Réflexion avant-projet

Avant de débuter le développement, nous avons entrepris une analyse des sites de zoos existants afin de nous inspirer des meilleures pratiques. Cette recherche a révélé que ces sites mettent souvent l'accent sur la simplicité, en privilégiant la mise en avant des images, en particulier des photos des animaux et des installations du zoo. Ces éléments visuels jouent un rôle central dans l'expérience utilisateur, permettant d'engager les visiteurs de manière immédiate et intuitive.

Compte tenu de ces observations, j'ai décidé de concevoir un site web qui adopte une approche minimaliste, avec une interface utilisateur épurée et une navigation fluide. L'objectif est de créer une expérience utilisateur à la fois simple et immersive, en mettant en avant les images et les informations essentielles sans surcharger l'utilisateur.

Choix technologiques

Pour l'environnement de travail, mon choix s'est porté sur **React**, et plus spécifiquement **Next.js**. Ce framework, développé par Vercel et largement utilisé par des géants comme Facebook, est reconnu pour sa robustesse et sa flexibilité. L'écosystème React/Next.js bénéficie d'une documentation exhaustive et d'une large communauté, ce qui facilite l'apprentissage et le partage de bonnes pratiques. De plus, l'adoption de React par des leaders technologiques tels que Facebook témoigne de la pérennité et de la fiabilité de ce framework. En choisissant cette technologie, je m'assure non seulement d'utiliser un outil performant, mais aussi un environnement soutenu par une entreprise stable, garantissant ainsi la pérennité de la technologie utilisée.

Front-end

Pour la partie front-end, **Next.js** s'impose également comme un choix stratégique. Grâce à sa capacité à gérer à la fois le rendu côté serveur (SSR) et la génération de sites statiques (SSG), il offre des performances optimisées, essentielles pour un site centré sur l'expérience utilisateur. De plus, pour accélérer le développement et garantir une interface utilisateur cohérente, j'ai opté pour **shadcn/ui**, une solution qui intègre **Tailwind CSS**. Ce framework CSS utilitaire permet de créer des interfaces rapidement tout en assurant une grande flexibilité de personnalisation.

Back-end

Du côté back-end, **Next.js** offre une intégration étroite avec le front-end via ses API REST. Cette approche intégrée permet de réduire les frictions entre les équipes front-end et back-end, assurant ainsi une communication fluide entre les deux parties de l'application. Les données seront principalement gérées via **MySQL** ou **MariaDB** en environnement local, avec une transition vers **PostgreSQL** pour la production. Pour les fonctionnalités de stockage de données NoSQL et le stockage de fichiers binaires, j'utiliserai **Firebase Firestore** et **Firebase Storage**.

Conclusion

En combinant ces technologies, je vise à créer un site web moderne, performant et évolutif. L'utilisation de **React** avec **Next.js** assure une cohérence et une maintenabilité du code, tandis que les choix de bases de données garantissent une gestion des données adaptée aux besoins spécifiques du projet. Enfin, le déploiement sur **Vercel** permettra de bénéficier de pipelines CI/CD intégrés, assurant des mises à jour rapides et sans interruption.

Environnement de travail :

Front : HTML 5, tailwind, Typescript, Javascript, react avec next.js

Back-end : Next.js avec api Rest

Base de données relationnelle : MySQL, MariaDB en local et PostgreSQL en production

Base de données NoSQL : Firebase

Base de donnée Storage : Firebase-Storage

Déploiement :vercel

Front-end :

- **HTML 5**: Standard pour la structure des pages web.
- **Tailwind CSS**: Framework CSS utilitaire pour la conception rapide et flexible des interfaces utilisateur.
- **TypeScript**: Langage de programmation typé qui améliore la maintenabilité et la fiabilité du code JavaScript.
- **JavaScript**: Langage principal pour le développement de scripts côté client.
- **React avec Next.js**: Framework React utilisé pour le développement côté client avec des capacités avancées de rendu côté serveur (SSR) et de génération de sites statiques. De plus React est un langage moderne développer par les ingénieurs de facebook on peut estimer qu'il tiendra dans le temps. Enfin la recherche de ressource que ce soit sur react natif ou next.js est très accessible que ce soit par la documentation complète de ces derniers ou par le partage des utilisateurs.

Back-end :

- **Next.js avec API REST**: Next.js peut être utilisé pour gérer à la fois le rendu côté serveur et la création d'API RESTful, ce qui permet une intégration étroite entre le front-end et le back-end.

Bases de données :

- **MySQL/MariaDB en local et PostgreSQL en production**: Utilisation de bases de données relationnelles populaires adaptées à différents environnements de développement et de production.
- **Firebase Firestore**: Base de données NoSQL pour le stockage de données non structurées et évolutives.
- **Firebase Storage**: Stockage de fichiers binaires comme des images, des vidéos, etc., intégré avec Firebase.

Déploiement

- **Vercel**: Plateforme de déploiement spécialisée pour les applications front-end et les sites Next.js, offrant une intégration continue (CI/CD) et des déploiements rapides.

Conclusion

Mon choix d'utiliser TypeScript avec Next.js pour le front-end et le back-end assure une cohérence dans les types de données et les interfaces à travers toute l'application. Cette approche garantit une meilleure maintenabilité et une réduction des erreurs grâce à la vérification statique des types.

L'adoption simultanée de bases de données relationnelles comme MySQL/MariaDB et PostgreSQL pour la production, ainsi que Firebase Firestore pour les besoins NoSQL, offre une flexibilité optimale dans la gestion des données. De plus, l'intégration de Firebase Storage pour le stockage de fichiers enrichit cette palette d'outils.

En résumé, mon environnement de développement est conçu pour répondre aux exigences modernes des applications web, en combinant des technologies adaptées à chaque aspect de mon projet. Cela garantit une performance optimale, une évolutivité future et une expérience utilisateur enrichie.

Les ressources utilisés lors projet :

initialisation du projet :

npx create-next-app@latest my-app --typescript --tailwind –eslint

→ Ajoute les dépendence suivantes :

- react
- react-dom
- next

→ Ajoute les dépendences dev suivantes :

- typescript
- @types/node
- @types/react
- @types/react-dom
- postcss
- tailwindcss
- eslint
- eslint-config-next

npx shadcn-ui@latest init

→ Ajoute les dépendence suivantes :

- clsx
- lucide-react
- tailwind-merge
- tailwindcss-animate

Les ressources back-end

→ Ajoute les dépendence suivantes :

- bcryptjs
- dotenv
- firebase
- jsonwebtoken
- pg
- mysql2
- nodemailer
- sequelize
- sequelize-cli

→ Ajoute les dépendences dev suivantes :

- @types/bcryptjs
- @types/jsonwebtoken
- @types/pg
- @types/nodemailer

Les ressources front-end

→ Ajoute les dépendence suivantes :

- react-icons
- framer-motion
- react-toastify

→ Ajoute les dépendences dev suivantes :

- @types/react-toastify

Récapitulatif de toutes les dépendances :

Front-end

Dépendances :

react :

- **Description** : Librairie JavaScript pour construire des interfaces utilisateur dynamiques.
- **Commandes** : `npm react`

react-dom :

- **Description** : Fournit des méthodes spécifiques au DOM pour manipuler les éléments React.
- **Commandes** : `npm react-dom`

react-icons :

- **Description** : Collection d'icônes pour React, offrant un accès à des milliers d'icônes de bibliothèques populaires.
- **Commandes** : `npm react-icons`

framer-motion :

- **Description** : Bibliothèque pour les animations fluides et interactives dans les applications React.
- **Commandes** : `npm framer-motion`

react-toastify :

- **Description** : Affichage de notifications toast pour React.
- **Commandes** : `npm react-toastify`

Dépendances de développement :

@types/react :

- **Description** : Types TypeScript pour la librairie React.
- **Commandes** : `npm @types/react`

@types/react-dom :

- **Description** : Types TypeScript pour la librairie `react-dom`.
- **Commandes** : `npm @types/react-dom`

@types/react-toastify :

- **Description** : Types TypeScript pour la bibliothèque `react-toastify`.
- **Commandes** : `npm @types/react-toastify`

eslint :

- **Description** : Outil de linting pour identifier et corriger les problèmes dans le code JavaScript/TypeScript.
- **Commandes** : `npm eslint`

eslint-config-next :

- **Description** : Configuration ESLint recommandée pour les projets Next.js.
- **Commandes** : `npm eslint-config-next`

postcss :

- **Description** : Outil pour transformer les styles CSS avec des plugins JavaScript.
- **Commandes** : `npm postcss`

tailwindcss :

- **Description** : Framework CSS utilitaire pour la création rapide d'interfaces utilisateur.
- **Commandes** : `npm tailwindcss`

typescript :

- **Description** : Superset typé de JavaScript qui compile en JavaScript.
- **Commandes** : `npm typescript`

ts-node :

- **Description** : Exécute TypeScript directement dans Node.js.
- **Commandes** : `npm ts-node`

Back-end

Dépendances :

next :

- **Description** : Framework React avec rendu côté serveur et génération de sites statiques.
- **Commandes** : `npm next`

bcryptjs :

- **Description** : Librairie JavaScript pour hacher les mots de passe.
- **Commandes** : `npm bcryptjs`
-

dotenv :

- **Description** : Charge les variables d'environnement à partir d'un fichier `.env`.
- **Commandes** : `npm dotenv`

firebase :

- **Description** : SDK pour utiliser les services Firebase comme Firestore, Auth, et Storage.
- **Commandes** : `npm firebase`

jsonwebtoken :

- **Description** : Librairie pour créer et vérifier les JSON Web Tokens (JWT).
- **Commandes** : `npm jsonwebtoken`

pg :

- **Description** : Client PostgreSQL pour Node.js.
- **Commandes** : `npm pg`

mysql2 :

- **Description** : Client MySQL pour Node.js, avec support des promesses.
- **Commandes** : `npm mysql2`

nodemailer :

- **Description** : Librairie pour envoyer des emails depuis Node.js.
- **Commandes** : `npm nodemailer`

sequelize :

- **Description** : ORM pour Node.js, facilitant les interactions avec les bases de données relationnelles.
- **Commandes** : `npm sequelize`

sequelize-cli :

- **Description** : Interface en ligne de commande pour Sequelize.
- **Commandes** : `npm sequelize-cli`

Dépendances de développement :

@types/bcryptjs :

- **Description** : Types TypeScript pour la librairie `bcryptjs`.
- **Commandes** : `npm @types/bcryptjs`

@types/jsonwebtoken :

- **Description** : Types TypeScript pour la librairie `jsonwebtoken`.
- **Commandes** : `npm @types/jsonwebtoken`

@types/pg :

- **Description** : Types TypeScript pour le client `pg`.
- **Commandes** : `npm @types/pg`

@types/nodemailer :

- **Description** : Types TypeScript pour la librairie `nodemailer`.
- **Commandes** : `npm @types/nodemailer`

@types/node :

- **Description** : Types TypeScript pour les fonctionnalités Node.js.
- **Commandes** : `npm @types/node`

Sécurité

La sécurité de l'environnement :

1. ### Next.js

→ **Rendu côté serveur (SSR) :** Next.js vous permet de rendre les pages côté serveur, ce qui réduit la surface d'attaque pour les scripts malveillants comme le Cross-Site Scripting (XSS). SSR génère du HTML côté serveur, ce qui signifie que les données sensibles peuvent être sécurisées avant d'arriver sur le client.

→ **API Routes sécurisées :** Next.js permet de créer des API routes internes sécurisées, où les données sont traitées sur le serveur. Ces routes peuvent être protégées par des mécanismes d'authentification et d'autorisation robustes. Protection automatique contre les XSS : Next.js échappe automatiquement les données rendues dans les balises HTML, limitant ainsi les risques de XSS.

2. ### TypeScript

→ **Typage statique :** TypeScript réduit les erreurs de programmation courantes en offrant un typage statique. Cela aide à éviter les bugs liés à la manipulation des types de données qui peuvent conduire à des vulnérabilités.

→ **Renforcement du code :** Le typage statique de TypeScript aide à prévenir les erreurs inattendues en s'assurant que les variables, les fonctions et les objets sont utilisés de manière cohérente. Cela renforce la robustesse du code et réduit la probabilité de failles de sécurité.

→ **Meilleure lisibilité et maintenabilité :** TypeScript rend le code plus lisible et plus maintenable, ce qui facilite la détection et la correction des vulnérabilités par les développeurs.

3. Sequelize ORM

→ **Protection contre l'injection SQL :** Sequelize, en tant qu'ORM, échappe automatiquement les entrées des utilisateurs avant de les inclure dans les requêtes SQL, ce qui protège contre les attaques par injection SQL.

→ **Gestion des permissions et des transactions :** Sequelize permet de mettre en place facilement des systèmes de gestion des permissions et des t ransactions, garantissant que seules les actions autorisées sont effectuées et que

les transactions sont traitées de manière sûre.

→ **Validation des données :** Sequelize offre des mécanismes de validation des données au niveau du modèle, empêchant ainsi les données invalides ou malveillantes d'entrer dans la base de données.

Conclusion sur la sécurité l'environnement

En combinant Next.js, shadcn/ui, TypeScript, et Sequelize, vous créez une application avec des couches supplémentaires de sécurité inhérente à chaque technologie. Ensemble, elles permettent de minimiser les risques liés aux injections, aux XSS, aux erreurs de validation de données, et aux bugs courants tout en offrant une base solide pour une application maintenable et sécurisée.

Les sécurité de l'application :

1. Utilisation de routes sécurisées avec un système de contexte : `UserContext.tsx`

c'est un context qui :

- **Gestion de la connexion** : Le `UserContext` vérifie si l'utilisateur est connecté en consultant le token stocké dans le local storage.
- **Décodeur de token** : Le token est décodé pour déterminer le rôle de l'utilisateur (employee, veterinarian, admin).
- **Gestion des accès** : En fonction du rôle, les droits d'accès aux différentes parties de l'application sont définis. Si l'accès à une route est refusé, l'utilisateur est redirigé vers une page appropriée, telle qu'une page de connexion ou une page d'erreur.

2. Création de compte et connexion :

Création de compte :

- Seule l'admin peut créer des comptes. Les comptes peuvent uniquement être de type employee ou veterinarian.
- Les comptes admin doivent être ajoutés directement dans la base de données par des moyens administratifs.
- Hashage du mot de passe : Lors de la création du compte, le mot de passe est

hashé à l'aide de la bibliothèque *bcryptjs* par la fonction *hashPassword()* dans *passwordsUtils.js*. La fonction *hashPassword()* assure que les mots de passe ne sont pas stockés en clair, tandis que la fonction *comparePassword()* est utilisée pour comparer les mots de passe lors de la connexion.

Connexion :

- Les utilisateurs se connectent en fournissant leur email et leur mot de passe non hashé. Le mot de passe fourni est comparé au mot de passe hashé stocké en base de données à l'aide de *comparePassword()*.
- Si la comparaison est valide, un jeton JWT est généré pour l'utilisateur et stocké dans le local storage. Ce jeton contient des informations telles que l'email et le rôle de l'utilisateur, permettant de maintenir la session de manière sécurisée.

3. Sécurité des API

- Typage fort avec TypeScript : Les API sont sécurisées grâce à TypeScript, qui offre un typage statique robuste. Cela réduit les erreurs de type et assure que les données envoyées et reçues respectent les formats attendus.
- Validation des données : Le système de validation des données est intégré à l'API. Par exemple :
  - Si une API attend un texte de longueur minimale de 4 caractères et maximale de 50 caractères, la validation des données garantit que les contraintes sont respectées.
  - Les erreurs de type, comme l'envoi d'un texte à la place d'un nombre, sont détectées et renvoient des erreurs appropriées.
- Validation côté client et serveur : La validation des données est également implémentée sur les pages front-end pour éviter des erreurs avant que les données ne soient envoyées à l'API.

Conclusion

En mettant en œuvre ces mesures de sécurité, l'application Arcadia est protégée contre diverses menaces courantes. La gestion sécurisée des rôles et des accès, le stockage sécurisé des mots de passe, et la validation rigoureuse des données contribuent à assurer une sécurité robuste et une protection efficace des informations sensibles. L'utilisation de technologies telles que Next.js, shadcn/ui, TypeScript, et Sequelize ORM ajoute une couche supplémentaire de sécurité en assurant des pratiques de codage sûres et en réduisant les risques de vulnérabilités.