simple n'est pas en mesure de stocker un fichier d'une taille excédant les 65 Ko. De plus, la conversion en base 64 augmente la taille du fichier de l'image d'environ 33%. Certaines images ne pouvaient donc pas être stockées sans que le liens soit raccourci car il n'était pas capable de stocker les trop longues chaines de caractères.

5.2.1.3 Solution(s)

Pour pallier ce problème, j'ai simplement dû modifier le type du champs de la DB qui stockait mes images. En utilisant LONGBLOB à la place, cela permet de stocker des données jusqu'à 4Go.

Une autre solution plus optimisée serait de stocker les images sur un serveur de fichier externe (voir axes d'amélioration pt. 7.1.8)

5.3 Spécifications détaillées

5.3.1 Sécurité des données (implémentation des fonctions cités au point 3.4)

- **Injection SQL**

  Mises en place des requêtes préparées pour la protection contre les injections SQL. C'est-à-dire que l'on définit d'abord la requête avec des paramètres vides et que l'on exécute ensuite la requêtes avec les paramètres définis.

  Ainsi l'injection sera traitée comme du texte simple

```php
$queryPrepared = $this->pdo->prepare($query);
$queryPrepared->execute($params);
```

- **Injection HTML et JavaScript**

  On utilise la fonction PHP « htmlspecialchars » pour contrer les injections HTML et JS dans notre base de données.

```php
$params = array("username" => htmlspecialchars($username));
```

- **Gestion des accès**

Une gestion des sessions du coté serveur est mis en place afin de vérifier que les actions, que le client tente d'effectuer, sont bien autorisées.

La gestion de session nous permet de mettre en place différente méthodes pour sécuriser notre application.

On bloque l'utilisation de certaines méthodes du serveur si l'utilisateur ne possède pas de session. Cela fonctionne également si des requêtes sont passées depuis un client différent que celui prévu initialement ou depuis une application comme Postman.

Une vérification de session est parfois également appelée depuis le client. Cela nous sert par exemple à rediriger un utilisateur s'il tente d'accéder à une page qui requiert d'être connecté et qu'il ne l'est pas.

Pour obtenir une session, il faut que l'utilisateur s'authentifie ou s'enregistre sur l'application.

Pour que la session soit supprimée, il y a deux possibilités. Lorsque l'utilisateur se déconnecte de l'application et qu'une requête vers le serveur est envoyée pour supprimer la session en cours. Soit lorsque la durée de la session a expiré.

- **Protection des données sensibles**

Lorsque l'utilisateur entre son mot de passe dans l'application, que ce soit pour s'authentifier ou s'enregistrer, le mot de passe n'est pas directement visible.

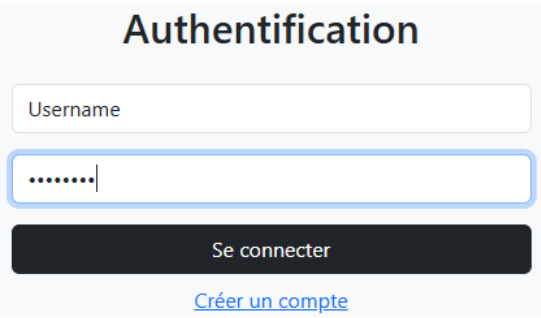


Figure 26 Captur d'écran protection des données de login

Les requêtes qui concernent l'utilisateur qui sont effectuées vers le serveur sont toutes des requêtes POST. De ce fait, les paramètres sont stockés dans la requête et ne sont pas directement visibles dans l'URL.

```
fetch(this.url, {
    method: "POST",
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    credentials: 'include',
    body: JSON.stringify({
        action: "login",
        username: username,
        password: password
    })
})
```

Finalement, les mots de passe des utilisateurs sont hachés avant d'être stockés dans la base de données. Ce qui empêche même les développeurs d'avoir accès aux données des utilisateurs.

```
$password = password_hash(htmlspecialchars($password), PASSWORD_DEFAULT);
```

	pk_user	username	password
▶	1	Admin	$2y$12$SXCYzZmyvZ3d.QY4c/03ne3hRYJwml3...

Figure 27 Capture d'écran base de données

5.3.2 Requêtes vers le serveur PHP

Voici un tableau récapitulatif des requêtes acceptées sur le serveur avec les paramètres nécessaires ainsi que le retour fourni.

Une session active est nécessaire pour effectuer les actions de création et modification sur les nichoirs, la suppression des observations ainsi que la déconnexion.

Retour de base :

success : true / false
message : « message en fonction de la requête ou erreur »

Serveur		https://bbt.emf-infopro-tpi.ch/BBT/SRV/index.php	
Méthode	Action	Paramètres	Retour
GET	getBirdboxs	action : getBirdboxs	Retour de base et liste des nichoirs
	getObservations	action : getObservations fk_birdbox : 1	Retour de base et liste des observations
	generateQR	action : generateQR fk_birdbox : 1 species : a	Fichier PNG

POST	login	{
		"action": "login",
		"username": "a",
		"password": "a"
		}
			Retour de base avec « pk » et « username » de l'utilisateur

	register	{
		"action": "register",
		"username": "a",
		"password": "a"
		}
			Retour de base avec « pk » et « username » de l'utilisateur

	checkSession	{
		"action": "checkSession"
		}
			Retour de base

	createBirdbox	{ "action": "createBirdbox", "designation": "a", "species": "a",	Retour de base et tous les attributs du nichoir créé

		"year": 1, "latitude": 1, "longitude": 1, "length": 1, "width": 1, "height": 1, "hole": "a", "picture": "" }	

	logout	{
		"action": "logout"
		}
			Retour de base

	createObservation	{
		"action": "createObservation",
		"autor": "a",
		"observation": "a",
		"fk": 1
		}
			Retour de base et tous les attributs de l'observation créé

PUT	updateBirdbox	{
		"action": "updateBirdbox",
		"pk_birdbox": 1,
		"designation": "a",
		"species": "a",
		"year": 1,
		"latitude": 1,
		"longitude": 1,
		"length": 1,
		"width": 1,
		"height": 1,
		"hole": "a",
		"picture": ""
		}
			Retour de base et tous les attributs du nichoir modifié

DELETE	deleteObservations	action : deleteObservations observations : [1,1,1]	Retour de base

Les champs de texte sont remplacés par des « a » ainsi que les valeurs numériques par des « 1 » pour simplifier la mise en page.

5.3.3 Requêtes vers la base de données MySQL

Voici la liste des requêtes SQL effectuées depuis le serveur vers la base de données.

5.3.3.1 Birdbox

Récupération de tous les nichoirs :

Cette requête sert à récupérer tous les nichoirs de la base de données pour les charger et les afficher ensuite sur la page principale du client.

```
SELECT * FROM t_birdbox
```

Récupération d'un nichoir précis :

Cette requête sert à récupérer les informations d'un nichoir en particulier pour afficher l'espèce sur la page d'envoi d'observation.

```
SELECT * FROM t_birdbox WHERE pk_birdbox = :pk_birdbox
```

Insertion d'un nouveau nichoir :

Cette requête crée un nouveau nichoir dans la base de données lorsqu'un nouveau nichoir est validé depuis la page principale du client.

```
INSERT INTO t_birdbox (designation, species, year, latitude, longitude, length,
width, height, hole, picture) VALUES (:designation, :species, :year, :latitude,
:longitude, :length, :width, :height, :hole, :picture)
```

Mise à jour d'un nichoir :

Cette requête modifie un nichoir dans la base de données lorsqu'un nichoir est modifié depuis la page principale du client.

```
UPDATE t_birdbox SET designation = :designation, species = :species, year = :year,
latitude = :latitude, longitude = :longitude, length = :length, width = :width,
height = :height, hole = :hole, picture = :picture WHERE pk_birdbox = :pk_birdbox
```

5.3.3.2 User

Récupération d'un utilisateur en particulier :

Cette requête est effectuée afin de vérifier si l'utilisateur qui tente de se connecter est enregistré dans la base de données.

```
SELECT * FROM t_user WHERE username = :username
```

Insertion d'un nouvel utilisateur :

Cette requête sert à créer un nouvel utilisateur dans la base de données lorsqu'un utilisateur s'enregistre sur le client.

```
INSERT INTO t_user (username, password) VALUES (:username, :password)
```

5.3.3.3 Observation

Insertion d'une nouvelle observation :

Création d'une nouvelle observation dans la base de données lorsqu'une observation est envoyée par un marcheur.

```
INSERT INTO t_observation (date, autor, observation, fk_birdbox) VALUES (:date,
:autor, :observation, :fk_birdbox)
```

Récupération de toutes les observation d'un nichoir précis :

Récupération des observations du nichoir pour les afficher sur la page de gestion des observations.

```
SELECT * FROM t_observation WHERE fk_birdbox = :fk_birdbox ORDER BY date DESC
```

Suppression d'une observation précise :

Cette requête est appelée dans une boucle du serveur. Elle sert à supprimer de la base de données toutes les observations que l'utilisateur a sélectionné pour la suppression sur la page de gestion des observations.

```
DELETE FROM t_observation WHERE pk_observation = :pk_observation
```

5.4 Design du système

Voici une présentation globale de l'application cliente. On peut voir les différentes pages avec et sans données. On voit également un aperçu des différents messages ou action qui sont déclenchées lors de certaines actions.

5.4.1 Authentification et Enregistrement

5.4.1.1 Affichage

Page d'authentification :

On cache le mot de passe entré par l'utilisateur.

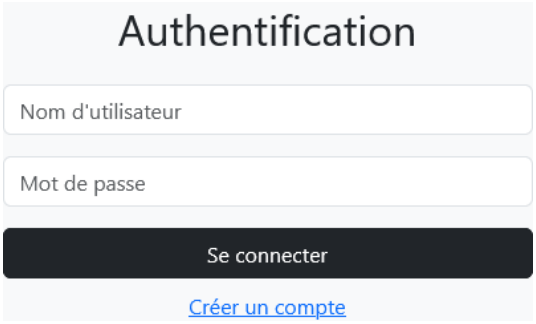


Figure 28 Authentification (sans données)

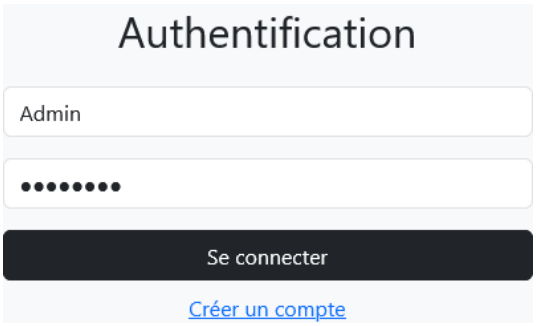


Figure 29 Authentification (avec données)

Page d'enregistrement :

On cache à nouveau le mot de passe entré par l'utilisateur.

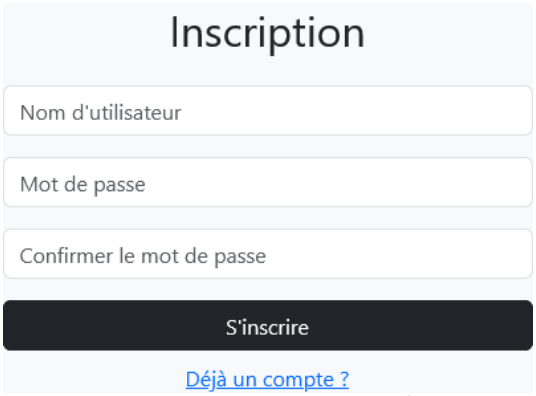


Figure 30 Inscription (sans données)

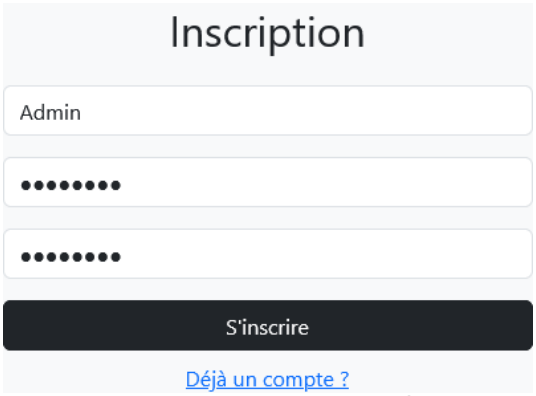


Figure 31 Inscription (avec données)

5.4.1.2 Messages

Action	Résultat possibles
Se connecter	Veuillez compléter tous les champs
	Identifiants incorrects
	Chargement de la page de gestion des nichoirs
S'inscrire	Veuillez compléter tous les champs
	Ce nom d'utilisateur est déjà pris
	Les mot de passe ne correspondent pas
	Chargement de la page de gestion des nichoirs

5.4.2 Page principale de gestion des nichoirs

5.4.2.1 Affichage

Page lors du chargement :

Les nichoirs enregistrés dans l'application sont affichés sur la carte. On affiche une image par défaut pour la photo du nichoir et le code QR.

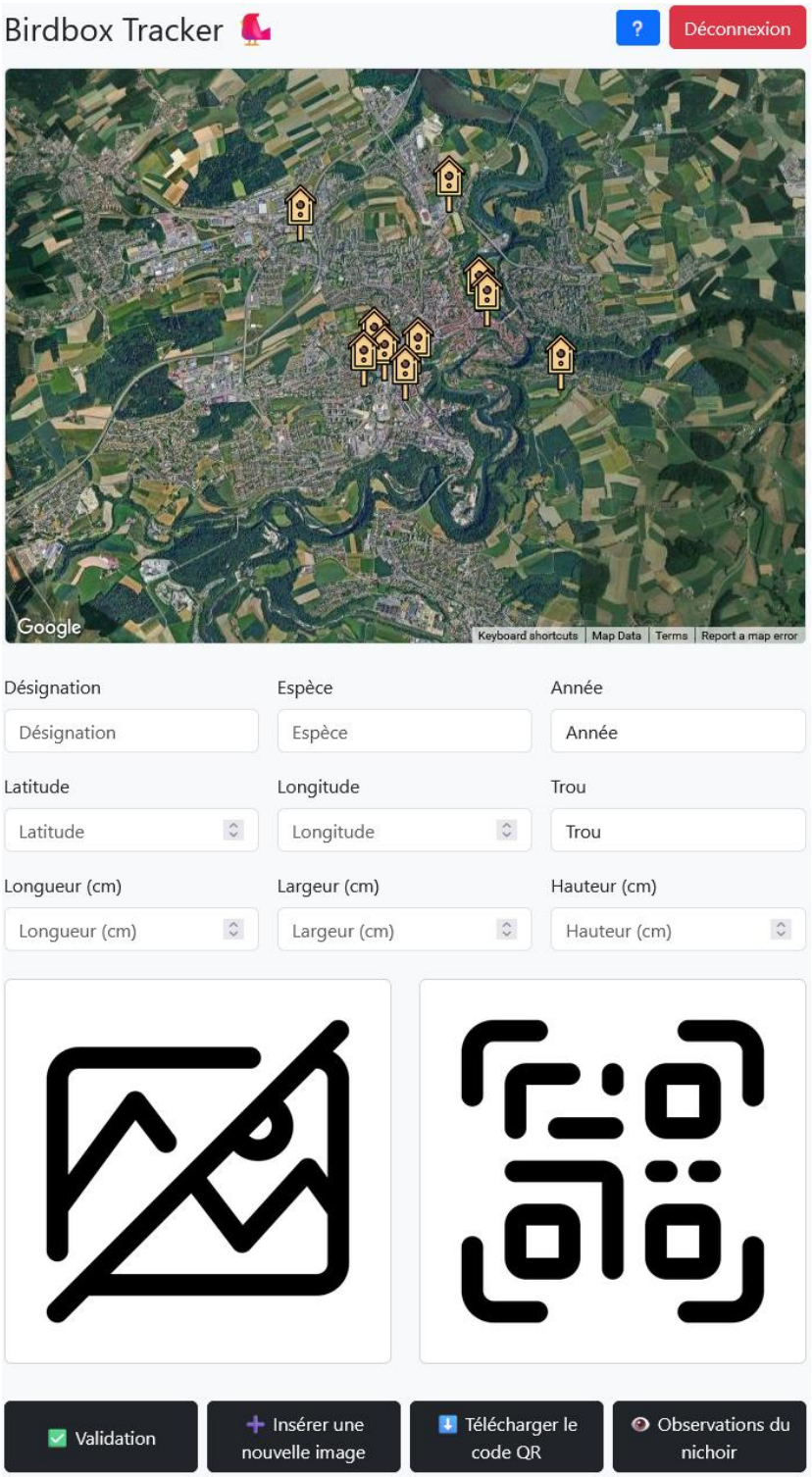


Figure 32 Page de gestion (sans données)

Page lors du chargement des informations d'un nichoir :

Les informations du nichoir sélectionné sont affichées dans les champs correspondants. L'image est affichée si elle existe, sinon on garde l'image par défaut. Le code QR du nichoir est généré et affiché également.

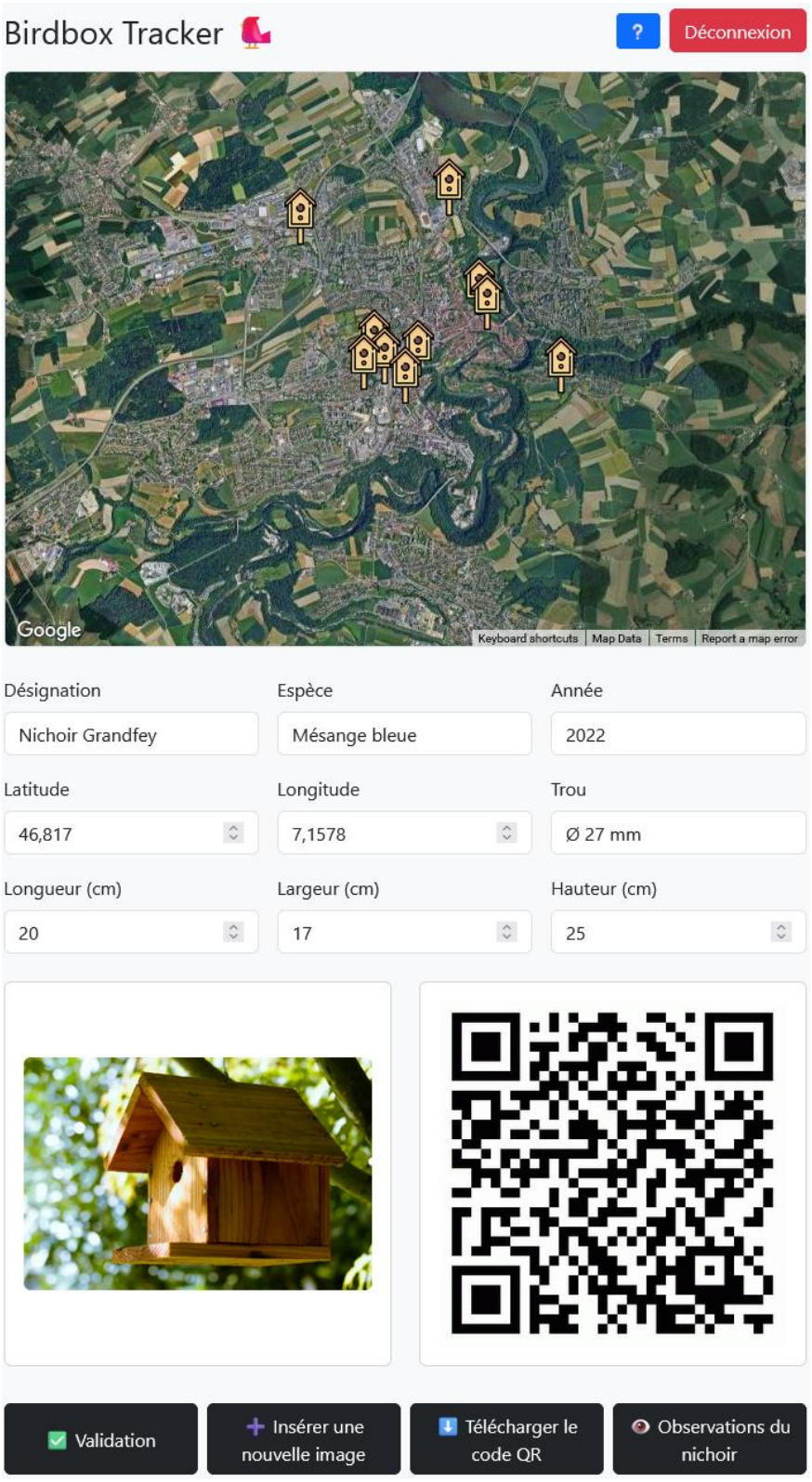


Figure 33 Page de gestion (avec données)

Page d'aide et instruction de l'application :

Cette page est accessible via le bouton « ? » en haut à gauche et guide l'utilisateur s'il ne comprend pas bien le fonctionnement de l'application.

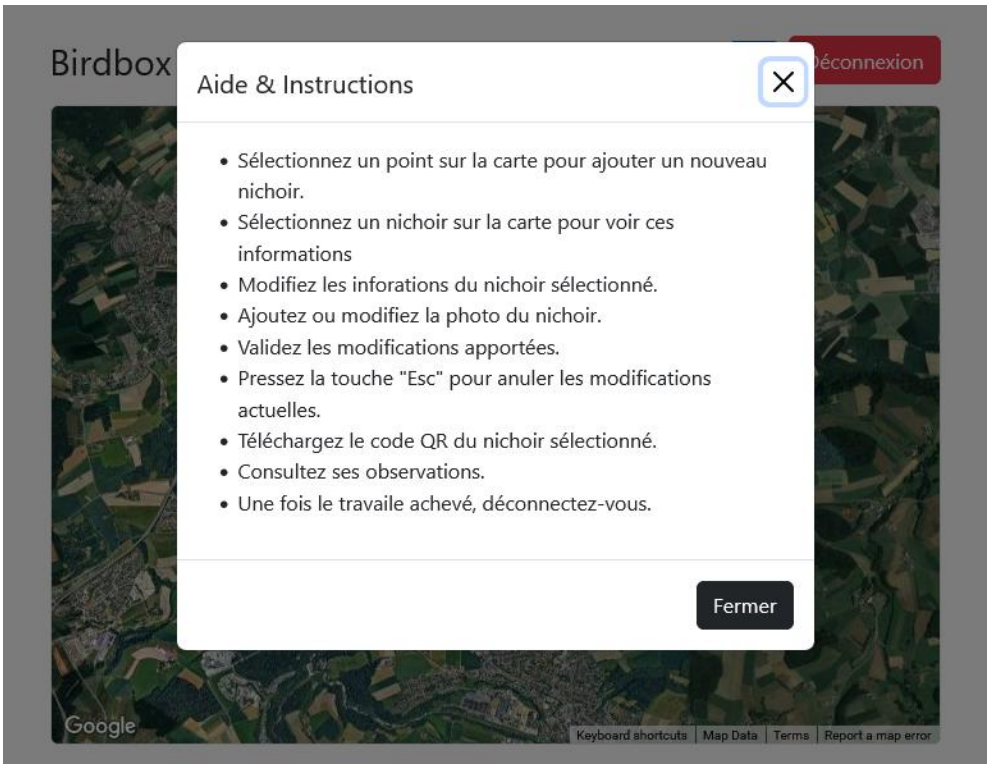


Figure 34 Affichage Aide & Instructions

5.4.2.2 Messages

Action	Message
Validation (nouveau nichoir)	Veuillez compléter tous les champs
	Nichoir ajouté
Validation (mise à jour d'un nichoir)	Veuillez compléter tous les champs
	Nichoir mis à jour
Insérer une image	L'image dépasse les 5 Mo autorisés
	Affichage de l'image
Télécharger le code QR	Sélectionnez d'abord un nichoir
	Téléchargement du code QR
Observations du nichoir	Sélectionnez d'abord un nichoir
	Chargement de la page de gestion des observations
Déconnexion	Chargement de la page d'authentification

5.4.3 Page de gestion des observation du nichoir

5.4.3.1 Affichage

Page lorsque le nichoir possède des observations :

Affichage des observations avec la possibilité de les sélectionner pour la suppression. Tri des observations par année pour simplifier l'affichage.

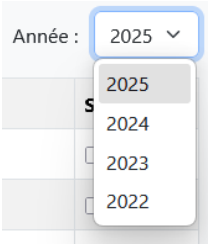


Figure 35 Liste déroulante années d'observations

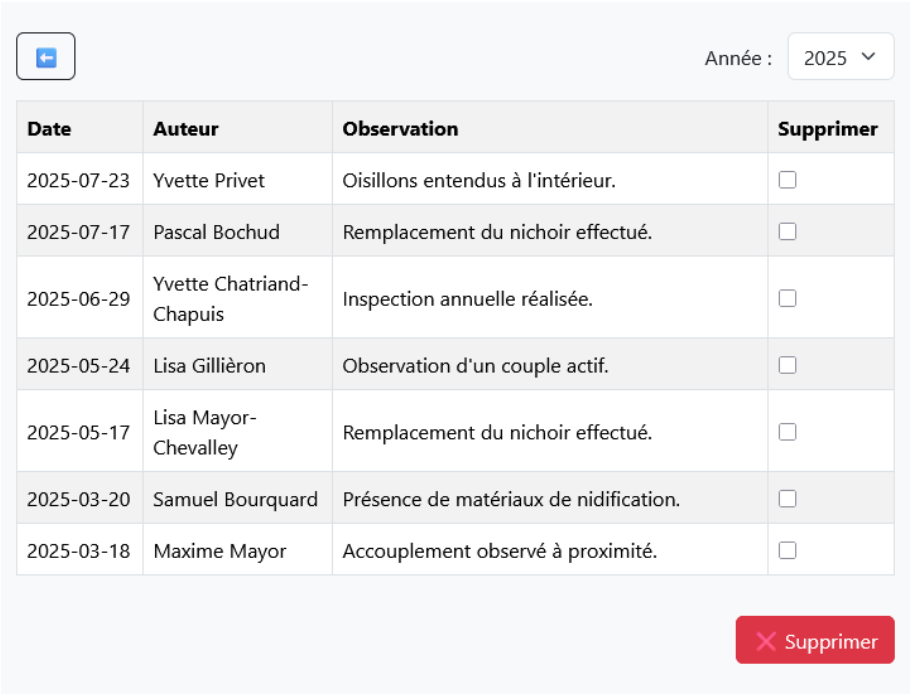


Figure 36 Gestion des observations (non-sélectionnées)

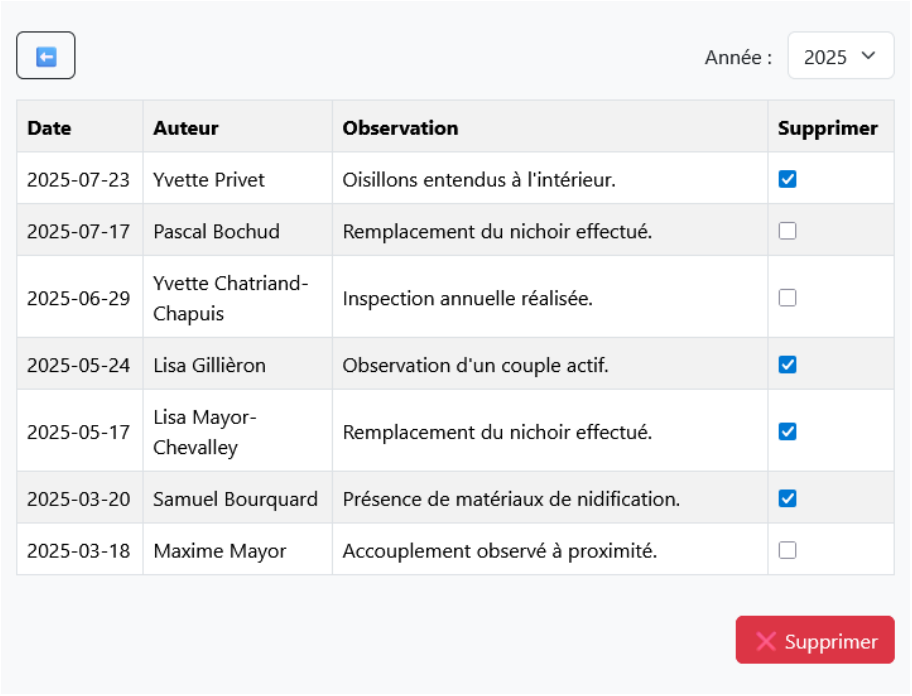


Figure 37 Gestion des observations (sélectionnées)

Page affichée lorsque le nichoir sélectionné ne possède aucune observation :

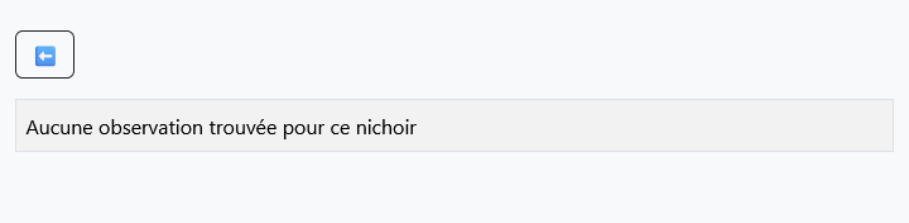


Figure 38 Gestion des observations (aucune observation)

5.4.3.2 Messages

Action	Message	
Sélectionner une année	Afficher les observations de l'année sélectionnée	
Supprimer	Aucune observations sélectionnée	
	Supprimer X observations ?	Annuler
		Confirmer
Retour	Chargement de la page de gestion des nichoirs	

5.4.4 Page d'envoi d'observations

5.4.4.1 Affichage

Page d'envoi d'observation chargée correctement :

On charge et on affiche le nom de l'espèce hôte du nichoir sur la page.

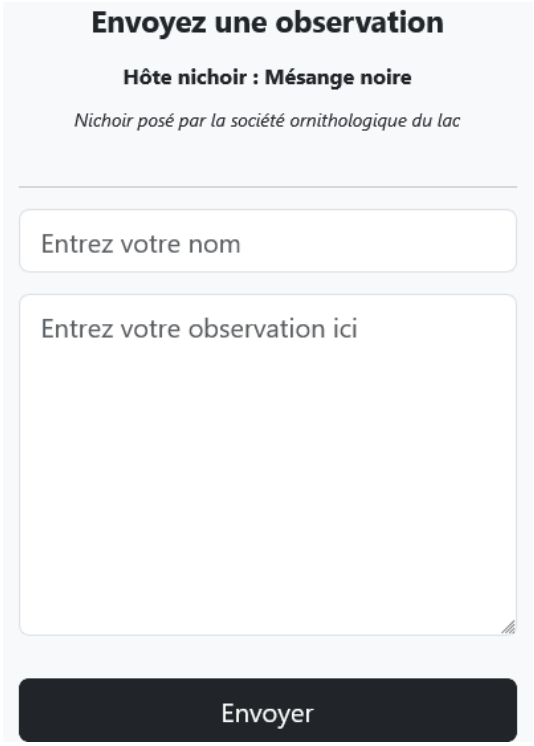


Figure 39 Envoi d'observations (sans données)

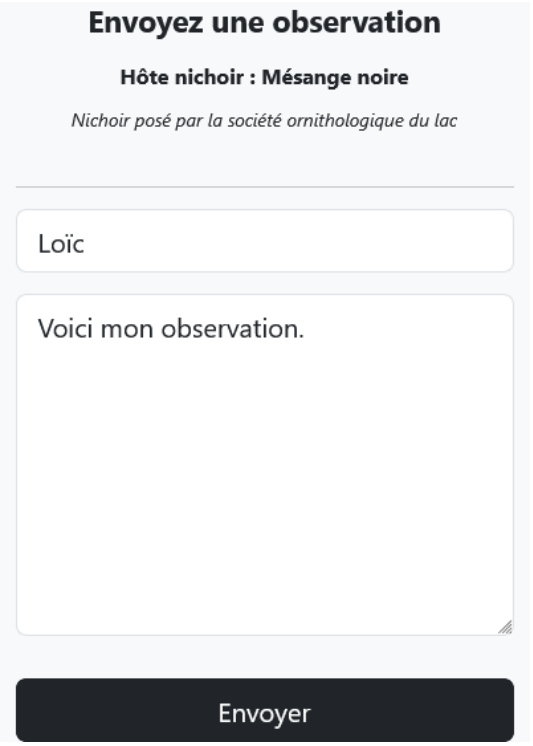


Figure 40 Envoi d'observations (avec données)

Page affichée lorsqu'une erreur survient pour récupérer l'espèce du nichoir (erreur dans l'url ou avec le chargement coté backend) :

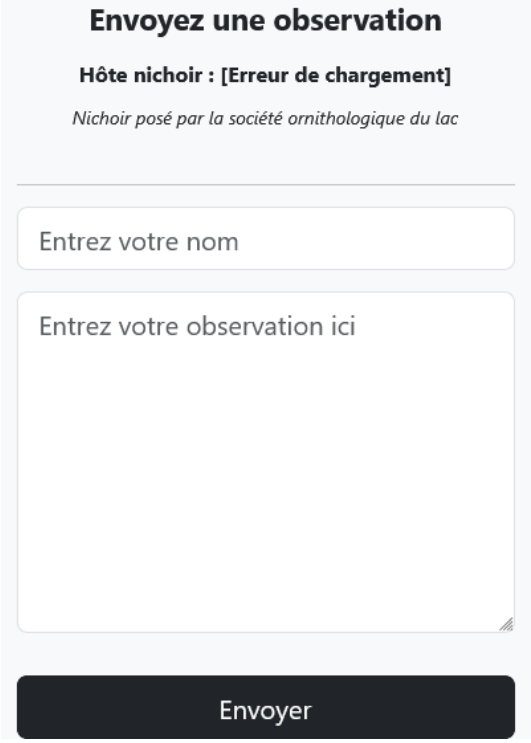


Figure 41 Envoi d'observations (erreur)

5.4.4.2 Messages

Action	Message
Envoyer	Erreur lors de l'envoi de l'observation
	Observation envoyée avec succès

5.5 Descente de code

Cette descente de code correspond à la réalisation du processus de création et de mise à jour d'un nichoir. Il reprend donc le diagramme de séquence d'interaction (création / mise à jour d'un nichoir) défini dans la conception au point 4.2.

5.5.1 Partie cliente

5.5.1.1 Class : CtrlIndex

Cette méthode est appelée lors de l'appui sur le bouton « Validation » de notre application. On commence par vérifier que tous les champs sont bien remplis, mis à part l'image qui peut ne pas être définie.

On vérifie ensuite la valeur de la variable « pkUpdate ». Elle va nous permettre de déterminer l'action à réaliser. Si elle est « null » on crée un nouveau nichoir, sinon on modifie celui qui possède l'identifiant stocké. La méthode correspondante est ensuite appelée avec les champs passé en paramètres.

On interprète finalement le résultat. Si le retour de succès est vrai, on affiche le message de confirmation, sinon on affiche le message d'erreur et dans tous les cas, la page est rechargée. Soit pour afficher les nouvelles informations, soit pour vider les champs.

```
async validation() {
    if (this.designation_field.value !== "" &&
        this.species_field.value !== "" &&
        this.year_field.value !== "" &&
        this.length_field.value !== "" &&
        this.width_field.value !== "" &&
        this.height_field.value !== "" &&
        this.hole_field.value !== "" &&
        this.latitude_field.value !== "" &&
        this.longitude_field.value) {
        var img = this.birdbox_img.src;
        if (this.birdbox_img.src === "https://bbt.emf-infopro-
tpi.ch/BBT/CLT/ressources/no_image.png") {
            img = null;
        }
        if (this.pkUpdate !== null) {
            var data = await this.wrk.updateBirdbox(
                this.pkUpdate,
                this.designation_field.value,
                this.species_field.value,
                this.year_field.value,
                this.latitude_field.value,
                this.longitude_field.value,
                this.length_field.value,
                this.width_field.value,
                this.height_field.value,
                this.hole_field.value,
                img
            );
            if (data.success) {
                alert(data.message);
                window.location.reload();
            } else if (!data.success) {
                alert(data.message);
                window.location.reload();
            } else {
                console.error(data);
            }
        } else {
            var data = await this.wrk.addBirdbox(
                this.designation_field.value,
                this.species_field.value,
                this.year_field.value,
                this.latitude_field.value,
                this.longitude_field.value,
                this.length_field.value,
                this.width_field.value,
                this.height_field.value,
                this.hole_field.value,
                img
            );
```

```
            if (data.success) {
                alert(data.message);
                window.location.reload();
            } else if (!data.success) {
                alert(data);
            } else {
                console.error(data);
            }
        }
    } else {
        alert("Veuillez remplir tous les champs");
    }
}
```

5.5.1.2 Class Wrk

Pour ajouter un nichoir :

Voici le requête « fetch » effectuée lorsqu'on appelle la fonction de mise à jour depuis le contrôleur. On effectue une requête avec la méthode PUT en passant (en JSON) dans le corps de la requête les paramètres reçus du client.

On retourne ensuite au client soit les données au format JSON, soit une erreur si elle survient durant l'exécution de la requête.

```
updateBirdbox(pk, designation, species, year, latitude, longitude, length, width,
height, hole, picture) {
    return fetch(this.url, {
        method: "PUT",
        headers: { 'Content-Type': 'application/json' },
        credentials: 'include',
        body: JSON.stringify({
            action: "updateBirdbox",
            pk_birdbox: pk,
            designation: designation,
            species: species,
            year: year,
            latitude: latitude,
            longitude: longitude,
            length: length,
            width: width,
            height: height,
            hole: hole,
            picture: picture
        })
    })
        .then(res => res.json())
        .catch(error => {
            console.error('[updateBirdbox]', error);
            throw error;
        });
}
```

Pour modifier un nichoir :

Voici le requête « fetch » effectuée lorsqu'on appelle la fonction d'ajout depuis le contrôleur. On effectue une requête avec la méthode POST en passant (en JSON) dans le corp de la requête les paramètres reçus du client.

On retourne ensuite au client soit les données au format JSON, soit une erreur si elle survient durant l'exécution de la requête.

```
addBirdbox(designation, species, year, latitude, longitude, length, width, height,
hole, picture) {
    return fetch(this.url, {
        method: "POST",
        headers: { 'Content-Type': 'application/json' },
        credentials: 'include',
        body: JSON.stringify({
            action: "createBirdbox",
            designation: designation,
            species: species,
            year: year,
            latitude: latitude,
            longitude: longitude,
            length: length,
            width: width,
            height: height,
            hole: hole,
            picture: picture
        })
    })
        .then(data => data.json())
        .catch(error => {
            console.error('[addBirdbox]', error);
            throw error;
        });
}
```

5.5.2 Partie serveur

5.5.2.1 File : index.php

Le fichier index.php récupère les requêtes et effectue plusieurs vérifications afin de déterminer s'il doit la traiter ou renvoyer un message d'erreur. Il commence par vérifier la méthode employée. Il vérifie ensuite, pour chacune des méthodes acceptées, l'action passée en paramètres. Et pour chaque action, il vérifie si toutes les données nécessaires sont bien présentes.

Pour ajouter un nichoir :

Pour l'action « createBirdbox », on commence donc, comme mentionnée ci-dessus, par vérifier que tous les paramètres nécessaires sont bien présents dans la requête.

Ensuite, on vérifie si la session est valide (si le client qui passe la requête possède une session valide) et si c'est le cas, on appelle la méthode du contrôleur. Sinon on renvoi une erreur.

```
case 'createBirdbox':
```

```
    if (isset($data['designation'], $data['species'], $data['year'], $data['lati-
tude'], $data['longitude'], $data['length'], $data['width'], $data['height'],
$data['hole'])) {
        if ($user->checkSessionCtrl()) {
            setResponseAndEcho($birdbox->createBirdboxCtrl($data['designation'],
$data['species'], $data['year'], $data['latitude'], $data['longitude'],
$data['length'], $data['width'], $data['height'], $data['hole'], $data['pic-
ture']));
        } else {
            http_response_code(401);
            echo json_encode([
                "success" => false,
                "message" => "Permission denied"
            ]);
        }
    }
```

Pour modifier un nichoir :

Le processus est très similaire au point précédent. Pour l'action « updateBirdbox », on commence, à nouveau, par vérifier que tous les paramètres nécessaires sont bien présents dans la requête.

Ensuite, on vérifie si la session est valide (si le client qui passe la requête possède une session valide) et si c'est le cas, on appelle la méthode du contrôleur. Sinon on renvoi une erreur.

```
if ($data['action'] === 'updateBirdbox') {
    if (
        isset($data['pk_birdbox'], $data['designation'], $data['species'],
$data['year'], $data['latitude'], $data['longitude'], $data['length'],
$data['width'], $data['height'], $data['hole'])
    ) {
        if ($user->checkSessionCtrl()) {
            setResponseAndEcho($birdbox->updateBirdboxCtrl($data['pk_birdbox'],
$data['designation'], $data['species'], $data['year'], $data['latitude'],
$data['longitude'], $data['length'], $data['width'], $data['height'],
$data['hole'], $data['picture'] ?? null));
        } else {
            http_response_code(401);
            echo json_encode([
                "success" => false,
                "message" => "Permission denied"
            ]);
        }
    } else {
        http_response_code(400);
        echo json_encode([
            "success" => false,
            "message" => "Missing datas"
        ]);
    }
}
```

5.5.2.2 Class : CtrlBirdbox

Pour ajouter un nichoir :

Le « Controller » transmet les données de l'index vers le « Worker ».

```
public function createBirdboxCtrl($latitude, $longitude, $designation, $species,
$year, $length, $width, $height, $hol, $picture)
{
    return $this->wrk->createBirdboxWrk( $latitude, $longitude, $designation, $spe-
cies, $year, $length, $width, $height, $hol, $picture);
}
```

Pour modifier un nichoir :

Le « Controller » transmet les données de l'index vers le « Worker ».

```
public function updateBirdboxCtrl($pk_birdbox, $latitude, $longitude, $designation,
$species, $year, $length, $width, $height, $hole, $picture)
{
    return $this->wrk->updateBirdboxWrk($pk_birdbox, $latitude, $longitude, $desig-
nation, $species, $year, $length, $width, $height, $hole, $picture);
}
```

5.5.2.3 Class : WrkBirdbox

Pour ajouter un nichoir :

Cette méthode commence par déclarer la requête « INSERT » qui va être effectuée sur la base de données pour la création de l'objet. On définit ensuite les paramètres de la requête en utilisant la fonction PHP « htmlspecialchars » qui permet de convertir les caractères spéciaux en entités HTML afin d'éviter les injection HTML et JavaScript.

On execute ensuite la requête grâce au « WrkDB » et on récupère le dernier identifiant inséré dans la table à nouveau à l'aide de « WrkDB ».

On retourne finalement tous les paramètres de l'entité crée.

```
public function createBirdboxWrk($designation, $species, $year, $latitude, $longi-
tude, $length, $width, $height, $hole, $picture)
{
    $query = "INSERT INTO t_birdbox (designation, species, year, latitude, longi-
tude, length, width, height, hole, picture) VALUES (:designation, :species, :year,
:latitude, :longitude, :length, :width, :height, :hole, :picture)";
    $params = array(
        ":designation" => htmlspecialchars($designation),
        ":species" => htmlspecialchars($species),
        ":year" => htmlspecialchars($year),
        ":latitude" => htmlspecialchars($latitude),
        ":longitude" => htmlspecialchars($longitude),
        ":length" => htmlspecialchars($length),
        ":width" => htmlspecialchars($width),
        ":height" => htmlspecialchars($height),
        ":hole" => htmlspecialchars($hole),
        ":picture" => htmlspecialchars($picture)
    );
    $response = $this->db->executeQuery($query, $params);
    $pk = $this->db->getLastId("t_birdbox");
    $response = json_encode([
```

```
        "success" => true,
        "message" => "Nichoir ajouté",
        "pk_birdbox" => $pk,
        "designation" => $designation,
        "spesies" => $species,
        "year" => $year,
        "latitude" => $latitude,
        "longitude" => $longitude,
        "length" => $length,
        "width" => $width,
        "height" => $height,
        "hole" => $hole,
        "picture" => $picture
    ]);
    return $response;
}
```

Pour modifier un nichoir :

Cette méthode commence par déclarer la requête « UPDATE » qui va être effectué sur la base de données pour la modification de l'objet. On définit ensuite les paramètres de la requête en utilisant la fonction PHP « htmlspecialchars » qui permet de convertir les caractères spéciaux en entités HTML afin d'éviter les injection HTML et JavaScript.

On execute ensuite la requête grâce au « WrkDB » et on retourne finalement tous les paramètres de l'entité modifiée.

```
public function updateBirdboxWrk($pk_birdbox, $designation, $species, $year, $lati-
tude, $longitude, $length, $width, $height, $hole, $picture)
{
    $query = "UPDATE t_birdbox SET designation = :designation, species = :species,
year = :year, latitude = :latitude, longitude = :longitude, length = :length, width
= :width, height = :height, hole = :hole, picture = :picture WHERE pk_birdbox =
:pk_birdbox";
    $params = array(
        ":pk_birdbox" => htmlspecialchars($pk_birdbox),
        ":designation" => htmlspecialchars($designation),
        ":species" => htmlspecialchars($species),
        ":year" => htmlspecialchars($year),
        ":latitude" => htmlspecialchars($latitude),
        ":longitude" => htmlspecialchars($longitude),
        ":length" => htmlspecialchars($length),
        ":width" => htmlspecialchars($width),
        ":height" => htmlspecialchars($height),
        ":hole" => htmlspecialchars($hole),
        ":picture" => htmlspecialchars($picture)
    );
    $response = $this->db->executeQuery($query, $params);
    $response = json_encode([
        "success" => true,
        "message" => "Nichoir mis à jour",
        "pk_birdbox" => $pk_birdbox,
        "designation" => $designation,
        "species" => $species,
```

```
        "year" => $year,
        "latitude" => $latitude,
        "longitude" => $longitude,
        "length" => $length,
        "width" => $width,
        "height" => $height,
        "hole" => $hole,
        "picture" => $picture
    ]);
    return $response;
}
```

5.5.2.4 Class WrkDB

Cette méthode permet d'exécuter les requêtes sur la base de données. On effectue une requête préparée, c'est-à-dire que l'on prépare d'abord la requête vide et on l'exécute ensuite avec les paramètres définis.

Cela nous permet de contrer les injections SQL.

Si une exception survient du côté de la base de données lors de l'exécution de la requête, on retourne une erreur avec l'exception.

```
public function executeQuery($query, $params)
{
    try {
        $queryPrepared = $this->pdo->prepare($query);
        $queryRes = $queryPrepared->execute($params);
        return $queryRes;
    } catch (PDOException $e) {
        print "Erreur !: " . $e->getMessage() . "<br/>";
        die();
    }
}
```

5.6 Déploiement de l'application

Déploiement du client ainsi que du serveur sur l'hébergement bbt.emf-infopro-tpi.ch à l'aide de l'outils de transfert de fichier WinSCP.

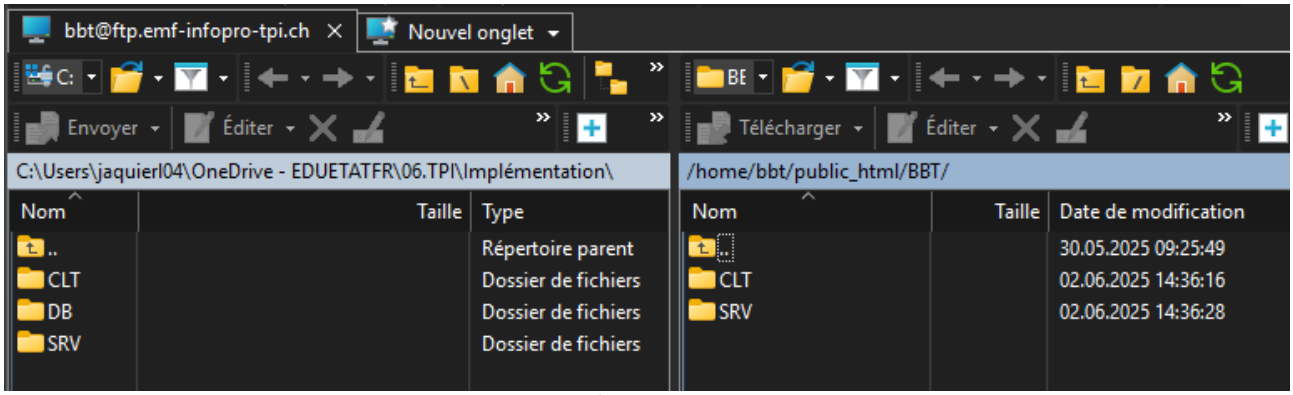


Figure 42 Déploiement via WinSCP

Création de la base de données ainsi que son utilisateur sur CPanel :

Figure 43Création de la DB

Figure 44 Création de l'utilisateur

Ajout de l'utilisateur a la base de données :

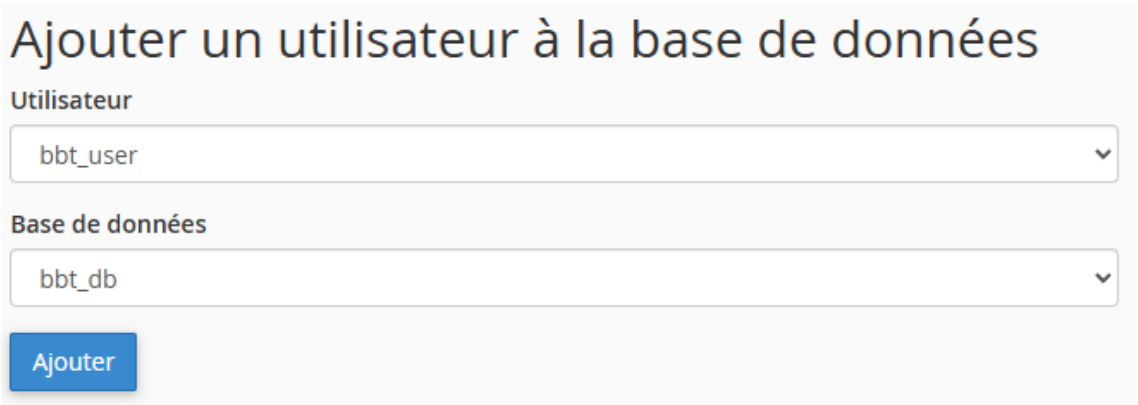


Figure 45 Ajout de l'utilisateur à la DB

Résultat finale de la création de la DB et de son utilisateur :

Base de données	Taille	Utilisateurs avec privilèges
bbt_db	1,58 MB	bbt_user

Figure 46 Résultat final création DB et utilisateur

Modification des données d'accès à la DB dans le code du serveur.

configConnexion.php :

```php
<?php
define('DB_TYPE', 'mysql');
define('DB_HOST', 'bbt.emf-infopro-tpi.ch');
define('DB_NAME', 'bbt_db');
define('DB_USER', 'bbt_user');
define('DB_PASS', '...');
```

5.7 Conclusion d'implémentation

Dans la globalité, l'implémentation respecte le cahier des charges ainsi que les concepts établis.

Le client diffère légèrement de la conception. L'utilisation des beans coté client étais spécifié sur les schémas de conception, mais cela n'a finalement pas été implémenté.

J'ai fait ce choix car je trouvais plus simple de récupérer uniquement les données utiles à notre application lors d'exécutions de requêtes et dans la plupart des cas, on affiche simplement un message de succès ou d'erreur.

6 Test

6.1 Accès à l'application

Pour vous connecter sur l'application afin de voir le résultat du travail réalisé ou pour effectuer des tests, vous pouvez utiliser les informations suivantes :

Accès	bbt.emf-infopro-tpi.ch
Nom d'utilisateur	Test
Mot de passe	Pa$$w0rd

6.2 Procédure de test

Voici le résultat de la réalisation des tests définis plus tôt au point 4.5. Ces tests nous permettent de vérifier notre application dans son ensemble (du coté client car c'est un protocole blackbox).

Environnement de test :

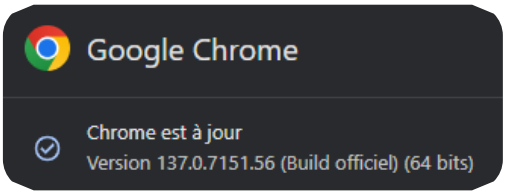


Figure 47 version du navigateur

6.3 Protocole de test

Nr.	Objet testé	Description du test	Paramètres	Résultat attendu	Résultat obtenu	Test réussi	Visa
1.01	Page d'inscription	Inscription avec les champs vides	Nom d'utilisateur : "" Mot de passe : "" Confirmation du mot de passe : ""	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
1.02	Page d'inscription	Inscription avec un utilisateur existant	Nom d'utilisateur : "Admin" Mot de passe : "Emf12345" Confirmation du mot de passe : "Emf12345"	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
1.03	Page d'inscription	Inscription avec un nouvel utilisateur mais deux mots de passe différents	Nom d'utilisateur : "Admin2" Mot de passe : "Emf12345" Confirmation du mot de passe : "Emf54321"	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
1.04	Page d'inscription	Inscription avec un nouvel utilisateur mais un mot de passe de moins de 8 caractères	Nom d'utilisateur : "Admin2" Mot de passe : "Emf123" Confirmation du mot de passe : "Emf123"	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
1.05	Page d'inscription	Inscription avec un nouvel utilisateur et des champs corrects	Nom d'utilisateur : "Admin2" Mot de passe : "Emf12345" Confirmation du mot de passe : "Emf12345"	Affichage de la page de gestion des nichoirs	Affichage de la page de gestion des nichoirs	Oui	Jaquier Loïc le 30 mai 2025 15H00
2.01	Page d'authentification	Connexion avec les champs vides	Nom d'utilisateur : "" Mot de passe : ""	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
2.02	Page d'authentification	Connexion avec un utilisateur inexistant	Nom d'utilisateur : "utilisateur1" Mot de passe : "mot de passe 1"	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
2.03	Page d'authentification	Connexion avec un utilisateur existant mais mauvais mot de passe	Nom d'utilisateur : "Admin" Mot de passe : "Emf12345"	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H00
2.04	Page d'authentification	Connexion avec des identifiants corrects	Nom d'utilisateur : "Admin" Mot de passe : "Pa$$w0rd"	Affichage de la page de gestion des nichoirs	Affichage de la page de gestion des nichoirs	Oui	Jaquier Loïc le 30 mai 2025 15H00
3.01	Page de gestion des nichoirs	Chargement de la page lorsque l'utilisateur se connecte	URL : "https://bbt.emf-infopro-tpi.ch/BBT/CLT/view/index.html"	Chargement de la carte Google Maps avec les points des nichoirs enregistrés	Chargement de la carte Google Maps avec les points des nichoirs enregistrés	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.02	Page de gestion des nichoirs	Déconnexion de l'application	-	Retour sur la page d'authentification	Retour sur la page d'authentification	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.03	Page de gestion des nichoirs	Sélection d'un point d'un nichoir sur la Map	-	Affichage des informations dans les champs de la page	Affichage des informations dans les champs de la page	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.04	Page de gestion des nichoirs	Validation avec champs incomplets après avoir créé un nouveau point sur la carte (nouveau nichoir)	Désignation : "" Espèce : "" Année : "" Latitude : "46.8" Longitude : "7.1" Trou : "" Longueur : "" Largeur : "" Hauteur : "" Image : Pas d'image	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.05	Page de gestion des nichoirs	Validation avec champs incomplets après avoir cliqué sur un nichoir sur la carte (mise à jour du nichoir)	Désignation : "Nichoir Bois de Moncor" Espèce : "Rougequeue noir" Année : "2023" Latitude : ""	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H15

			Longitude : "" Trou : "Ø 32 mm" Longueur : "22" Largeur : "18" Hauteur : "28" Image : Pas d'image				
3.06	Page de gestion des nichoirs	Validation avec champs complets après avoir créé un nouveau point sur la carte (nouveau nichoir)	Désignation : "" Espèce : "Rougequeue noir" Année : "2023" Latitude : "46.8" Longitude : "7.1" Trou : "Ø 32 mm" Longueur : "22" Largeur : "18" Hauteur : "28" Image : Pas d'image	Message de confirmation	Message de confirmation	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.07	Page de gestion des nichoirs	Validation avec champs complets après avoir cliqué sur un point sur la carte (mise à jour du nichoir)	Désignation : "" Espèce : "" Année : "" Latitude : "" Longitude : "" Trou : "" Longueur : "" Largeur : "" Hauteur : "" Image : Pas d'image	Message de confirmation	Message de confirmation	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.08	Page de gestion des nichoirs	Déplacement d'un nichoir sur la carte	-	Demande de confirmation du nouvel emplacement	Demande de confirmation du nouvel emplacement	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.09	Page de gestion des nichoirs	Insertion d'une image		Affichage de l'image	Affichage de l'image	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.10	Page de gestion des nichoirs	Télécharger le code QR sans avoir sélectionné de nichoir	Nichoir sélectionné : aucun	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.11	Page de gestion des nichoirs	Télécharger le code QR après avoir sélectionné un nichoir	Nichoir sélectionné (désignation) : "Nichoir Bois de Moncor"	Téléchargement du code QR	Affichage du code QR sur la page	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.12	Page de gestion des nichoirs	Accès aux observations sans avoir sélectionné de nichoir	Nichoir sélectionné : aucun	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H15
3.13	Page de gestion des nichoirs	Accès aux observations après avoir sélectionné un nichoir	Nichoir sélectionné (désignation) : "Nichoir Bois de Moncor"	Affichage de la page des observations du nichoir	Affichage de la page des observations du nichoir	Oui	Jaquier Loïc le 30 mai 2025 15H15
4.01	Gestion des observations	Valider la suppression sans avoir sélectionnée d'observations (pour nichoir sélectionnée au test 3.13)	-	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H45
4.02	Gestion des observations	Valider la suppression après avoir sélectionné des observations(pour nichoir sélectionnée au test 3.13)	Contenu : "2025-03-19 / Clara Uldry / Nichoir nettoyé récemment."	Message de confirmation	Message de confirmation	Oui	Jaquier Loïc le 30 mai 2025 15H45

5.01	Code QR	Scan du code QR d'un nichoir	-	Chargement de la page d'envoi d'observation avec les informations concernant l'espèce du nichoir	Chargement de la page d'envoi d'observation avec les informations concernant l'espèce du nichoir	Oui	Jaquier Loïc le 30 mai 2025 15H45
6.01	Ajout d'observation	Validation de la nouvelle observation avec des champs vides	Nom : "" Observation : ""	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H45
6.02	Ajout d'observation	Validation de la nouvelle observation avec des champs corrects	Nom : "Loïc" Observation : "Voici mon observation"	Message de confirmation	Message de confirmation	Oui	Jaquier Loïc le 30 mai 2025 15H45
7.01	Gestion des accès	Chargement de la page de gestion des nichoirs sans être connecté	URL : "https://bbt.emf-infopro-tpi.ch/BBT/CLT/view/index.html"	Redirection sur la page d'authentification	Redirection sur la page d'authentification	Oui	Jaquier Loïc le 30 mai 2025 15H45
7.02	Gestion des accès	Chargement de la page de gestion des observations sans être connecté	URL : "https://bbt.emf-infopro-tpi.ch/BBT/CLT/view/observations.html"	Redirection sur la page d'authentification	Redirection sur la page d'authentification	Oui	Jaquier Loïc le 30 mai 2025 15H45
8.01	Sécurité	Injection HTML/JS (page de gestion des nichoirs) Validation de la mise à jour du nichoirs avec une injection de script JS dans des balises HTML dans un des champs du nichoir	Désignation : " <script>alert("Test d'injection")</script> " Espèce : "Rougequeue noir" Année : "2023" Latitude : "" Longitude : "" Trou : "Ø 32 mm" Longueur : "22" Largeur : "18" Hauteur : "28" Image : Pas d'image	Affichage de l'injection sans modification du comportement de l'application	Affichage de l'injection sans modification du comportement de l'application	Oui	Jaquier Loïc le 30 mai 2025 15H45
8.02	Sécurité	Injection SQL (page de login) Tentative de connexion avec une injection SQL a la place du nom d'utilisateur	Nom d'utilisateur : "; DROP TABLE t_user; --" Mot de passe : "Test injection"	Message d'erreur	Message d'erreur	Oui	Jaquier Loïc le 30 mai 2025 15H45

6.4 Conclusion de la réalisation des tests

Tous les tests définis lors de la conception on bien pu être réalisés et documentés. On peut constater que le résultat obtenu pour chacun des tests correspond bien à celui qui étais attendu. Cela démontre que l'implémentation à bien pu être réalisée selon les concepts établis (en termes d'utilisation).

6.5 Signature du protocole de test

Date	Nom Prénom	Signature
Le 30 mai 2025	Jaquier Loïc	

7 Conclusion

7.1 Axes d'améliorations possibles

Voici une liste de propositions de fonctionnalités qui n'étaient pas imposées par le cahier des charges mais qui pourraient être des bonnes perspectives d'amélioration de l'application.

7.1.1 Suppression des comptes utilisateur

Un utilisateur peut créer un compte sur l'application mais il devrait également avoir la possibilité de le supprimer. Cela serait intéressant, d'autant plus si l'application est utilisée dans une association d'ornithologie. Si un membre quitte l'association, il doit pouvoir supprimer son compte d'accès.

7.1.2 Suppression des nichoirs

Les nichoirs peuvent être ajoutés ou modifiés sur l'application de gestion mais il n'est pas spécifié dans le cahier des charges qu'ils doivent pouvoir être supprimés.

Dans la pratique, les nichoirs sont parfois installés ou modifiés mais il se peut également qu'ils soient retirés de leur emplacement pour diverses raisons (inoccupés, trop vieux, …).

Cette fonctionnalité permettrait une gestion plus complète des nichoirs et c'est pourquoi elle serait l'une des plus importantes à implémenter, pour le développement futur de notre application.

7.1.3 Gestion de plusieurs nichoirs au même emplacement

Notre application actuelle ne permet pas de gérer correctement l'affichage de plusieurs nichoirs sur une même coordonnée géographique.

Ajouter sur le côté de la carte une liste des nichoirs enregistrés nous permettrait de contourner ce problème en nous permettant de visualiser tous les nichoirs peu importe leur emplacement.

Cela permettrait également d'afficher et visualiser plus simplement les nichoirs.

7.1.4 Affichage dynamique de la carte

L'affichage actuel de la carte est simplement défini sur les coordonnées de la ville de Fribourg avec un zoom défini.

Cet affichage pourrait être modifié afin d'afficher la carte en fonction des nichoirs enregistrés. Il faudrait calculer le centre géographique par rapport à tous les nichoirs et définir le niveau de zoom afin de bien tous les voir.

Ceci nous permettrait de proposer une meilleure expérience pour l'utilisateur.

7.1.5 Liste des espèces d'oiseaux

Il existe une multitude d'espèces d'oiseaux. Sur notre application de gestion de nichoir, il serait plus pratique de pouvoir sélectionner une espèce directement dans une liste déroulante lors de la création ou de la modification d'un nichoir plutôt que de devoir l'entrer à la main.

La plus grande complication de cette fonctionnalité est qu'il faut que la liste reste constamment à jour. De ce fait, nous ne pouvons pas simplement entrer une liste de noms d'espèces pour la sélection.

En revanche, nous avons d'autres possibilités. Nous pouvons faire une requête pour charger les espèces depuis une API externe (si elle est régulièrement mise à jour). Ou ajouter une table à notre base de données qui serait accessible par les administrateurs pour être mise à jour.

7.1.6 Envoi d'image avec les observations

Une autre fonctionnalité intéressante ajouter à notre application serait de pouvoir joindre une image lors de l'envoi d'une observation du nichoir.

Ce serait très utile si un utilisateur signale par exemple un dommage sur un nichoir, l'administrateur saura directement quoi faire sans même avoir besoin de se rendre sur place.

Ou encore si un utilisateur constate que l'espèce d'oiseaux du nichoir ne correspond pas aux informations de l'application. Il peut envoyer une image à l'appuis de son commentaire pour que le nichoir soit mis à jour.

7.1.7 Adapter l'application de gestion pour les appareils mobiles

Adapter l'application de gestion des nichoirs pour une utilisation sur mobile serait très intéressant. Les administrateurs sont parfois en déplacement pour la maintenance ou la pose de nichoir et n'ont donc pas forcément accès à un ordinateur. Il peut donc être intéressant de proposer une interface adaptée aux smartphones afin qu'ils puissent accéder à la gestion des nichoirs à tout moment.

7.1.8 Stockage des images sur un serveur externe

Le stockage des images se fait actuellement directement dans ma base de données (encodé en base 64). Il serait préférable d'opter pour une solution de stockage sur un serveur de fichier externe en ne conservant plus que le liens de l'image en texte dans la base de données.

Ceci réduirait premièrement l'espace utilisé pour stocker les images car l'encodage en base 64 augment la taille des images. Et cette méthode serait plus adaptée car un serveur de fichier est plus adapté pour la gestion des images qu'une base de données.

7.2 Auto-évaluation

J'ai appliqué une démarche stricte tout au long du projet, en utilisant des méthodes de travail et d'organisation adaptées. Celles-ci m'ont permis de respecter les exigences du cahier des charges ainsi que les critères définis dans le cadre du TPI.

De plus, les méthodes de travail mises en place m'ont permis de gérer efficacement les problèmes et les imprévus rencontrés sans perdre trop de temps et en limitant l'impact sur l'avancée du projet.

J'estime que le résultat final correspond aux critères fixés et présente un bon niveau de qualité.

Ce projet m'a permis d'utiliser mes compétences acquises au cours de ma formation à l'EMF mais également de développer mes connaissances en découvrant de nouvelles technologies telles que l'API Google Maps ou la génération de codes QR.

7.3 Conclusion personnelle

J'appréhendais la réalisation de ce projet dû au fait qu'il aura un impact important sur ma formation professionnel. J'ai tout de même éprouvé beaucoup de plaisir à réaliser un projet comme celui-ci car, quoi que plus complexe, il était similaire à un projet réalisé durant ma troisième année de formation sur lequel j'avais déjà eu beaucoup de plaisir à travailler.

Ma seconde crainte a été de ne pas réussir à correctement utiliser les méthodes que nous avions apprises à l'école. Car, en effet, je n'ai pas eu beaucoup d'exercices pratiques techniques à réaliser durant mes stages effectués en quatrième année.

Malgré mes appréhensions, je suis fière du travail que j'ai fourni j'ai eu un bon ressenti durant la globalité de la réalisation. Je pense avoir été capable de démontrer mes capacités à réussir ce projet.

8 Bibliographie : liste des sources et références

8.1 Liste des images

8.2 Sources d'informations

Source	Utilisation
Module EMF N°151	• Structure de l'application • Architecture Client-serveur • Requêtes HTTP • PHP
Module EMF N°306	Organisation de projet
Module EMF N°140	Base de données SQL
Standards d'entreprise de l'EMF	• Schémas UML • Structure MVC • Schémas et structure DB
Bootstrap Documentation	Implémentation du style
Google Maps Platform Documentation	Informations sur l'utilisation de l'API Google Maps
Chat GPT	• Corrections orthographiques et grammaticale • Structurations d'idées • Aide au débogage
Github Copilot	• Auto-complète de code (gain de temps) • Aide au débogage

9 Glossaire

Terme	Signification
API	Serveur mettant à disposition des services ou des données via des requêtes
Application web	Programme accessible via un navigateur web
Backend	Partie serveur qui traite la logique métier et les requêtes
BLOB	Type de champ stockant des données binaires dans une DB
chillerlan/php-qrcode	Bibliothèque PHP permettant de générer des QR codes
Clé étrangère (FK)	Clé de référence a une PK
Clé primaire (PK)	Identifiant unique d'une entrée dans une table d'une DB
Client-serveur	Architecture où un client envoie des requêtes à un serveur qui les traite
CPanel	Interface web de gestion d'hébergement
CRUD	Opérations de base d'un système de gestion de données : Create, Read, Update, Delete
Diagramme d'activité	Représentation des étapes d'un processus
Diagramme de cas d'utilisation	Représentation des actions possibles pour chaque acteur du système
Diagramme de classe	Représentation des classes et de leurs relations dans un système
Diagramme de contexte	Vue d'ensemble des interactions entre le système et ses acteurs externes
Diagramme de séquence	Représentation chronologique des échanges entre les différents acteurs
Frontend	Partie visible par l'utilisateur
Google Maps API	Interface permettant d'afficher des cartes Google dans une application
Hachage de mot de passe	Transformation irréversible d'un mot de passe pour sécuriser son stockage
HTML	Langage de balisage utilisé pour structurer les pages web
htmlspecialchars()	Fonction PHP qui empêche l'exécution de code HTML/JS malveillant
HTTPS	Protocole sécurisé de communication sur le web
Injection HTML / JavaScript	Insertion de code non désiré dans une page web pour manipuler l'affichage

Injection SQL	Technique malveillante qui insère du code SQL dans une requête
JavaScript (JS)	Langage de programmation utilisé côté client pour traiter la logique de l'application
LONGTEXT	Type de champ texte pouvant contenir de grandes quantités de texte dans une DB
Maquette	Prototype visuel d'une interface utilisateur
Modèle MVC	Modèle de conception séparant les données (Modèle), l'interface (Vue) et la logique (Contrôleur)
MySQL	Système de gestion de base de données
MySQL Workbench	Outil de gestion des bases de données MySQL
PHP	Langage côté serveur utilisé pour traiter les requêtes web
QR Code	Image contenant un lien ou des informations accessibles avec la caméra d'un smartphone
Requête GET / POST / PUT / DELETE	Types de requêtes HTTP utilisées pour lire, créer, modifier ou supprimer des données
Requête HTTP	Message envoyé par un client à un serveur pour échanger des données
Requête préparée	Requête SQL sécurisée contre les injections
Schéma ER	Schéma de représentation des entités et de leurs relations dans une DB
Schéma relationnel	Représentation des tables et relations d'une base de données
Session	Méthode permettant de garder l'utilisateur connecté
Test blackbox	Test sans connaissance du code interne
Test fonctionnel	Test des fonctionnalités d'une application du point de vue utilisateur
Visual Studio Code (VS Code)	Éditeur de code
WinSCP	Logiciel pour transférer des fichiers en SSH/SFTP

10 Signatures

Je soussigné déclare que les informations contenues dans ce rapport de travail pratique individuel rendu ce jour le 02.06.2025 dans le cadre de la procédure de qualification de mon CFC d'informaticien, ne sont pas plagiées. Toutes les informations de sources extérieures ainsi que les informations fournies par des tiers durant le déroulement du travail sont consignées.

Date	Nom Prénom	Signature
02.06.2025	Jaquier Loïc	

11 Annexes

11.1 Annexes externes

- **Code source du client, du serveur et de la DB :**
  Jaquier_Code_source_2025 (dossier)
- **Schémas d'analyse et de conception UML :**
  Jaquier_UML_2025.qea
- **Planning organisationnel :**
  Jaquier_Planning_2025.xlsx
- **Journal de travail :**
  Jaquier_Journal_2025.xlsx
- **Schémas relationnel Notation de Chen :**
  Jaquier_Schema_ER_2025.vsdx
- **Schémas relationnel Notation patte d'oie :**
  Jaquier_Modele_Workbench_2025.mwb

11.2 Tests technologiques (pré-TPI)

11.2.1 Bootstrap

Bootstrap est un Framework frontend open source qui permet facilement de mettre en style un site web en utilisant des composants prédéfinis ainsi que des classes CSS.

Installation :

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/boot-
strap.min.css" rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/boot-
strap.bundle.min.js"></script>
```

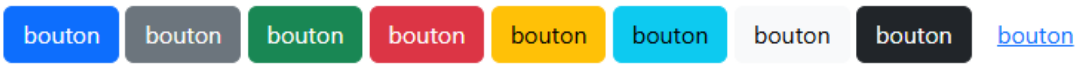
Quelques composants :

- **Grille :**

Colonne 1 Colonne 2

```
<div class="container">
    <div class="row">
        <div class="col-6 text-center">Colonne 1</div>
        <div class="col-6 text-center">Colonne 2</div>
    </div>
</div>
```

- **Bouton :**




```
<button class="btn btn-primary">bouton</button>
<button class="btn btn-secondary">bouton</button>
<button class="btn btn-success">bouton</button>
<button class="btn btn-danger">bouton</button>
<button class="btn btn-warning">bouton</button>
<button class="btn btn-info">bouton</button>
<button class="btn btn-light">bouton</button>
<button class="btn btn-dark">bouton</button>
<button class="btn btn-link">bouton</button>
```

- **Carte :**

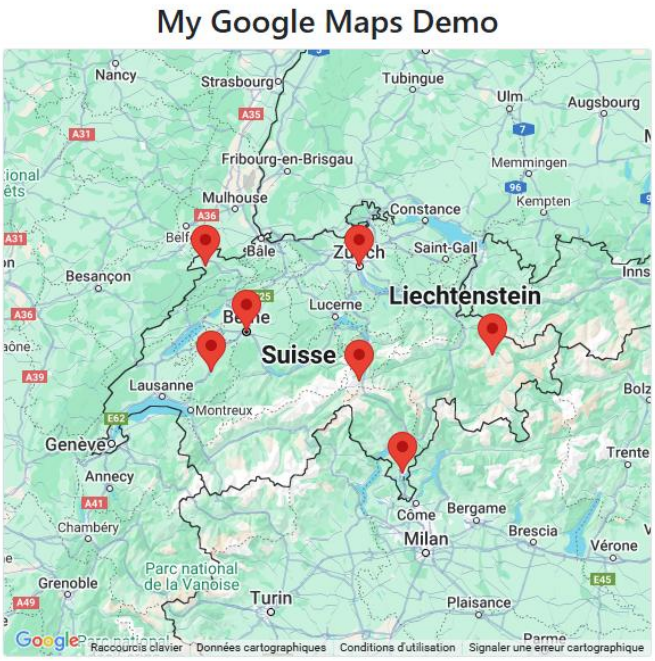

```
<div class="card text-center mx-auto" style="width: 18rem;">
    <div class="card-body">
        <h5 class="card-title">Titre</h5>
        <p class="card-text">Description rapide.</p>
    </div>
</div>
```

Documentation officielle ou l'on peut retrouver tous les composants disponibles :

Bootstrap · The most popular HTML, CSS, and JS library in the world.

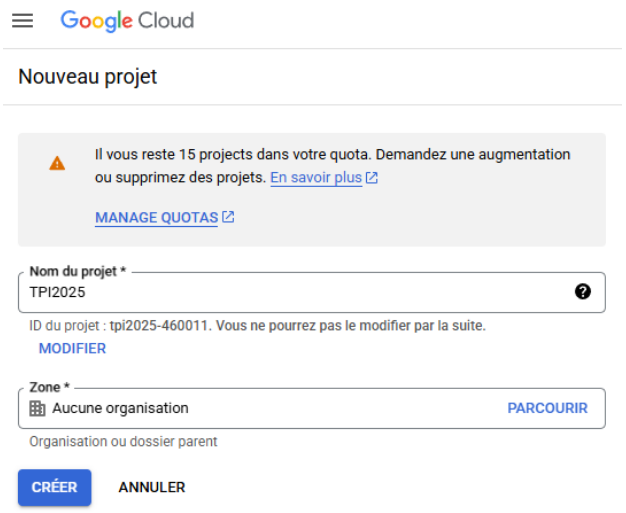
11.2.2 Google Maps

L'API Google Maps va nous servir à afficher une carte avec des markers comme dans l'exemple ci-dessous :

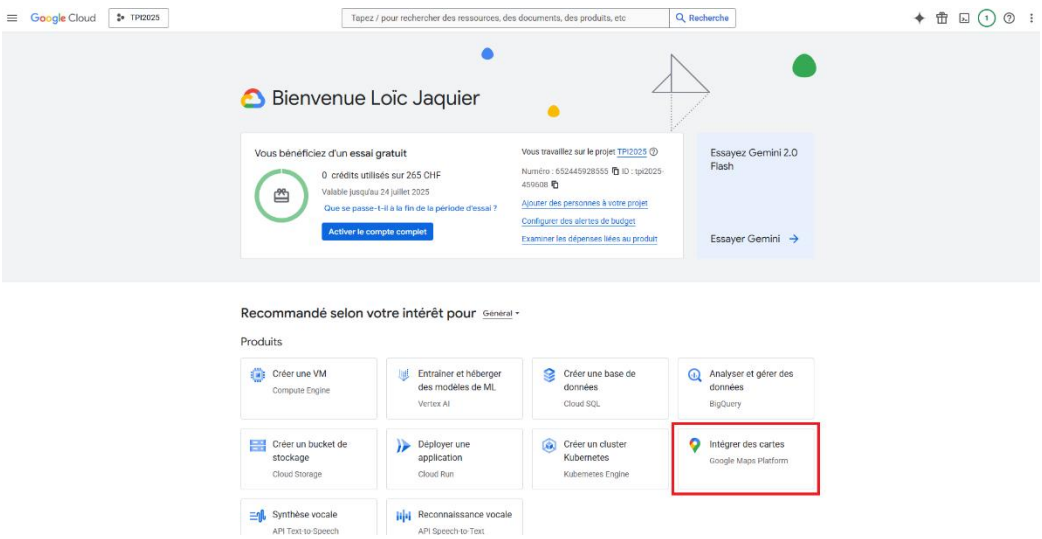


Avant de pouvoir afficher une carte, nous devons nous inscrire sur la plateforme Google Cloud, et effectuer les étapes suivantes afin de générer et récupérer une clé qui nous servira pour importer l'API.

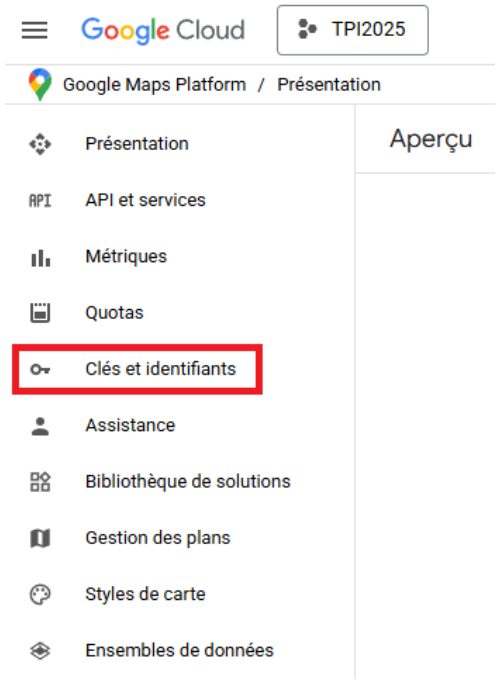
Création d'un nouveau projet :



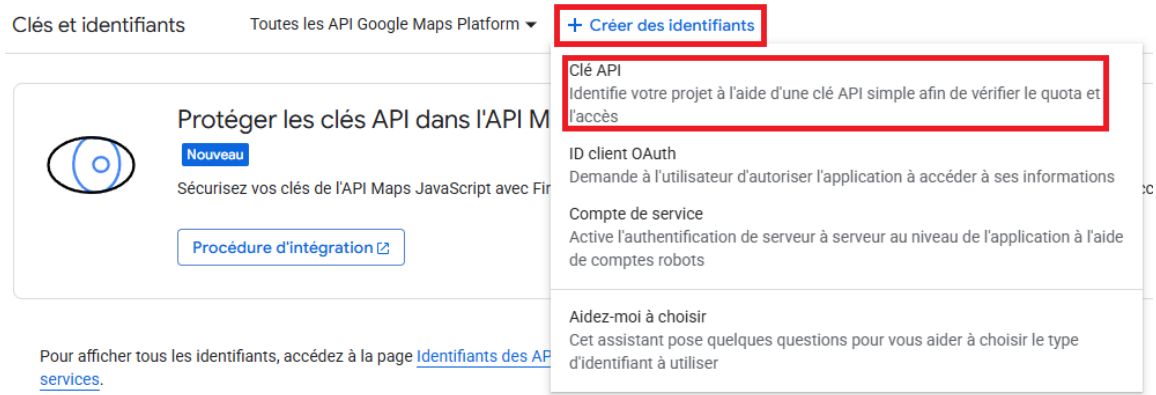
Accès à Google Maps Platform :

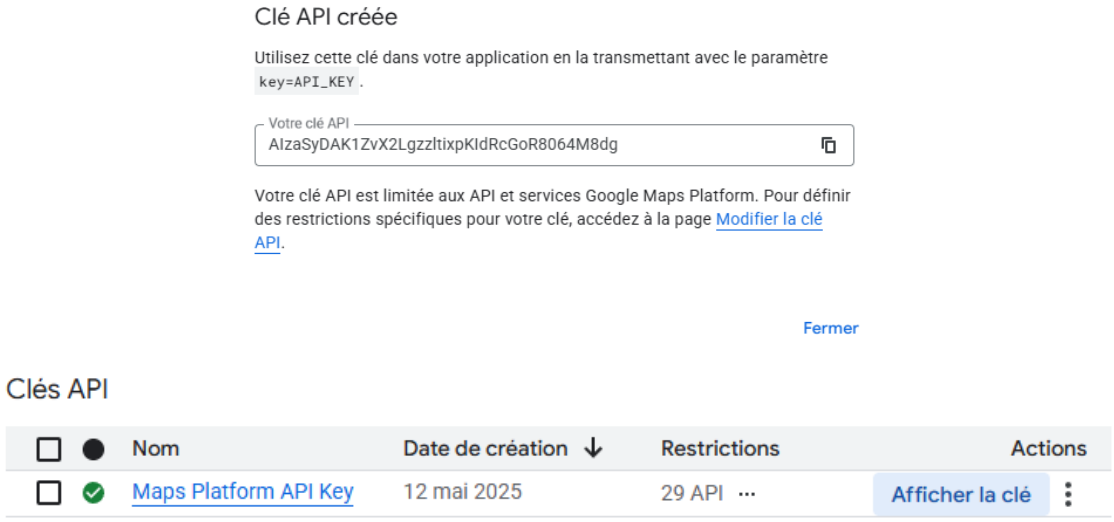


Se rendre dans Clés et identifiants :



Création d'une nouvelle clé :





Maintenant la clé générée, nous pouvons l'utiliser pour importer l'API dans notre projet.

L'API peut être importé de deux manières différentes, dans l'HTML ou dans le JavaScript.

Import de l'API du coté JavaScript :

Maps JavaScript API

```
(g => { var h, a, k, p = "The Google Maps JavaScript API", c = "google", l = "im-
portLibrary", q = "__ib__", m = document, b = window; b = b[c] || (b[c] = {}); var
d = b.maps || (b.maps = {}), r = new Set, e = new URLSearchParams, u = () => h ||
(h = new Promise(async (f, n) => { await (a = m.createElement("script"));
e.set("libraries", [...r] + ""); for (k in g) e.set(k.replace(/[A-Z]/g, t => "_" +
t[0].toLowerCase()), g[k]); e.set("callback", c + ".maps." + q); a.src =
`https://maps.${c}apis.com/maps/api/js?` + e; d[q] = f; a.onerror = () => h = n(Er-
ror(p + " could not load.")); a.nonce = m.querySelector("script[nonce]")?.nonce ||
""; m.head.append(a) })); d[l] ? console.warn(p + " only loads once. Ignoring:", g)
: d[l] = (f, ...n) => r.add(f) && u().then(() => d[l](f, ...n)) })({key: "API_KEY",
v: "weekly"});
```

Import de l'API du coté HTML :

```
<script type="module" src="https://maps.googleapis.com/maps/api/js?key=API_KEY"
async defer></script>
```

Import des librairies dans le JavaScript :

```
const { Map } = await google.maps.importLibrary("maps");
const { Marker } = await google.maps.importLibrary("marker");
```

Création d'une nouvelle carte (et définition de ses attributs) :

```
map = new Map(document.getElementById("map"), {
    zoom: 7,
    center: position,
    mapId: "map1",
    disableDefaultUI: false
});
```

Quelques attributs :

- Zoom : niveau de zoom sur la carte
- Center : position du centre de la carte
- mapId : identifiant de la carte
- disableDefaultUI : utiliser ou non l'affichage par défaut de la carte google maps

Création d'un nouveau marker (et définition de ses attributs) :

```
var newMarker = new Marker({
     map: map,
     position: position,
     title:"Titre du nouveau marker",
});
```

Quelques attributs :

- map : carte sur laquelle il est affiché
- position : position du marker
- draggable : le marker peut-il être déplacé ou non

Pour la carte et les markers il est possible d'utiliser tous les attributs mis à disposition par l'API pour personnaliser le comportement de ces éléments.

Ajout d'une action lorsqu'un marker est sélectionné :

```
newMarker.addListener('gmp-click', () => {
    …
});
```

Ajout d'une action lorsqu'un marker est déplacé :

```
newMarker.addListener('dragend', (event) => {
    …
});
```

11.2.3 Génération de QR Codes avec chillerlan/php-qrcode

Chillerlan/php-qrcode est une librairie PHP qui va nous permettre de générer des codes QR du côté de notre backend.

Pour se faire, nous devons la télécharger depuis GitHub à l'aide de composer.

Installation de PHP et composer depuis les sites web respectifs.

Commande composer pour l'installation de la librairie :

Chargement des dépendances

```
composer require chillerlan/php-qrcode
```

Chargement des classes nécessaires

```
use Endroid\QrCode\QrCode;
use Endroid\QrCode\Writer\PngWriter;
```

Création d'un code QR

```
$qrCode = new QrCode("https://bbt.emf-infopro-tpi.ch/Tests/QR_Code_BE/Frontend/in-
dex2.html?name=" . $name . "&fk=" . $fk);
```

Utilisation de PngWriter afin de générer un fichier png du code QR

```
$writer = new PngWriter();
$result = $writer->write($qrCode);
```

Résultat final