Boule de poils

Dossier de projet

François-Louis Toussaint

Titre Professionnel Développeur Web et Web Mobile

Ecole O'clock - Promotion Zeus - 2022

SOMMAIRE

REMERCIEMENTS

Six mois de cours intensifs, de pratiques mais aussi parfois de découragements viennent de s'écouler depuis le début de cette reconversion. Que de chemin parcouru, quelle satisfaction d'observer le résultat atteint, en étant parti d'aucunes connaissances en matière de développement web.

Mes premiers remerciements seront envers les youtubers Benjamin Code et Hugo Taschet, qui m'ont permis de découvrir ce métier, m'ont fait comprendre qu'il ne m'était pas inaccessible, et surtout m'ont transmis l'envie de l'exercer.

Vient ensuite Grafikart, pour le travail titanesque qu'il réalise, m'ayant permis de me préparer un tant soit peu avant de démarrer la formation, et qui plus d'une fois fut une ressource pour me sortir d'un mauvais pas.

Je tiens bien sûr à remercier tout le staff de O'clock pour le travail qu'ils accomplissent.

Notre super-référent quasi-omniscient et multi-tâche qui nous à suivi tout au long de cette aventure :
Loïc 😸

Nos professeurs qui ont su nous transmettre un maximum de leur savoir, nous encourager et de temps en temps nous faire rire :
Greg 🥥
Rémi 🎶
Solène 🍰
Alexis 🤘
Julien 🦄
Cécile 👩‍💻

Ainsi que Romain et Céline qui nous ont accompagnés durant la préparation du Titre Professionnel.

Merci aussi à tous mes camarades de la promotion Zeus, pour l'esprit d'entraide que le groupe à su développer, ainsi que pour la bonne ambiance au quotidien.

Et enfin, merci à mes coéquipiers, grâce auxquels le projet *Boule de poil*s a pu voir le jour, le tout sereinement et dans la bonne humeur.

I. INTRODUCTION

Après une carrière enrichissante d'une douzaine d'années dans le secteur de la coutellerie, une certaine lassitude s'est faite ressentir. Associée à d'autres facteurs survenus au fil du temps, j'ai alors été amené à envisager de prendre un nouveau départ dans ma vie.

S'envuivit dès lors une phase d'introspection, renouant avec des vieux démons issus de l'adolescence, à propos de la question "Mais que vais-je bien pouvoir faire ?". Et bien justement, pourquoi ne pas renouer avec l'intérêt que je porte à l'informatique depuis mon enfance ? Mais dont l'accès aux études ne m'a pas été rendu possible à l'époque.

Dès lors, j'ai pu découvrir un tout nouvel univers au cours de mes recherches, et la décision fut prise de m'aventurer sur la voie du développement web. Mon choix s'est porté sur l'école O'clock pour sa durée de formation équilibrée, ainsi que pour l'état d'esprit qu'elle véhicule. Il ne me restait plus qu'à arpenter les méandres de l'administration pour réaliser cette aventure.

Cette formation m'a donc offert l'opportunité d'acquérir de nouvelles connaissances afin d'exercer le métier de développeur web dans le futur. Ayant découvert le html/css, ainsi que les langages php et javascript, j'ai choisi d'effectuer une spécialisation basée sur l'utilisation de React, ayant une appétence particulière pour le front.

Ce projet *Boule de poils*, nous à permis, mes collègues et moi, d'affûter nos compétences. Après cinq mois de cours intensifs, et malgré la fatigue, il était temps pour nous de mettre en pratique tout ce que nous avions appris. En associant React et Symfony, dans le cadre d'une mise en situation professionnelle, tout en planchant sur une thématique qui nous tenait à cœur, la protection animale.

II. LISTE DES COMPÉTENCES COUVERTES PAR LE PROJET

A. Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

CP1 - Maquetter une application

Lors de la phase de mise au point de notre projet, tous les membres de l'équipe ont pu travailler conjointement sur les différents supports à établir préalablement, par le biais de différents outils en ligne, facilitant alors la discussion et la prise de décision. Le cahier des charges et les user stories que vous retrouverez un peu plus loin, ont été réalisés à l'aide des outils Google doc et Google sheet.
Les wireframes quant à eux ont été élaborés à l'aide de Figma, permettant à l'équipe de définir un design responsive ainsi qu'une charte graphique, approuvés en temps réel par tous les membres, en partant sur une idée de créer une sorte de "Tinder de l'adoption d'animaux" .

CP2 - Réaliser une interface utilisateur web statique et adaptable

A partir des wireframes, les différentes pages de notre site ont ainsi pu être intégrées, les composants du front office étant créés avec React.
A l'exception des formulaires, ou la bibliothèque Material UI fut utilisée, le front office à été stylisé intégralement en css personnalisé, à l'aide du préprocesseur Sass.
Le framework Bootstrap à été associé à Twig de Symfony pour la réalisation du back office, auquel fut adjoint quelques lignes de css afin d'obtenir une correspondance des couleurs avec la charte graphique établie dans le cahier des charges.
Cela nous à permis d'obtenir une application pensée en premier lieu pour les ordiphones, et dont l'affiche se module en fonction du passage vers des écrans plus larges.

CP3 - Développer une interface utilisateur web dynamique

L'utilisation de React sur le front office nous permet d'obtenir un site dynamique, réagissant aux différentes interactions de l'utilisateur avec l'application. Chaque élément se voit chargé dans un DOM virtuel, qui à chaque changement d'état d'un composant met à jour l'affichage qui en découle.
Axios va quant à lui permettre d'effectuer des requêtes http vers les endpoints de notre API afin d'y récupérer ou envoyer des données.
En ce qui concerne le back office, les éléments importés de bootstrap contiennent déjà du code javascript, leurs permettant de réagir aux interactions de l'utilisateur.

B. Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

CP5 - Créer une base de donnée

Notre base de données a été conçue avec la méthode Merise. Une fois le MCD établi avec l'ensemble des données qui nous semblaient nécessaires au fonctionnement de notre application, nous avons pu réaliser le MLD ainsi que le dictionnaire de données. A l'aide de l'ORM Doctrine de Symfony, les différentes tables ont par la suite pu être créées, au sein de notre SGBD, MySQL, qui accueille la base de données.

CP6 - Développer les composants d'accès aux données

Afin de pouvoir récupérer et envoyer des données depuis le front office, une API a été développée, avec la mise en place de différents endpoints, dont les requêtes seront traitées par le Controller dédié.
Les consultations et manipulations de la base de données s'effectuent par l'intermédiaire de l'ORM Doctrine, par le biais des Entities associées aux Repositories qui vont pouvoir définir les requêtes SQL à exécuter sur les données souhaitées.

CP7 - Développer la partie back-end d'une application web ou web mobile

L'architecture du projet sur la partie back end est conçue avec le design pattern MVC. Lorsqu'une requête HTTP est reçue, celle-ci sera récupérée par le Controller dédié via la route configurée grâce aux Annotations. Le traitement est alors effectué, faisant appel si besoin aux Entity et Repository nécessaires qui traiteront les données via l'ORM Doctrine. L'affichage retourné à l'utilisateur sera quant à lui traité par le moteur de template Twig, qui génère le code html en conséquence.
L'usage de la POO nous permet d'obtenir un code organisé et réutilisable en différentes parties du projet, en définissant chaque élément en tant qu'objet, avec ses propriétés et actions, et régulant leur contexte d'usage.
En ce qui concerne les notions de sécurité, Doctrine possède des fonctions d'origine, permettant les filtrages des valeurs et la prévention des attaques par injection SQL. Ont été mis en place des rôles utilisateurs afin de restreindre l'accès à certaines fonctionnalités selon les droits accordés. Ainsi que l'usage de jetons JWT stockés en cookie pour les utilisateurs authentifiés. L'API est quant à elle régulée par des règles CORS, autorisant uniquement l'accès depuis le front office.

III. RÉSUMÉ DU PROJET

Problématique

Il est parfois compliqué d'adopter un animal de compagnie, il faut souvent se renseigner sur l'existence des refuges et associations autour de chez soi, et s'y déplacer sans pour autant être sûr de trouver l'animal qui correspond à nos critères. Les adoptants y passent vite des week-ends entiers, y dépensant du temps, de l'énergie, leurs déplacements engendrant aussi des frais kilométriques.

Les associations quant à elles, n'ont pas forcément à disposition un site internet leur permettant de présenter leurs animaux disponibles à l'adoption.

Objectifs

Boule de poils se propose donc d'être une plateforme regroupant les animaux d'associations de protection animalière et de refuges. Permettant à une personne de trouver son compagnon idéal, selon ses critères, ainsi que sa position géographique, depuis chez elle. *Boule de poils* permet par la même occasion aux refuges et associations collaboratrices, en proposant leurs animaux, d'augmenter leur visibilité, et ainsi d'en faciliter l'adoption.

Description du projet

Notre équipe composée de développeurs React et Symfony a donc, durant presque un mois, planché sur la réalisation d'une application avec pour idée originale de reproduire une sorte de "Tinder" de l'adoption d'animaux. Avec une interface destinée au grand public, offrant l'opportunité de rechercher des profils d'animaux selon ses critères; et une console d'administration permettant aux associations partenaires de gérer les animaux qu'elles proposent, ainsi qu'aux administrateurs de gérer toutes les données utiles à l'application. Au fil du temps, l'intégration de certaines fonctionnalités, comme la mise en favoris d'animaux, a été reporté à des versions ultérieures de l'application, afin de se concentrer pleinement sur l'accomplissement des objectifs principaux.

IV. CAHIER DES CHARGES

A. Conceptualisation de l'application

Les Fonctionnalités du projet

Lors de la phase de conception de l'application, plusieurs idées de fonctionnalités ont été émises. Néanmoins, au cours de l'avancement du projet et au vu du temps qui nous était imparti, certaines d'entre elles ont dû être écartées, afin d'obtenir une première version d'application suffisamment stable pour son utilisation.

Le Minimum Viable Product comporte donc les fonctionnalités suivantes :

- Front-office :
  - Slider d'affichage d'animaux aléatoires en page d'accueil
  - Recherche de profils d'animaux avec critères filtrants :
    - Espèce
    - Sexe
    - Âge
    - Compatibilité avec les enfants
    - Compatibilité avec d'autres animaux
    - Besoin d'un accès à l'extérieur
    - Situation géographique par département
  - Consultations des profils d'animaux résultant de la recherche
  - Inscription et authentification d'un utilisateur
- Back-office :
  - Connexion à la console d'administration
  - Gestion CRUD des associations
  - Gestion CRUD des animaux
  - Gestion CRUD des espèces
  - Gestion CRUD des utilisateurs

Les fonctionnalités envisagées pour faire évoluer l'application :

- Mise en favori d'un animal
- Commentaire sur le profil d'un animal
- Signalement d' un contenu inapproprié
- Recherche de profils d'animaux avec critère géographique kilométrique
- Messagerie de contact entre utilisateur et association
- Enregistrement des critères personnels dans le profil d'utilisateur
- Notification de nouveaux animaux enregistrés correspondant à ces critères
- Recadrage d'une photo d'animal avant son upload

B. Utilisateurs et user stories

Le public visé et les rôles

L'application est destinée aux personnes en recherche d'un animal à adopter, et aux associations souhaitant proposer leurs animaux réfugiés à l'adoption.
Quatre profils d'utilisateurs ont ainsi pu être identifiés au cours de nos réflexions, auxquels ont été accordés des droits d'accès différents selon les fonctionnalités :

Les visiteurs non connectés, ont accès à :

- L'ensemble des pages disponibles sur le front-office
- La fonctionnalité de recherche d'animaux
- La consultation des profils
- Un formulaire d'inscription
- Un formulaire de connexion

Les visiteurs connectés, ont accès à :

- Un lien dans la barre de navigation vers la section Mes Favoris (qui redirige vers une page 404 personnalisée tant que la fonctionnalité n'a pas été intégrée)

Les associations et administrateurs non connectées ont accès à :

- Un formulaire de connexion afin de s'authentifier

Les associations connectées , ont accès aux :

- Pages leur permettant de gérer les animaux qu'elles proposent

Les administrateurs connectés, ont accès aux:

- Pages de gestion des associations
- Pages de gestion de tous les animaux présents dans la base de données
- Pages de gestion des espèces présentes dans la base de données
- Pages de gestion des utilisateurs

Les User Stories

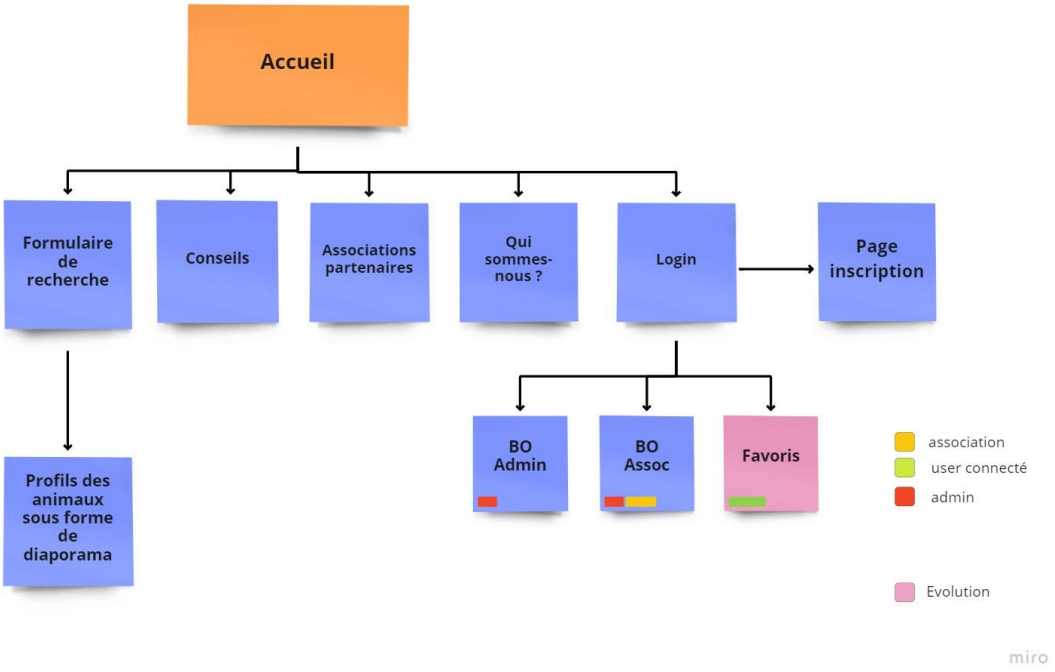
En se basant sur les différents rôles et leurs droits définis en amont, les user stories ont ainsi pu être rédigé :

En tant que	Je veux	Afin de (si nécessaire)
visiteur	Pouvoir m'inscrire	Créer un profil d'utilisateur
visiteur	Pouvoir me connecter	Accéder aux droits user
visiteur	Pouvoir me déconnecter	Se déconnecter de l'application
visiteur / user	accéder au formulaire de recherche d'animaux	rechercher un animal en fonction de mes critères
visiteur / user	pouvoir visualiser les animaux répondant à mes critères	trouver ma boule de poils :)
visiteur / user	pouvoir accéder à la liste de refuges / associations partenaires	Identifier les associations participant à l'application
visiteur / user	pouvoir accéder à une page de conseils à l'adoption	
visiteur / user	pouvoir avoir les informations de contact d'un refuge / association	
admin / association	pouvoir me connecter à la console d'administration	gérer les contenus me concernant
admin / association	pouvoir me déconnecter	se déconnecter de l'application
association	visualiser la liste des animaux de mon association	
association	pouvoir ajouter un animal	
association	afficher le profil d'un animal	
association	modifier le profil d'un animal	
association	mettre à jour la disponibilité de l'animal	
association	supprimer le profil de l'animal	
admin	visualiser la liste des associations	
admin	ajouter une association	

En tant que	Je veux	Afin de (si nécessaire)
admin	modifier une association	
admin	supprimer une association	
admin	visualiser la liste des utilisateurs	
admin	ajouter un utilisateur	
admin	modifier un utilisateur	
admin	mettre à jour les rôles des utilisateurs	
admin	supprimer un utilisateur	
admin	visualiser la liste des espèces	
admin	ajouter une espèce	
admin	modifier une espèce	
admin	supprimer une espèce	
admin	visualiser la liste de tous les animaux	
admin	pouvoir ajouter un animal	
admin	afficher le profil d'un animal	
admin	modifier le profil d'un animal	
admin	mettre à jour la disponibilité de l'animal	
admin	supprimer le profil de l'animal	

C. L'arborescence de l'application

Une fois les fonctionnalités primaires et les rôles utilisateurs cernés, il nous a été possible de mettre au point la structure que devrait prendre l'application, dans l'organisation des différentes pages la composant. Et d'en schématiser les parcours que pourraient emprunter les différents utilisateurs.

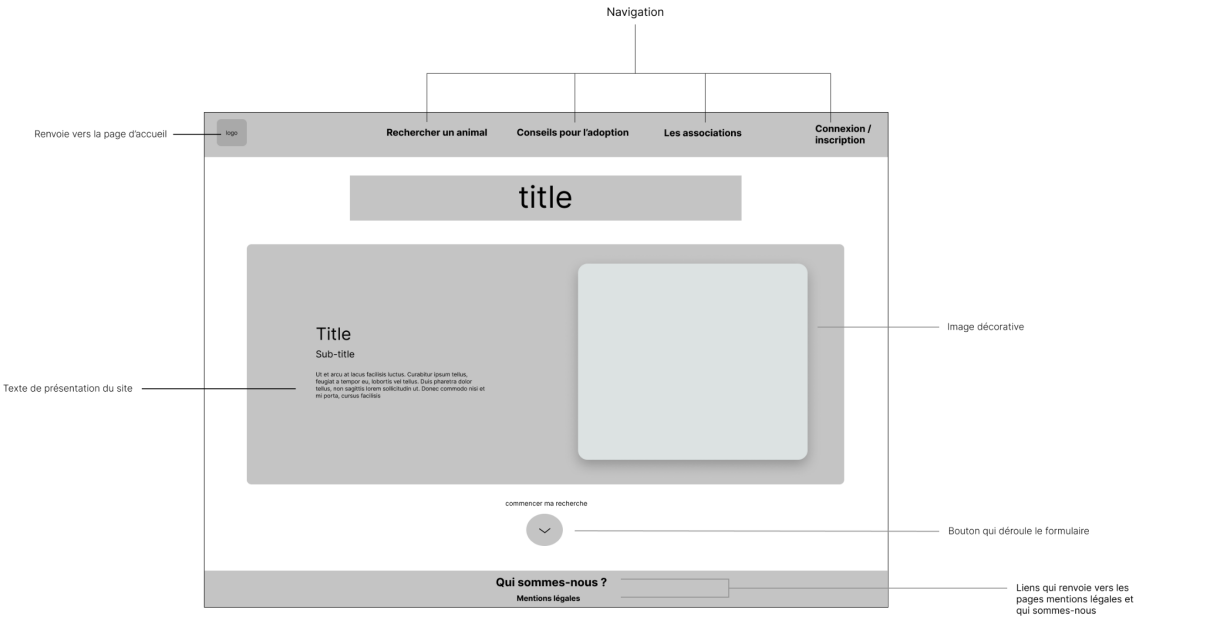


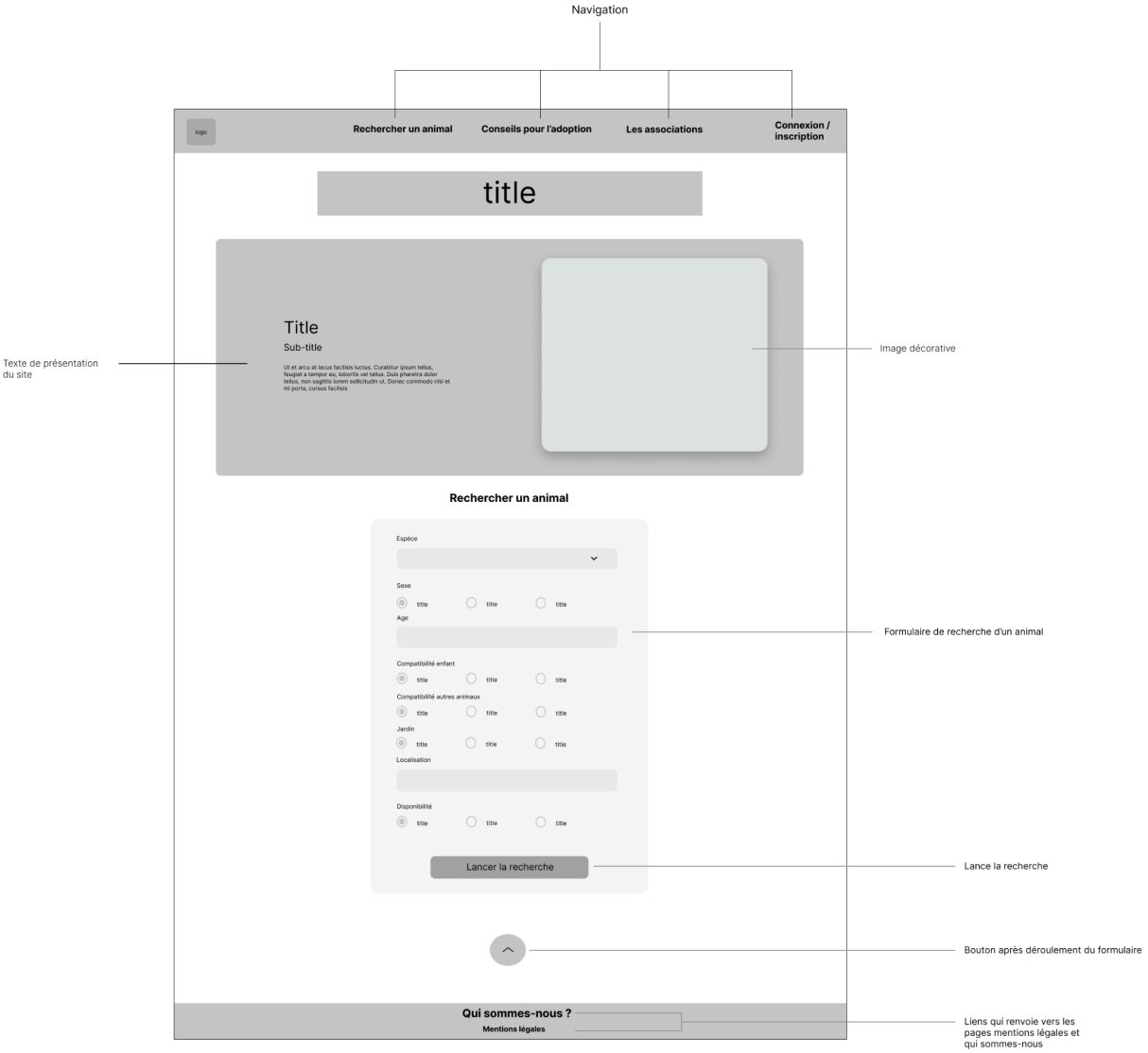
D. Mise en dimension de l'application

Les Wireframes de l'application

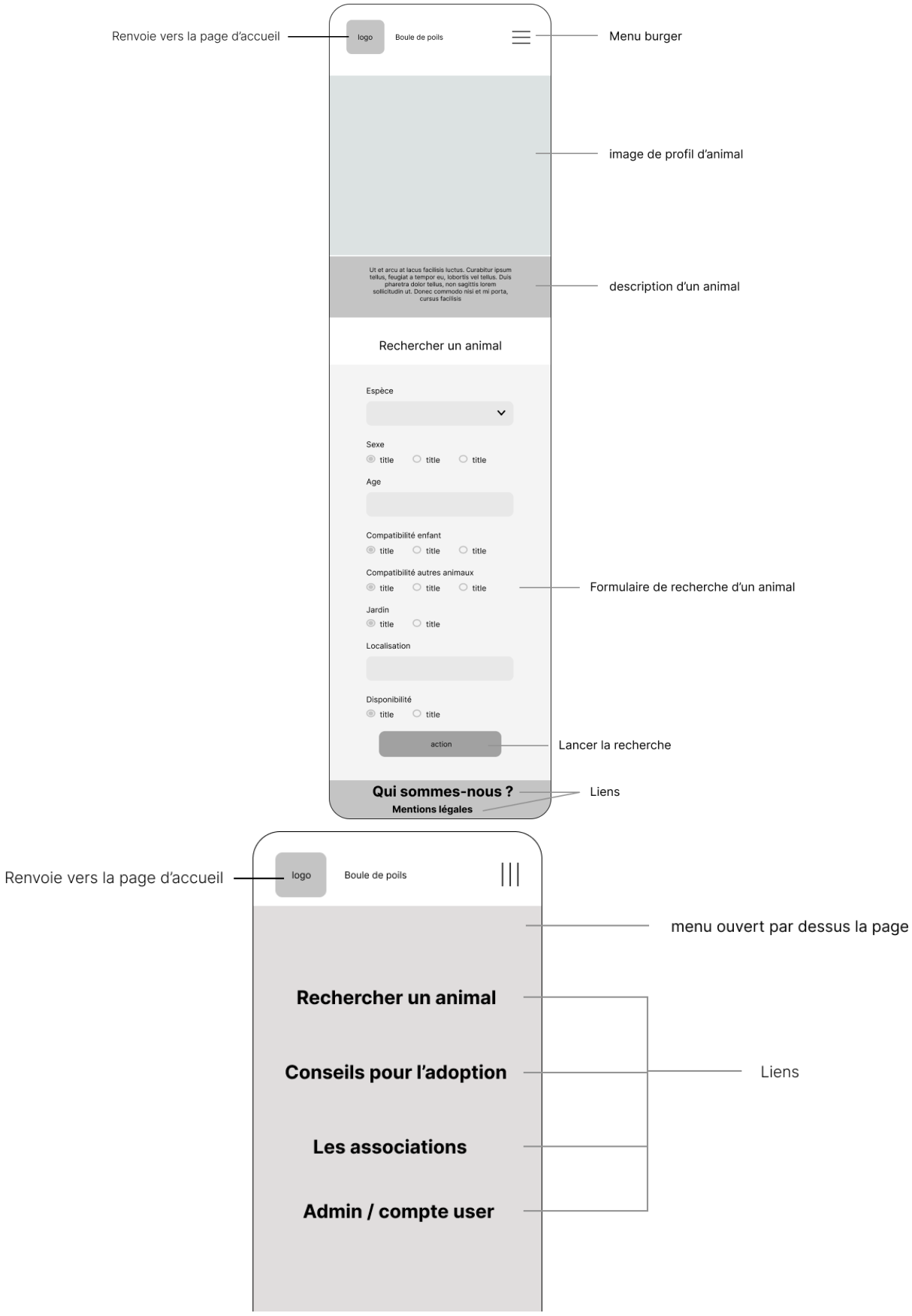
A partir de l'arborescence, il nous fut alors possible de maquetter les différentes pages. Mettant au point le design de l'application pour les différents supports d'affichage en plaçant les éléments composant les pages.

Wireframe Page d'accueil Desktop

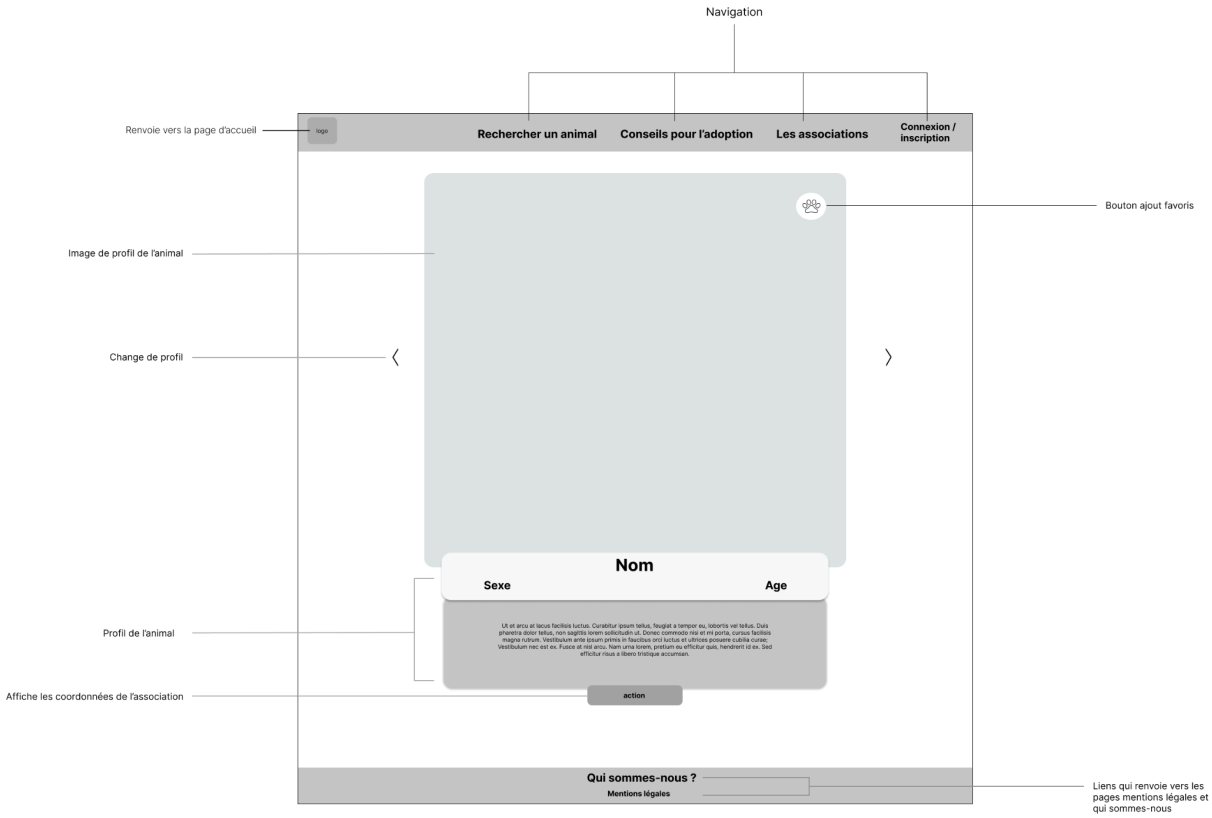


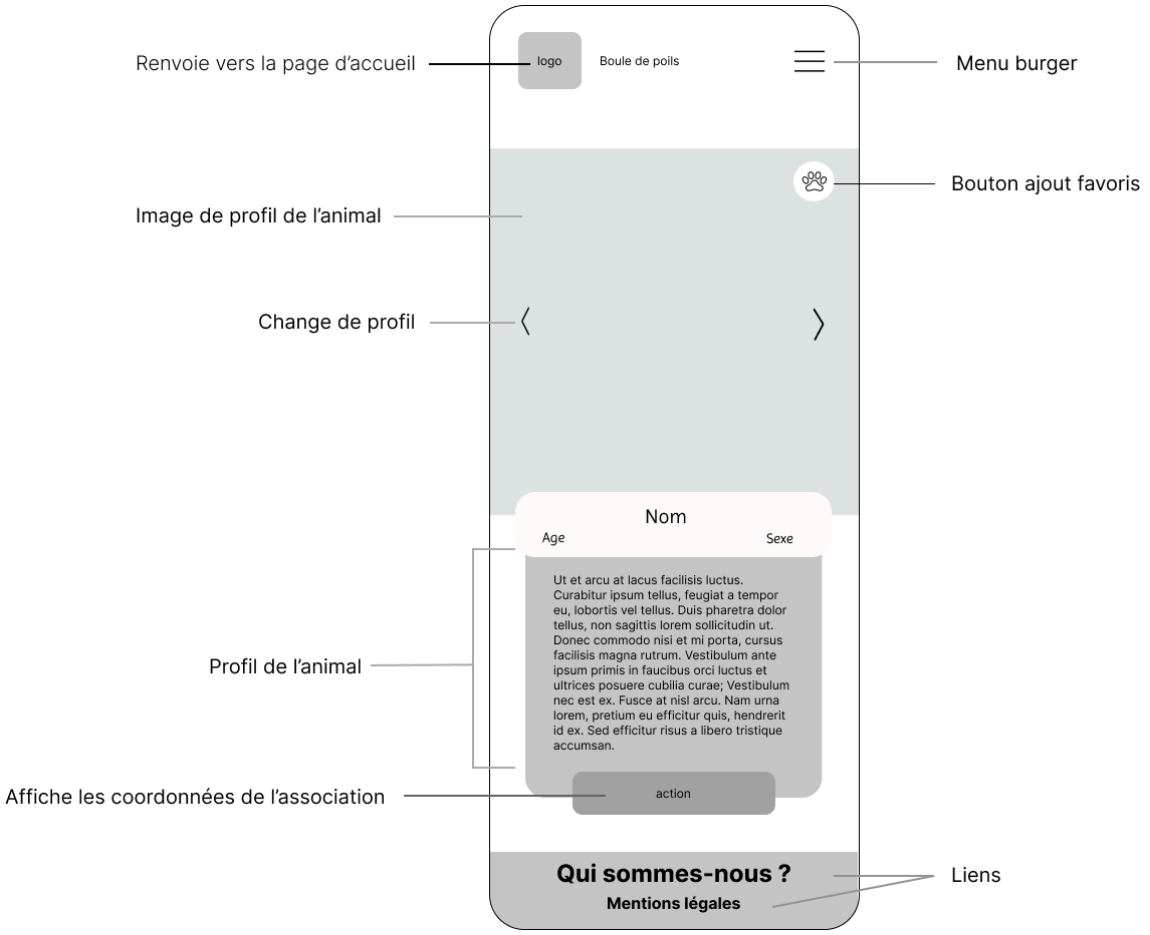


Wireframe Page d'accueil Mobile

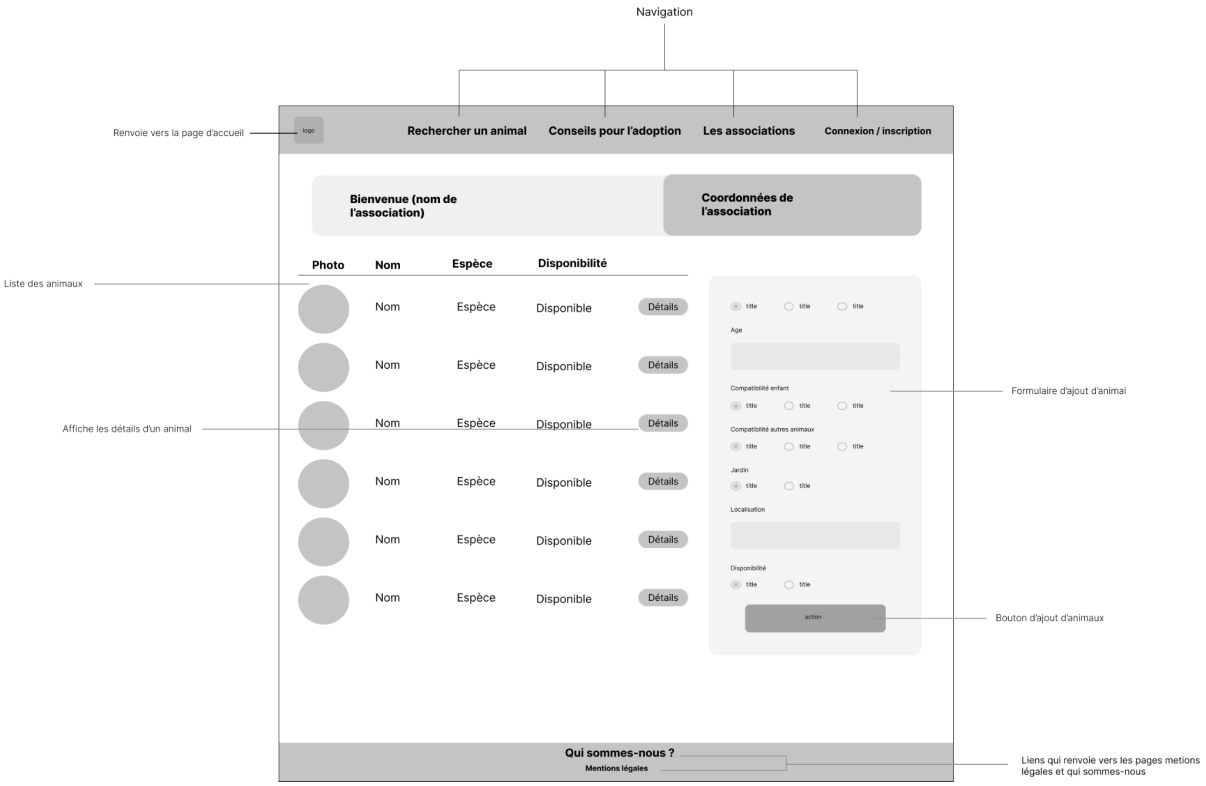


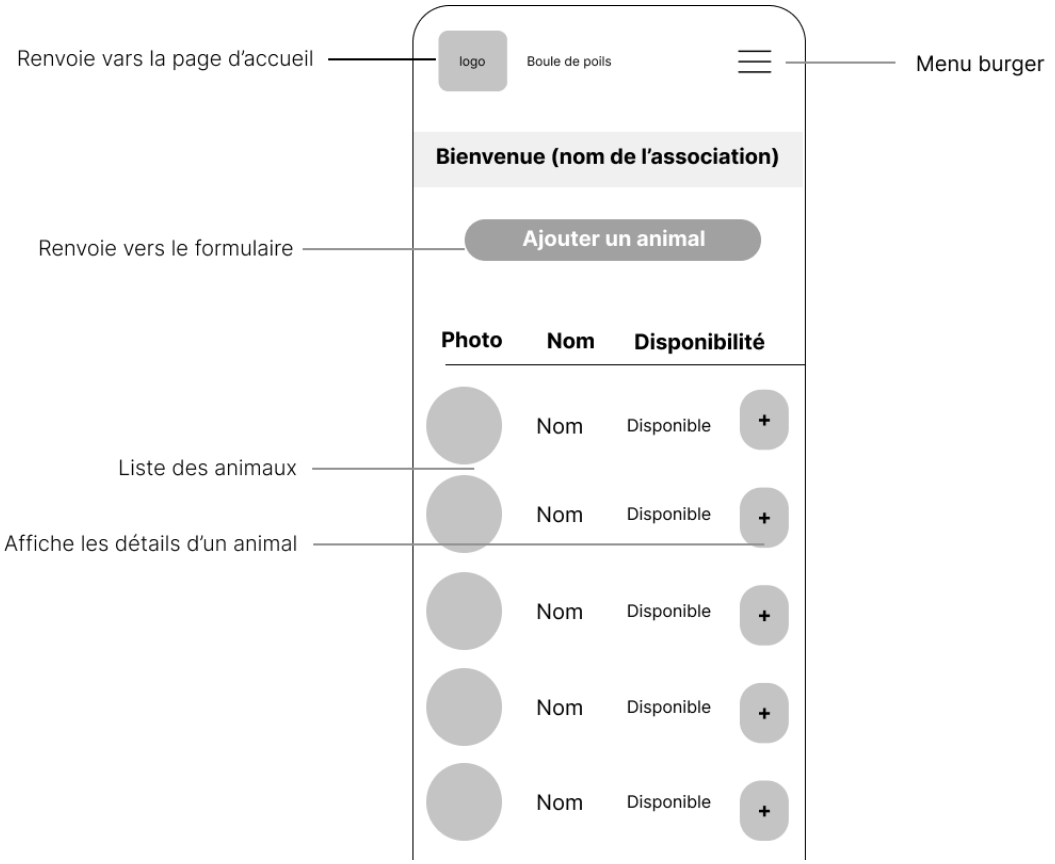
Wireframe de la page de profil d'un animal



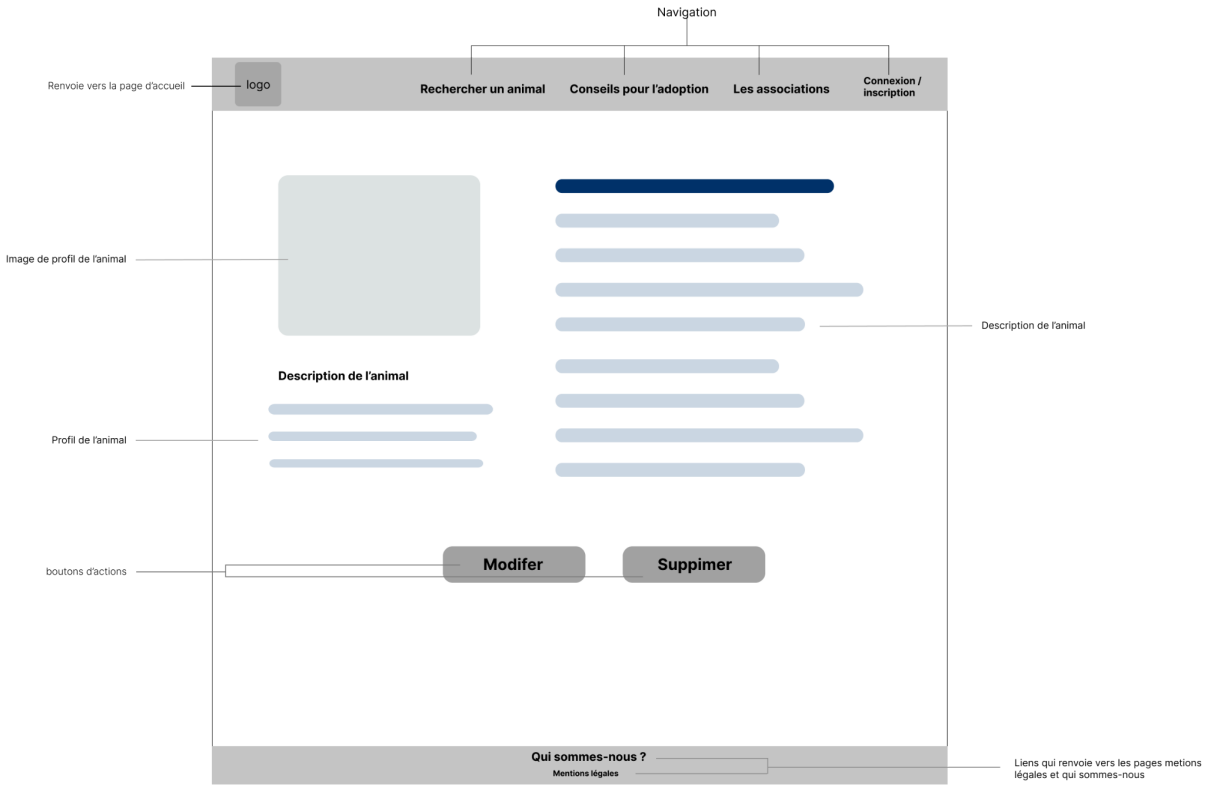


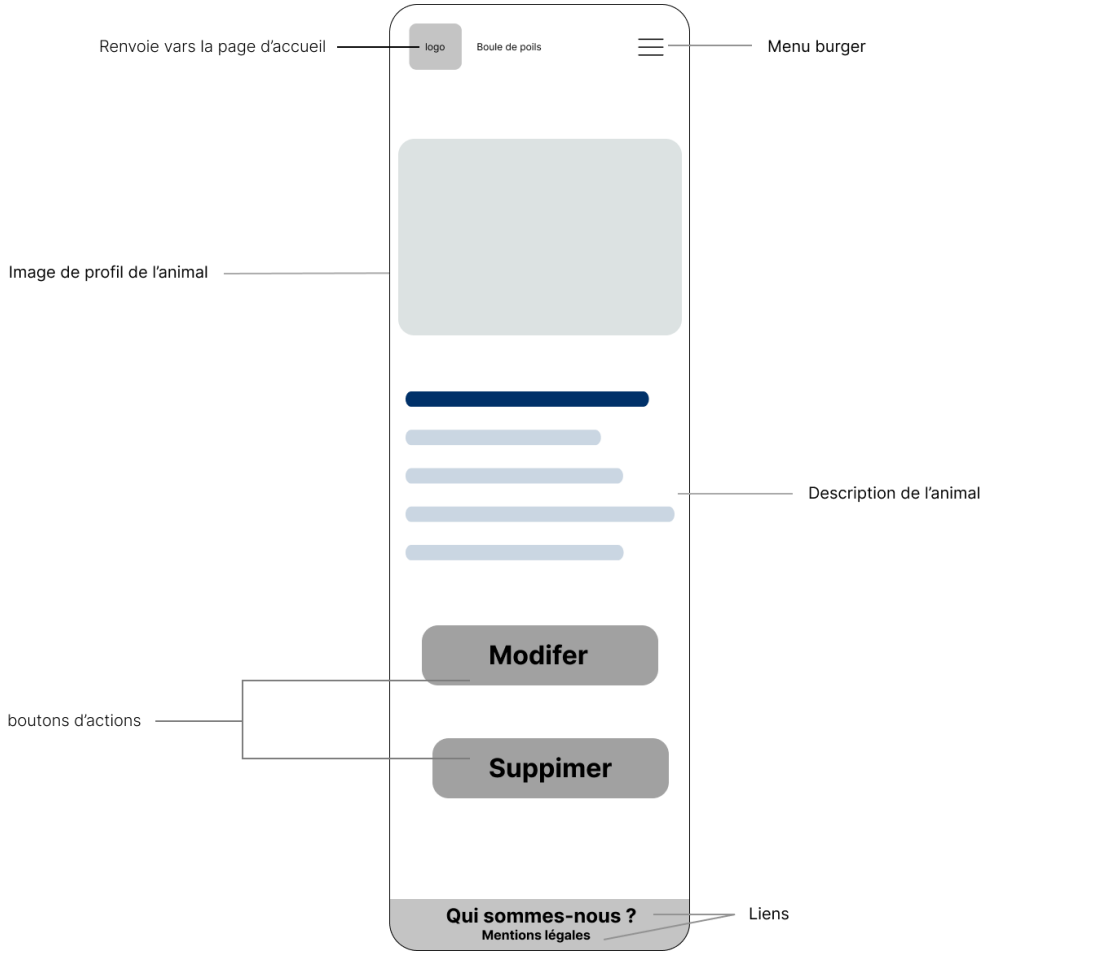
Wireframe de la page de gestion pour les associations





Wireframe d’un profil d’animal dans le back-office





La charte graphique

Nous nous sommes concertés afin de trouver un code couleur et des typographies qui nous plaisent, à partir de plusieurs exemples dénichés sur Pinterest et Google Font. Un logo libre de droit à aussi été déniché et fut accordé aux couleurs choisies via Adobe Illustrator. Il a donc été convenu de suivre les règles suivantes :

- Couleurs :
  - Primary :
    - main : #735F6E
    - light : #8C7488
    - dark : #594A55
    - contrast text : #F2F0F2
  - Secondary :
    - main : #D97C0B
    - light : #F2B33D
    - dark : #BF6D0A
    - contrast text : #F2F0F2

- Polices d'écriture:
  - Formulaires: Roboto
  - Titres : Arima Madurai
  - Paragraphes : Lato

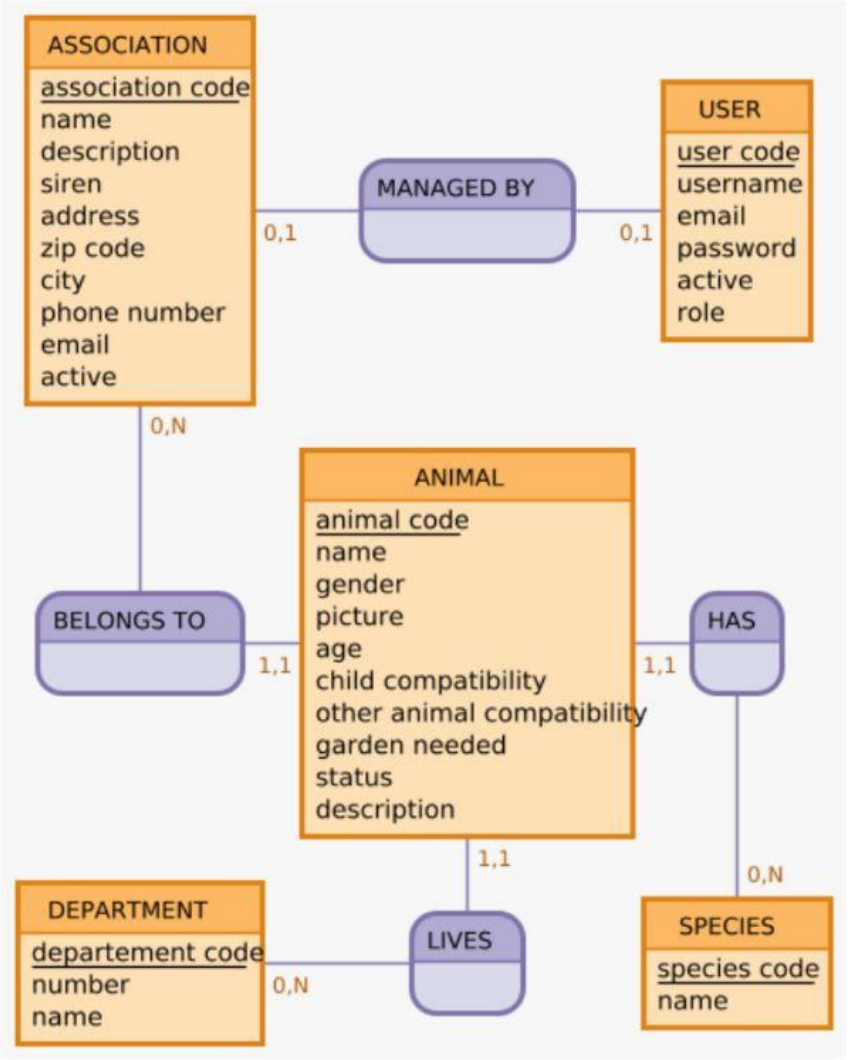
E. Mise en place de la base de données

MCD

Nous avons établi une modélisation de notre base de données, incluant les entités avec leurs propriétés, ainsi que les relations les liant entre elles.

L'entité principale de l'application se trouve être naturellement l'animal, qui appartient à une espèce, se situe dans un département et qui a trouvé refuge auprès d'une association. L'association quant à elle, est gérée par un utilisateur et peut accueillir plusieurs animaux.

Précisons que la situation géographique d'un animal n'est effectivement pas liée à celle de l'association dont il dépend. En effet, certaines associations opèrent sur le plan national, effectuant des transferts entre différents refuges lui appartenant à travers le territoire, et certains jeunes animaux peuvent se voir placés en famille d'accueil.



MLD

A partir du MCD, le modèle logique a pu être mis au point, afin de déterminer les tables nécessaires au sein de notre base de données, accompagnées des clés primaires et étrangères qui s'y trouveront.

```
ASSOCIATION (_association code_, name, description, siren, address, zip code,
city, phone number, email, active)
USER (_user code_, username, email, password, active, role, #association code)
ANIMAL (_animal code_, name, gender, picture, age, child compatibility, other
animal compatibility, garden needed, status, description, #species code,
#departement code, #association code)
DEPARTMENT (_departement code_, number, name)
SPECIES (_species code_, name)
```

Dictionnaire de données

La phase de conception de la base de données s'est achevée par la rédaction du dictionnaire de données. Chaque champ s'est alors vu attribuer un type et des spécifications selon les valeurs qu'ils devront contenir. Nous avons pu nous baser sur ce document afin de développer les Entities de Symfony dans le répertoire back end du projet, et ainsi créer les différentes tables au sein de notre base de données en effectuant une migration avec l'ORM Doctrine.

❖ Animal

Champ	Type	Spécificité	*Description*
id	INTEGER	PRIMARY KEY, NOT NULL, AUTO INCREMENT	*id de l'animal*
name	VARCHAR(255)	NOT NULL	*nom de l'animal*
gender	SMALLINT	NOT NULL	*genre de l'animal - 0 = femelle / 1 = male / 2 = indifférent*
imageName	VARCHAR(255)	NOT NULL	*photo de l'animal*
species	ENTITY	NOT NULL	*clé étrangère - espèce de l'animal*
age	INTEGER	NOT NULL	*âge de l'animal*
child_compatibility	BOOLEAN	NOT NULL	*compatibilité avec des enfant*
other_animal_compatibility	BOOLEAN	NOT NULL	*compatibilité avec d'autres animaux*
garden_needed	BOOLEAN	NOT NULL	*besoin d'un jardin*
status	SMALLINT	NOT NULL	*0 = disponible / 1 = réservation en cours / 2 = adopté / 3 = indisponible*
description	TEXT	NOT NULL	*description de l'animal*
department	ENTITY	NOT NULL	*clé étrangère - localisation de l'animal*
association	ENTITY	NOT NULL	*clé étrangère - association de l'animal*

❖ Association

Champ	Type	Spécificité	*Description*
id	INTEGER	PRIMARY KEY, NOT NULL, AUTO INCREMENT	*id de l'association*
name	VARCHAR(255)	NOT NULL	*nom de l'association*
description	TEXT	NOT NULL	*description de l'association*
siren	VARCHAR(255)	NOT NULL	*numéro de SIREN de l'association*
street	VARCHAR(255)	NOT NULL	*numéro et nom de la rue*
zip_code	VARCHAR(255)	NOT NULL	*code postal*
city	VARCHAR(255)	NOT NULL	*Ville*
phone_number	VARCHAR(255)	NULLABLE	*numéro de téléphone (facultatif)*
email	VARCHAR(255)	NOT NULL	*email de l'association*
active	BOOLEAN	NOT NULL	*0 = actif* *1 = inactif*

❖ Department

Champ	Type	Spécificité	*Description*
id	INTEGER	PRIMARY KEY, NOT NULL, AUTO INCREMENT	*id du département*
number	VARCHAR(10)	NOT NULL	*code postal du département*
name	VARCHAR(255)	NOT NULL	*nom du département*

❖ Species

Champ	Type	Spécificité	*Description*
id	INTEGER	PRIMARY KEY, NOT NULL, AUTO INCREMENT	*id de l'espèce*
name	VARCHAR(255)	NOT NULL	*nom de l'espèce*

❖ User

Champ	Type	Spécificité	Description
id	INTEGER	PRIMARY KEY, NOT NULL, AUTO INCREMENT	*id de l'utilisateur*
pseudo	VARCHAR(255)	NOT NULL, UNIQUE	*username de l'utilisateur*
email	VARCHAR(255)	NOT NULL, UNIQUE	*email de l'utilisateur*
password	VARCHAR(255)	NOT NULL	*mot de passe de l'utilisateur*
active	BOOLEAN	NOT NULL	*0 = actif* *1 = inactif*
rôle	JSON	NOT NULL	*droits d'accès*

F. Les routes

Les routes ont par la suite été déterminées, celles du front-office seront gérées en Javascript grâce à React-Router-Dom. Concernant le back-office, plus de détails on été apportés afin d'assurer une bonne organisation pour le traitement des données, et les différents endpoints de l'API ont été définis.

URL - Front	Content
/	Slider + formulaire de recherche d'animaux
/favoris	Page regroupant les animaux favoris du user
/associations	Liste des associations participantes
/conseils	Conseils aux adoptants
/equipe	Page statique présentant l'équipe et contact
/rejoindre	Page invitant les associations à contacter l'admin
/signin	Formulaire d'inscription
/login	Formulaire de connexion

URL - Back Office	HTTP Method	Controller	Method	Title	Content
backoffice/animal/	GET	AnimalController	browse	Back Office	Affichage de la liste des animaux
backoffice/animal/{id}	GET	AnimalController	read	Back Office	Affichage de la fiche d'un animal
backoffice/animal/{id}/edit	GET, PUT	AnimalController	edit	Back Office	Modification de la fiche d'un animal
backoffice/animal/new	GET, POST	AnimalController	add	Back Office	Ajout de la fiche d'un animal

URL - Back Office	HTTP Method	Controller	Method	Title	Content
backoffice/animal/{id}	POST	AnimalController	delete	Back Office	Suppression de la fiche d'un animal
backoffice/species/	GET	SpeciesController	browse	Back Office	Affichage de la liste des éspèces
backoffice/species/{id}	GET	SpeciesController	read	Back Office	Affichage d'une éspèce
backoffice/species/{id}/edit	GET, PUT	SpeciesController	edit	Back Office	Modification d'une éspèce
backoffice/species/new	GET, POST	SpeciesController	add	Back Office	Ajout d'une éspèce
backoffice/species/{id}	POST	SpeciesController	delete	Back Office	Suppression d'une éspèce
backoffice/user/	GET	UserController	browse	Back Office	Affichage de la liste des utilisateurs
backoffice/user/{id}	GET	UserController	read	Back Office	Affichage d'un utilisateur
backoffice/user/{id}/edit	GET, PUT	UserController	edit	Back Office	Modification d'un utilisateur
backoffice/user/new	GET, POST	UserController	add	Back Office	Ajout d'un utilisateur
backoffice/user/{id}	POST	UserController	delete	Back Office	Suppression d'un utilisateur
backoffice/association/	GET	AssociationController	browse	Back Office	Affichage de la liste des associations
backoffice/association/{id}	GET	AssociationController	read	Back Office	Affichage d'une association
backoffice/association/{id}/edit	GET, PUT	AssociationController	edit	Back Office	Modification d'une association
backoffice/association/new	GET, POST	AssociationController	add	Back Office	Ajout d'une association
backoffice/association/{id}	POST	AssociationController	delete	Back Office	Suppression d'une association

API - EndPoint	HTTP Method	Controller	Method	Title	Content
/api/animal/form	GET, POST	AnimalController	getResults		Traitement du formulaires et renvoie vers la vue
/api/animal/caroussel	GET	AnimalController	animalCaroussel		Envoie 10 animaux aléatoire de la BDD
/api/department	GET	DeparmentController	departmentList		Envoie la liste des départements
/api/species	GET	SpeciesController	speciesList		Envoie la liste des espèces
/api/association	GET	AssociationController	associationList		Envoie la liste des associations
/api/login	GET, POST	Login			Gère l'authentification
/api/user/form	GET, POST	UserController			Traitement du formulaires et renvoie vers la vue

V. SPÉCIFICATIONS TECHNIQUES

Versioning

Afin d'optimiser l'organisation de notre projet, qui pour sa première version contient 42 branches avec 228 commits pour la partie front-end et 20 branches avec 78 commits pour la partie back-end; nous avons choisi d'établir des conventions de nommage pour l'application en elle-même, mais aussi de ses différentes fonctionnalités lors de leur développement, à l'aide de l'outil Git.

Pour l'application mise en production, en définissant sa version et ses mises à jour. La première version se trouve nommée v1, une correction de défauts d'affichage à été apportée en toute fin de projet sur le code css, faisant passer la version actuelle à la v1.1.

Lors du développement des différentes fonctionnalités nous avons choisi la convention suivante pour gérer nos branches et commits :

- Branch
  - feature-nomDeLaFeature, développement d'une nouvelle fonctionnalité
  - fix-nomDeLaCorrection, correction d'une fonctionnalité
  - experiment-nomDuTest, essai d'une nouvelle fonctionnalité
- Commit
  - feat(nom du fichier ou composant): description du travail effectué
  - fix(nom du fichier ou composant): description du travail effectué
  - wip(nom du fichier ou composant): description de l'avancement du travail
  - merge: nomDeLaFeature

Les technologies du front

- **Yarn** : gestionnaire de packages javascript, utilisé afin d'installer les dépendances nécessaires au fonctionnement de l'application, mais aussi lors de son développement.
- **Webpack** : permet de gérer les différentes dépendances, en automatisant par exemple des tâches comme la compilation du code.
- **Babel** : nous utilisons la version ES6 de javascript, qui nous permet par exemple l'usage des fonctions fléchées. Malheureusement, certains navigateurs, surtout s'ils ne sont pas mis à jour, ne sont pas compatibles avec cette version. Babel résout celà en retranscrivant le code dans une version antérieure lors de sa compilation.

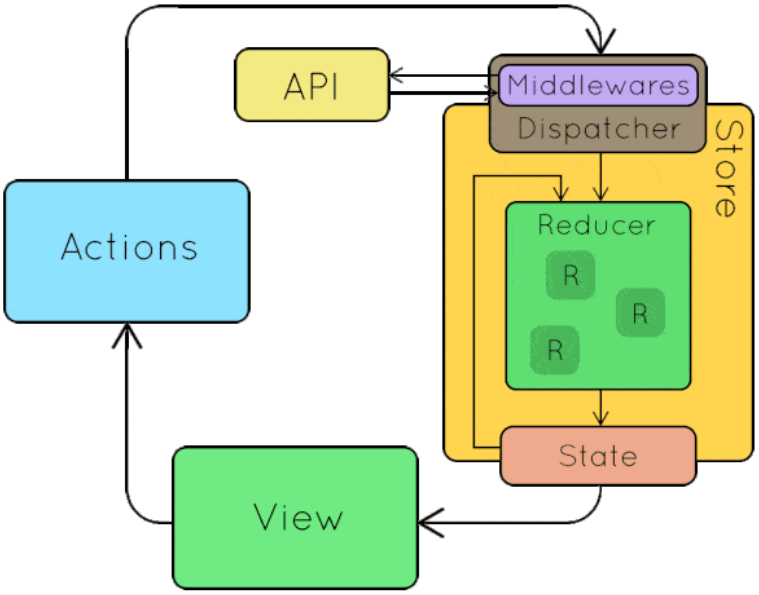
- **ESlint** : cet outil analyse notre code lorsque nous développons, afin de s'assurer du suivi de règles syntaxiques en vue d'obtenir un code propre et compréhensible par d'autres, et permet aussi d'identifier certains oublis dans le code qui pourraient occasionner un bug.
- **Jest et Enzyme** : ces bibliothèques nous permettent de faciliter l'écriture de tests unitaires, afin de pouvoir valider le fonctionnement des composants de l'application.
- **React** : bibliothèque javascript, cœur de l'application. Facilitant la création d'une application monopage, par la génération d'objets javascript dans un langage simple. Les composants se trouvent chargés dans un DOM virtuel, permettant la mise à jour de l'affichage selon leur sollicitation.
- **Sass** : préprocesseur css, qui offre donc une simplification dans l'établissement de règles css, la possibilité d'utiliser des variables en amont de la compilation du code.
- **React-Router-Dom** : bibliothèque de routage pour React. Permet la synchronisation de l'interface de l'application et ses informations avec l'url demandé par l'utilisateur.
- **Redux** : bibliothèque assurant la gestion des états de l'application.
- **Axios** : client HTTP, nous permettant d'effectuer des requêtes auprès de notre API.
- **Material UI** : bibliothèque de composants pour React, dans le style des interfaces de Google et personnalisables.

Les technologies du back

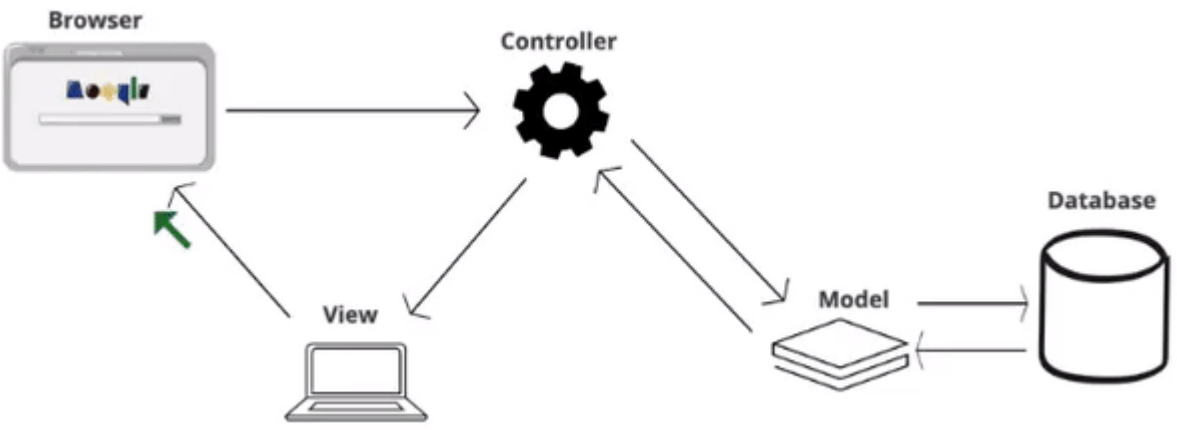
- **Composer** : gestionnaire de dépendances php, utilisé afin d'installer les dépendances nécessaires au fonctionnement de l'application, mais aussi lors de son développement.
- **Symfony** : ensemble de composants php et framework MVC, utile à la structuration de notre application et en facilitant le développement.
- **Doctrine** : ORM par défaut de Symfony, permettant la manipulation des données.
- **Twig** : moteur de template par défaut de Symfony, qui générera les pages html à renvoyer à l'utilisateur.
- **Lexik/jwt** : permet de mettre en place l'utilisation des tokens d'authentification.
- **Nelmio/cors** : permet de mettre en place les autorisations d'accès à l'API
- **Sensio/extra-bundel** : permet de configurer les Controllers à l'aide d'annotations.
- **Vich/uploader** : facilite l'upload de fichiers, en lien avec l'ORM.
- **MySQL** : Système de gestion de base de données.
- **Faker** : générateur de données aléatoires utilisé lors du développement.

L'architecture de l'application

L'utilisation de Redux sur la partie front-end nous a permis d'assurer une meilleure organisation du code, en découpant les différents états utiles au fonctionnement de l'application au sein de reducers, eux même regroupés dans un store. Des middlewares ont été développés en fonction des besoins de chaque composant, afin de gérer plus aisément les échanges d'informations entre le front-office et l'API.



La partie Back-end est quant à elle architecturée en MVC. Optimisant là aussi l'organisation du code, en séparant les rôles en plusieurs fichiers interagissant entre eux. Le Controller traite les actions demandées par l'utilisateur, le Model gère les interactions avec la base de données, et la View génère l'affichage à retourner à l'utilisateur. Nous obtenons donc un code structuré, compréhensible et aisé à maintenir.



La sécurité de l'application

Côté client, des fonctions de validation de champs de formulaires ont été incluses. Néanmoins, pouvant être contournées assez facilement, une deuxième couche de sécurité de ce même type est appliquée côté serveur.

Côté serveur, se retrouvent donc les validations des valeurs récupérées auprès des formulaires, ainsi que les préventions d'injection SQL incluses dans Doctrine.
L'accès à l'API est régulé par des règles Cors, vérifiant les en-têtes dans les requêtes HTTP qu' elle reçoit afin d'y autoriser ou non l'émetteur.
L'authentification d'un utilisateur est effectuée grâce à l'utilisation de tokens générés aléatoirement et stockés en cookie de session.

Autres services utilisés

Pour réaliser le déploiement de l'application, j'ai personnellement eu recours à un service d'hébergement. Cette solution, obtenue auprès de la société O2Switch, présente l'avantage d'offrir un serveur web déjà configuré, une interface de gestion plus visuelle qu'avec l'utilisation d'un terminal, et une documentation aidant à réaliser les différentes opérations que l'on souhaite effectuer.
Un sous domaine à par ailleurs été configuré afin de recevoir la partie back end du projet, suivi de la création de la base de données associée à un utilisateur.

VI. RÉALISATION DU PROJET

Présentation de l'équipe et répartition des rôles

L'équipe du projet *Boule de poils* se voit composée de cinq développeurs fullstack, chacun ayant suivi une spécialisation dans une technologie à l'issue de leur socle de formation commun. Au lancement du projet, et après concertation, chaque membre s'est vu attribuer un rôle à assurer pendant cette période de travail.

- **Julien Laurent** : Dev Symfony - Product Owner et Lead Dev Back

A l'initiative du projet, Julien fut chargé de prendre les décisions sur la direction que devait prendre l'application, ainsi que sur les choix à effectuer concernant l'élaboration de la partie Back-end

- **Mathilde Janssen** : Dev React - Scrum Master

Chaque journée se voyant débuter par une réunion permettant de faire le point sur l'avancement du projet et répartir les tâches, Mathilde était en charge de mener celles-ci et de consigner les différents sujets abordés.

- **Anthony Mayor** : Dev React - Lead Dev Front

Anthony eut pour rôle de prendre les décisions concernant l'élaboration de la partie Front-End.

- **Virginie Touzalin** : Dev Symfony - Git Master

En tant que référente Git, Virginie avait pour responsabilité de veiller au bon usage des conventions de versioning, d'assurer le bon déroulement des merge, mais aussi d'aider à résoudre les petites erreurs de branches ou de commit qui pouvaient survenir.

- **François-Louis Toussaint** : Dev React - Référent React

François-Louis, avait pour tâche de rechercher des dépendances utiles à l'application, et d'assurer la compréhension de leur documentation.

Organisation de travail

Nous avons organisé notre travail selon la méthode agile scrum, structurant la réalisation de l'application en quatre sprint d'environ une semaine chacun, tout en gardant une capacité d'adaptation en fonction de l'avancement du projet.

Les différentes tâches ont été planifiées et leur difficulté évaluée grâce à l'outil Trello, par lequel nous pouvions suivre leur état d'accomplissement, ainsi que se les répartir.

Nous nous sommes par la suite divisés en deux sous-équipes selon nos spécialisations, soit Julien et Virginie pour travailler sur le back-end, Mathilde, Anthony et François-Louis pour le front-end. Restant néanmoins en contact en cas de questionnement ou de besoin d'œuvrer en commun, tel lors de la connexion du front-office à l'API ou bien la personnalisation du css sur le back-office.

Nous avons choisi de garder les mêmes horaires de travail que nous respections lors de nos cours, soit 9h-12h30 et 13h30-17h.

Chaque journée commençait par un rapide entretien en commun, afin de faire le point sur le travail accompli la veille par chacun, évoquer les difficultés et solutions rencontrées, débattre de nouvelles idées ou prendre des décisions, et finalement se répartir les tâches quotidiennes.

Le reste de la journée s'orientait donc essentiellement sur le développement de fonctionnalités ou la correction de bugs, en solitaire mais aussi parfois en pair-programming selon la difficulté du travail à effectuer.

Programme des sprints

- **Sprint 0** : la phase de conception, soit une semaine de réunion continue entre l'équipe afin de mettre au point le projet. Rédaction du cahier des charges, mise au point de la base données, et établissement de tous les documents nécessaires à nous guider dans le développement de l'application.

- **Sprint 1** : côté front, l'intégration des différents composants fut réalisée lors de cette étape, en se basant sur les wireframes. Côté back, les éléments de la base de données furent créés, ainsi que l'API mise au point.

- **Sprint 2** : Durant cette étape, la connexion entre le front-office et l'API a été réalisée afin d'effectuer les transferts d'informations, et ainsi dynamiser les données. La mise en place de la sécurité du front fut aussi réalisée à cette occasion. L'équipe réalisant le back-end, s'est penchée quant à elle sur la réalisation du back-office assurant la gestion des données par l'utilisateur, ainsi que sur les fonctionnalités d'authentification.

- **Sprint 3** : lors de cette dernière semaine, nous avons mergé nos fonctionnalité sur une branche principale, résolvant au fur et à mesure les conflits engendrés. S'ensuivit alors des phases de test pour vérifier le bon fonctionnement de l'application, associées aux corrections de bugs nécessaires au parachèvement du projet. L'ultime action fut alors la mise en production de l'application.

Outils utilisés

- **Google docs** et **sheets** : outils de traitement de texte et de tableur en ligne, proposés par Google et permettant de travailler en équipe. Utilisés pour la rédaction des différents documents lors de la phase de conception du projet.
- **Slack** : plateforme nous ayant servi à communiquer ponctuellement par écrit au sein de l'équipe.
- **Discord** : logiciel nous ayant permis d'effectuer des réunions en communiquant à l'oral, et avec la possibilité de partager nos écrans.
- **Miro** : espace de tableau blanc collaboratif en ligne, ayant servi à créer l'arborescence du site.
- **Figma** : éditeur graphique vectoriel à vocation de prototypage, ayant permis l'élaboration des wireframes.
- **Adobe Illustrator** : logiciel de dessin vectoriel, utilisé pour la réalisation du logo et de la favicon du site.
- **Mocodo** : outil en ligne, servant à conceptualiser la base de données grâce à une schématisation.
- **Trello** : outil de gestion de projet en ligne, ayant listé les différentes tâches à accomplir selon le sprint en cours.
- **VS Code** : éditeur de code proposé par Microsoft, ayant servi tout au long de la phase de développement de l'application.
- **Git** : gestionnaire de version utilisée pour sauvegarder, compartimenter puis assembler les différentes fonctionnalités codées.
- **Insomnia** : client API, nous ayant servi afin de vérifier les endpoints ainsi que les informations transmises.
- **Filezilla** : client FTP, ayant permis l'upload des fichiers de l'application sur le serveur d'hébergement.

VII. RÉALISATIONS PERSONNELLES

Menu burger

Dans le but d'obtenir un site à l'affiche responsive et conçu en mobile first, nous avons souhaité mettre en place un menu burger. Mon collègue Anthony a réalisé l'intégration des éléments de la barre de navigation, j'ai par la suite pris la main pour mettre en place les règles css nécessaires à l'affichage et aux animations. Quelques ajouts de codes javascript ont aussi été apportés afin d'optimiser l'expérience utilisateur.

Ci dessous, le rendu du menu de navigation pour le mobile, fermé puis ouvert lors du click sur le bouton burger.

```
const NavBar = () => {
  const role = useSelector((state) => (state.Login.role));
  const dispatch = useDispatch();
  const connectedUser = useSelector((state) => (state.Login.tokenUserConnected));

  const [showLinks, setShowLinks] = useState(false);

  const handleShowLinks = () => {
    dispatch(emptyAnimalResults());
    dispatch(notSubmit());
    if (window.matchMedia("(max-width: 767px)").matches) {
      setShowLinks(!showLinks);
      console.log('coucou')
    }
  };

  const handleLogo = () => {
    dispatch(emptyAnimalResults());
    dispatch(notSubmit());
  };
```

Le hook state showLinks contient une valeur booléenne qui permet l'apparition et la disparition du menu sur mobile, en échangeant la classe de la balise <nav> par show-nav ou hide-nav, appliquant les règles css particulières à chacune.

La fonction handleShowLinks, est appelée au click sur un élément du menu. Excepté le bouton burger qui contrôle l'apparition et la disparition du menu, ils "renvoient" tous vers d'autres parties du site. Deux autres fonctions sont appelées au début, vidant le state des résultats d'animaux, et définissant le formulaire des animaux comme non soumis en guise de rafraichissement de page. Par la suite, une condition vérifie si la largeur d'affichage est inférieure à 767px, grâce à la méthode window.matchMedia, et effectue le changement de la valeur de showLinks si c'est le cas.

La fonction handleLogo aura quant à elle, uniquement le rôle de rafraîchir la page en vidant le state des résultats d'animaux, et définissant le formulaire des animaux comme non soumis.

```
  return (
    <nav className={`navbar ${showLinks ? 'show-nav' : 'hide-nav'}`}>
      <Link to="/" label="homepage"><img className="logo__link" src={logo} alt="logo" onClick=
{handleLogo}/></Link>
      <h1 className="navbar__title">Boule de poils</h1>
      <ul className="navbar__links">
        {connectedUser && (
          <li className="navbar__item slideInDown-1 " onClick={handleShowLinks}>
            <Link to="/favoris" className="navbar__link">
              Mes favoris
            </Link>
          </li>
        )}
        <li className="navbar__item slideInDown-2 " onClick={handleShowLinks}>
          <Link to="/#searchForm" className="navbar__link">
            Rechercher un animal
          </Link>
        </li>
        <li className="navbar__item slideInDown-3" onClick={handleShowLinks}>
          <Link to="/conseils" className="navbar__link">
            Conseils pour l'adoption
          </Link>
        </li>
        <li className="navbar__item slideInDown-4" onClick={handleShowLinks}>
          <Link to="/associations" className="navbar__link">
            Les associations
          </Link>
        </li>
        {(role !== 'ROLE_USER' && role !== null) && (
          <li className="navbar__item slideInDown-5" onClick={handleShowLinks}>
            <Link to="/login" className="navbar__link">
              Admin/compte user
            </Link>
          </li>
        )}
        {connectedUser ? (
          <li className="navbar__item slideInDown-6" onClick={handleShowLinks}>
            <Link
              to="/"
              className="navbar__link items"
              onClick={() => (dispatch(logout()))}
            >
              Logout
            </Link>
          </li>
        ) : (
          <li className="navbar__item slideInDown-6" onClick={handleShowLinks}>
            <Link
              to="/login"
              className="navbar__link items"
            >
              Login
            </Link>
          </li>
        )}
      </ul>
      <button type="button" className="navbar__burger" onClick={handleShowLinks}>
        <span className="burger-bar" />
      </button>
    </nav>
  );
};

export default NavBar;
```

Chaque lien dans la navbar à une classe permettant de l'identifier, afin d'enchainer l'animation de leur apparition.

Certains éléments voient leur présence conditionnée par la connexion et le rôle de l'utilisateur.

```css
/* toggle menu */
.show-nav .navbar__burger {
  position: fixed;
  top:1rem;
  right:1rem;
}
.navbar__burger:hover {
  cursor: pointer;
}
.burger-bar,
.burger-bar::before,
.burger-bar::after {
  display: block;
  width: 40px;
  height: 3px;
  position: relative;
  border-radius: 3px;
  background: #FFF;
  transition: all .5s ease-in-out;
}
.burger-bar::before,
.burger-bar::after {
  content: "";
  position: absolute;
  left: 0;
}
.burger-bar::before {
  transform: translateY(-12px)
}
.burger-bar::after {
  transform: translateY(12px)
}
/* burger button animation */
.show-nav .burger-bar {
  width:0;
  background: transparent;
}
.show-nav .burger-bar::before {
  transform: rotate(45deg);
}
.show-nav .burger-bar::after {
  transform: rotate(-45deg);
}
```

Dans le css, la balise <button> est masquée pour être remplacée par le burger.

On définit un style de curseur lorsqu'on pointe dessus.

Les trois barres du burger sont créées en css, à partir d'une unique barre on ajoute les deux autres à l'aide des pseudo-classes ::before et ::after.

Pour l'animation, lorsque le menu doit apparaître, la barre principale du milieu est masquée, et les deux autres effectuent une rotation afin d'obtenir une forme de croix, invitant l'utilisateur à la fermeture du menu.

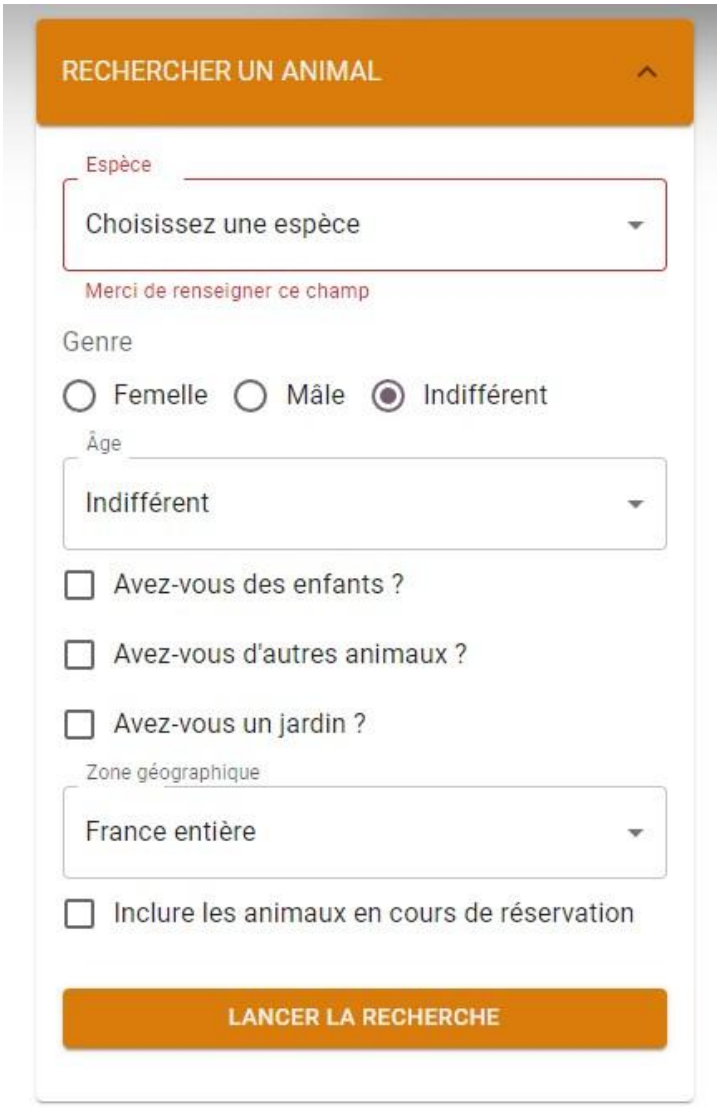
```
  /* Bonus - Animations */
   .navbar__item {
     transform: translateY(100vh);
   }
   .show-nav .navbar__item {
     transform: translateY(0);
   }
   .show-nav .slideInDown-1 {
     transition: all 1s ease-out;
   }
   .show-nav .slideInDown-2 {
     transition: all 1.1s ease-out;
   }
   .show-nav .slideInDown-3 {
     transition: all 1.2s ease-out;
   }
   .show-nav .slideInDown-4 {
     transition: all 1.3s ease-out;
   }
   .show-nav .slideInDown-5 {
     transition: all 1.4s ease-out;
   }
   .show-nav .slideInDown-6 {
     transition: all 1.5s ease-out;
   }
}
```

L'apparition du menu va s'effectuer progressivement en recouvrant tout l'écran de la droite vers la gauche.

Les liens de navigation vont alors entrer en scène par le bas, les uns à la suite des autres avec un petit délai entre chaque.

Formulaire de recherche d'un animal

La fonctionnalité principale de notre application, la toute première à avoir été codée durant la phase de développement, et dont je me suis chargé.
Elle va permettre à l'utilisateur de remplir un formulaire avec des critères qui permettront de lui présenter une liste d'animaux filtrée selon ses choix.



Nous voulions que notre fonctionnalité de recherche se présente sous la forme d'un bouton call to action, qui déploie le formulaire lorsque l'on clique dessus. Souhaitant utiliser Material Ui pour gagner un peu de temps, j'ai quand même dû en dépenser dans la compréhension de sa documentation, ainsi qu'effectuer quelques essais à l'aide de CodeSandBox. J'ai finalement réussi à combiner les composants Button et Accordion de Material UI pour obtenir le résultat souhaité.

```
useEffect(() => {
    dispatch(fetchSpecies());
    dispatch(fetchDepartments());
  }, []);
```

Le hook d'effet useEffect, permet d'exécuter une action selon l'affichage du composant. Dans notre cas, au premier rendu de notre formulaire, deux requêtes vont successivement être envoyées vers notre API afin de récupérer la liste des espèces et des départements existants dans notre base de données, afin de dynamiser les inputs leur correspondant.

```
{locError
    ? (
      <FormControl fullWidth error>
        <InputLabel id="demo-simple-select-label" sx={{ mt: 1 }}>
          Zone géographique
        </InputLabel>
        <Select
          labelId="demo-simple-select-label"
          id="demo-simple-select"
          name="department"
          value={locValue}
          label="species"
          onChange={handleChangeLoc}
          sx={{ mt: 2 }}
        >
          <ListSubheader>Pays</ListSubheader>
          <MenuItem value={0}>
            <em>France entière</em>
          </MenuItem>
          <ListSubheader>Départements</ListSubheader>
          {departments.map((item) => (
            <MenuItem key={item.id} value={item.id}>{item.name}</MenuItem>
          ))}
        </Select>
        <FormHelperText>Merci de renseigner ce champ</FormHelperText>
      </FormControl>
    )
    : (
      <FormControl fullWidth>
        <InputLabel id="demo-simple-select-label" sx={{ mt: 1 }}>
          Zone géographique
        </InputLabel>
        <Select
          labelId="demo-simple-select-label"
          id="demo-simple-select"
          name="department"
          value={locValue}
          label="species"
          onChange={handleChangeLoc}
          sx={{ mt: 2 }}
        >
          <ListSubheader>Pays</ListSubheader>
          <MenuItem value={0}>
            <em>France entière</em>
          </MenuItem>
          <ListSubheader>Départements</ListSubheader>
          {departments.map((item) => (
            <MenuItem key={item.id} value={item.id}>{item.name}</MenuItem>
          ))}
        </Select>
      </FormControl>
    )}
```

Chaque champ du formulaire est donc créé à l'aide de composants issus de Material UI.

Il y a aussi une gestion des erreurs. Ici, locError est une propriété du state du formulaire, contenant une valeur booléenne. Selon cette valeur le composant affiché diffère, s'il y a erreur, le contour du champ deviendra rouge.

Ce champ de localisation étant un select, les données en son sein ont été dynamisées, en utilisant la méthode .map sur la liste des départements récupérée auprès de l'API.

```
const locValidate = (locVerify) => {
    if (locVerify >= 0 && locVerify < departments.length) {
      const action = changeLocError(locError, false);
      dispatch(action);
    }
    else {
      const action = changeLocError(locError, true);
      dispatch(action);
    }
  };

  const handleChangeLoc = (event) => {
    const { value: inputValue, name } = event.target;
    const action = changeLocField(name, parseInt(inputValue, 10));
    dispatch(action);
    locValidate(inputValue);
  };
```

```
import {
  CHANGE_AGE_FIELD,
  CHANGE_GENDER_FIELD,
  CHANGE_SPECIES_FIELD,
  CHANGE_CHILD_FIELD,
  CHANGE_OTHERS_FIELD,
  CHANGE_GARDEN_FIELD,
  CHANGE_LOC_FIELD,
  CHANGE_STATUS_FIELD,
} from 'src/actions/formActions';

const initialState = {
  species: 999,
  gender: 2,
  age: 4,
  childCompatibility: 0,
  otherAnimalCompatibility: 0,
  gardenNeeded: 0,
  status: 0,
  department: 0,
  animalResults: [],
  displayProfile: 0,
  favorites: [],
  showContact: false,
  formSubmit: false,

};
```

Chaque champ possède aussi une fonction permettant de récupérer la valeur qui y est saisie ou choisie, et ainsi de la sauvegarder dans le state associé dans le reducer. Ici, lorsque l'utilisateur choisit un département, l'action changeLocField est exécutée (la méthode parseInt permettant de convertir la valeur de entier à chaîne de caractère) et modifie donc la valeur contenue dans le state.

Certains champs nécessitent une validation, ce pour quoi une fonction de validation peut être appelée afin de vérifier que la valeur entrée correspond bien aux attentes de l'application. Pour la localisation, ce qui est censé être un index de département est comparé avec le nombre de départements présents listés dans le tableau récupéré auprès de l'API. S'il n'y a pas correspondance, la soumission du formulaire sera bloquée et la valeur de locError passera à vraie, occasionnant l'affichage d'une bordure rouge sur l'élément input, invitant l'utilisateur à revoir sa saisie.

```
const handleSubmit = (event) => {
    event.preventDefault();
    // eslint-disable-next-line no-unused-vars
    const data = new FormData(event.currentTarget);
    if (speciesError === false && ageError === false && locError === false) {
      dispatch(formSubmit());
    }
  };
```

Lorsque l'utilisateur clique sur le bouton de soumission du formulaire, la fonction handleSubmit est alors appelée. Cette dernière vérifiera qu'il n'y pas d'erreurs dans les champs nécessitant une validation, avant d'exécuter l'action qui lancera la requête vers notre API, sinon rien ne se passe.

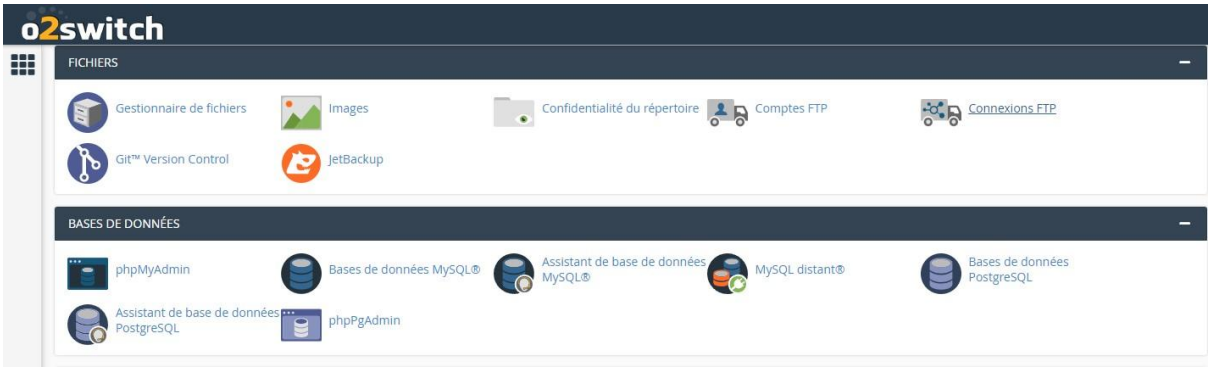
```
const animalSearchedMiddleware = (store) => (next) => (action) => {
  switch (action.type) {
    case FETCH_ANIMALS:
      axios.post('http://localhost:8081/api/animal/form', {
        "species": store.getState().SearchedAnimals.species,
        "gender": store.getState().SearchedAnimals.gender,
        "age": store.getState().SearchedAnimals.age,
        "childCompatibility": store.getState().SearchedAnimals.childCompatibility,
        "other_animal_compatibility": store.getState().SearchedAnimals.otherAnimalCompatibility,
        "garden_needed": store.getState().SearchedAnimals.gardenNeeded,
        "status": store.getState().SearchedAnimals.status,
        "department": store.getState().SearchedAnimals.department,
      } )
        .then((response) => {
          // handle success
          console.log(response.data);
          response.data.length === 0 ? store.dispatch(saveFetchedAnimals('void')) :
store.dispatch(saveFetchedAnimals(response.data));
        })
        .catch((error) => {
          // handle error
          console.log(error);
          store.dispatch(saveFetchedAnimals('error'));
        });

      break;
```

Depuis le middleware dédié, à l'aide du client HTTP Axios, une requête va alors être envoyée à notre API, lui transmettant les informations récupérées dans le state des animaux. Selon la réponse reçue, un tableau d'objets contenant les animaux recherchés sera sauvegardée dans le même state, donnant lieu à l'affichage des composants des profils d'animaux. S'il aucun animal n'a été trouvé, la valeur 'void' sera ajoutée dans ce state, permettant l'affichage d'un composant spécifique pour prévenir l'utilisateur, de même en cas d'une réponse d'erreur de la part de l'API.

Déploiement

Ayant donc souscrit à un hébergement auprès de la société O2Switch, j'ai pu effectuer la mise en production de l'application sur ce serveur.



Le front office fut aisément déployé. A l'issue de la commande yarn build dans le dossier le contenant, une compilation est effectuée par webpack, créant un nouveau dossier dist au sein duquel se trouvent les fichiers à transférer sur le serveur. J'ai pu m'aider pour celà du logiciel Filezilla, permettant d'utiliser le protocole FTP.

Le fichier javascript principal a alors été modifié à l'aide de l'interface en ligne afin de configurer l'url des appels à l'API de l'application vers le sous domaine du back office.
En dernier lieu, j'ai créé un fichier htaccess dans le répertoire du front office afin de configurer le fonctionnement des différentes url, étant donné qu'il n'y a de base qu'un seul fichier html et que les routes sont gérées par javascript.

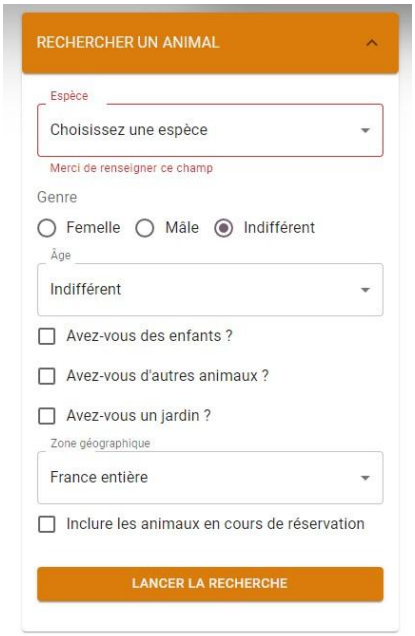
Pour le back office, et suivant les conseils prodigués par la documentation, une connexion ssh a été configurée entre le serveur et Github afin d'y récupérer le repository. A l'aide du terminal de commande en ligne, ce dernier a alors été chargé au sein du sous-domaine. Les dépendances ont alors été installées avec la commande composer install, ainsi que l'ajout ici aussi d'un fichier htaccess avec la commande symfony require/apache-pack. Par la suite, le fichier env a été modifié directement via l'interface en ligne, afin de configurer le mode production, les cors et l'accès à la base de données sur le serveur. Achevant ce déploiement, la commande demandant à Doctrine d'effectuer la création des tables et des données principales fut lancée.

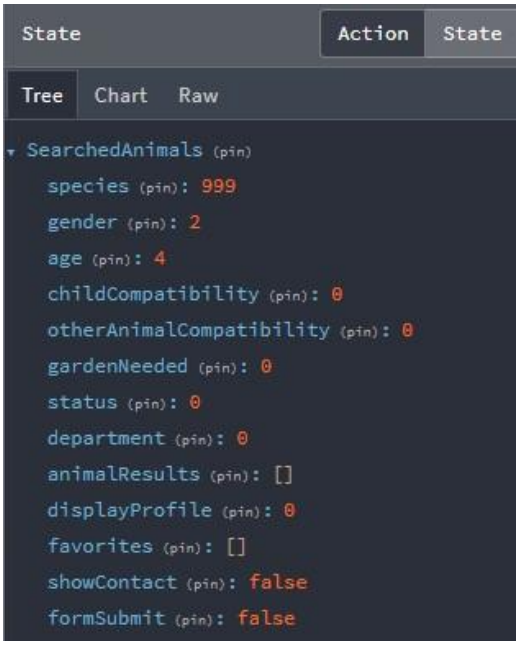
VIII. PRÉSENTATION DU JEU D'ESSAI DE LA FONCTIONNALITÉ PRINCIPALE

Recherche d'un animal

Voici le résultat escompté lors d'utilisation de la fonctionnalité qui est au cœur de notre application, en effectuant son test.

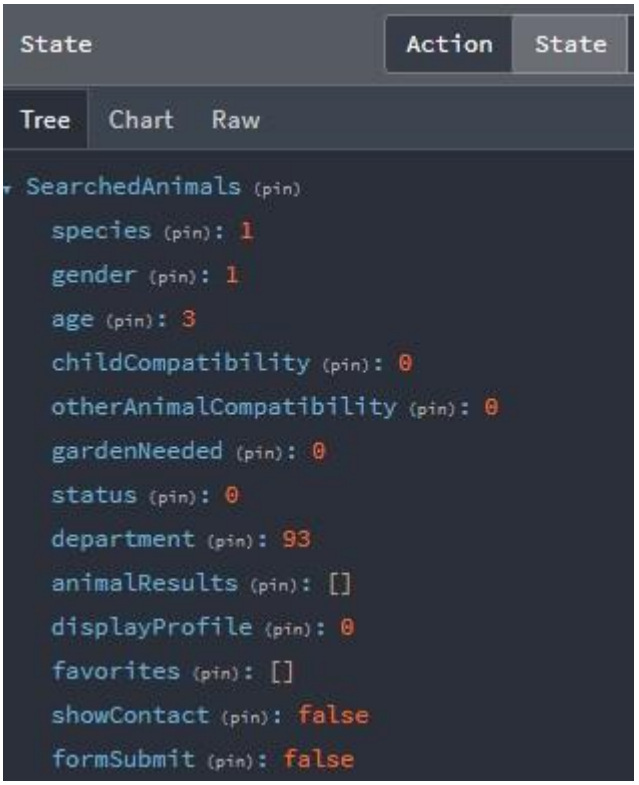
Au click sur le bouton call to action, le formulaire de recherche se déroule, et nous pouvons observer les valeurs de base dans le state du formulaire à l'aide de l'outil de développement Redux dans le navigateur :



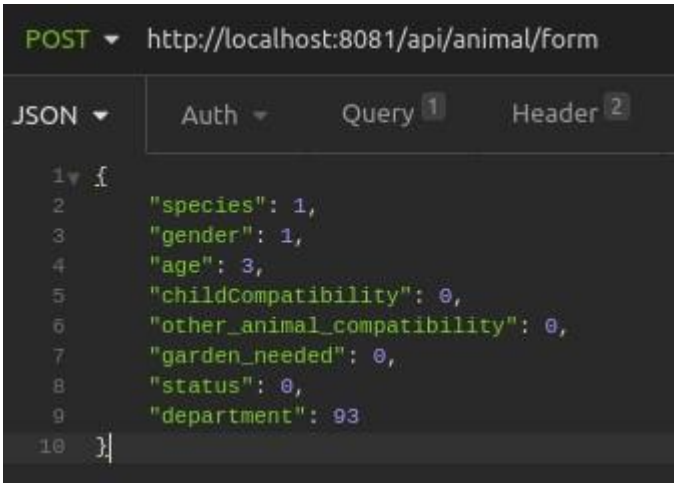


Recherchons donc un chat ! Mâle, de plus de 10 ans, situé dans les Hauts de Seine.

Nous pouvons observer qu'à chaque sélection d'un critère, la propriété associée dans le state change de valeur.



Lorsque l'on clique par la suite sur le bouton "Lancer la recherche", une requête va être envoyée à l'API avec les valeurs obtenues auprès du formulaire soit en ce cas :



Le temps que l'API nous retourne la réponse des données demandées, un loader rotatif va apparaître à l'écran :

Une fois la réponse de l'Api chargé dans la propriété du state recevant le tableau d'objet des animaux recherchés, les cartes de profil s'affichent. En ce cas il n'y a qu'un animal, s'il y en avait plusieurs, les petites flèches permettraient de passer d'un profil à l'autre pour les consulter.

IX. VEILLE TECHNOLOGIQUE ET VULNÉRABILITÉ DE SÉCURITÉ

Validation des valeurs saisies dans les champs de formulaire

M'étant occupé du développement des formulaires de l'application sur la partie front-end, j'ai dû effectuer quelques recherches afin d'assurer un premier niveau de sécurité.

Mes recherches ont donc porté sur l'utilisation des expressions régulières, j'ai pu pour cela, me référer principalement au site de la fondation pour la sécurité du web OWASP.

https://owasp.org/www-community/OWASP_Validation_Regex_Repository

J'y ai donc trouvé une liste des principales regex à mettre en place afin d'assurer la validation des champs.

Selon le champ en question il va être nécessaire de vérifier qu'il contient bien un nombre de caractères défini, uniquement des chiffres, un caractère spécial comme le @, ou encore qu'il ne contienne pas ceux permettant d'exécuter une attaque xss par exemple.

Une faille xss permettrait à un utilisateur malveillant d'injecter du code javascript, afin d'exécuter des actions qui ne seraient pas voulues par les administrateurs du site.

Exemple d'expression régulière permettant d'assainir un champ de texte :

```
/^[a-zA-Z0-9 .-]+$/
```

Elle va permettre d'autoriser uniquement la saisie de caractères alphanumériques, en minuscule et majuscule, et ce en nombre illimité.

X. RESOLUTIONS DE PROBLEMES

Au cours de l'élaboration du projet, J'ai pu rencontrer quelques difficultés, ou nécessités de revenir sur certaines notions, m'amenant à effectuer des recherches sur les sujets suivants :

Conventions de nommage Git

Problématique : Au démarrage du projet, nous avons choisi de suivre des conventions de nommage pour nos branches et commits afin que chaque membre de l'équipe puisse aisément les identifier. N'ayant jamais mis en œuvre une telle pratique, nous avons dû nous renseigner à ce propos.

Recherche effectuée : "git name convention"

J'ai filtré un peu les résultats en les parcourant un à un avant de choisir celui qui me paraissait le plus complet.

https://dev.to/couchcamote/git-branching-name-convention-cch

Les media queries

Problématique : L'expérience de l'utilisation des média queries se trouva quelque peu succincte et lointaine. Un petit rafraichissement de mémoire s'imposa afin de réaliser la partie responsive de l'affiche de l'application.

Recherche effectuée : "media queries"

Après la lecture de quelques résultats de recherche, j'ai choisis de me concentrer sur une documentation qui me semblait la plus claire, et surtout présentant des exemples visuels

https://www.w3schools.com/css/css_rwd_mediaqueries.asp

Validation d'un fichier à uploader en javascript

Problématique : Durant le développement d'une fonctionnalité d'upload et recadrage de photo, j'ai cherché à ajouter une couche de sécurité afin d'interdire certains formats de fichiers, comme le svg qui peut représenter un risque s'il comporte du code malveillant. HTML le permet dans la configuration de l'input d'upload, mais cela reste trop facilement contournable, d'où la volonté d'ajouter une fonction en javascript.

Recherche effectuée : "how to validate image input js"

Une fois les premiers résultats de recherche analysés, j'ai fait le choix de m'orienter vers une page regroupant des tutoriels à ce sujet, principalement présentés sous forme de code simple à analyser.

https://www.geeksforgeeks.org/file-type-validation-while-uploading-it-using-javascript/

Traduction d'un extrait du résultat de la recherche portant sur les conventions de nommage Git

"Working on a big company with projects that could scale from a one-man team then suddenly to a 20 developers team, having a manageable code repository is a need. Most *Proof of Concept* projects start with a repository with all changes being applied directly to the master branch. Elevating one into a proper big scale repository is a common path being taken by new developers when their small-scale project is suddenly noticed.
On my end, having to work on dozens of different projects like these, I came up with this branch naming convention.
I separated these branches into two categories:
Code Flow Branches
These branches which we expect to be permanently available on the repository follow the flow of code changes starting from development until the production.

- Development (*dev*)
  All new features and bug fixes should be brought to the development branch. Resolving developer codes conflicts should be done as early as here.
- QA/Test (*test*)
  Contains all codes ready for QA testing.
- Staging (*staging* , Optional)
  It contains tested features that the stakeholders wanted to be available either for a demo or a proposal before elevating into the production. Decisions are made here if a feature should be finally brought to the production code.
- Master (*master*)
  The production branch, if the repository is published, this is the default branch being presented.

Except for Hotfixes, we want our codes to follow a one-way merge starting from development > test > staging > production."

En travaillant dans une grosse entreprise pouvant passer soudainement d'une équipe de un à vingt développeurs, avoir un dépôt de code maintenable est une nécessité. La plupart des projets "Preuve de concept" débutent avec un dépôt contenant toutes les modifications directement sur la branche principale. Il est courant que les nouveaux développeurs intègrent le dépôt d'un petit projet à un plus gros lorsque celui-ci est remarqué.
Pour ma part, ayant travaillé sur des dizaines de projets différents comme cela, j'ai instauré cette convention de nommage de branche.
J'ai séparé ces branches en deux catégories :

- Développement (dev)
  Toutes les nouvelles fonctionnalités et les corrections de bugs devraient se trouver dans la branche de développement. Les résolutions de conflits de codes devraient être effectuées dès que possible.
- CQ/Test (test)
  Contient tous les codes prêts pour les tests de contrôle qualité.
- En attente (staging, Optionnel)
  Contient les fonctionnalités testées que toutes les parties prenantes du projet souhaitent avoir à disposition pour effectuer une démonstration ou les proposer avant d'être intégré au code de production. Les décisions d'intégrer une fonctionnalité au code de production sont donc prises ici.
- Principale (master)
  La branche de production, si le dépôt est publié, c'est cette branche par défaut qui est utilisée.

A l'exception des correctifs, nous voulons que nos codes suivent le parcours d'intégration suivant : développement->test->en attente->production.

XI. CONCLUSION

La réalisation de ce projet fut réellement enrichissante, me permettant d'asseoir les connaissances accumulées durant la formation. Je dois bien avouer qu'après cinq mois de cours intensifs, il me tardait de passer à la pratique sur un projet complet. Mais aussi au niveau humain, j'appréhendais un peu le travail en équipe, qui s'est révélé tout à fait agréable, l'ambiance étant bonne et chacun étant à l'écoute et disponible en cas de difficulté d'un autre membre.

Depuis la fin de formation, je continue de découvrir certaines notions et technologies par moi même, symfony par exemple, pour obtenir une meilleure compréhension de la partie back-end du projet mais aussi afin d'échafauder des projets personnels. Et côté front-end, j'envisage d'aborder Typescript, la bibliothèque javascript Greensock, ainsi que la thématique de l'accessibilité au web.

Mon projet pour l'avenir est de poursuivre mes études dans cette voie, en préparant le Titre Professionnel Concepteur-Développeur d'Application, dans le cadre d'une alternance en contrat de professionnalisation, toujours au sein de l'école O'clock, et bien sûr avec une entreprise qu'il me faut désormais dénicher.