Arcadia

Site web d'un zooparc (vitrine et gestion)

Projet réalisé dans le cadre de la présentation au

Titre Professionnel Développeur Web et Web Mobile

présenté par

Martin Terrier

Studi – Promotion septembre-octobre 2024

Sommaire

Introduction

J'ai longtemps vécu loin du code. Après une classe préparatoire qui m'a emmenée vers une école de commerce, j'ai exercé quelques années des fonctions dans le marketing communautaire, d'abord dans la veille puis dans la gestion de communauté. Ces deux activités étaient étroitement en rapport avec l'univers d'Internet, et pourtant complètement déconnectées de ses rouages. Le web était un univers à explorer, mais je n'avais que des notions très vagues de son fonctionnement, comme finalement des lois de la thermodynamique qui régissent le monde physique ou de l'architecture de l'immeuble dans lequel j'habite.

Vers la fin des années 2000, le marketing ne répondant plus à mes aspirations, je me suis réorienté. Toujours pas vers le code, mais vers la traduction. S'il m'est arrivé de traduire du contenu pour des sites web ou des applications, me rapprochant un peu des coulisses d'Internet, mon activité en restait malgré tout assez éloignée. Toutefois, à défaut de code, j'ai travaillé en tant que traducteur technique avec le langage, et je m'en suis servi pour restituer aussi efficacement que possible la logique des textes qui m'étaient confiés. En cela, je pense que j'étais finalement moins éloigné qu'on pourrait le croire de la logique du développement web.

J'ai exercé le métier de traducteur pendant près d'une quinzaine d'années, avec plaisir dans l'ensemble. Mon principal problème sur le long terme est que cette activité était essentiellement solitaire, le marché étant essentiellement tourné vers le *freelancing*. Au fil du temps, cette solitude m'a pesé de plus en plus, et avec la période chaotique du COVID couplée à une charge très lourde de travail, j'ai fait un burnout en2022. C'est alors que s'est présentée à moi une opportunité inespérée de reconversion vers le développement informatique, en alternance.

La période n'était pas idéale pour reprendre un apprentissage à zéro : ma deuxième fille venait tout juste de naître. Mais la curiosité, le goût du défi et l'envie de découvrir l'envers d'un monde numérique qui, en vingt ans, avait envahi le monde analogique, m'ont poussé à accepter. Aujourd'hui, je suis convaincu d'avoir fait le bon choix, puisque mon alternance a débouché sur une embauche qui va à son tour entraîner un déménagement : le nouveau départ est total.

Compétences du référentiel couvertes par le projet

Le site web d'Arcadia est un projet développé avec le *framework* NestJS 10.3 et l'ORM *(Object Relational Mapping)* TypeORM 0.3.20 pour sa partie back-end, et le *framework* CSS Bootstrap 5.3 pour sa partie front-end.

Un *framework* – « cadre de travail » en français - est un ensemble de composants logiciels structurels qui permet aux développeurs d'être plus efficaces dans leur développement. Il offre une architecture et des composants prêts à l'emploi et réutilisables.

ACTIVITÉ TYPE N° 1 : Développer la partie front-end d'une application web ou web mobile sécurisée

Maquetter des interfaces utilisateur web ou web mobile

En amont de mon projet, j'ai synthétisé les besoins du client et étudié les *user stories* pour réaliser le diagramme de cas d'utilisation de l'application, qui m'a permis de définir sa structure et la liste des pages à développer. J'ai aussi défini grâce au site web coolors.co une palette de couleurs adaptée aux consignes du client sur la charte graphique. J'ai réalisé sur Figma des wireframes pour les versions mobile des principales pages, que j'ai ensuite adaptées en version desktop, puis déclinées sous forme de maquette pour visualiser l'apparence de mon application.

Réaliser des interfaces utilisateur statiques web ou web mobile

J'ai ensuite réalisé une version *mobile first* de l'interface statique du site, même si celle-ci intégrait déjà des éléments dynamiques du fait de l'utilisation de composants Bootstrap.

Développer la partie dynamique des interfaces utilisateur web ou web mobile

Une fois l'interface statique réalisée, j'ai utilisé Javascript sans framework pour la dynamiser en gérant par exemple des vérifications en temps réel des informations saisies pour déposer un commentaire (pour les visiteurs) ou se connecter à son compte (pour le personnel du zoo). Cette étape a aussi été inextricablement liée avec l'ajout d'appels à la partie back-end de l'application. Ceux-ci ont été nécessaires pour gérer authentification et autorisation ; pour construire le contenu de certaines pages (Services, Habitats, Avis et les espaces utilisateurs) à partir d'informations stockées en base de données ; et pour saisir dans des formulaires des informations devant être stockées dans la base. Ces trois besoins étaient souvent entremêlés, par exemple quand un formulaire contenant une sélection d'éléments

extraits de la base de données envoyait à celle-ci de nouvelles données à enregistrer, l'opération nécessitant naturellement une autorisation.

ACTIVITÉ TYPE N° 2 : Développer la partie back-end d'une application web ou web mobile sécurisée

Mettre en place une base de données relationnelle

En partant des user stories, j'ai déterminé la structure de la base de données qu'utiliserait mon application et construit son modèle physique de données dans le gestionnaire de base de données PGAdmin. J'ai ensuite écrit la requête SQL correspondante, permettant de créer une base de données PostgreSQL et d'y injecter le minimum de données nécessaires au développement et au test de fonctionnalités. Cette requête m'a été utile lors de réinitialisations de la base en cours de projet, mais aussi lors de changements de poste et quand je suis changé de gestionnaire de base de données, passant de PGAdmi à DBeaver.

Développer des composants d'accès aux données SQL et NoSQL

La couche d'accès aux données de mon application utilise TypeORM. J'ai créé les *Entity* (entités), et pour certaines des *Repository* (répertoires) personnalisés, correspondant aux tables de ma base de données. J'ai ensuite défini une variable d'environnement DATABASE_URL, stockée dans le fichier .env à la racine de mon application back-end, permettant à mon application de se connecter à ma base de données.

Développer des composants métier coté serveur

J'ai développé le back-end comme une application Node.js en utilisant le framework NestJS. J'ai donc suivi l'architecture implémentée par ce framework. J'ai créé un module pour chaque fonctionnalité générale, puis pour chaque module un service et un controller. Le service contient les différentes méthodes nécessaires pour manipuler les données correspondantes, tandis que le controller définit les routes grâce auxquelles le front-end communique avec le back-end. C'est aussi dans le controller que sont définis quels utilisateurs peuvent accéder à une méthode, en fonction du rôle qu'ils possèdent, et que la validation des données reçues du front-end se fait grâce aux *pipes* fournies par NestJS ou à la définition de *Data Transfer Objects* (DTO, objets de transfert de donénes) et à l'usage de la bibliothèque class-validator .

Résumé du projet

Le cadre de ce projet a été fourni par Studi, l'organisme dont j'ai suivi la formation. Il s'agit de développer une application web pour le compte de l'entreprise fictive Arcadia, dont l'activité est la gestion d'un zoo.

Arcadia est un zoo situé en France près de la forêt de Brocéliande, en Bretagne depuis 1960. Ils possèdent tout un panel d'animaux, réparti par habitat (savane, jungle, marais) et font extrêmement attention à leurs santés. Chaque jour, plusieurs vétérinaires viennent afin d'effectuer les contrôles sur chaque animal avant l'ouverture du zoo afin de s'assurer que tout se passe bien, de même, toute la nourriture donnée est calculée afin d'avoir le bon grammage (qui est précisé dans le rapport du vétérinaire). De plus, le site est entièrement indépendant au niveau des énergies. Le zoo, se porte très bien financièrement, les animaux sont heureux. Cela fait la fierté de son directeur, José, qui a de grandes ambitions. A ce jour, l'informatique et lui ça fait deux, mais, il a envie d'une application web qui permettrait aux visiteurs de visualiser les animaux, leurs états et visualiser les services ainsi que les horaires du zoo. C'est pourquoi il se tourne vers un prestataire extérieur pour développer cette application et augmenter ainsi la notoriété et l'image de marque du zoo.

Cahier des charges

Objectifs

L'application Arcadia doit répondre à deux objectifs distincts. D'une part, elle doit présenter aux internautes potentiellement intéressés une vitrine du zooparc. D'autre part, elle doit permettre aux différentes personnes intervenant dans l'activité quotidienne du zoo (employés et vétérinaires) de se communiquer des informations utiles à celle-ci. De plus, José précise qu'il souhaite que les couleurs et le thème de l'application évoquent l'écologie, afin que les visiteurs ressentent l'importance de cette valeur pour l'équipe du zoo lors de leur consultation du site.

User Stories

US 1 : Page d'accueil

Utilisateur concerné : Visiteur

La page d’accueil doit comporter :

- Présentation du zoo en y incorporant quelques images
- Mentionnez les différents habitats, services ainsi que les animaux que possède le zoo
- Les avis du Zoo

US 2 : Menu de l’application

Utilisateur concerné : Visiteur

Le menu est essentiel à l’application, il permet de faciliter la navigation et de fluidifier le trafic, ainsi, il doit composer au minimum :

- Retour vers la page d’accueil
- Accès à tous les services
- Accès à tous les habitats
- Connexion (uniquement pour les vétérinaires, employés et administrateur)
- Contact

US 3 : vue globale de tous les services

Utilisateur concerné : Visiteur

Une vue globale est nécessaire afin de proposer une interface simple et récapitulative de tous les services que propose le parc. Pour le moment, le parc dispose de plusieurs services : restauration, visite des habitats avec un guide (gratuit), visite du zoo en petit train. Les services doivent être modifiables par les employés, tandis que l’administrateur pourra en supprimer ou en créer de nouveaux. Un service dispose des caractéristiques suivantes :

- Un nom
- Une description

US 4 : vue globale des habitats

Utilisateur concerné : Visiteur

Cette page, doit mentionner tous les habitats que propose le zoo et les animaux associés. José vous propose une vue tout d'abord de tous les habitats (en n'affichant que l'image et le nom), puis, au clic sur celui-ci, on affiche le détail avec les animaux ainsi que la description.
Un habitat est caractérisé par :

- Un nom
- Une image
- Une description de l'habitat
- Une liste d'animaux

Un animal est caractérisé par :

- Un nom
- Une espèce
- Une image
- Un habitat où il est affecté

Le vétérinaire, passe régulièrement et saisie des informations depuis son espace sur un animal donné en mentionnant :

- L'état de l'animal
- La nourriture proposée
- Le grammage de la nourriture
- La date de son passage
- Le détail de l'état de l'animal (information facultative)

Il est très important de mentionner l'état de l'animal sur cette page. Enfin, au clic sur un animal de l'habitat, le visiteur doit visualiser ses propriétés mais aussi l'avis du vétérinaire.

US 5 : Avis

Utilisateur concerné : Visiteur

Un visiteur, peut laisser un commentaire s'il le désire. Ce commentaire, contiendra juste un pseudo ainsi qu'un champ "avis" texte. Cet avis sera ensuite soumis à validation par

l'employé. Depuis l'espace employé, il pourra ensuite autoriser ou non l'avis à apparaitre sur la page des avis. La soumission d'un avis se fait par cette même page.

US 6 : Espace Administrateur

Utilisateur concerné : Administrateur

L'administrateur peut créer un compte de type "employé" et "vétérinaire", il doit pour cela, fournir un courriel (qui sera l'username) ainsi qu'un mot de passe. L'utilisateur devra se rapprocher de l'administrateur afin d'obtenir son mot de passe. Il peut également modifier les habitat et animaux du zoo (création, mise à jour et suppression pour ces derniers). Ensuite, l'espace administrateur doit avoir un emplacement avec tous les comptes rendus du vétérinaire, des filtres sont à positionner afin de pouvoir trier et filtrer les comptes rendus sur un animal ou une date. Enfin, il doit avoir un Dashboard lui montrant le nombre de consultation par animal (ce besoin est décrit en US 11). José précise que vous devez lui créer ce compte et qu'il ne doit pas être possible de créer un compte Administrateur depuis l'application.

US 7 : Espace Employé

Utilisateur concerné : Employé

Un employé depuis son espace peut valider un avis de visiteur ou l'invalider. Comme il va donner leur alimentation quotidienne aux animaux, il devra depuis son espace pouvoir sélectionner un animal et lui ajouter une consommation de nourriture. Pour cela il donnera la date et l'heure du repas, puis la nourriture donnée ainsi que la quantité.

US 8 : Espace Vétérinaire

Utilisateur concerné : Vétérinaire

Un vétérinaire passe quotidiennement dans le zoo, ainsi, depuis son espace, il remplira les comptes rendus par animaux. Le détail est disponible en US 4. Le vétérinaire voit également

sur son espace et par animal, tout ce que l'animal a pu manger via la saisie de l'employé sur son propre espace.

US 9 : Connexion

Utilisateur concerné : Administrateur, Vétérinaire, Employé

Seul une personne de type administrateur, vétérinaire ou employé peut se connecter. Pour se connecter, il devra juste saisir son username (mail) suivi de son mot de passe.

US 10 : Contact

Utilisateur concerné : Visiteur

Un visiteur peut contacter le zoo s'il le souhaite, pour cela, il devra accéder à la page contact depuis le menu applicatif. Pour donner suite à cela, il aura accès à un formulaire qui va lui demander un titre, une description ainsi que son mail afin qu'il puisse obtenir une réponse. Pour donner suite à cet envoie, la demande est envoyée par mail au zoo. L'employé peut alors répondre lui-même à la demande directement par mail.

US 11 : Statistique sur la consultation des animaux

Utilisateur concerné : Visiteur

Un visiteur, quand il cliquera sur l'animal, va augmenter la consultation de 1 pour l'animal donné. Cette information devra être stockée dans une base de données non relationnelle. Ces données sont exploitées dans le Dashboard administrateur afin que José puisse visualiser quels animaux plaisent le plus.

Diagramme des cas d'usage

À partir du brief client, j'ai pu établir le diagramme de cas d'usage figurant en annexe 1.

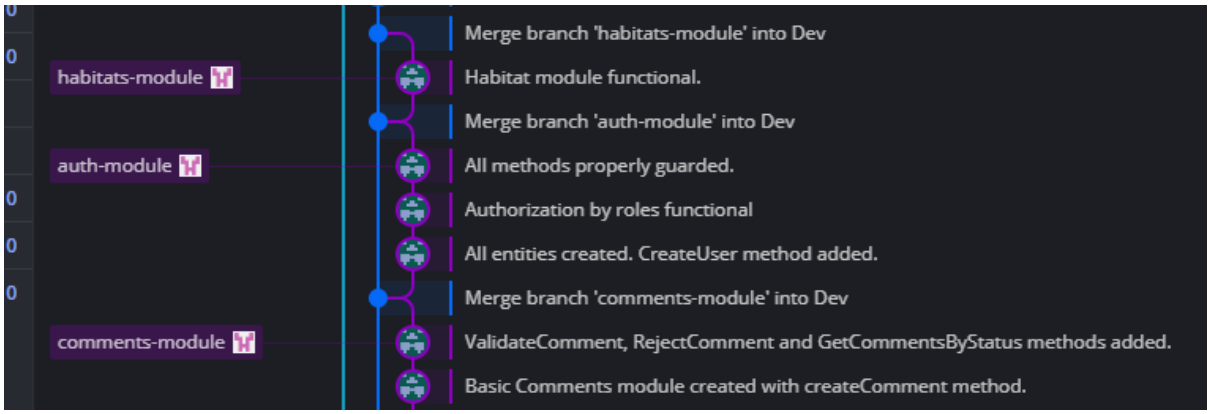
Spécifications techniques

Versioning

La gestion de versions est un élément essentiel de tout projet informatique, même lorsqu'il n'implique qu'une seule personne. En effet, elle permet de revenir à une version fonctionnelle du code en cas si on découvre que les derniers ajouts ont créé des problèmes critiques. Elle permet aussi de gérer les différentes phases de l'évolution d'un projet, y compris après sa livraison si des développements ultérieurs sont commandés.

J'ai utilisé GitHub pour ce projet, avec l'interface graphique de l'outil GitKraken. Mon front-end et mon back-end étant développés comme deux applications indépendantes, j'ai donc utilisé deux repository GitHub différents. Chacun comporte une branche *Dev* sur laquelle se faisait le gros du développement, ainsi qu'une branche *Main* sur laquelle la branche Dev était mergée uniquement lorsque j'arrivais à une version que je considérais comme stable et suffisamment testée.

Lors du déploiement de chaque élément, j'ai suivi le modèle git-flow de Vincent Driessen. J'ai donc créé une nouvelle branche sur laquelle travailler, puis j'ai mergé cette branche dans la branche Dev une fois la fonctionnalité terminée et stable. Voici un extrait des commits effectués sur le répertoire Github contenant le back-end de l'application.

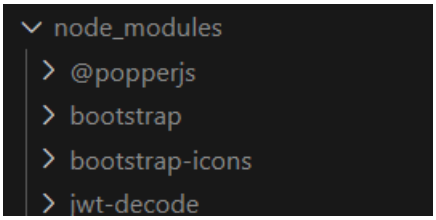


Technologies utilisées

Pour la partie front-end de l'application, j'ai évidemment utilisé le langage de structuration de page HTML 5 (*HyperText Markup Language*) et les Feuilles de Style en Cascade CSS 3 (*Cascading Style Sheets*). La dynamisation de l'interface s'est faite grâce au langage de programmation JavaScript. De plus, j'ai choisi d'utiliser le framework Bootstrap, qui propose de nombreuses librairies HTML, CSS et JS pour gérer rapidement certains éléments de mon front-end. Pour modifier certaines options de Bootstrap dont notamment la palette de couleurs de mon application, j'ai utilisé ponctuellement le pré-processeur CSS SASS.

J’ai par contre renoncé à utiliser un framework JavaScript comme Angular ou Vue, ou la bibliothèque React. Mon raisonnement était que l’apprentissage d’un tel outil ne m’épargnerait pas celui du langage JavaScript en lui-même ; il m’aurait donc pris un temps précieux, plus fructueusement consacré pour l’instant à bien maîtriser les subtilités du JavaScript. Je compte bien cependant apprendre à maîtriser au moins un framework JavaScript pour mes projets futurs.

Enfin, je me suis servi pour faciliter la gestion des JSON Web Tokens utilisés pour l’authentification des utilisateurs la librairie Node.js jwt-decode. Voici à quoi ressemble le dossier node_modules de mon projet front-end.



Pour la partie back-end de mon application, j’ai choisi d’utiliser aussi le langage JavaScript, ou plutôt son sur-ensemble Typescript, grâce à l’environnement d’exécution Node.js. Ce choix, plutôt que celui de PHP par exemple, m’a permis de concentrer mon apprentissage sur ce langage de programmation et de maximiser la pratique que j’en ai.

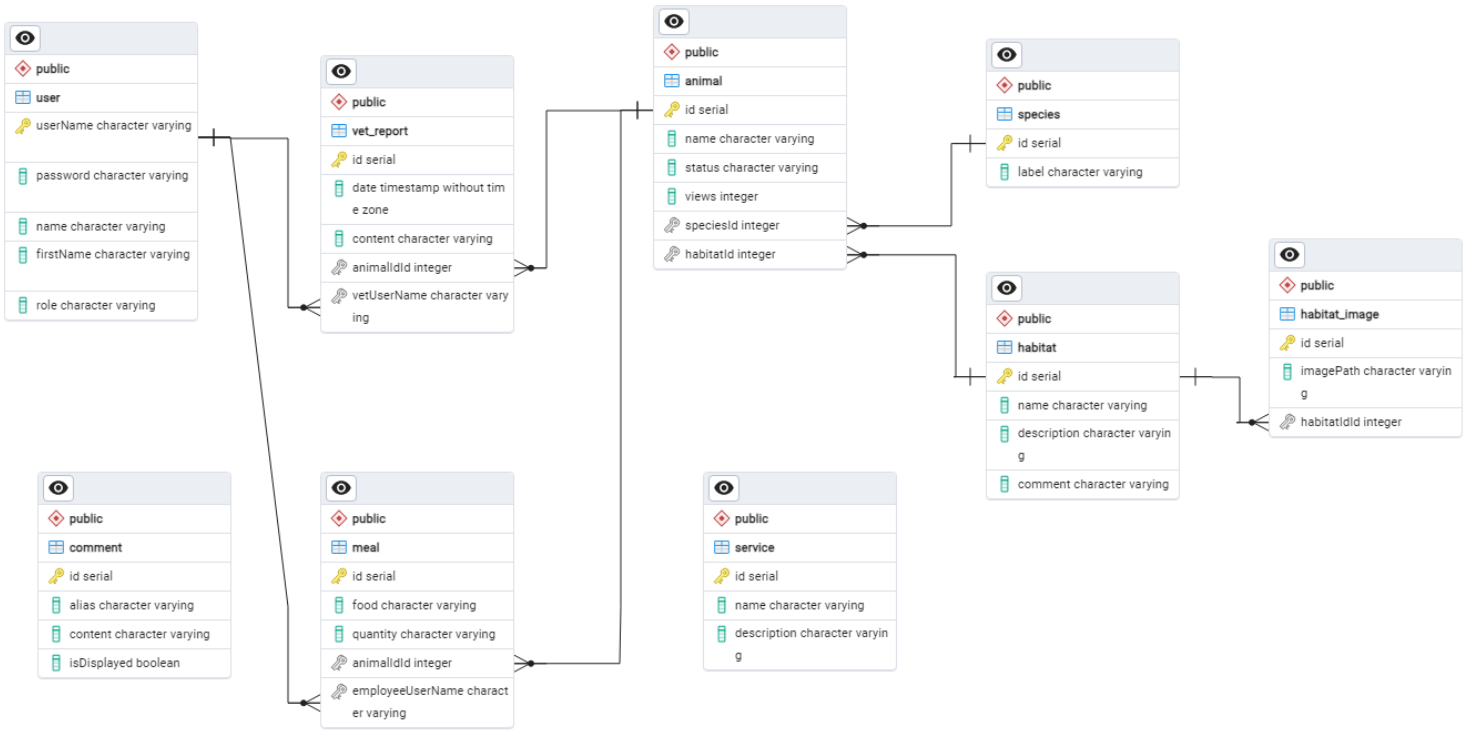
J’ai aussi choisi d’utiliser le framework NestJS afin de construire mon back-end. Le fait que ce framework soit assez prescriptif, notamment en termes d’architecture, m’a paru idéal pour un premier gros projet. J’apprécie aussi le fait qu’il gère l’injection de dépendances. J’ai de plus installé un certain nombre de librairies supplémentaires pour me faciliter la gestion de différentes fonctionnalités de l’application.

En ce qui concerne la base de données, j’ai choisi d’utiliser le système de gestion de base de données PostgreSQL avec l’interface utilisateur pgAdmin. J’ai créé ma base de données et l’ai alimentée en données d’essai grâce à des scripts SQL, mais j’ai utilisé l’ORM TypeORM pour manipuler les données depuis l’application.

J’ai choisi de déployer mon application et sa base de données via la plateforme Heroku.

Création de la base de données

En partant du brief client, j'ai établi le modèle physique de données suivant :



Malgré mon travail d'analyse préliminaire, ce schéma a évolué au fur et à mesure du développement, quand il me semblait qu'une autre organisation faciliterait l'écriture de requêtes. La structure finale de ma base se trouve en Annexe 2.

J'ai ensuite écrit un script suivant pour créer la base de données, et je l'ai corrigé lui aussi au fur et à mesure des changements de structure :

```
CREATE TABLE "user" (
    "userName" VARCHAR(50) PRIMARY KEY NOT NULL,
    password VARCHAR(100) NOT NULL,
    name VARCHAR(50) NOT NULL,
    "firstName" VARCHAR(50) NOT NULL,
    role VARCHAR(10) NOT NULL
);

CREATE TABLE service_image (
    id SERIAL PRIMARY KEY NOT NULL,
```

```
    "fileName" VARCHAR NOT NULL,
    "imageFile" BYTEA NOT NULL,
);

CREATE TABLE service (
    id SERIAL PRIMARY KEY NOT NULL,
    name VARCHAR(50) NOT NULL,
    description VARCHAR NOT NULL,
    "imageId" INT,
    FOREIGN KEY ("imageId")
        REFERENCES service_image(id)
);

CREATE TABLE comment (
    id SERIAL PRIMARY KEY NOT NULL,
    alias VARCHAR(50) NOT NULL,
    content TEXT NOT NULL,
    "isDisplayed" BOOLEAN NOT NULL
);

CREATE TABLE habitat_image (
    id SERIAL PRIMARY KEY NOT NULL,
    "fileName" VARCHAR NOT NULL,
    "imageFile" BYTEA NOT NULL
);

CREATE TABLE habitat (
    id SERIAL PRIMARY KEY NOT NULL,
    name VARCHAR(50) NOT NULL,
    description TEXT NOT NULL,
    comment VARCHAR,
    "imageId" INT,
    FOREIGN KEY ("imageId")
        REFERENCES habitat_image(id)
);

CREATE TABLE species (
    id SERIAL PRIMARY KEY NOT NULL,
    label VARCHAR(100) NOT NULL
);

CREATE TABLE animal_image (
    id SERIAL PRIMARY KEY NOT NULL,
    "fileName" VARCHAR NOT NULL,
    "imageFile" BYTEA NOT NULL,
);

CREATE TABLE animal (
    id SERIAL PRIMARY KEY NOT NULL ,
    name VARCHAR(50) NOT NULL,
    status VARCHAR(50) NOT NULL,
    views INT NOT NULL,
    "speciesId" INT,
    "habitatId" INT,
    "imageId" INT,
    FOREIGN KEY ("imageId")
        REFERENCES animal_image(id),
    FOREIGN KEY ("speciesId")
        REFERENCES species(id),
    FOREIGN KEY ("habitatId")
        REFERENCES habitat(id)
```

```
);

CREATE TABLE meal (
    id SERIAL PRIMARY KEY NOT NULL,
    food VARCHAR(50) NOT NULL,
    quantity VARCHAR(50) NOT NULL,
    "animalId" INT,
    "employeeUserName" VARCHAR(50),
    FOREIGN KEY ("animalId")
        REFERENCES animal(id),
    FOREIGN KEY ("employeeUserName")
        REFERENCES "user"("userName")
);

CREATE TABLE vet_report (
    id SERIAL PRIMARY KEY NOT NULL,
    date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    content TEXT NOT NULL,
    food VARCHAR(50) NOT NULL,
    quantity VARCHAR(50) NOT NULL,
    "animalId" INT,
    "vetUserName" VARCHAR(50),
    FOREIGN KEY ("animalId")
        REFERENCES animal(id),
    FOREIGN KEY ("vetUserName")
        REFERENCES "user"("userName")
);
```

Et j'ai créé un jeu de données d'essai minimal à insérer dans ma base à l'aide d'une autre script SQL. Ces données d'essai m'ont permis de tester tous les appels au back-end effectués par mon front-end, notamment pour composer le contenu html de pages comme celle des Services. De plus, il était absolument indispensable de créer au moins un compte administrateur, qui pourrait à son tour créer d'autres utilisateurs – mais pour plus de commodité, j'ai aussi créé un compte employé et un compte vétérinaire.

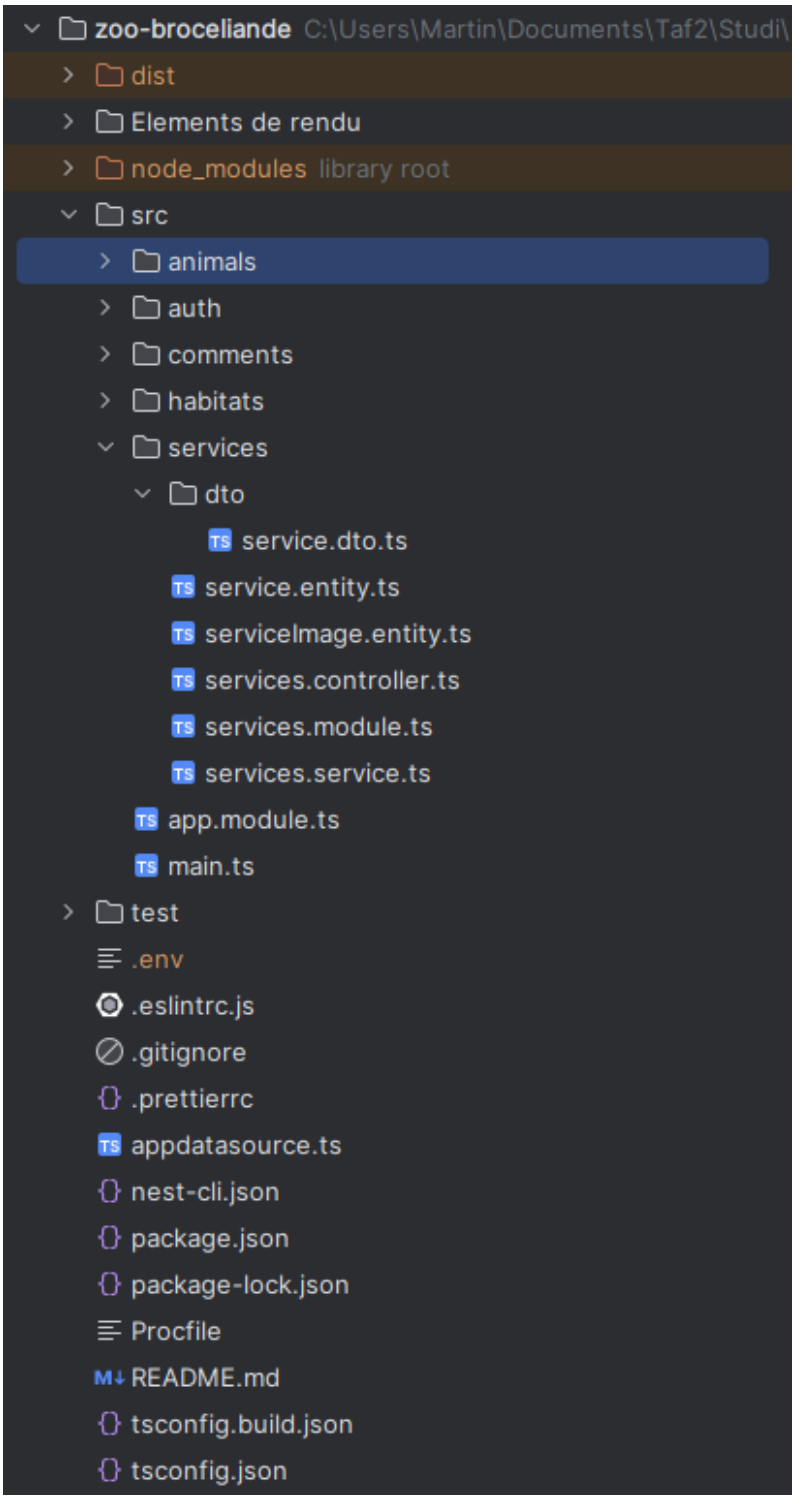
```
INSERT INTO "user" ("userName", password, name, "firstName", role)
VALUES ('admin@example.com',
'$2b$10$JxNxhrtSq.tzk53VF7jZoekgaKuHsx3L/RilYJOooLNU8zrJEuZ5u', 'admin',
'adm', 'admin'),
       ('john@example.com',
'$2b$10$JxNxhrtSq.tzk53VF7jZoekgaKuHsx3L/RilYJOooLNU8zrJEuZ5u', 'Doe',
'John', 'vet'),
       ('jane@example.com',
'$2b$10$JxNxhrtSq.tzk53VF7jZoekgaKuHsx3L/RilYJOooLNU8zrJEuZ5u', 'Smith',
'Jane', 'employee');
INSERT INTO service (name, description)
VALUES ('Visite guidée du parc', 'Venez passer un moment exceptionnel au
zooparc de Brocéliande ! Apprenez comment fonctionne le parc, ses projets,
la gestion des animaux au quotidien. Au programme également, l'engagement
du zoo pour la conservation des espèces en milieu naturel, les échanges
avec d''autres parcs zoologiques, des anecdotes etc. Ces visites sont
gratuites.'),
        ('Visite en train', 'Profitez d''une visite à travers le zooparc à
bord de notre petit train de caractère. Un départ a lieu toutes les heures
de 10 heures à 18 heures, et la visite dure une demi-heure. Tous à bord !
Coût du billet : 5 euros.');
INSERT INTO species (label) VALUES ('Lion'), ('Tigre'), ('Eléphant');
```

```
INSERT INTO comment (alias, content, "isDisplayed")
VALUES ('User1', 'Great zoo!', true),
       ('User2', 'Amazing animals!', false);
INSERT INTO habitat (name, description)
VALUES ('La savane', 'La savane est un type de paysage qui est composé
surtout d''herbes.'),
       ('La jungle', 'La jungle est la forme de végétation naturelle des
régions équatoriales de basse altitude.'),
       ('Le marais', 'Un marais est un terrain qui reste très humide la
plupart du temps, souvent envahi de nappes d''eau stagnante peu
profonde.');
INSERT INTO animal (name, status, "speciesId", "habitatId")
VALUES ('Alex', 'En bonne santé', 2, 1),
       ('Hathi', 'Malade', 152, 2);
INSERT INTO meal (food, quantity, "animalId", "employeeUserName")
VALUES ('viande', '5 kg', 1, 'jane@example.com'),
       ('feuilles', '10 kg', 2, 'jane@example.com');
INSERT INTO vet_report (content, food, quantity, "animalId", "vetUserName")
VALUES ('Visite mensuelle de routine', 'viande', '10 kg' 1,
'john@example.com'),
       ('Opération d''urgence', 'feuilles', '50 kg', 2,
'john@example.com');
```

Afin d'établir la connexion entre l'application et la base de données, j'ai défini la variable d'environnement DATABASE_URL dans un fichier .env placé à la racine du projet. Pour des raisons de sécurité, j'ai aussi créé à cette occasion un fichier .gitignore auquel j'ai ajouté le fichier .env, afin qu'il ne soit pas publié.

Back End

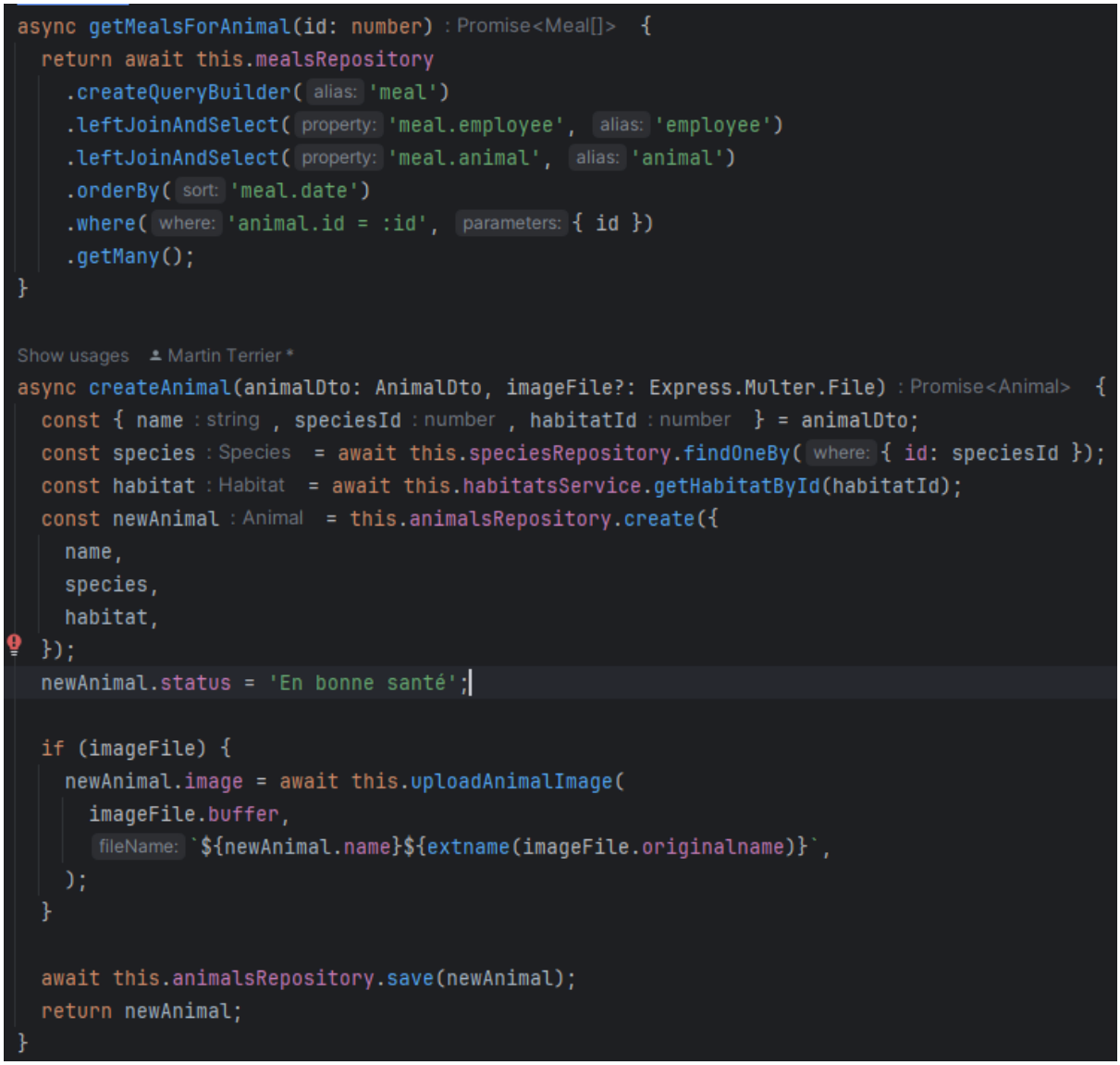
J’ai installé et utilisé l’outil d’interface en ligne de commande Nest CLI pour créer la structure de mon application. J’ai créé un dossier pour chaque fonctionnalité principale de l’application, puis au sein de ce dossier, un module, un controller et un ou plusieurs services, ainsi que les entity et les repository correspondant aux tables de ma base de données. Voici un aperçu de l’architecture finale de l’application :



Au sein de cette structure, une entity est simplement la transcription sous forme de classe TypeScript d'une structure d'objet de ma base de données. À titre d'exemple, voici le contenu du fichier animal.entity.ts, qui définit la classe Animal correspondant aux objets de la table *animal* de la base de données.

```
Show usages  Martin Terrier
@Entity()
export class Animal {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column()
  status: string;

  @Column()
  views: number;

  @JoinColumn()
  @ManyToOne( typeFunctionOrTarget: () => Species,  inverseSide: (species : Species ) => species.id)
  species: Species;

  @JoinColumn()
  @ManyToOne( typeFunctionOrTarget: () => Habitat,  inverseSide: (habitat : Habitat ) => habitat.id)
  habitat: Habitat;

  @JoinColumn()
  @OneToOne( typeFunctionOrTarget: () => AnimalImage,  inverseSide: (animalImage : AnimalImage ) => animalImage.id)
  image: AnimalImage;

  @Column()
  imageId: number;
}
```

Le service, lui contient la logique métier nécessaire à une fonctionnalité : créer un service, le mettre à jour, enregistrer l'image liée à un service… Voici un extrait du service animals.service.ts, qui rassemble toutes les opérations liées de près ou de loin aux animaux (et donc aussi aux repas et aux rapports vétérinaires).



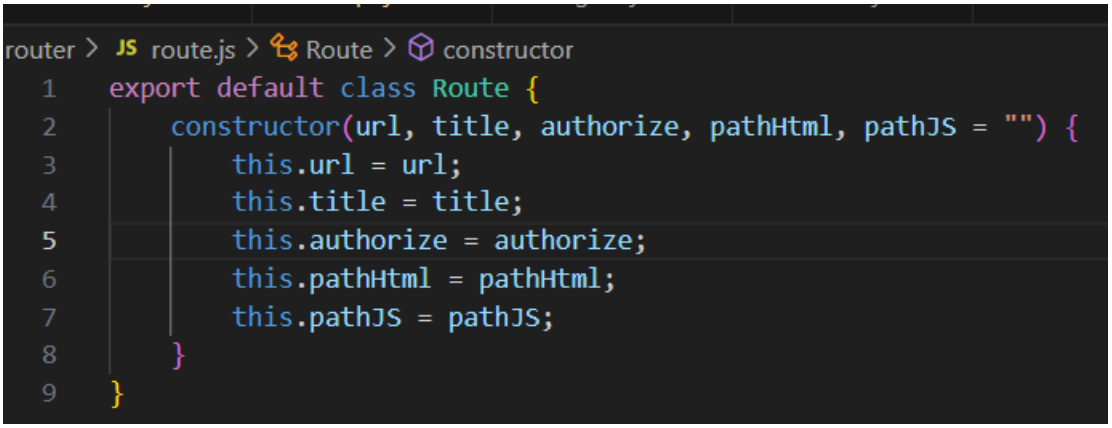
```
async getMealsForAnimal(id: number) : Promise<Meal[]>  {
  return await this.mealsRepository
    .createQueryBuilder( alias: 'meal')
    .leftJoinAndSelect( property: 'meal.employee',  alias: 'employee')
    .leftJoinAndSelect( property: 'meal.animal',  alias: 'animal')
    .orderBy( sort: 'meal.date')
    .where( where: 'animal.id = :id',  parameters: { id })
    .getMany();
}

Show usages  Martin Terrier *
async createAnimal(animalDto: AnimalDto, imageFile?: Express.Multer.File) : Promise<Animal>  {
  const { name : string , speciesId : number , habitatId : number  } = animalDto;
  const species : Species  = await this.speciesRepository.findOneBy( where: { id: speciesId });
  const habitat : Habitat  = await this.habitatsService.getHabitatById(habitatId);
  const newAnimal : Animal  = this.animalsRepository.create({
    name,
    species,
    habitat,
  });
  newAnimal.status = 'En bonne santé';

  if (imageFile) {
    newAnimal.image = await this.uploadAnimalImage(
      imageFile.buffer,
      fileName: `${newAnimal.name}${extname(imageFile.originalname)}`,
    );
  }

  await this.animalsRepository.save(newAnimal);
  return newAnimal;
}
```

Enfin, le controller définit les routes par lesquelles le front-end va pouvoir accéder aux méthodes du service. Chacune précise notamment quels rôles d'utilisateur permettent d'y accéder ; si le front-end tente d'accéder à une méthode alors que l'utilisateur actif ne possède aucun des rôles définis pour celle-ci, l'appel renverra une erreur 401 "unauthorized".

```
no usages  Martin Terrier
@Get( path: '/meal/:id')
async getMealsForAnimal(@Param( property: 'id', ParseIntPipe) animalId: number) : Promise<Meal[]> {
  return await this.animalsService.getMealsForAnimal(animalId);
}

no usages  Martin Terrier
@Post()
@UseGuards(RolesGuard)
@Roles(Role.Admin)
@UseInterceptors(FileInterceptor( fieldName: 'imageFile'))
async createAnimal(
  @Body() animalDto: AnimalDto,
  @UploadedFile() imageFile: Express.Multer.File,
) : Promise<Animal> {
  if (imageFile) {
    return await this.animalsService.createAnimal(animalDto, imageFile);
  } else {
    return await this.animalsService.createAnimal(animalDto);
  }
}

no usages  Martin Terrier
@Patch( path: '/:id')
@UseGuards(RolesGuard)
@Roles(Role.Admin)
@UseInterceptors(FileInterceptor( fieldName: 'imageFile'))
async updateAnimal(
  @Param( property: 'id', ParseIntPipe) id: number,
  @Body() animalDto: AnimalDto,
  @UploadedFile() imageFile: Express.Multer.File,
) : Promise<Animal> {
  if (imageFile) {
    return await this.animalsService.updateAnimal(id, animalDto, imageFile);
  } else {
    return await this.animalsService.updateAnimal(id, animalDto);
  }
}
```

Front End

Mon front-end est une “single page application” (application à page unique), qui n’utilise techniquement qu’une seule page index.html. En réalité, celle-ci ne contient que le header et le footer du site. Le premier contient le menu de navigation, et le second des informations utiles et le lien de contact. Le contenu qui se trouve entre ces deux éléments est modifié à chaque événement de navigation par un router codé en JavaScript. Pour cela, j’ai d’abord créé une classe Route :



Puis j’ai défini toutes les routes que j’allais utiliser pour la navigation de mon front-end, dans un fichier allRoutes.js :

Et enfin, j’ai créé un fichier router.js dans lequel j’ai codé la logique de routage. Tout d’abord, j’ai défini une fonction routeEvent qui, lorsque l’utilisateur clique sur un lien pour naviguer à l’intérieur du site, empêche la réponse normale du navigateur de se déclencher, modifie l’historique du navigateur et appelle une fonction loadPageContent, qui va se charger de modifier dynamiquement le contenu de la page sans la recharger.

```
// Fonction pour gérer les événements de routage (clic sur les liens)
const routeEvent = (event) => {
    event.preventDefault();
    // Mise à jour de l'URL dans l'historique du navigateur
    window.history.pushState({}, "", event.target.href);

    // Chargement du contenu de la nouvelle page
    LoadPageContent();
};
```

La fonction loadPageContent, elle, commence par appeler une troisième fonction getRouteByUrl pour récupérer les données de la route appelée.

```
// Fonction pour charger le contenu de la page
const LoadPageContent = async () => {
    const path = window.location.pathname;
    // Récupération de l'URL actuelle
    const currentRoute = getRouteByUrl(path);
```

La fonction getRouteByUrl parcourt le tableau du fichier allRoutes.js pour déterminer laquelle a été appelée lors de l'événement. Si aucune ne correspond, elle renvoie vers la page 404 de l'application. Si elle trouve la route appelée, elle renvoie l'objet correspondant dans le tableau.

```
// Fonction pour récupérer la route correspondant à une URL donnée
const getRouteByUrl = (url) => {
    let currentRoute = null;
    // Parcours de toutes les routes pour trouver la correspondance
    allRoutes.forEach((element) => {
        if (element.url === url) {
            currentRoute = element;
        }
    });
    // Si aucune correspondance n'est trouvée, on retourne la route 404
    // return currentRoute || route404;
    if (currentRoute) {
        return currentRoute;
    } else {
        return route404;
    }
};
```

Une fois la nouvelle route récupérée, la fonction loadPageContent vérifie que l'utilisateur a bien le droit d'y accéder (en fonction de son statut de connexion et de son rôle le cas échéant), comme nous le verrons plus loin, puis elle injecte le contenu html de la page correspondante dans l'élément <main> de la page actuelle.

```
// Récupération du contenu HTML de la route
const html = await fetch(currentRoute.pathHtml).then((data) => data.text());
// Ajout du contenu HTML à l'élément avec l'ID "main-page"
document.getElementById("main-page").innerHTML = html;
```

Elle récupère aussi le fichier JavaScript qui dynamise la nouvelle page, s'il y en a un, et l'injecte aussi.

```
// Ajout du contenu JavaScript
if (currentRoute.pathJS) {
    // Création d'une balise script
    const scriptTag = document.createElement("script");
    scriptTag.setAttribute("type", "module");
    scriptTag.setAttribute("src", currentRoute.pathJS);

    // Ajout de la balise script au corps du document
    document.querySelector("body").appendChild(scriptTag);
}
```

Enfin, elle change dynamiquement le titre de la page et masque les éléments affichés que l'utilisateur ne devrait pas voir, en fonction de son statut de connexion et de son rôle.

```
    // Changement du titre de la page
    document.title = `${currentRoute.title} - ${websiteName}`;

    // Masquer les éléments indisponibles en fonction du rôle
    manageRoleElements();
};
```

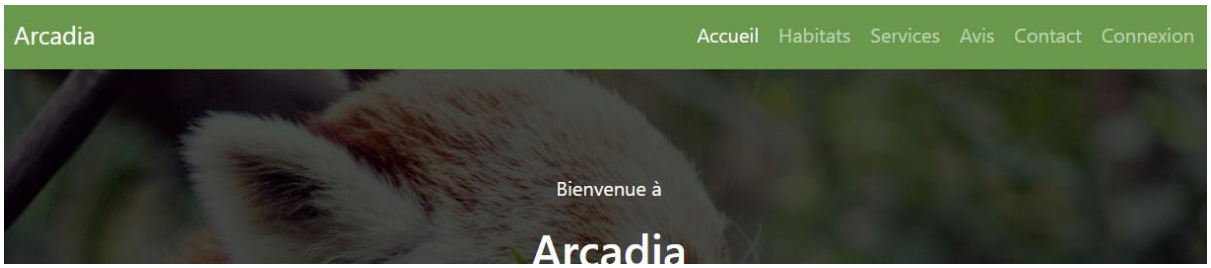
Le script router.js est appelé sur la page index.html, et lance la fonction loadPageContent lorsqu'il est lancé ainsi qu'à chaque retour en arrière dans l'historique du navigateur.

```
// Gestion de l'événement de retour en arrière dans l'historique du navigateur
window.onpopstate = LoadPageContent;
// Assignation de la fonction routeEvent à la propriété route de la fenêtre
window.route = routeEvent;
// Chargement du contenu de la page au chargement initial
LoadPageContent();
```

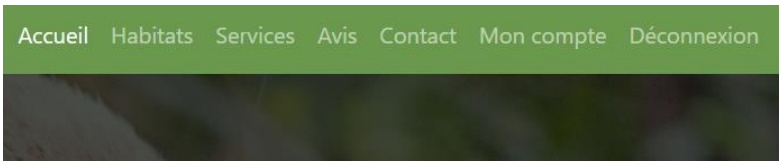
Sécurité

Gestion de l'authentification et des rôles

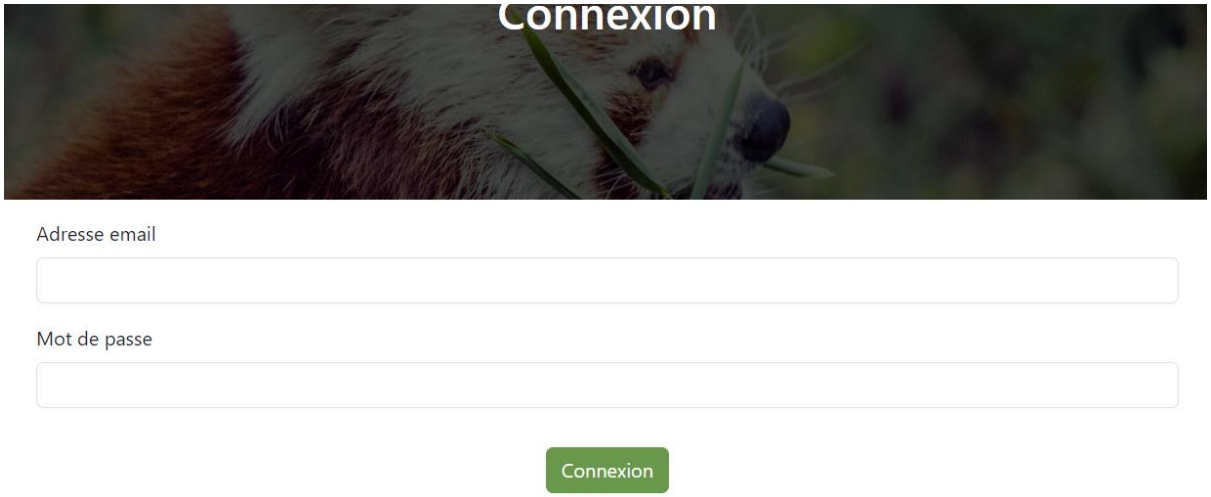
Pour gérer l'authentification des utilisateurs connectés, ainsi que les différents rôles qu'ils peuvent avoir, j'ai choisi d'utiliser des JSON web tokens stockés dans le local storage du navigateur. Je n'ai pas eu besoin de gérer de page d'inscription, puisque le brief client précisait que c'était l'administrateur qui créerait les utilisateurs. J'ai par contre une page Connexion, accessible depuis le menu de navigation en haut de page.



En réalité, ce lien vers la page de Connexion n'est visible que pour les visiteurs déconnectés ; quand un utilisateur est connecté, la fonction manageRoleElements que nous avons vue appelée par loadPageContent masque dynamiquement ce lien et le remplace par un autre, Déconnexion. La même fonction rend accessible un lien vers la page Mon compte, pointant vers une page différente enfonction du rôle de l'utilisateur connecté (Employé, Vétérinaire ou Administrateur).



La page de connexion ne comporte qu'un formulaire permettant au visiteur souhaitant se connecter de saisir son adresse email ainsi que son mot de passe.



Grâce à la méthode addEventListener, nous associons au clic sur le bouton “Connexion” une fonction checkCredentials qui va vérifier si un utilisateur existe pour le couple email, mot de passe saisi.

```
const checkCredentials = async () => {
    let dataForm = new FormData(signinForm);

    let myHeaders = new Headers();
    myHeaders.append('Content-Type', 'application/json');

    let raw = JSON.stringify({
        'userName': dataForm.get('email'),
        'password': dataForm.get('password'),
    });

    let requestOptions = {
        method: 'POST',
        headers: myHeaders,
        body: raw,
        redirect: 'follow',
    };

    fetch(`${apiBaseUrl}/auth/login`, requestOptions)
```

Pour cela, nous récupérons les données saisies dans les champs du formulaire et nous créons un objet JSON les contenant. Nous passons ensuite cet objet JSON dans le corps d’un appel au back-end, en utilisant la fonction fetch pour effectuer une requête POST vers la route “/auth/login” exposée par le back-end.

Dans le fichier auth.controller.ts du back-end, celle-ci est déclarée ainsi :

```
@Post( path: '/login')
async signIn(@Body() signInDto: SignInDto) : Promise<{accessToken: string}>  {
  return await this.authService.signIn(signInDto);
}
```

On voit que la fonction signIn du service authService accepte comme argument un DTO, c’est-à-dire un *Data Transfer Object* (Objet de transfert de données) que j’ai défini dans son propre fichier. Grâce à la bibliothèque class-validator, je peux préciser que la propriété userName doit être un email, ce que NestJS va donc vérifier automatiquement. Si cette vérification sur les données reçues du front-end échoue, le back-end renverra une erreur.

```
Show usages  Martin Terrier
export class SignInDto {
  @IsEmail()
  userName: string;

  password: string;
}
```

La méthode signIn récupère donc le DTO contenant email et mot de passe, et commence par chercher dans la base de données si un utilisateur existe pour cet email. Si c'est le cas, elle compare le mot de passe qu'elle a reçu avec celui de cet utilisateur en base de données. Si cette comparaison échoue, soit parce que l'utilisateur n'existe pas, soit parce que le mot de passe est incorrect, elle renvoie une erreur.

```
async signIn(signInDto: SignInDto) : Promise<{accessToken: string}>  {
  const { userName : string , password : string  } = signInDto;

  const user : User  = await this.usersRepository.findOneBy( where: { userName });

  if (!(await bcrypt.compare(password, user.password))) {
    throw new UnauthorizedException( objectOrError: 'Please check your login credentials.');
  }
```

On peut voir cependant que la méthode appelle pour faire sa comparaison une bibliothèque, bcrypt. En réalité, la comparaison ne se fait pas entre la valeur de la base de données et celle que l'utilisateur a saisie. En effet, à la création de l'utilisateur, le mot de passe est hashé avant d'être stocké grâce à la méthode .hash de becrypt.

```
const salt : string  = await bcrypt.genSalt();
const hashedPassword : string  = await bcrypt.hash(password, salt);

const newUser : User  = this.create({
  userName,
  name,
  firstName,
  role,
  password: hashedPassword,
});
```

Si l'utilisateur existe et que son mot de passe est bien celui qui a été transmis par le front-end, la méthode signIn renvoie un JSON Web Token. Celui-ci contient le nom et le rôle de l'utilisateur. De plus, l'une des chaînes de caractères qui le compose est encodée à l'aide d'un "secret", c'est à dire une autre chaîne de caractères. Ce token va être associé à chaque requête au back-end faite par la suite par l'utilisateur, et permettra à l'application de

vérifier en le décodant que l'utilisateur est bien autorisé à faire cette requête. Pour des raisons de sécurité, j'ai stocké le secret non pas dans le code de l'application mais dans une variable d'environnement JWT_SECRET_KEY, qui se trouve dans le fichier .env.

```
const payload : {userName: string, role: Role}  = { userName, role: user.role };
const accessToken : string  = await this.jwtService.signAsync(payload, options: {
  secret: process.env.JWT_SECRET_KEY,
});
return { accessToken };
```

Le token est intégré à la réponse qu'envoie le back-end au front-end et récupéré par notre fonction checkCredentials. Si le back-end renvoie une erreur, le front-end modifie dynamiquement les champs du formulaire pour montrer que les informations saisies étaient incorrectes. Si le statut de la réponse est "ok", la fonction checkCredentials appelle une autre fonction pour stocker ce token, puis renvoie l'utilisateur sur la page d'accueil.

```
fetch(`${apiBaseUrl}/auth/login`, requestOptions)
.then(async (response) => {
    if (response.ok) {
        const json = await response.json();
        console.log(json);
        return json;
    } else {
        emailInput.classList.add('is-invalid');
        passwordInput.classList.add('is-invalid');
    }
})
.then((result) => {
    setConnexionToken(result);

    window.location.replace('/');
})
.catch((error) => console.error(error));
```

La fonction setConnexionToken se contente de placer le token reçu dans le local storage du navigateur. Une fonction getConnexionToken permettra de le récupérer pour l'intégrer au header des futures requêtes nécessitant une autorisation.

```
function setConnexionToken(token){
    localStorage.setItem('accessToken', token.accessToken);
}
function getConnexionToken(){
    const token = localStorage.getItem('accessToken');
    return token;
}
```

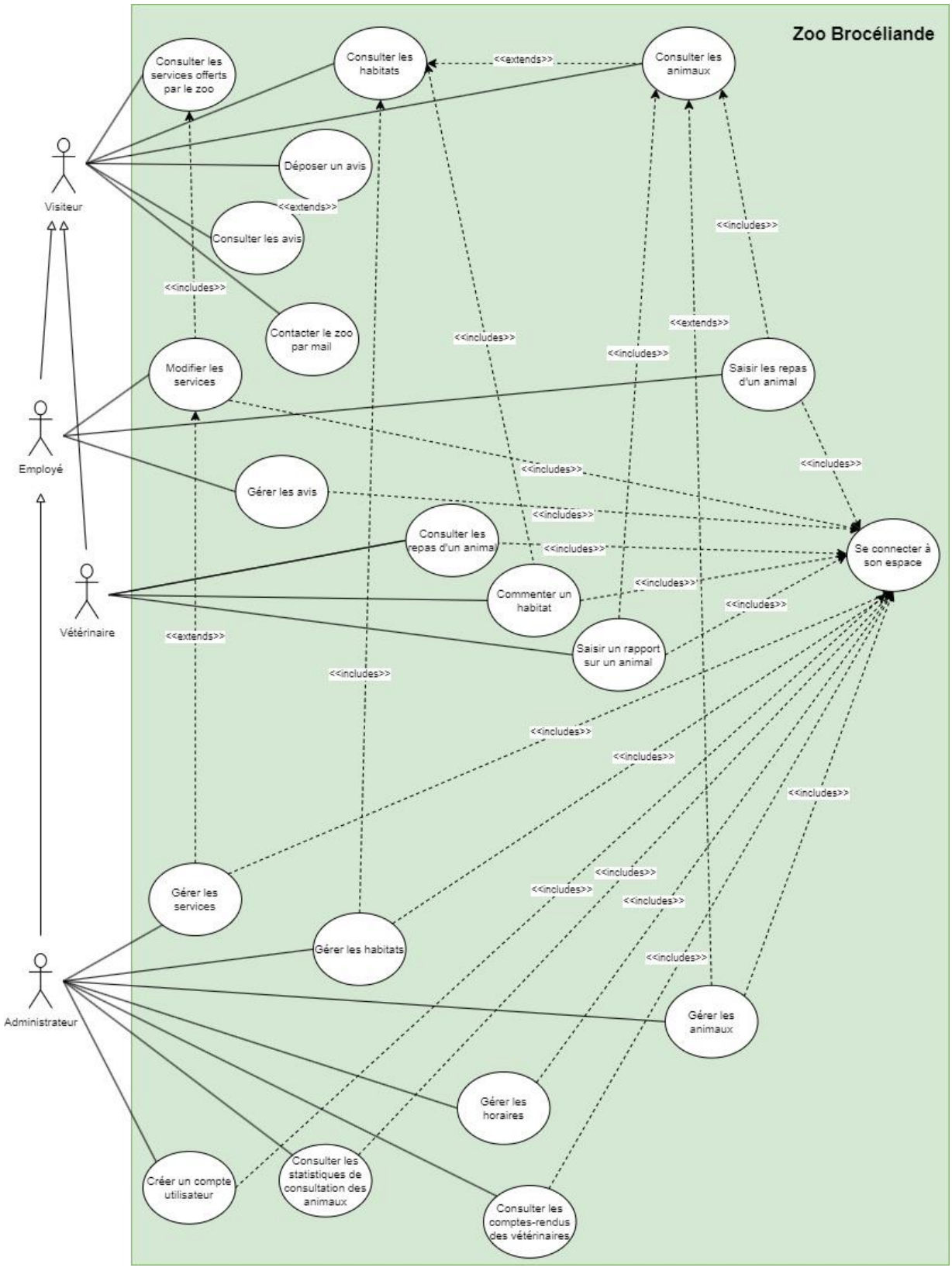
Conclusion

Ce projet, comme toute cette année, a été compliqué. J'ai eu du mal à trouver du temps pour lui. J'ai fait des choses dans le mauvais ordre. J'ai fait des choix discutables, et j'ai eu le temps de m'en rendre compte mais pas de les corriger. Pour autant, j'ai beaucoup appris. Je peux même dire qu'à côté du travail en entreprise et du soutien de mes collègues, sur le front de la formation, c'est indéniablement ce sujet qui m'a fait le plus progresser.

Aucun exercice, aucun cours sur un sujet précis ne peut enseigner autant que le fait de s'attaquer à un projet complet, dans son entièreté. Ce n'est, j'en suis sûr, que le premier auquel je m'attelle ; il y en aura d'autres, quelle que soit son issue, pour me permettre de progresser encore. Dans mon nouveau poste au sein de l'entreprise qui m'embauche, bien sûr, mais aussi à côté sur des sujets que je trouverai plus proches de moi. Si j'ai eu toute cette année l'impression de gravir une pente raide, je découvre de là où je suis arrivé que je ne suis qu'au début du voyage, certes, mais que mon point de départ paraît lui aussi tellement loin à présent qu'il n'y aurait aucun sens à faire demi-tour. Le meilleur est devant moi, et maintenant, je suis échauffé.

Annexes

Annexe 1 : diagramme de cas d'utilisation



Annexe 2 : structure finale de la base de données