Rapport ECF : Développement d'un Site Web

Introduction

Dans le cadre de mon Examen en Cours de Formation (ECF), j'ai entrepris le développement d'un site web intégrant des fonctionnalités dynamiques et interactives. Ce projet avait pour objectif de mettre en pratique les compétences acquises tout au long de ma formation, en respectant un cahier des charges précis et en utilisant des technologies modernes telles que PHP pour le développement côté serveur et MongoDB pour la gestion des données. Ce rapport présente les différentes étapes de conception, les choix technologiques, ainsi que les difficultés rencontrées et les solutions apportées.

Activités Réalisées

Développement de la Partie Front-End

- Installation et configuration de l'environnement de travail : Mise en place de l'environnement de développement en fonction des besoins du projet web ou web mobile.
- Maquettage des interfaces utilisateur : Création de maquettes pour les interfaces utilisateur du site, en prenant en compte l'ergonomie et l'expérience utilisateur.
- Développement des interfaces utilisateur statiques : Création des pages statiques en HTML, CSS, et JavaScript.
- Développement de la partie dynamique des interfaces utilisateur : Intégration de fonctionnalités interactives et dynamiques pour améliorer l'expérience utilisateur.

Développement de la Partie Back-End

- Mise en place d'une base de données relationnelle : Conception et création d'une base de données pour stocker les informations nécessaires au fonctionnement de l'application.
- Développement des composants d'accès aux données : Mise en place des composants pour accéder aux données SQL et NoSQL, en utilisant MongoDB et MySQL.
- Développement des composants métier côté serveur : Programmation des fonctionnalités logiques du serveur pour assurer la cohérence des données et des interactions utilisateur.
- Documentation du déploiement : Rédaction de la documentation pour le déploiement de l'application web dynamique, incluant les configurations nécessaires pour un bon fonctionnement.

US 1 : Page d'accueil

Utilisateur concerné : Visiteur

La page d'accueil doit comporter une présentation du zoo en y intégrant quelques images, mentionner les différents habitats, services, ainsi que les animaux du zoo, et inclure les avis des visiteurs.

Retour d'expérience

Pour la réalisation de la page d'accueil, j'ai d'abord créé un squelette de base en HTML et CSS, ce qui m'a permis de poser les fondations visuelles du site. Après avoir créé le dépôt GitHub et organisé les branches de développement, j'ai également consulté mon fichier de gestion de projet pour planifier les tâches. En raison de changements personnels, notamment la naissance de mon enfant, j'ai dû adapter mon emploi du temps et la gestion du projet. Malgré ces ajustements, j'ai réussi à maintenir une certaine cohérence en suivant les User Stories définies. Le développement de cette page a progressé étape par étape, se concentrant sur la structure HTML et CSS, avant d'ajouter progressivement des fonctionnalités interactives.

US 2 : Menu de l'application

Utilisateur concerné : Visiteur

Le menu est essentiel à l'application, permettant de faciliter la navigation et de fluidifier le trafic. Il doit inclure : un retour vers la page d'accueil, l'accès à tous les services et habitats, une section de connexion (pour vétérinaires, employés et administrateurs uniquement), et une page de contact.

Retour d'expérience

Pour la création du menu de l'application, j'ai décidé de rendre le site plus dynamique en ajoutant des balises hypertexte (<a href>), permettant de lier les différentes pages entre elles. À ce stade, seules la page d'accueil, la page des services, la page des habitats, et la page de contact étaient disponibles. Les pages employé, vétérinaire, et administrateur ont été ajoutées plus tard.

C'est également à cette étape que j'ai commencé à réfléchir au design plus avancé, notamment en ajoutant des effets de survol (hover) sur les liens pour améliorer l'expérience utilisateur. J'ai conçu une barre de navigation cohérente qui serait disponible sur toutes les pages du site.

Avec le recul, je constate que le code du menu est devenu redondant, car il a été répété sur chaque page. Cela m'a permis de comprendre l'importance de la modularité et de la réduction de la charge côté client. Pour de futurs projets, j'utiliserai une approche plus modulaire, en séparant la logique d'affichage, les variables, et les calculs côté serveur. Cela permettrait d'améliorer les performances et de rendre le code plus maintenable.

À ce stade, la base de données n'avait pas encore été implémentée, ce qui signifiait qu'il n'était pas possible de créer un espace de connexion sécurisé. J'avais prévu de créer une base de données MariaDB pour stocker les utilisateurs, afin d'assurer une gestion sécurisée des connexions. Bien que l'espace de connexion n'ait pas encore été opérationnel, j'ai continué à travailler sur le menu de navigation, en le rendant disponible sur toutes les pages existantes.

US 3 : Vue globale de tous les services

Utilisateur concerné : Visiteur

Une vue globale est nécessaire pour proposer une interface simple et récapitulative de tous les services proposés par le parc. Les services doivent être configurables depuis l'espace administrateur.

Retour d'expérience

Pour l'implémentation de la User Story 3, j'ai principalement travaillé sur la page des services. À ce stade, je n'avais pas encore de base de données et je n'avais pas l'intention de l'implémenter immédiatement. J'ai donc conçu la page des services en utilisant des balises HTML, y compris des titres, et en utilisant Flexbox pour structurer la mise en page de manière esthétique.

J'ai également intégré des images des différents services, en les récupérant depuis le dossier `ressources`. À ce moment du développement, je n'avais pas encore anticipé le problème que cela poserait : si l'administrateur devait changer ou ajouter des services, il serait compliqué de gérer ces images directement depuis le code HTML sans une base de données. Cela rendrait la page moins flexible et plus difficile à maintenir.

Bien que je n'aie pas pris en compte ces contraintes à ce stade, j'étais satisfait du rendu visuel de la page, qui semblait fonctionnelle et offrait une bonne expérience utilisateur. Ce processus a cependant été formateur, me permettant de mieux comprendre les limites de l'approche non dynamique et les avantages qu'une base de données aurait apportés dès le début du projet.

US 4 : Vue globale des habitats

Utilisateur concerné : Visiteur

Cette page doit mentionner tous les habitats du zoo et les animaux associés. José a proposé une vue avec l'image et le nom des habitats, puis, au clic, un détail avec les animaux et la description.

Retour d'expérience

La User Story 4 a introduit un niveau de complexité supplémentaire, nécessitant l'utilisation de JavaScript pour gérer l'affichage dynamique. Lorsque l'utilisateur clique sur un habitat, la liste des animaux présents dans cet habitat doit s'afficher, et lorsqu'il clique sur un animal, des informations plus détaillées doivent apparaître.

C'était la première fois que je développais ce type de fonctionnalité, et cela m'a demandé beaucoup de réflexion. J'ai dû apprendre à manipuler le DOM avec JavaScript pour permettre une interaction fluide entre les différents éléments de la page. Bien que la version initiale ait été assez basique, elle m'a permis de comprendre comment rendre la navigation plus interactive.

Cependant, en l'absence d'une base de données, le contenu restait statique. Les informations sur les habitats et les animaux étaient codées en dur dans le HTML, ce qui posait des problèmes évidents si l'administrateur souhaitait modifier les habitats ou les animaux. Avec le recul, je comprends que pour rendre cette fonctionnalité vraiment dynamique, il aurait été préférable d'intégrer une base de données dès le début.

US 5 : Avis

Utilisateur concerné : Visiteur

Un visiteur peut laisser un commentaire s'il le désire. Ce commentaire est soumis à validation par un employé avant d'être publié.

Retour d'expérience

La User Story 5 a marqué le début de l'intégration de la base de données dans le projet. Pour permettre aux visiteurs de laisser des avis, il était nécessaire de mettre en place une base de données. J'ai opté pour l'utilisation de PHPMyAdmin, une technologie que je connaissais déjà. Compte tenu des contraintes de temps dues à ma nouvelle situation personnelle, j'ai préféré travailler avec des outils familiers.

J'ai commencé par créer une base de données de test et ajouté plusieurs tables : habitats, animaux, rapports vétérinaires, utilisateurs, etc. Cependant, la table concernant les avis n'a pas été créée immédiatement, car je n'avais pas encore finalisé la conception de la section des avis.

En parallèle, j'ai initialement mis en place une base de données MongoDB localement sur le site, mais je me suis rapidement rendu compte que cela alourdissait considérablement le projet en raison du poids de MongoDB. Cette version locale de MongoDB est encore présente sur plusieurs branches du projet GitHub, notamment sur certains commits qui conservent cette base de données locale pour les fonctionnalités liées aux clics. Lors de l'importation sur l'hébergeur, j'ai décidé de passer à une solution en ligne pour MongoDB. Cela m'a demandé beaucoup de temps, car il a fallu que je cherche une solution adéquate, remanie le code de mon site, et lise la documentation de la base de données distante pour bien comprendre son implémentation. Cette solution en ligne permet d'éviter les contraintes de configuration locale et le poids important de l'outil, rendant le projet plus simple à déployer et à maintenir.

L'intégration de la base de données a ouvert la voie à la création d'espaces sécurisés pour les différents types d'utilisateurs (employés, vétérinaires, administrateurs) et a rendu le site plus dynamique. Bien que la section des avis n'ait pas été finalisée à ce stade, l'ajout de la base de données a posé les bases nécessaires pour développer cette fonctionnalité ultérieurement.

À environ les trois quarts du projet, j'ai mis en place la solution des avis. J'avais déjà MongoDB installé et PHPMyAdmin configuré. J'avais aussi mis en place les espaces employés et vétérinaires, mais pas encore l'espace administrateur. J'ai dû réfléchir à la manière de traiter les messages des utilisateurs. Initialement, je voulais utiliser MongoDB, mais j'ai trouvé plus simple de passer par PHPMyAdmin, un outil que je connaissais déjà.

Pour gérer les avis, j'ai créé deux tables dans la base de données PHPMyAdmin. Une première table reçoit les avis dès qu'un utilisateur clique sur le bouton "Laisser un avis" et saisit un pseudonyme et un texte. Ces avis sont alors en attente de confirmation. Lorsqu'un employé consulte son espace, il voit la liste des messages en attente de validation ou de suppression. Bien que cela n'ait pas été spécifié dans la User Story, j'ai ajouté une fonctionnalité de suppression pour permettre aux employés de nettoyer la base de données des messages en attente. J'ai donc ajouté deux boutons : "Supprimer" et "Valider". Si l'employé clique sur "Supprimer", le message est effacé de la base de données. Si l'employé clique sur "Valider", le message est transféré dans une autre table, celle des messages validés, qui est affichée sur la page d'accueil (index).

US 6 : Espace Administrateur

Utilisateur concerné : Administrateur

L'espace administrateur permet de gérer les différents aspects du site, notamment la création de comptes pour les employés et les vétérinaires, ainsi que la gestion des habitats, animaux, services, et horaires. L'administrateur peut créer, modifier, et supprimer ces éléments depuis son espace.

Retour d'expérience

J'ai créé cet espace administrateur vers la fin du projet, juste avant la phase de peaufinage. Cette fonctionnalité a beaucoup chamboulé mon projet, car elle a introduit la possibilité pour José de supprimer des éléments existants, en plus de pouvoir en ajouter ou modifier. Initialement, j'avais conçu une page fixe en PHP, avec des balises statiques pour chaque habitat, service, et animal. Ajouter ou modifier un élément était faisable, mais supprimer des éléments de manière dynamique présentait des difficultés.

Pour résoudre ce problème, j'ai dû remanier complètement mon approche et opter pour une solution modulaire. J'ai transformé le code existant pour utiliser des boucles qui allaient chercher les données directement dans la base de données, puis les afficher dynamiquement. Cela m'a permis de rendre l'application beaucoup plus flexible et d'intégrer la possibilité de créer, modifier, et supprimer des éléments en toute simplicité. En passant à une approche modulaire, j'ai pu réduire plusieurs centaines de lignes de code en quelques lignes qui bouclent sur les éléments des tables de la base de données.

Cette réflexion sur la modularité m'a fait réaliser l'importance de concevoir des sites dynamiques basés sur des templates et des données gérées en base. J'ai donc tout refait : animaux, services, habitats, en utilisant des boucles pour rendre chaque élément dynamique et facilement modifiable. Dans ce projet, cependant, je n'ai pas appliqué cette approche de pré-calcul côté serveur. Chaque page interroge la base de données pour afficher les

éléments dynamiquement, ce qui n'est pas optimal en termes de performance. C'est une leçon que je retiens et que j'appliquerai pour de futurs projets.

Maintenant, pour mes futurs projets, je pense systématiquement à la modularité. Tous les éléments devant être dynamiques seront gérés par une base de données et affichés en fonction des variables pré-calculées. Cela permet une gestion simplifiée des éléments à ajouter, modifier ou supprimer, en évitant de devoir tout recharger ou recalculer côté client. Bien que cette approche m'ait pris du temps et des efforts, elle s'est révélée très efficace et m'a donné envie de continuer à pousser mes compétences en conception modulaire.

José est maintenant capable d'ajouter un utilisateur de type employé ou vétérinaire. Lorsqu'il ajoute un utilisateur, un mail est automatiquement envoyé. Il peut également modifier, ajouter ou supprimer des éléments. Initialement, j'avais inclus tous les éléments du site, mais je me suis recadré pour ne mettre que les éléments essentiels.

J'ai aussi ajouté un tableau de bord (dashboard) pour suivre certaines activités. Après avoir mis en place la base MongoDB, j'ai développé un petit script JavaScript pour enregistrer le nombre de clics sur chaque animal de la section habitat. Ces données sont ensuite stockées dans MongoDB, et l'espace administrateur permet de les récupérer et de les afficher sous forme de tableau.

Au début, avec la base de données locale, j'avais des problèmes de synchronisation, ce qui provoquait des erreurs fréquentes. En passant à une solution en ligne, la synchronisation s'est grandement améliorée, et chaque clic est désormais pris en compte correctement. Je pense que les problèmes de synchronisation étaient liés au rafraîchissement de la page au moment du clic, mais avec la solution en ligne, tout fonctionne de manière beaucoup plus fluide.

US 7 : Espace Employé

Utilisateur concerné : Employé

L'espace employé permet aux employés de valider ou supprimer les avis laissés par les visiteurs et de gérer l'alimentation quotidienne des animaux. Un employé peut ainsi ajouter une consommation de nourriture pour un animal, en mentionnant la date, le type de nourriture et la quantité donnée.

Retour d'expérience

L'espace employé a été relativement simple à développer, car il s'agissait principalement d'une fonctionnalité de gestion de l'alimentation des animaux. J'ai réutilisé le formulaire existant de l'espace vétérinaire, en effectuant quelques modifications pour l'adapter à l'espace employé. Bien que ce soit essentiellement un copier-coller, j'ai dû ajuster certaines parties, notamment le fonctionnement du bouton de soumission (submit) pour éviter des problèmes de synchronisation.

Lors de la soumission, il y avait un problème de double envoi du formulaire, ce qui posait des soucis de mise à jour des données. Pour résoudre cela, j'ai ajouté une redirection après la soumission du formulaire, afin de rafraîchir la page sans renvoyer les données une

deuxième fois. Cela a permis de stabiliser le processus et d'éviter les erreurs de synchronisation.

En résumé, l'espace employé permet de valider ou invalider des avis et de gérer la distribution de nourriture aux animaux. Bien que la création de cet espace ait été simple et reposait en grande partie sur des fonctionnalités déjà existantes, cela m'a tout de même permis de renforcer mes compétences en réutilisation de code et en gestion des formulaires.

US 8 : Espace Vétérinaire

Utilisateur concerné : Vétérinaire

L'espace vétérinaire permet aux vétérinaires de remplir des comptes rendus sur l'état des animaux, de laisser des commentaires sur les habitats, et de visualiser l'historique de l'alimentation des animaux.

Retour d'expérience

Le développement de l'espace vétérinaire a été une étape importante, car il m'a permis de mieux comprendre la structure des interactions nécessaires entre les différents éléments du site. J'ai utilisé des formulaires HTML et des requêtes SQL pour gérer l'ajout, la modification, et la suppression des données. Initialement, j'avais combiné la prise de nourriture et le rapport vétérinaire, mais j'ai dû ajuster cette approche en séparant les formulaires pour faciliter la saisie des informations.

L'espace vétérinaire a été le premier à être développé parmi les espaces vétérinaire, employé et administrateur. J'ai rencontré plusieurs défis, notamment des problèmes de synchronisation lors de l'envoi des formulaires. Au départ, j'avais tout intégré dans un seul formulaire, mais cela s'est révélé peu pratique. J'ai donc divisé le formulaire en plusieurs parties distinctes (nourriture, habitat, observation), ce qui a permis de mieux gérer les champs requis et de résoudre les problèmes de synchronisation.

Ces défis m'ont permis de renforcer mes compétences en gestion des formulaires et en résolution des problèmes de double soumission. Bien que cela ait demandé plusieurs tentatives et ajustements, c'était une expérience très enrichissante qui m'a préparé pour le développement des autres espaces utilisateurs.

US 9 : Connexion

Utilisateur concerné : Administrateur, vétérinaire, employé

La fonctionnalité de connexion permet aux administrateurs, vétérinaires et employés de se connecter au site web de manière sécurisée. Un utilisateur doit saisir son identifiant (adresse e-mail) et son mot de passe pour accéder aux espaces qui lui sont réservés. Les visiteurs classiques ne peuvent pas se créer de compte.

Retour d'expérience

Au début, la fonctionnalité de connexion était simple et réalisée rapidement après la mise en place de la base de données, principalement pour tester la connectivité. Les mots de passe étaient stockés en clair, sans considération pour la sécurité, car le projet était développé localement à titre d'étude, sans utilisateurs réels. J'ai choisi cette approche pour me concentrer sur la maîtrise des connexions en PHP, que je n'avais pas pratiquées depuis un moment.

Une fois que j'ai commencé à héberger le projet sur Hostinger, la situation a changé en raison de l'exposition publique du site. J'ai dû revoir la connectivité et la sécurité de la base de données, y compris l'ajout de noms d'utilisateur et de mots de passe sécurisés. J'ai remanié l'ensemble des pages du site qui avaient des connexions, pour qu'elles s'adaptent à la base de données PHPMyAdmin de Hostinger. Cela a impliqué de nombreux ajustements, notamment l'importation table par table à cause de la taille des bases de données et la mise en place de types de données appropriés pour gérer les images.

La sécurité a également été améliorée avec l'hébergement public, mais je reconnais que certaines parties ne sont pas optimisées. Par exemple, chaque page a un espace de connexion distinct, ce qui n'est pas idéal d'un point de vue sécurité et performance. À l'avenir, j'opterai pour un fichier externe unique pour la connexion, qui serait appelé depuis chaque page. De même, le bandeau de navigation devrait être un module externe unique plutôt que d'être copié sur chaque page, afin d'améliorer la maintenabilité et l'efficacité du code.

Ce retour d'expérience m'a permis de comprendre l'importance d'une architecture modulaire et d'une gestion centralisée des éléments communs pour réduire la redondance et améliorer les performances.

US 10 : Contact

Utilisateur concerné : Visiteur

La fonctionnalité de contact permet aux visiteurs de contacter le zoo via un formulaire. Le visiteur doit fournir un titre, une description de sa demande, ainsi que son adresse e-mail pour recevoir une réponse. La demande est ensuite envoyée par e-mail au zoo.

Retour d'expérience

La mise en place du formulaire de contact semblait simple au début, mais s'est avérée plus complexe en raison de l'absence de configuration SMTP. Initialement, j'avais prévu de recevoir les demandes sans SMTP, mais après avoir commencé l'hébergement du site sur Hostinger, j'ai pu configurer une adresse liée à l'hébergeur pour gérer l'envoi des e-mails. Cela m'a permis de rendre la page contact pleinement fonctionnelle.

C'est également à ce moment-là que j'ai implémenté l'envoi de mails automatiques lors de la création d'un nouvel utilisateur par José. La configuration SMTP a donc été l'une des dernières étapes du projet, qui a permis de finaliser la fonctionnalité de contact.

US 11 : Statistique de la consultation des habitats

Utilisateur concerné : Visiteur, Administrateur

La fonctionnalité de statistique permet de suivre le nombre de consultations des animaux dans chaque habitat. Chaque fois qu'un visiteur clique sur un animal, une consultation est enregistrée et stockée dans une base de données non relationnelle (MongoDB). Ces données sont ensuite exploitées par l'administrateur via un tableau de bord pour visualiser les animaux les plus populaires.

Retour d'expérience

Le développement de cette fonctionnalité a nécessité l'utilisation de MongoDB pour stocker les consultations de manière dynamique. Initialement, MongoDB était configuré localement, mais des problèmes de synchronisation et de performance sont apparus. En passant à une solution en ligne, la gestion des consultations s'est grandement améliorée.

Lors de l'hébergement sur Hostinger, j'ai réalisé que le module MongoDB n'était pas directement pris en charge. J'ai donc dû trouver une solution alternative, impliquant l'ajout d'une extension MongoDB après contact avec le support technique, ce qui s'est révélé fastidieux. En raison du poids important de MongoDB, j'ai décidé de privilégier une solution en ligne pour alléger le site et faciliter son transfert. La version locale reste disponible sur les commits précédents, mais l'utilisation de la version en ligne s'est avérée plus adaptée à ce projet, compte tenu de la simplicité et du faible nombre de requêtes nécessaires.

Réflexions Initiales, Technologique et Débrief

Contexte du projet

Le projet de développement du site web du zoo Arcadia a commencé par une lecture sur ce que voulait José, le client. Il avait des idées très claires : il voulait que les habitats et les services du zoo soient bien mis en avant, et il souhaitait pouvoir interagir avec les visiteurs via des avis. Dès le début, je me suis dit qu'il serait judicieux de structurer le projet autour de User Stories (US) pour chaque fonctionnalité. Ça m'a aidé à avancer pas à pas, même si j'ai dû adapter ce plan en cours de route, surtout avec la naissance de mon enfant qui a chamboulé un peu mon organisation et mon temps disponible.

Développements et Choix Techniques

Le développement s'est fait étape par étape. J'ai d'abord travaillé sur des pages statiques en HTML et CSS, puis j'ai ajouté des fonctionnalités dynamiques avec JavaScript, PHP, et des bases de données MySQL (avec laquelle j'avais déjà travaillé par le passé) pour la modularité et MongoDB. Chaque US a été abordée une par une, mais je gardais toujours en tête la cohérence du site dans son ensemble.

Pour l'hébergeur j'ai choisi Hostinger en fin de parcours, j'ai déjà travaillé avec eux et je n'ai pas été déçu, seul bémol : MongoDB n'est pas fonctionnel nativement sur Hostinger, j'ai du adapter mon choix technologique vers une solution Online de MongoDB, ce qui après coup m'arrange au du poid de l'outils (pas très optimisé pour les petits site, 90% du poid du site ne servant qu'à une petite fonctionnalité)

Le SMTP a été celui de l'hébergeur, ne souhaitant pas mettre un serveur SMTP en place, en sachant qu'il devrait changer pour celui de l'Hébergeur et ne souhaitant pas adopter une solution annexe (google, microsoft, aws etc), la data sera supprimé (comprenant le email et ce qu'il contient) lors de la suppression de ce projet.

Il y a eu un autre point très révélateur pour moi : la question de la modularité. Je me suis rendu compte que certaines parties du code, comme la barre de navigation ou les formulaires de connexion, auraient été beaucoup plus faciles à maintenir si je les avais rendues modulaires dès le début.

Retour sur le Processus et Enseignements

Ce projet m'a vraiment appris l'importance d'une architecture modulaire et flexible, surtout pour un projet qui est amené à évoluer. Avec le recul, certaines décisions que j'ai prises pour aller plus vite se sont révélées problématiques. J'ai compris que la modularité aide non seulement à éviter la redondance, mais aussi à améliorer les performances.

Passer d'une version locale à un hébergement en ligne m'a aussi fait réaliser des choses auxquelles je ne m'attendais pas, comme la gestion des extensions et l'indisponibilité de certain modules, l'imposition de norme de sécurité (par exemple pour PhpMyadmin, l'impossibilité de configurer des users particuliers pour la connectique. Ça a été un vrai apprentissage sur l'écosystème d'hébergement et la sécurité des applications.

Prochaines Étapes

Pour mes futurs projets, je veux vraiment me concentrer sur l'optimisation des performances, avec des pratiques comme le pré-calcul côté serveur, et une meilleure modularité. Le but est d'avoir un code plus efficace, moins redondant, et d'améliorer l'expérience pour tout le monde, que ce soit les visiteurs du site ou les administrateurs qui doivent le gérer.

Retour Personnel

Je viens d'un milieu orienté système d'information et transformation digitale, donc la réflexion autour des besoins de l'utilisateur et les User Stories, c'est un peu mon terrain de jeu habituel. Mais je n'avais pas le retour d'expérience pratique d'un développeur chevronné, et ça s'est vu. Quand j'ai commencé, j'avais cette vision très séquentielle : d'abord HTML, puis CSS, ensuite la base de données, etc. Mais en réalité, ça ne marche pas toujours comme ça. La prochaine fois, je commencerai par réfléchir à la structure de communication entre les différentes parties du site, à la logique des fonctionnalités, et à la modularité pour rendre le code plus simple à maintenir.

Il y a eu des moments où je me suis dit : "Ah, ça ne va pas du tout, il faut tout refaire". Et là, ce n'était pas évident, surtout en étant devenu papa en plein milieu du projet. Refaire des choses, apprendre de nouvelles approches, puis les appliquer, ça prend du temps et de l'énergie, mais c'est aussi ce qui rend l'apprentissage intéressant. Je sais que je referai des erreurs, mais c'est ça qui fait avancer.

Le code, on l'a vu, est assez brouillon. J'ai testé beaucoup de choses différentes, et à la fin, c'était difficile de m'y retrouver. Pour la prochaine fois, je vais vraiment me concentrer sur la modularité et l'importance des commentaires. J'en ai mis beaucoup, mais parfois, ils ne veulent plus rien dire parce que j'ai supprimé certaines parties du code sans mettre à jour les commentaires. Le prochain projet sera, je l'espère, beaucoup plus organisé. Cette organisation ne dépend que de moi, et je compte bien y parvenir.

Je veux aussi, pour le futur, créer des fiches de conception plus poussées, surtout sur l'interaction entre les pages. Ce n'était pas demandé pour ce projet, mais je pense que c'est important de savoir pourquoi et comment une page interagit avec une autre, notamment pour la cohérence du système d'information. Pour le prochain projet, je prendrai le temps de m'amuser avec cette idée, et j'espère pouvoir vous montrer le résultat lors de mon passage à l'examen.

J'ai aussi un regret, mais j'ai dû faire des choix stratégiques en termes de timing. Avec le boulot de papa, c'était compliqué, et il y a une fonctionnalité que je n'ai pas pu ajouter, et je le regrette : permettre à l'administrateur de consulter les rapports vétérinaires.

En fait, si je ne l'ai pas ajouté, ce n'est pas par manque de compétences, car je suis là pour apprendre et acquérir ces compétences au travers de ce projet. Le vrai problème est que j'ai conçu les rapports vétérinaires sans avoir en tête qu'il fallait conserver un historique (souvenez vous quand je vous disais que j'avais vu le projet en séquences US après US). Quand le vétérinaire fait un rapport, il sélectionne un animal et met à jour ses données (ce

qu'il mange, combien, etc.). Du coup, il n'y a pas de notion d'historique, et c'est là où ça coince.

Pour intégrer l'historique, il faudrait redéfinir toute la partie sur les observations. Ça pourrait être une évolution future, une V0.2 de cette fonctionnalité. Si je devais la remanier, je sais exactement ce qu'il faudrait faire : créer une table d'observations séparée qui contiendrait les détails, comme l'ID de l'observation, l'animal concerné (clef étrangère associé aux ID des animaux [pidAnimal]) et la date. Ensuite, l'administrateur pourrait sélectionner un animal, puis choisir une date pour afficher les détails de l'observation. Ce n'est pas un manque de savoir-faire, mais juste un manque de temps. Il me reste 4 heures avant de rendre ce projet, et je cours après le temps. Donc, j'ai fait le choix d'expliquer ici ce que j'aurais fait plutôt que de l'implémenter maintenant.