

Dossier projet



Nom de naissance : SAEZ

Prénom : Fanny

Adresse : 143 Bis Boulevard Lafayette – 63000 Clermont-Ferrand

Titre professionnel visé : développeur-se web et web mobile - niveau III

Modalité d'accès :

- Parcours de formation
- Validation des Acquis de l'Expérience (VAE)

SOMMAIRE

SOMMAIRE	2
INTRODUCTION	4
COMPETENCES COUVERTES PAR LE PROJET	5
I. Développer la partie front-end d'une application web ou web mobile sécurisée	
✓ Installer et configurer son environnement de travail	
✓ Maquetter des interfaces utilisateur web	
✓ Réaliser des interfaces utilisateur statiques	
✓ Développer la partie dynamique des interfaces utilisateur web	
II. Développer la partie back-end d'une application web ou web mobile sécurisée	
✓ Mettre en place une base de données relationnelle	
✓ Développer des composants d'accès aux données SQL	
✓ Développer des composants métier coté serveur.....	
✓ Documenter le déploiement d'une application dynamique	
RESUME DU PROJET	6
CAHIER DES CHARGES	7
I. Objectifs du projet 9	
a) Objectifs généraux	
b) Évolutions prévues	
II. Stack technique 10	
a) Next.js	
b) Prisma & Néon (PostgreSQL)	
c) JWT & BcryptJS	12
d) Email.Js	13
e) Resend	

SOMMAIRE

III. Outils	14
a) Méthodologie & Trello	15
b) Git	16
c) GitHub	
IV. conceptions	19
a) Diagramme UML	
b) Réalisations de la maquette	21
V. Architecture générale du projet	22
a) Architecture projet Next.js	
b) Schéma Next.js	23
c) App Router	
VI. Développement technique	25
a) Front-end	
b) Back-end	28
c) API	31
VII. Fonctionnalités principales	35
a) Espace administrateur	
b) Espace utilisateur	37
VIII. Authentification & Sécurité	39
IX. Tests & Validation	41
X. Déploiement	44
XI. Conclusion	47
Annexes	49

INTRODUCTION

Depuis plusieurs années, je m'intéresse de près aux technologies numériques et à la création web. Issue à l'origine d'un parcours dans le secrétariat et la communication visuelle, j'ai toujours continué, en parallèle de mon activité professionnelle, à me former de manière autonome au développement web et mobile, aux CMS comme WordPress et aux bonnes pratiques du design numérique. Cet apprentissage personnel m'a permis de réaliser des projets bénévoles, de tester différentes approches créatives et techniques et de rester en veille sur l'évolution du secteur.

Animée par l'envie de donner un nouveau souffle à ma carrière, j'ai entrepris une reconversion professionnelle et choisi d'intégrer une formation intensive en développement web et mobile. Cette étape m'a permis de consolider mes acquis, d'approfondir mes connaissances techniques et d'aborder des thématiques telles que l'éco-conception et l'accessibilité lors de Hackathon et de projets collectifs.

J'ai également eu l'opportunité d'effectuer un stage de 2,5 mois en Freelance, une expérience extrêmement enrichissante qui m'a permis de développer la partie front-end et back-end d'applications web et mobiles sécurisées, de configurer des environnements de travail adaptés, de maquetter et de documenter mes réalisations. Cette immersion dans un contexte professionnel réel m'a apporté une vision concrète des contraintes, de l'organisation et de la communication nécessaires à la réussite d'un projet numérique.

Au-delà de l'acquisition de nouvelles compétences, mon objectif était aussi d'apprendre à travailler en groupe sur un projet. Autodidacte et à mener mes propres réalisations, je tenais à découvrir les bonnes pratiques d'une organisation collective : gestion du temps, répartition des tâches, communication au sein de l'équipe et coordination technique.

En somme, cette reconversion, combinant formation, projets collaboratifs et stage en freelance, m'a permis d'ancrer mes savoir-faire dans un cadre professionnel et d'expérimenter le développement web sous un angle plus structuré et collaboratif, ce qui correspond aujourd'hui à l'orientation que je souhaite donner à mon parcours dans le numérique.

COMPETENCES COUVERTES PAR LE PROJET

I. Développer la partie front-end d'une application web ou web mobile sécurisée

- ✓ Installer et configurer son environnement de travail en fonction du projet
- ✓ Maquette des interfaces utilisateurs
- ✓ Réaliser des interfaces utilisateurs statiques
- ✓ Développer la partie dynamique des interfaces

II. Développer la partie back-end d'une application web ou web mobile sécurisée

- ✓ Mettre en place une base de données relationnelle
- ✓ Développer des composants d'accès aux données SQL
- ✓ Développer des composants métiers côté serveur
- ✓ Développer le déploiement d'une application dynamique

J'ai mis en place un environnement **Next.js/React** pour bénéficier du rendu côté serveur. J'ai conçu et intégré des interfaces responsives avec **CSS Modules**, géré les formulaires avec **React Hook Form** et Zod, uniformisé l'interface avec **React Icons** et ajouté l'envoi d'e-mails côté front via **Email.js** pour améliorer l'expérience utilisateur.



J'ai mis en place une base de données relationnelle **PostgreSQL (Neon Tech)** et simplifié son accès avec **Prisma**. J'ai développé les composants métiers via les **API Routes** de Next.js, sécurisé l'authentification avec **BcryptJs** et **JWT**, configuré l'envoi d'e-mails transactionnels avec **Resend**, testé les **API avec Postman** et déployé l'application dynamique sur **Vercel** pour un hébergement performant et documenté.



RESUME DU PROJET

La cliente, praticienne expérimentée dans le domaine du bien-être, exprimait le besoin urgent de moderniser sa présence digitale pour développer son activité. Elle souhaitait dépasser le simple site vitrine statique pour créer une véritable plateforme interactive qui reflète le professionnalisme et la qualité de ses services. Son principal défi était de proposer une expérience client complète, depuis la découverte de ses services jusqu'à la réservation de consultations, tout en développant une nouvelle source de revenus via des formations en ligne.

L'enjeu majeur résidait dans l'automatisation de sa gestion administrative, particulièrement chronophage dans son activité. Elle avait besoin de libérer du temps pour se concentrer sur ses consultations plutôt que sur les tâches de planification, de suivi client et de gestion de contenu. La praticienne souhaitait également pouvoir proposer des formations à distance de manière sécurisée, avec un système de paiement intégré et un suivi personnalisé des apprenants.

Au niveau technique, elle exigeait une solution robuste et évolutive, capable de gérer simultanément les aspects commerciaux, pédagogiques et administratifs de son activité. La sécurisation des données clients et des contenus de formation constituait une priorité absolue, de même que l'optimisation pour les moteurs de recherche afin d'attirer de nouveaux clients.

Elle souhaitait également une interface d'administration intuitive lui permettant de gérer autonomement l'ensemble de sa plateforme sans compétences techniques particulières.

Cette application répond donc à un besoin global de digitalisation d'une activité de bien-être traditionnelle, en créant un écosystème numérique complet qui valorise l'expertise de la praticienne tout en optimisant l'expérience client et la gestion opérationnelle.

CAHIER DES CHARGES

Objectifs

L'objectif principal d'Alexia Énergies est de digitaliser complètement l'activité d'une praticienne en bien-être en créant un écosystème numérique tout-en-un. Cette plateforme vise à automatiser la prise de rendez-vous, professionnaliser la présence en ligne, et développer une nouvelle source de revenus via des formations sécurisées, tout en centralisant la gestion administrative dans une interface unique et intuitive.

User Stories

Le visiteur peut naviguer librement sur le site public après avoir découvert les services et contenus. Il peut consulter les articles, contacter la praticienne via le formulaire, et prendre rendez-vous via l'interface Calendly intégrée. L'utilisateur peut créer un compte.

L'utilisateur connecté, bénéficie de toutes les fonctionnalités visiteur plus l'accès à son espace personnel. Il peut ajouter des articles en favoris, gérer sa liste de favoris avec pagination, et se déconnecter.

L'administrateur, correspond à la praticienne propriétaire de l'application web. Elle a accès à toutes les fonctionnalités utilisateur plus l'interface d'administration complète. Elle peut gérer les articles (CRUD complet), consulter et modifier les utilisateurs inscrits, et gérer la newsletter.

Technologies utilisées

➤ Developer Experience & Gestionnaire de paquets

- **Yarn** – *Gestionnaire de paquets rapide et fiable*
- **ESLint** – *Linter JavaScript pour la qualité du code*
- **Prettier** – *Formateur automatique pour la cohérence de style*

➤ Front-end

- **React**⁽¹⁹⁾ – *Bibliothèque pour interface utilisateur*
- **CSS Modules** – *Styles isolés par composant*
- **React Icons** – *Bibliothèque d'icônes*
- **Embla Carousel** – *Carrousels d'images*

- **Back-end**
 - **Next.js API Routes** – API REST *intégrée*
 - **Prisma ORM** – Gestion de *base de données*
 - **Neon PostgreSQL** – Base de données *cloud*
 - **BcryptJS** – Chiffrement des *mots de passe*
 - **JsonWebToken** – *Authentification* sans état

- **Services externes**
 - **Cloudinary** – *Hébergement et optimisation* d’images
 - **Calendly** – Système de *prise de rendez-vous*
 - **EmailJS** – Envoi d’emails *côté client*
 - **Resend** – Emails transactionnels *côté serveur*
- **Déploiement et outils**
 - **Vercel** – *Hébergement et déploiement* continu
 - **Git** – *Versioning*
 - **GitHub** – *Hébergement & mutualisation*
 - **React Hook Form + Zod** – Gestion et validation des formulaires

Fonctionnalités de l’application

- **Site dynamique professionnel**
 - Présentation générale et biographie de la praticienne
 - Pages dédiées pour chaque service (magnétisme, sophrologie, human design)
 - Blog avec articles de bien-être, système de catégories et compteur de vues
 - FAQ avec questions/réponses fréquentes
 - Formulaire de contact avec envoi d’emails automatisé
- **Système de rendez-vous**
 - Intégration Calendly pour la prise de rendez-vous en ligne
 - Choix entre consultation en présentiel ou à distance
 - Redirection automatique vers l’interface de planification

- ***Gestion des utilisateurs***
 - Inscription et connexion sécurisées
 - Système de favoris pour sauvegarder des articles
 - Réinitialisation de mot de passe par email
- ***Interface d'administration***
 - Gestion complète des articles (CRUD)
 - Administration des newsletter (consultation, modification, suppression)
 - Dashboard avec navigation par sections
- ***Fonctionnalités techniques avancés***
 - Upload et optimisation d'images avec Cloudinary
 - Emails transactionnels automatisés
 - Application web et web mobile responsive (desktop/mobile)
 - Optimisation SEO (balises méta, performances)
 - Sécurisation des accès et protection des données

I. OBJECTIFS DU PROJET

a) Des évolutions prévues - Phase 2 (Future – planifiée)

1. Espace formation en ligne

- Achat de formations via Stripe (paiement sécurisé)
- Création automatique de compte utilisateur après paiement
- Authentification renforcée : BcryptJs + JsonWebToken
- Espace personnel sécurisé avec accès aux contenus payants

2. Gestion des chapitres de formation

Cette application va être améliorée avec une plateforme de formation en ligne avec un contenu payants.

3. Système de contenus multimédia

- Textes enrichis avec éditeur WYSIWYG
- Vidéos intégrées (hébergement Cloudinary/Vimeo)
- Audios de méditation/sophrologie

- Fichiers PDF téléchargeables (guides, exercices)

4. Interface utilisateur formation

5. Intégration Stripe Webhooks (nouvelle fonctionnalité)

- Gestion automatisée des paiements et accès

II. STACK TECHNIQUE

a) Next.js¹

J'ai choisi une techno « Next.js » en vue de mon stage que j'ai effectuée pendant les 2,5 mois de stage.

Next.js est une solution idéale pour développer des projets modernes et interactifs grâce à son intégration avec React, qui permet de concevoir des interfaces fluides, rapides et réactives. En réunissant front-end et back-end dans un même environnement, il facilite la gestion, la maintenance et la collaboration, tout en offrant un SEO optimisé via le SSR et le SSG pour un meilleur référencement et une performance accrue. Son écosystème riche, avec une large communauté et des composants réutilisables, accélère le développement et l'intégration de fonctionnalités avancées.

De plus, son déploiement est simplifié grâce à des plateformes comme Vercel, rendant le processus rapide et efficace. Ces atouts font de Next.js un choix parfaitement adapté pour un projet nécessitant une interface moderne, responsive et riche en interactions, comme un site vitrine, un espace client, un blog ou une plateforme de réservation.

b) Prisma(ORM) & Neon Tech (PostgreSQL)

Comprendre Prisma(ORM)² avec Next.js et Neon PostgreSQL. Qu'est-ce que Prisma ORM et pourquoi l'utiliser ?

Prisma est un **Object-Relational Mapping** (ORM) moderne qui fait le pont entre votre code JavaScript et votre base de données. Imaginez que vous voulez parler à quelqu'un qui ne

¹ **Next.js** : Framework basé sur React qui facilite la création de sites web rapides, optimisés et structurés.

² **Prisma(ORM)** : est un outil moderne pour JavaScript/TypeScript qui simplifie l'accès aux bases de données en générant un client typé et en gérant automatiquement les migrations.

parle pas votre langue – vous auriez besoin d'un traducteur. C'est exactement ce que fait Prisma : il traduit vos instructions JavaScript en requêtes SQL que votre base de données PostgreSQL peut comprendre.

Pourquoi utiliser un ORM comme Prisma ?

```
SELECT
  a.id, a.title, a.slug, a.description, a.image,
  c.name AS category_name,
  f.createdAt AS favorite_date
FROM articles a
INNER JOIN favorites f ON a.id = f.articleId
INNER JOIN users u ON f.userId = u.id
LEFT JOIN category c ON a.categoryId = c.id
WHERE u.email = 'fanny.saez.0486@gmail.com'
  AND a.status = 'published'
ORDER BY f.createdAt DESC;
```

Sans ORM, pour récupérer un utilisateur et ses articles dans votre projet Next.js, vous devriez écrire du SQL brut comme ceci :

Prisma est un outil puissant pour un projet Next.js, car il simplifie considérablement la gestion des bases de données. Il permet

d'écrire des requêtes en JavaScript intuitif plutôt qu'en SQL complexe, tout en validant ces requêtes à l'avance pour éviter les erreurs. Il prend aussi en charge automatiquement les relations entre les tables, comme celles des utilisateurs, articles ou commentaires, sans nécessiter de manipulation manuelle des jointures. Enfin, Prisma traduit directement les modèles définis dans ses fichiers en requêtes SQL, ce qui facilite la création et l'évolution de la base de données.

Qu'est-ce que Néon Tech³, base de données PostgreSQL⁴ ?

Neon Tech est une plateforme cloud qui repose sur PostgreSQL, l'une des bases de données les plus solides et populaires, adaptée aux applications modernes comme Next.js.

Pourquoi ?

Elle simplifie l'utilisation de PostgreSQL avec un déploiement en quelques minutes, un **scaling automatique** selon le trafic, et un **mode veille économique** pour optimiser les coûts.

À quoi ça sert ?

³ **Neon Tech** : est un service cloud qui fournit un PostgreSQL managé, scalable et facile à utiliser.

⁴ **PostgreSQL** : est un **système de gestion de base de données relationnelle (SGBDR) open-source**, puissant et extensible, compatible SQL et orienté vers la fiabilité et la performance.

Elle permet de gérer toutes les données du projet (utilisateurs, produits, commandes) avec la fiabilité de PostgreSQL, tout en offrant une innovation clé : les **branches de données**, qui permettent de tester sans risque, à la manière de Git pour le code.

JWT (JSON Web Tokens)⁵ et bcrypt.js dans mon projet Next.js Qu'est-ce que JWT et pourquoi l'utiliser ?

c) Jwt (JasonWebToken) & bcrypt.js

JWT : Une carte d'identité numérique

JWT est un token signé cryptographiquement qui contient les informations utilisateur (ID, permissions). Il fonctionne comme une carte d'identité électronique vérifiable sans interroger la base de données.

Pourquoi JWT ?

Il évite de stocker les sessions côté serveur. Le token signé est infalsifiable, permettant au serveur de faire confiance aux informations qu'il contient. Résultat : application plus rapide et moins gourmande en ressources.

Usage concret

Connexion → génération du token → stockage navigateur → présentation automatique à chaque requête. Cela sécurise les pages privées, protège les API et maintient la connexion utilisateur sans solliciter constamment la base de données.

Qu'est-ce que bcrypt.js⁶ ?

bcrypt.js : Un coffre-fort pour mots de passe

bcrypt.js est une bibliothèque JavaScript qui transforme les mots de passe en codes indéchiffrables (hash) avec un "salt" unique pour chaque mot de passe. Elle utilise un algorithme adaptatif ajustable en complexité.

Pourquoi bcrypt ?

⁵ **Jwt** : sont des **jetons d'authentification** au format JSON, signés (et parfois chiffrés), utilisés pour transmettre de façon sécurisée des informations entre un client et un serveur.

⁶ **Bcrypt.js** : est une **bibliothèque JavaScript** qui permet de hacher et vérifier des mots de passe de manière sécurisée, en utilisant l'algorithme bcrypt.

Stocker des mots de passe en clair ou avec un simple hash est dangereux. bcrypt rend les attaques par dictionnaire et force brute quasi-impossibles grâce à son algorithme lent et adaptatif qui évolue avec la puissance des ordinateurs.

Usage concret

Inscription → hachage du mot de passe → stockage du hash en base. Connexion → comparaison du mot de passe saisi avec le hash stocké. Même si la base de données est compromise, les vrais mots de passe restent secrets

d) Email.js (coté client) & Resend (coté serveur)

EmailJS⁷ : Envoi d'emails simplifié pour Next.js

EmailJS permet d'envoyer des emails directement depuis le frontend Next.js sans serveur backend. Il établit une connexion sécurisée avec des services comme Gmail ou Outlook via un système de templates et clés publiques qui protègent vos identifiants.



The screenshot shows the EmailJS web application interface. The left sidebar contains navigation links: Email Services, Email Templates, Email History (which is selected and highlighted in blue), Suppressions, Contacts, Events, Statistics, Team Members, Account, and Personal Settings. The main content area is titled 'Email History' and displays a table of recent activity. The table has columns for 'Created', 'Result', 'Service', and 'Template'. One entry is visible: '16/07/2025 13:59:29', 'OK', 'f.saez.0486@gmail.com', and 'Contact Us' under 'Template'. At the top of the main area, there are dropdown menus for 'All history', 'All templates', 'All services', and a search bar labeled 'Search in template parameters'.

C'est quoi : Service cloud d'envoi d'emails depuis le navigateur sans serveur backend.

Pourquoi : Évite de développer une API serveur tout en gardant les identifiants sécurisés via templates et clés publiques. **À quoi ça sert :** Formulaires de contact, portfolios, notifications simples sur sites vitrine avec logique basique.

Cas d'usage idéaux : Sites vitrine, portfolios et applications avec envois d'emails occasionnels (formulaire de contact, notifications simples, confirmations). Moins adapté pour une logique métier complexe nécessitant un traitement serveur approfondi.

⁷ **Email.js** : est un service qui permet d'envoyer des emails directement depuis du JavaScript côté client (navigateur) sans serveur backend, en utilisant un compte EmailJS et des templates préconfigurés.

Resend⁸ : La puissance de l'envoi d'emails côté serveur

C'est quoi : API moderne d'envoi d'emails transactionnels côté serveur pour développeurs.

Pourquoi : Offre meilleure délivrabilité, contrôle avancé (tracking, analytics) et sécurité renforcée. **À quoi ça sert :** Emails critiques (confirmations, factures), campagnes marketing, applications avec suivi professionnel des envois.

Resend : API d'emails moderne pour développeurs

Resend est une API moderne d'envoi d'emails pour développeurs, alternative élégante à SendGrid/Mailgun, s'intégrant parfaitement dans les API Routes Next.js. Elle permet de créer des emails dynamiques avec logique métier complexe, gestion d'envois en masse, programmation différée et historique complet, offrant une excellente délivrabilité et des outils de monitoring avancés. Idéale pour les emails transactionnels professionnels (confirmations, notifications, factures, newsletters) avec des templates HTML/CSS personnalisés intégrés au design system.

Quand choisir l'un ou l'autre dans Next.js ?

Le choix dépend de la complexité du projet. EmailJS convient pour les interactions simples côté client comme les formulaires de contact basiques, où vous pouvez styliser avec du CSS personnalisé et gérer directement les états dans vos composants JSX. Resend devient indispensable dès que vous avez besoin de logique métier, de sécurité renforcée ou d'intégration base de données : emails de bienvenue après inscription, notifications e-commerce, rappels automatiques ou rapports périodiques nécessitent la puissance du traitement côté serveur.

III. OUTILS

a) Méthodologie & Trello (Organisation des tâches)

Pour ce projet, j'ai choisi d'appliquer une **méthode inspirée de l'Agile / Scrum**, adaptée à un contexte freelance. L'objectif était d'avancer **par itérations courtes** avec des livrables fréquents et des retours réguliers du tuteur et de la cliente.

- **Découpage en tâches et priorisation** dans un backlog (Trello).

⁸ **Resend** : est un **service d'envoi d'emails** moderne. Côté serveur, l'API Resend te permet d'envoyer des emails transactionnels ou marketing directement depuis ton backend (Node.js, Python, etc.) via des requêtes HTTP ou SDK officiels, sans avoir à gérer d'infrastructure SMTP.

- **Organisation en sprints courts** pour livrer rapidement des versions fonctionnelles.
- **Point quotidien de 15 minutes sur Discord** pour suivre l'avancement, lever les blocages et recueillir les feedbacks.
- **Validation régulière** des fonctionnalités terminées avant de passer aux suivantes.

Cette approche permettait :

- de **donner de la visibilité** à la cliente sur l'état du projet,
- de **réduire les risques** d'incompréhension ou de retard,
- d'**ajuster rapidement** les priorités selon ses retours.

Pour la gestion opérationnelle, j'ai mis en place un **tableau Trello** afin de suivre les tâches et visualiser l'avancement.

Le tableau est structuré en plusieurs listes :

- **Backlog / Modifs à apporter** : nouvelles fonctionnalités ou corrections à prévoir.
- **À faire / En cours** : tâches planifiées pour le sprint en cours.
- **Terminé** : tâches finalisées et validées.

Chaque carte Trello correspond à une fonctionnalité ou une section du site. Elle contient :

- une **description claire** de la tâche,
- une **checklist** pour les sous-étapes (ex. « 5/5 » ou « 6/6 » cochées),
- des **étiquettes colorées** pour identifier les thématiques (front-end, back-end, responsive, contenu...),
- parfois des **pièces jointes** ou captures d'écran.

TRELLO (ORGANISATION DE TRAVAIL)

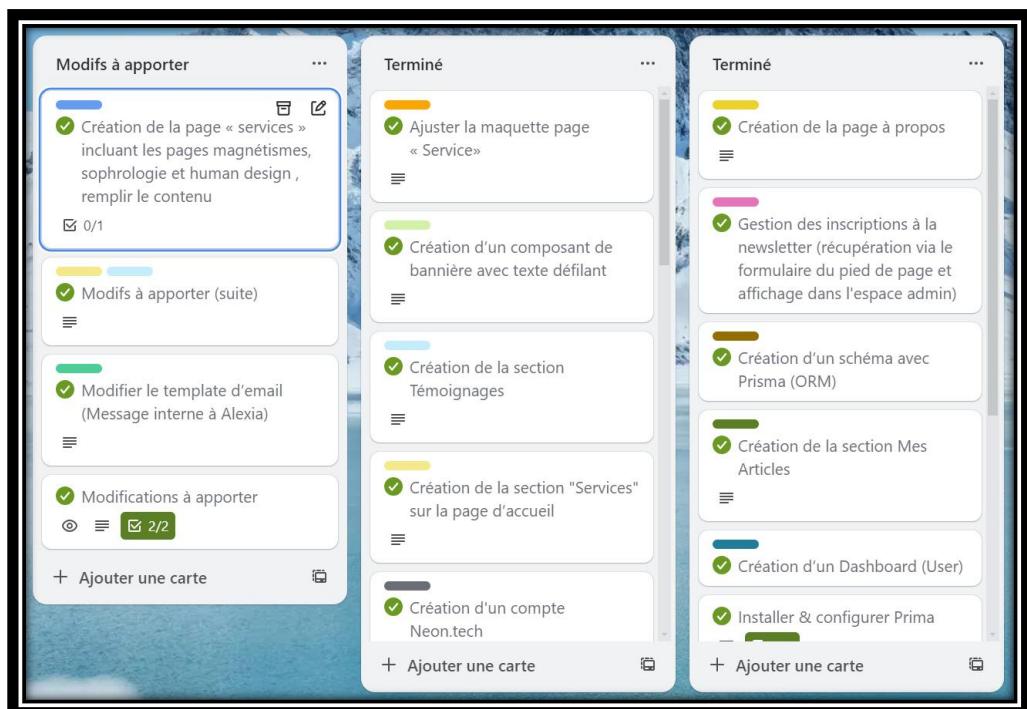
Exemples de cartes finalisées :

- Création des sections **Header, Hero, About, Footer**
- Création de la page **FAQ** et de la page **Mentions légales**
- Mise en place de composants réutilisables et d'un **dashboard utilisateur**
- Gestion des inscriptions à la newsletter et affichage dans l'espace admin

- Ajustements du responsive sur le header et le dashboard

Cette organisation m'a permis de :

- **visualiser l'avancement** du projet en un coup d'œil,
- **prioriser les tâches** selon l'urgence et la valeur pour la cliente,
- **documenter les réalisations** de manière claire et datée.



GIT & GITHUB

a) Git

Qu'est-ce que c'est ?

Un logiciel libre de **gestion de versions** (VCS – Version Control System)

Pourquoi ?

Pour suivre l'évolution du code dans le temps, travailler en parallèle sur plusieurs fonctionnalités et garder un historique complet des changements.



À quoi ça sert ?

- ✓ Sauvegarder ton code localement avec son historique.
- ✓ Créer des branches pour développer des fonctionnalités sans casser le reste du projet.
- ✓ Fusionner, revenir en arrière, comparer des versions.
- ✓ Travailler en équipe sans écraser le travail des autres.

b) GitHub

A screenshot of a GitHub repository page. At the top, it shows a merge commit from 'fannysaez' to 'Merge branch 'fanny''. Below this, there is a list of individual commits. The commits include modifications to 'prisma/schema.prisma', 'public', 'src', '.gitignore', 'README.md', 'eslint.config.mjs', 'jsconfig.json', 'next.config.mjs', 'package-lock.json', and 'package.json'. Each commit has a timestamp indicating when it was made, such as '2 months ago' or '3 months ago'. This visual representation of the commit history on GitHub provides a clear record of the project's evolution and the contributions of different team members.

Projet développé à partir du repository GitHub de mon tuteur pour appliquer les acquis de formation et découvrir de nouvelles technologies. Travail sur une **branche dédiée "Fanny"** avec création et modification de fichiers JSX, styles CSS

personnalisés et modules Node. **410 commits** témoignent de l'ampleur du développement réalisé.

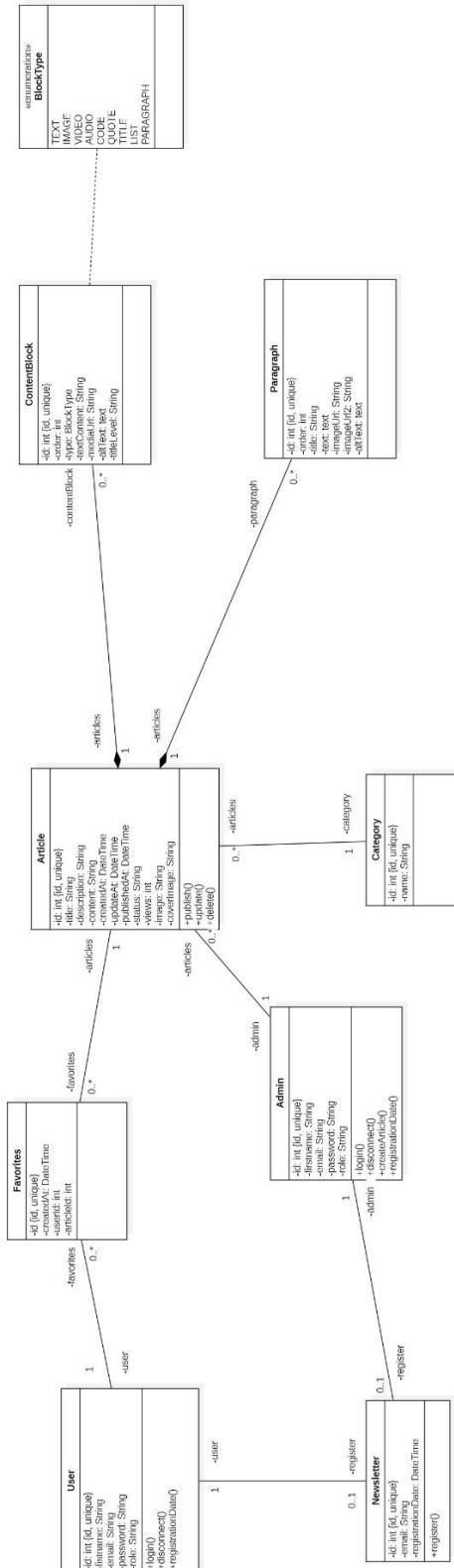
De plus, j'ai pu mettre une copie de ce projet sur mon ordinateur au cas où. Par la suite, je me suis créé un dépôt Git pour poursuivre ce projet ou j'ai pu le finaliser en 3 mois. (**190 commits**), car j'ai rajouté une fonctionnalité concernant l'utilisateur connecté. J'ai travaillé sur nouvelle branche « dev » pour éviter de toucher à la production directement. Le projet client est terminé et prêt à être livré depuis mon dépôt. Des modifications restent possibles après livraison.

The screenshot shows a GitHub repository page for 'alexia-energies'. The repository has 190 commits, 2 branches, and 0 tags. The commits are listed in chronological order, mostly 'initial commit' for files like .gitignore, README.md, cahier-des-charges.md, eslint.config.mjs, jsconfig.json, next.config.mjs, package-lock.json, and package.json. The repository is public and has 1 star. The 'About' section describes the project as a Next.js application for Alexia Energies, featuring responsive design, newsletter management, and user management. It uses technologies like Prisma ORM, Route.js, and Neon PostgreSQL. The repository has 0 forks and 0 watching.

Lien du projet

IV. CONCEPTIONS

a) Diagramme de classe « UML ⁹ »



⁹ UML : est un langage visuel standardisé pour **modéliser et représenter graphiquement** la structure et le fonctionnement d'un système

Relations principales :

Admin → Article (1 → 0..*) Une instance d'Admin gère plusieurs instances d'Article. Chaque instance d'Article a un seul auteur. **Relation One-to-Many**.

Article → ContentBlock (1 → 0..*) Une instance d'Article contient plusieurs instances de ContentBlock. Composition avec suppression en cascade **(ON DELETE CASCADE)**.

Article → Paragraph (1 → 0..*) Une instance d'Article contient plusieurs instances de Paragraph. Protection contre suppression accidentelle **(ON DELETE RESTRICT)**.

ContentBlock → BlockType. Chaque instance de ContentBlock utilise une valeur de l'énumération BlockType (TEXT, IMAGE, VIDEO, AUDIO, CODE, QUOTE, LIST, PARAGRAPH).

User → Newsletter (0..1 → 0..1) Une instance d'User peut s'abonner à une instance de Newsletter. **Relation One-to-One** optionnelle avec SET NULL.

Admin → Newsletter (1 → 0..1) Une instance d'Admin peut gérer une instance de Newsletter. **Relation One-to-One** avec SET NULL.

User → Favorites (1 → 0..*) Une instance d'User possède plusieurs instances de Favorites. Suppression en cascade (ON DELETE CASCADE).

Favorites → Article (0..* → 1) Plusieurs instances de Favorites réfèrentent une instance d'Article. **Relation Many-to-One** avec CASCADE.

User ↔ Article (via Favorites) Relation Many-to-Many : Une instance d'User favorise plusieurs instances d'Article via la classe d'association Favorites.

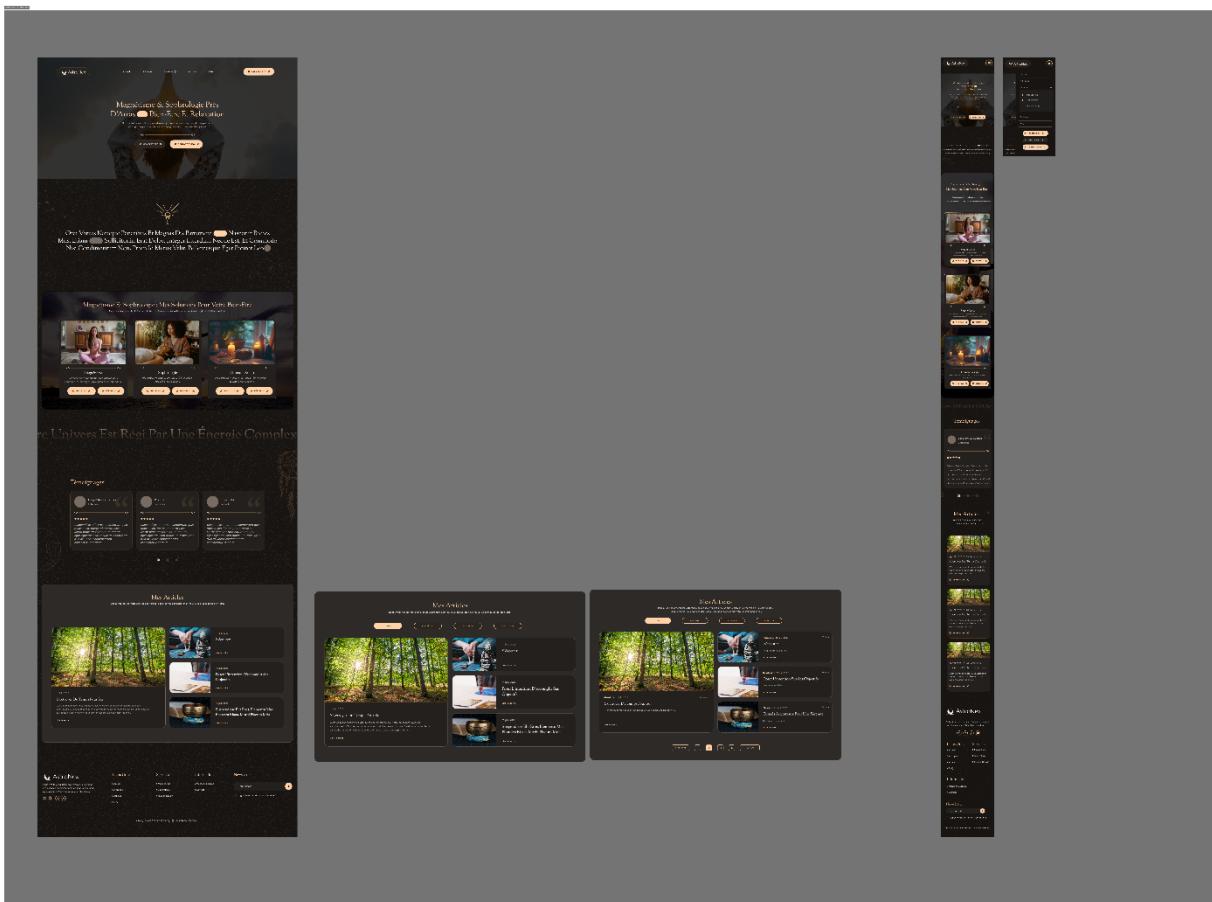
Category → Article (1 → 0..*). Une instance de Category catégorise plusieurs instances d'Article. Relation One-to-Many avec SET NULL.

Approche méthodologique « Front First »

Le projet **AstroNex** suit une approche « Front First », en concevant d'abord l'interface pour clarifier le produit, accélérer les validations et anticiper les contraintes techniques. Son nom évoque l'harmonie cosmique et la connexion spirituelle, appuyé par un logo provisoire. La maquette, réalisée sur **Figma** en design responsive, offre des éléments graphiques originaux et harmonieux, et sa version interactive facilite la visualisation et les échanges avec les parties prenantes.

b) Réalisations de la maquette - (*annexe 4 : Lighthouse*)

Des captures d'écran illustrent la maquette. La version complète est disponible sur Figma, offrant une navigation interactive et facilitant les échanges et validations avec les parties prenantes. J'ai choisi cette plateforme car elle permet une visualisation interactive du projet et facilite les échanges avec les parties prenantes lors des phases de validation et d'itération.



Maquette - [Figma¹⁰](#) | Alexia-energies

¹⁰ **Figma** : est un outil collaboratif en ligne de **conception d'interfaces (UI/UX)** qui permet de créer, prototyper et partager des maquettes ou designs en temps réel directement depuis un navigateur.

Création originale et identité visuelle

J'ai créé l'ensemble de la maquette de A à Z, depuis la conceptualisation jusqu'aux derniers détails d'interface. J'ai pensé et designé chaque élément graphique, chaque section et chaque composant spécifiquement pour ce projet. Cette approche m'a permis de développer une identité visuelle unique et parfaitement adaptée au positionnement de la marque.

Version Mobile : La déclinaison mobile reprend les codes visuels du desktop tout en s'adaptant aux contraintes d'affichage des smartphones. J'ai repensé la hiérarchie des informations, optimisé les zones de clic et simplifié la navigation pour offrir une expérience utilisateur fluide sur petit écran.

Version Desktop : J'ai développé une interface optimisée pour les écrans larges, privilégiant une navigation horizontale intuitive et une mise en page aérée. J'ai disposé les éléments visuels de manière à exploiter pleinement l'espace disponible, avec des sections bien définies qui guident naturellement l'œil de l'utilisateur vers les informations essentielles.

V. ARCHITECTURE GENERALE DU PROJET

a) Architecture du projet « Next.js »

Alexia-énergies est une application web construite avec **Next.js 15.3.3** et son **App Router** moderne. L'architecture est **modulaire et structurée**, ce qui facilite la **maintenabilité** et la **scalabilité**.

- Le **dossier racine** centralise les fichiers de configuration (Next.js, Prisma, ESLint) et les variables d'environnement.
- Le **dossier src/** contient :
 - **app/** : le nouveau système de routage basé sur les dossiers.
 - **lib/** : les utilitaires, configurations et logiques métier (Prisma, templates d'email, etc.).

```

```
alexia-energies/
├── src/
│ ├── app/
│ │ ├── layout.js # App Router (Next.js 15)
│ │ ├── page.js # Layout racine global
│ │ └── globals.css # Page d'accueil
│ ├── api/
│ │ ├── articles/ # Styles globaux
│ │ ├── auth/ # API Routes (Backend)
│ │ ├── favorites/ # CRUD Articles
│ │ ├── login/ # Authentification
│ │ ├── register/ # Gestion favoris
│ │ └── middleware/ # Connexion utilisateur
│ ├── components/
│ │ ├── button/ # JWT & Sécurité
│ │ ├── header/ # Inscription
│ │ ├── footer/ # Composants réutilisables
│ │ ├── FavoriteButton/ # Pied de page
│ │ └── dashboard/ # Composant Button universel
│ ├── admin/
│ │ ├── dashboard/ # En-tête navigation
│ │ ├── articles/ # Espace administrateur
│ │ ├── users/ # Dashboard admin
│ │ └── newsletter/ # Gestion articles
│ ├── lib/
│ │ ├── auth.js # Gestion utilisateurs
│ │ ├── prisma.js # Gestion newsletter
│ │ └── emailTemplates.js # Dashboard utilisateur
│ ├── prisma/
│ │ ├── schema.prisma # Pages articles publics
│ │ ├── migrations/ # Page connexion
│ │ └── schema/ # Page inscription
│ │ ├── base.prisma # Page contact
│ │ └── models/ # Utilitaires & configurations
│ │ ├── enums/ # Templates emails
│ │ └── enums/ # Fonctions authentication
│ │ └── enums/ # Configuration Prisma
│ ├── public/
│ │ ├── img/ # Base de données
│ │ ├── articles/ # Schéma unifié
│ │ └── fonts/ # Historique migrations
│ └── next.config.mjs # Schémas modulaires
└── package.json # Modèles de données
└── eslint.config.mjs # Énumérations

```

J'ai conçu l'application **Alexia Énergies** en suivant l'architecture **App Router** de Next.js 13+, qui représente l'évolution moderne du framework. Cette approche me permet de bénéficier des dernières fonctionnalités de Next.js tout en maintenant une structure de code claire et évolutive.

**Pourquoi cette architecture ?** L'App Router offre plusieurs avantages cruciaux pour mon projet : **Routage basé sur les dossiers** : Chaque page correspond à un dossier, rendant la navigation intuitive. **Layouts imbriqués** : Possibilité de créer des mises en page réutilisables à différents niveaux. **Composants serveur par défaut** : Amélioration des performances et du SEO. **API Routes intégrées** : Backend et frontend dans la même structure

```

```
src/
  └── app/
    ├── layout.js          # Layout racine avec Header/Footer
    ├── page.js            # Page d'accueil
    ├── globals.css         # Styles globaux
    ├── components/        # Composants réutilisables
    │   ├── header/          # En-tête de navigation
    │   ├── footer/          # Pied de page
    │   ├── button/          # Boutons personnalisés
    │   ├── modal/           # Modales et popups
    │   ├── accueil/         # Composants spécifiques à l'accueil
    │   └── dashboard/       # Interface utilisateur
    ├── api/                # Routes API Next.js
    └── [pages]/             # Pages de l'application
  └── lib/
    ├── prisma.js          # Configuration Prisma
    └── auth.js              # Fonctions d'authentification
  └── globals.css          # Styles globaux et variables CSS
  └── layout.js            # Layout principal de l'application
````
```

```
26 export default function Home() {
27 return (
28 <>
29 <HeroSection />
30 <About /> saez.fanny.63
31 <Process />
32 <Marque />
33 <Testimonials />
34 <MesArticles />
35 </>
36);
37 }
```

Le **système de routage** adopte le *file-system routing* de l'**App Router**, permettant une navigation intuitive entre les différentes sections de l'application. On y retrouve les routes publiques (articles, services, contact, FAQ), le système d'authentification complet (connexion, inscription), ainsi qu'un Dashboard utilisateur avec des sections dédiées.

Les **API Routes** sont également intégrées pour gérer les *endpoints backend* de manière cohérente.

## VI. DEVELOPPEMENT TECHNIQUE

### a) Front-end

J'ai conçu le front-end d'une application web et mobile avec Next.js, React et des outils modernes pour allier performance, SEO et expérience utilisateur. L'architecture modulaire et les styles CSS Modules assurent un code réutilisable et facile à maintenir, tandis que React Icons, Embla Carousel et Next/Image enrichissent l'interface et optimisent les visuels.

...

```
src/app/
└── components/ # Composants réutilisables
 └── button/ # Boutons personnalisés
 └── globals.css # Styles globaux et variables CSS
 └── layout.js # Layout principal de l'application
...
```

J'ai structuré le projet avec une architecture hiérarchique claire et des composants réutilisables et testables, regroupés par fonction ou par page, afin de faciliter la navigation, la collaboration, la maintenance et garantir cohérence et qualité du code.

#### Composant Hero Section avec optimisation d'images

```
src > app > components > accueil > hero > section1.jsx > HeroSection
...
1 // Importation des dépendances nécessaires pour la section d'accueil
2 import React from "react";
3 import style from "./style.module.css";
4 import Button from "@/app/components/button/button";
5 import Image from "next/image";
6 import fleche from "/public/img/accueil/HeroSection/VectorFlecheAccueil.svg";
7 import StarWhite from "/public/img/boutons/VectorStarWhite.svg";
8
9 export default function HeroSection() {
10
11 return (

```

J'ai conçu un composant Next/Image optimisant automatiquement les images (lazy loading, formats modernes, dimensions responsives) avec gestion systématique de l'attribut alt pour l'accessibilité. La **section d'accueil** d'une page, construite avec Next.js et React. On y retrouve une structure composée d'un titre (`<h1>`), d'un texte de présentation et d'une ligne décorative sous forme de SVG.

```

// /* Section 1 */
<section className={style.section}>
 /* L'overlay */
 <div className={style.overlay}></div>
 <div className={style.container}>
 <h1>Magnétisme & Sophrologie Prés de Lens Bien-être Et Relaxation</h1>
 <p> Situé à Lillas, près de Lens, je propose des séances de magnétisme et sophrologie pour vous accompagner vers un mieux-être global.</p>
 /* Ligne décorative avec le SVG */
 <div className={style.decorativeLine}>
 <Image src={fleche} alt="Fleche" width={380} height={20}/>
 </div>
 /* Si style.buttons n'existe pas, on peut utiliser une div sans classe ou ajouter le style en ligne pour tester */
 <div className={style.buttons ? style.buttons : undefined} style={!style.buttons ? { display: "flex", gap: "20px", justifyContent: "center", marginTop: "20px" } : ()}>
 /* Bouton menant à la page des services */
 <Button
 text="Mes services"
 link="#services"
 variant="primary"
 leftVector={<Image src={StarWhite} alt="" width={16} height={16} />}
 rightVector={<Image src={StarWhite} alt="" width={16} height={16} />}>
 />
 /* Bouton menant à la page de contact */
 <Button
 text="Contactez-moi"
 link="/contact"
 variant="secondary"
 leftVector={<Image src={StarBlack} alt="" width={16} height={16} />}
 rightVector={<Image src={StarBlack} alt="" width={16} height={16} />}>
 />
 </div>
 </div>
</section>
);
}

```

Ensuite, deux **boutons réutilisables** sont affichés : l'un mène vers la page des services et l'autre vers la page de contact.

Chaque bouton utilise des **props personnalisables** comme text, link, variant ( primaire ou secondaire), ainsi que des icônes placées à gauche et à droite (leftVector et rightVector).

Cette approche permet de créer une interface claire et attrayante, tout en exploitant les composants réutilisables définis précédemment. Cela garantit une **cohérence visuelle**, une **facilité de maintenance** et une **expérience utilisateur fluide**.

La gestion des composants suit une approche fonctionnelle avec une organisation par responsabilité dans le dossier **src/app/components/**. Cette structure inclut les composants d'interface générique (header, footer, boutons, modales), les composants métier spécifiques (formulaires d'authentification, dashboard, sidebar des services), et les composants de contenu (page d'accueil, témoignages, éléments de marque).

L'utilisation de composants réutilisables, comme le bouton modulaire, permet d'adapter facilement un même élément à

```

globals.css > @font-face
saez.fanny.63@gmail.com, le mois dernier | 1 author (saez.fanny.63@gmail.com)
root {
 --background: #181411;
 --primary-color: #FED1A7;
 --secondary-color: #2C2520;
 --tertiary-color: #FFFFFF;
 --quaternary-color: #23201D;
 --paragraph-color: #CAC3BC;
 --p-color: #FFFFFF;
 --background-pattern: url('/img/patterns/points.png');
 --decorative-line: url('/img/HeroSection/VectoFlecheAccueil.svg');
 --footer-separator-vector: url('/img/footer/VectorFooterCopyright.svg');
 --decorative-line-color: #776B62;
 --lineround-color: #776B62;
 --footer-decorative-line-color: #5C4228;
 --navigation-cross-default: url('/img/accueil/testimonials/VectorCroixMarron.svg');
 --navigation-cross-active: url('/img/accueil/testimonials/VectorCroixBeige.svg');
 --text: #CAC3BC;
 --title: #FED1A7;
 --text2: #776B62;
 --benefit-strong-color: var(--primary-color);
 --line-width-desktop: 60px;
 --line-height-desktop: 24px;
 --linegrey-width-desktop: 50px;
 --linegrey-height-desktop: 24px;
 --lineround-size-desktop: 24px;
 --vector-central-desktop: 140px;
 --vector-decoratif-desktop: 350px;
 --section-padding-desktop: 120px;
 --container-max-width: 1350px;
 --text-max-width: 1024px;
}

```

différents contextes grâce à des propriétés personnalisables. Associé à un CSS modulaire, cela assure un code clair, maintenable et cohérent, tout en rendant l'interface homogène et évolutive.

```
7 export default function Button({
8 text,
9 link,
10 onClick,
11 variant = "primary",
12 className = "",
13 leftVector,
14 rightVector,
15 type = "button",
16 isReserveButton = false,
17 ariaLabel | You, hier • Unc
18 }) {
```

Le composant Button est défini avec plusieurs **props 11 personnalisables** (texte, lien, icônes à gauche/droite, type de bouton, variante primaire ou secondaire, etc.). Ces paramètres permettent d'adapter le bouton à différents contextes sans avoir à dupliquer du code. Le fichier importe aussi Link de Next.js pour la navigation et un **fichier CSS modulaire (button.module.css)** qui gère le style de manière

maintenable et isolée.

**La logique conditionnelle** : la classe CSS appliquée dépend de la variante choisie (**primaire ou secondaire**), et une classe spéciale est ajoutée si le bouton est réservé. Le contenu du bouton est ensuite construit dynamiquement : il peut afficher un texte seul ou être enrichi avec des icônes placées à gauche et/ou

à droite. Cette approche rend le bouton **flexible, personnalisable** et facile à maintenir dans le temps.

```
/* Style pour le bouton principal */
.primary {
 background-color: var(--secondary-color);
 color: var(--tertiary-color);
 border-radius: 30px;
 text-transform: uppercase;
 border: 1px solid #5C422B;
 font-size: 14px;
 letter-spacing: 1px;
 padding: 15px 17px;
 display: flex;
 align-items: center;
 justify-content: center;
 white-space: nowrap;
}
```

```
/* Style pour le bouton secondaire */
.secondary {
 background-color: var(--primary-color);
 color: var(--secondary-color);
 border-radius: 30px;
 text-transform: uppercase;
 border: 1px solid white;
 font-size: 14px;
 font-weight: 600;
 letter-spacing: 1px;
 padding: 12px 12px;
 display: flex;
 align-items: center;
 justify-content: center;
 gap: 6px;
 white-space: nowrap;
}
 saez.fanny.63@gmail.com, il y a 2 mois +
```

Système de styles avec CSS Modules - Variables CSS globale

<sup>11</sup> **Props** : sont des paramètres transmis à un composant pour modifier son contenu, son style ou son comportement, ce qui le rend flexible et réutilisable sans changer son code interne

## Page d'accueil



### b) Back-end

#### Pourquoi Next.js API Routes ?

J'ai choisi **Next.js API Routes** comme base du back-end d'Alexia Énergies pour construire une architecture **full-stack cohérente** où front et back partagent la même base de code. Cela simplifie la maintenance, accélère le développement et facilite le déploiement.

**Quoi** ? Des routes API intégrées à Next.js permettant de créer des *endpoints back-end* directement dans le projet. **Pourquoi** ? Pour avoir une **cohérence technologique**, des **performances optimisées** et un **déploiement simplifié** (ex. Vercel). **À quoi ça sert** ? À gérer les données et la logique serveur (authentification, formulaires, intégrations externes...) sans créer une application serveur séparée.

#### Compléments back-end : Prisma + PostgreSQL (Neon Tech)

```
64 // Configuration du générateur (crée le client Prisma)
65 Generate
66 generator client {
67 provider = "prisma-client-js"
68 }
69 // Configuration de la base de données
70 datasource db {
71 provider = "postgresql"
72 url = env("DATABASE_URL")
```

**Configuration** dans  
**schema.prisma**.

```
You, hier | 2 authors (saez.fanny.63@gmail.com and one other)
105 model Favorite {
106 id String @id @default(cuid())
107 createdAt DateTime @default(now())
108 userId String
109 articleId String
110
111 user User @relation(fields: [userId], references: [id], onDelete: Cascade)
112 article Article @relation(fields: [articleId], references: [id], onDelete: Cascade)
113
114 @@unique([userId, articleId])
115 @@map("favorites")
116 }
```

## Models dans schema.prisma

```
You, hier | 2 authors (You and one other)
10 model Article {
11 id String @id @default(cuid())
12 title String
13 slug String @unique
14 author String
15 createdAt DateTime @default(now())
16 updatedAt DateTime? @updatedAt
17 publishedAt DateTime?
18 status String @default("draft")
19 content String
20 image String?
21 coverImage String?
22 categoryId String?
23 description String?
24 views Int @default(0)
25 category Category? @relation(fields: [categoryId], references: [id])
26 contentBlocks ContentBlock[]
27 paragraphs Paragraph[]
28 favorites Favorite[]
29
30 @@map("articles")
31 }
32
```

**PostgreSQL (Neon Tech)** : une **base de données robuste et scalable**, hébergée dans le cloud, qui stocke toutes les données du projet. Neon facilite le déploiement, le scaling automatique et propose les *branches de données* pour tester en toute sécurité.

```
203 ```sql
204 -- CreateTable
205 CREATE TABLE "favorites" (
206 "id" TEXT NOT NULL,
207 "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
208 "userId" TEXT NOT NULL,
209 "articleId" TEXT NOT NULL,
210
211 CONSTRAINT "favorites_pkey" PRIMARY KEY ("id")
212);
213
214 -- CreateIndex (index unique pour éviter les doublons)
215 CREATE UNIQUE INDEX "favorites_userId_articleId_key" ON "favorites"("userId", "articleId");
216 -- AddForeignKey (clés étrangères avec suppression en cascade)
217 ALTER TABLE "favorites" ADD CONSTRAINT "favorites_userId_fkey"
218 FOREIGN KEY ("userId") REFERENCES "users"("id") ON DELETE CASCADE ON UPDATE CASCADE;
219
220 ALTER TABLE "favorites" ADD CONSTRAINT "favorites_articleId_fkey"
221 FOREIGN KEY ("articleId") REFERENCES "articles"("id") ON DELETE CASCADE ON UPDATE CASCADE;
222```
```

## Les commandes utiles avec `npx prisma migrate dev --name init`

```

Initialiser Prisma dans le projet

```bash
npx prisma init
```

Crée le dossier `prisma/` et le fichier `*.env`.

Générer le client Prisma (après modification du schéma)

```bash
npx prisma generate
```

Met à jour le client Prisma utilisé dans le code.

Visualiser la base de données avec Prisma Studio

```bash
npx prisma studio
```

Ouvre une interface graphique pour explorer et modifier les données.

```

La commande **génère et exécute automatiquement les requêtes SQL** nécessaires pour créer la base de données dans PostgreSQL (Neon Tech). Prisma se charge de traduire tes modèles en tables et de gérer leurs relations. **PostgreSQL (Neon Tech)** : héberge la base dans le cloud. Une URL de connexion (fournie par Neon) est ajoutée dans le fichier **.env**

## Gestion des migrations avec Prisma + PostgreSQL (Neon Tech)

Une fois les modèles définis dans le fichier **schema.prisma**, Prisma permet de créer et maintenir la structure de la base de données grâce au système de migrations. Chaque migration

```

prisma/migrations/
├── migration_lock.toml
├── 20250727081554_init/
│ └── migration.sql # Verrou pour éviter les conflits
└── 20250804140849_add_description_to_articles/
 ├── migration.sql # Migration initiale (création des tables)
 └── migration.sql # Ajout colonne description
└── 20250806072220_add_views_to_articles/
 ├── migration.sql # Ajout colonne views
 └── migration.sql # Création table favorites

```

correspond à une évolution du schéma (ajout de tables, colonnes, relations...).

Dans le projet, les migrations sont stockées dans le dossier `prisma/migrations/`. On y retrouve par exemple :

- Migration initiale : création des premières tables (init).
- Ajouts successifs : nouvelles colonnes (description, views) ou nouvelles tables (favorites).
- Fichier SQL généré automatiquement : chaque migration contient un `migration.sql` qui traduit les modèles Prisma en requêtes SQL exécutées sur la base PostgreSQL hébergée par Neon Tech.

## Connexion à la base de données de neon tech

Cette configuration permet à **Prisma** et aux API Routes Next.js de **se connecter** directement à la base. Neon facilite le scaling automatique et propose les *branches de données* pour tester sans impacter la base principale.

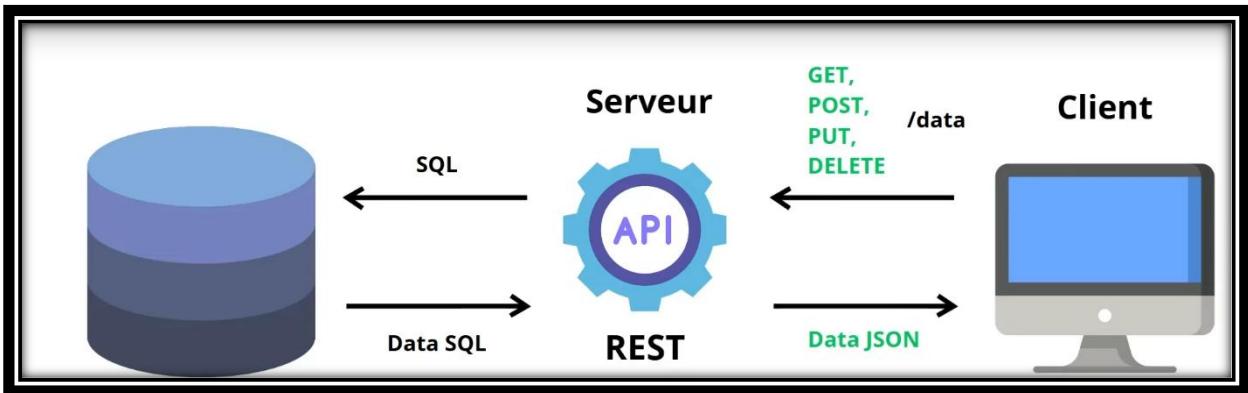
```
Connexion à la base Neon

Pour connecter Prisma à votre base PostgreSQL hébergée sur Neon, ajoutez l'URL de connexion dans le fichier `*.env` à la racine du projet :

```env
DATABASE_URL="postgresql://user:password@neon.tech/dbname"
```

Remplacez `user`, `password` et `dbname` par vos identifiants Neon.
Cette URL est utilisée par Prisma pour appliquer les migrations et interagir avec la base de données.
```

### c) API Rest



#### Qu'est-ce qu'une API REST ?

Une **API REST** (*Representational State Transfer*) est une interface qui permet à une application (*client*) de communiquer avec **un serveur** via *HTTP*.

- Elle s'appuie sur les 4 actions principales (CRUD) :
  - **GET** → Lire des données
  - **POST** → Créer des données
  - **PUT** → Mettre à jour des données
  - **DELETE** → Supprimer des données
- Les données circulent généralement au format **JSON**, simple à lire et échanger.

## À quoi ça sert ?

À faire le lien entre l'application et la base de données : *par exemple*, quand un **utilisateur remplit un formulaire de contact**, la **requête passe par l'API** → qui *enregistre les informations dans la base PostgreSQL* → et **renvoie une confirmation au client**.

## Pourquoi ?

Parce qu'une API REST offre une architecture **standardisée, flexible et indépendante du langage ou de la plateforme**. Elle rend le projet plus **modulaire, maintenable et évolutif**, ce qui facilite l'intégration avec d'autres services (paiements, envoi d'e-mails, etc.).

L'API du projet est structurée en plusieurs volets qui assurent à la fois la consultation publique, la gestion complète du contenu et l'interaction personnalisée des utilisateurs. Elle se compose de l'API publique, de l'API admin et de l'API favoris.

- ✓ **API Publique** : consultation libre (articles publiés).

L'API publique, disponible via la route /api/articles, offre un accès libre à tous les visiteurs sans nécessiter d'authentification. Elle permet uniquement de consulter les articles publiés, tout en incluant leurs relations (catégorie, blocs de contenu, paragraphes). Cette API est idéale pour l'affichage sur le site car elle expose uniquement le contenu validé et visible par le grand public.

```
1. Lire tous les articles publiés (GET)

Backend


```
export async function GET(req) {
  // Récupérer tous les articles dont le statut est 'published'
  const articles = await prisma.article.findMany({
    where: { status: 'published' }, // Filtrer uniquement les articles publiés
    orderBy: { createdAt: 'desc' }, // Trier par date de création décroissante
    include: {
      category: true, // Inclure la catégorie associée
      contentBlocks: true, // Inclure les blocs de contenu
      paragraphs: true // Inclure les paragraphes
    },
  });
  // Retourner la liste des articles au format JSON
  return NextResponse.json(articles);
}

Frontend (fetch)

useEffect(() => {
  // Appeler l'API publique pour récupérer les articles publiés
  fetch("/api/articles")
    .then((res) => res.json()) // Convertir la réponse en JSON
    .then((data) => setArticles(Array.isArray(data) ? data : [])); // Mettre à jour le state avec la liste des articles
}, []);
```


```

```

2. Lire un article par slug (GET)

Backend

export async function GET(req, { params }) {
 // Extraire le slug depuis les paramètres de la requête
 const { slug } = params;
 // Rechercher l'article correspondant au slug
 const article = await prisma.article.findUnique({
 where: { slug }, // Filtrer par slug
 include: {
 category: true, // Inclure la catégorie
 contentBlocks: true, // Inclure les blocs de contenu
 paragraphs: true // Inclure les paragraphes
 }
 });
 // Si aucun article trouvé, retourner une erreur 404
 if (!article) {
 return Response.json({ error: 'Article not found' }, { status: 404 });
 }
 // Retourner l'article trouvé au format JSON
 return Response.json(article);
}

Frontend (fetch)

const fetchArticle = async () => {
 // Appeler l'API pour récupérer un article par son slug
 const response = await fetch('/api/articles/${slug}');
 if (response.ok) {
 const data = await response.json(); // Convertir la réponse en JSON
 setArticle(data); // Mettre à jour le state avec l'article
 } else {
 setError("Article introuvable"); // Afficher une erreur si l'article n'est pas trouvé
 }
};

```

✓ **API Admin** : gestion complète (CRUD) protégée par JWT admin.

L'API admin, disponible via la route /api/admin/articles, est protégée par JWT et réservée aux administrateurs. Contrairement à l'API publique, elle permet une gestion complète des articles, qu'ils soient publiés ou en brouillon. Elle prend en charge toutes les opérations CRUD : création avec génération automatique d'un slug unique et gestion des relations, lecture de l'ensemble des articles enrichis, mise à jour des données avec régénération du slug si nécessaire, et suppression définitive avec prise en charge des dépendances. Cette API constitue ainsi un outil central de gestion éditoriale, combinant flexibilité et sécurité.

 API Admin – Articles (JWT obligatoire, rôle admin)

Route : /api/admin/articles  
Protégée par JWT (Authorization: Bearer <token>)

1. CREATE – Créer un article (POST)

Backend

```
export async function POST(req) {
 // Vérifier l'authentification et le rôle admin
 const user = await verifyAuth(req);
 if (!user || user.role !== 'admin') {
 return NextResponse.json({ error: 'Unauthorized access' }, { status: 403 }); // Refuser si non admin
 }

 // Extraire les champs du corps de la requête
 const { title, author, description, content, categoryId, status = 'draft' } = await req.json();

 // Générer un slug unique à partir du titre
 let slug = title.toLowerCase().replace(/[^a-z0-9]+/g, "-");
 while (await prisma.article.findUnique({ where: { slug } })) {
 slug = slug + "-" + Math.floor(Math.random() * 1000); // Ajouter un nombre si le slug existe déjà
 }

 // Crée le nouvel article avec les champs en anglais
 const newArticle = await prisma.article.create({
 data: { title, slug, author, description, content, status, categoryId }
 });

 // Retourner l'article créé au format JSON
 return NextResponse.json(newArticle, { status: 201 });
}
```

Frontend (fetch)

```
const createArticle = async (formData) => {
 // Récupérer le token admin depuis le localStorage
 const token = localStorage.getItem("adminToken");
 // Appeler l'API admin pour créer un article
 const res = await fetch("/api/admin/articles", {
 method: "POST",
 headers: {
 "Content-Type": "application/json", // Spécifier le type de contenu
 "Authorization": `Bearer ${token}` // Envoyer le token JWT
 },
 body: JSON.stringify(formData), // Envoyer les données du formulaire
 });
 const data = await res.json(); // Convertir la réponse en JSON
 if (!res.ok) throw new Error(data.error || "Error during creation"); // Gérer les erreurs
 alert("Article created!"); // Afficher une confirmation
};
```

## ✓ API Favoris : gestion personnalisée des favoris protégée par JWT utilisateur.

L’API favoris, accessible via la route /api/user/favoris, est protégée par JWT et réservée aux utilisateurs connectés. Elle permet d’ajouter, consulter et retirer des articles favoris, tout en évitant les doublons et en limitant l’ajout aux seuls articles publiés. Cette API offre une expérience personnalisée en permettant à chaque utilisateur de sauvegarder et retrouver facilement ses contenus préférés.

```

Route : /api/user/Favorites
JWT utilisateur obligatoire

1. Ajouter aux favoris (POST)

const addFavorite = async (articleId) => {
 // Récupérer le token utilisateur depuis le localStorage
 const token = localStorage.getItem("userToken");
 // Appeler l'API pour ajouter un favori
 const res = await fetch("/api/user/Favorites", {
 method: "POST", // Méthode POST pour ajouter
 headers: {
 "Content-Type": "application/json", // Spécifier le type de contenu
 "Authorization": `Bearer ${token}` // Envoyer le token JWT
 },
 body: JSON.stringify({ articleId }), // Envoyer l'id de l'article à ajouter
 });
 const data = await res.json(); // Convertir la réponse en JSON
 if (!res.ok) throw new Error(data.error || "Error adding favorite"); // Gérer les erreurs
 alert("Added to favorites!"); // Afficher une confirmation
};

2. Lire ses favoris (GET)

useEffect(() => {
 // Fonction pour récupérer les favoris de l'utilisateur
 const fetchFavorites = async () => {
 const token = localStorage.getItem("userToken"); // Récupérer le token utilisateur
 const res = await fetch("/api/user/Favorites", {
 headers: { "Authorization": `Bearer ${token}` }, // Envoyer le token JWT
 });
 const data = await res.json(); // Convertir la réponse en JSON
 setFavorites(data.data); // Mettre à jour le state avec la liste des favoris
 };
 fetchFavorites(); // Appeler la fonction au montage du composant
}, []);

3. Retirer un favori (DELETE)

const removeFavorite = async (articleId) => {
 // Récupérer le token utilisateur depuis le localStorage
 const token = localStorage.getItem("userToken");
 // Appeler l'API pour retirer un favori
 const res = await fetch("/api/user/Favorites", {
 method: "DELETE", // Méthode DELETE pour retirer
 headers: {
 "Content-Type": "application/json", // Spécifier le type de contenu
 "Authorization": `Bearer ${token}` // Envoyer le token JWT
 },
 body: JSON.stringify({ articleId }), // Envoyer l'id de l'article à retirer
 });
 const data = await res.json(); // Convertir la réponse en JSON
 if (!res.ok) throw new Error(data.error || "Error removing favorite"); // Gérer les erreurs
 alert("Removed from favorites!"); // Afficher une confirmation
};

```

## VII. FONCTIONNALITES PRINCIPALES

### a) Espace administrateur – Gestion des articles

L'espace Administration est une interface dédiée aux administrateurs du site. Elle centralise toutes les opérations liées à la gestion des articles, permettant ainsi de garantir la qualité et la pertinence des contenus publiés. Grâce à cet espace, l'administrateur peut facilement créer de nouveaux articles, les modifier, les publier ou les supprimer selon les besoins du site. Un système de filtres et de statuts facilite la recherche et le suivi des contenus. Dans ce composant React, la méthode **.map** sert à parcourir le tableau articles et à générer dynamiquement une balise **<tr>** pour chaque article.

**Explication détaillée :**

```
// Exemple de composant pour la liste des articles
export default function ArticleList({ articles }) {
 return (
 <table>
 <thead>
 <tr>
 <th>Titre</th>
 <th>Catégorie</th>
 <th>Statut</th>
 <th>Actions</th>
 </tr>
 </thead>
 <tbody>
 {articles.map(article => (
 <tr key={article.id}>
 <td>{article.titre}</td>
 <td>{article.categorie}</td>
 <td>{article.statut}</td>
 <td>
 <button>Éditer</button>
 <button>Supprimer</button>
 </td>
 </tr>
))}
 </tbody>
 </table>
);
}
```

- articles est un tableau d'objets représentant les articles à afficher dans le tableau.
- La syntaxe {articles.map(article => (...))} signifie : pour chaque élément du tableau articles, exécute la fonction et retourne le contenu entre parenthèses.
- À chaque tour de boucle, on crée une balise <tr> contenant :
  - Le titre de l'article
  - La catégorie
  - Le statut
  - Les boutons d'action (Éditer, Supprimer)

### Fonctionnalités principales :

- Création d'un nouvel article (titre, description, image, catégorie)
- Modification et mise à jour des articles existants
- Publication ou dépublication des articles
- Suppression définitive d'un article
- Visualisation de la liste des articles avec filtres (statut, catégorie)

**Ce que fait le code :** Il affiche la liste des articles dans un tableau HTML, chaque ligne représentant un article avec ses informations et ses actions possibles. **En résumé :** .map permet ici de transformer le tableau articles en une liste de lignes HTML affichées à l'écran, de façon dynamique et automatique.

### Explication de la propriété key dans React

Dans React, la propriété key est utilisée dans les listes générées dynamiquement (comme avec .map) pour aider React à identifier chaque élément de façon unique.

### Pourquoi utiliser key ?

- Elle permet à React d'optimiser le rendu et la mise à jour de la liste.
- Si tu ajoutes, modifies ou supprimes un élément, React sait exactement lequel doit être mis à jour dans le DOM.

```
{articles.map(article => (
 <tr key={article.id}>
 ...
 </tr>
))}
```

Ici, **key={article.id}** signifie que chaque article aura une clé unique basée sur son identifiant. C'est important pour éviter des bugs d'affichage et garantir la performance de l'application. **En résumé :** La clé (key) doit être unique pour chaque élément de la liste, généralement l'id de l'objet, pour que React gère correctement les mises à jour.

## b) Espace utilisateur – Gestion des favoris

Dans mon application, j'ai développé un **espace utilisateur** qui offre à chaque membre un **dashboard personnalisé**. Cet espace leur permet de gérer leur profil et de retrouver facilement leurs articles favoris.

Dans ce composant React, la méthode **.map** sert à parcourir le tableau favoris et à générer dynamiquement une balise **<li>** pour chaque article favori.

### Explication détaillée :

- favoris est un tableau d'objets représentant les articles favoris de l'utilisateur.
- La syntaxe **{favoris.map(article => (...))}** signifie : pour chaque élément du tableau favoris, exécute la fonction et retourne le contenu entre parenthèses.
- À chaque tour de boucle, on crée une balise **<li>** contenant :
  - L'image de l'article
  - Le titre de l'article
  - Un bouton « Retirer » qui, lorsqu'on clique dessus, appelle la fonction **onRemove** avec l'identifiant de l'article.

```
{favoris.map(article => (
 <li key={article.id}>
 ...

))}
```

**Ce que fait le code :** Il affiche la liste des favoris de l'utilisateur, chaque favori étant présenté avec son image, son titre et un bouton pour le retirer de la liste. **En résumé :** .map permet ici de transformer le tableau favoris en une liste d'éléments HTML affichés à l'écran, de façon dynamique et automatique.

### Explication de la propriété key dans React

Dans React, la propriété key est utilisée dans les listes générées dynamiquement (comme avec .map) pour aider React à identifier chaque élément de façon unique.

#### Pourquoi utiliser key ?

- Elle permet à React d'optimiser le rendu et la mise à jour de la liste.
- Si tu ajoutes, modifies ou supprimes un élément, React sait exactement lequel doit être mis à jour dans le DOM.

Ici, **key={article.id}** signifie que chaque favori aura une clé unique basée sur son identifiant. C'est important pour éviter des bugs d'affichage et garantir la performance de l'application.

```
// Exemple de composant pour la gestion des favoris
export default function Favoris({ favoris, onRemove }) {
 return (
 <div>
 <h2>Mes articles favoris</h2>

 {favoris.map(article => (
 <li key={article.id}>

 {article.titre}
 <button onClick={() => onRemove(article.id)}>Retirer</button>

))}

 </div>
);
}
```

L'espace Utilisateur est conçu pour offrir à chaque membre une expérience personnalisée. Il permet à l'utilisateur de sélectionner ses articles préférés, de les retrouver facilement et de gérer sa liste de favoris en toute autonomie. Cette fonctionnalité améliore l'engagement et la fidélisation, en donnant la possibilité de créer un espace de lecture adapté à ses besoins.

#### Fonctionnalités principales :

- Ajout d'un article à la liste des favoris depuis la page de consultation
- Suppression d'un article des favoris
- Visualisation rapide de tous les articles favoris dans une section dédiée

- Accès direct aux articles favoris pour une lecture simplifiée

## VIII. AUTHENTIFICATION & SECURITE

L'application est sécurisée grâce à l'usage de tokens JWT pour les routes sensibles, au chiffrement des mots de passe et à la séparation des rôles. Les secrets sont protégés via des variables d'environnement et les contrôles centralisés respectent les standards OWASP et RGPD.

### Coté client & Coté Serveur

#### a) Middleware de vérification JWT

J'ai développé un **middleware verifyJWT** que j'utilise sur toutes mes routes protégées. Il vérifie que le token transmis dans l'en-tête **Authorization** est valide et, si nécessaire, contrôle le rôle (admin, utilisateur).

```
// src/app/api/middleware/route.js
export function verifyJWT(req) {
 try {
 const authHeader = req.headers.get('authorization');
 if (!authHeader || !authHeader.startsWith('Bearer ')) {
 return { isValid: false, error: 'Token manquant' };
 }
 const token = authHeader.split(' ')[1];
 const decoded = jwt.verify(token, SECRET);
 req.user = decoded;
 return { isValid: true, user: decoded };
 } catch (err) {
 return { isValid: false, error: 'Token invalide' };
 }
}
```

**C'est quoi ?** Middleware qui vérifie la validité du token JWT sur chaque route protégée.

**À quoi ça sert ?** Protéger les routes sensibles et contrôler l'accès selon le rôle (admin, utilisateur).

**Pourquoi ?** Centraliser la sécurité, éviter la duplication de logique, garantir que chaque accès est vérifié.

#### b) Hachage des mots de passe

**C'est quoi ?** Mécanisme qui transforme les mots de passe en une chaîne illisible avant stockage. **À quoi ça sert ?** Empêcher qu'un mot de passe soit lisible en cas de fuite de la base de données. **Pourquoi ?** C'est la norme en sécurité web, protège les utilisateurs contre le vol de données.

Lors de l'inscription d'un utilisateur, je ne stocke jamais le mot de passe en clair. J'utilise bcrypt pour le hacher avec un "salt". Lors de la connexion, je compare le mot de passe saisi avec le hash enregistré.

```
// src/app/api/register/route.js
const salt = bcrypt.genSaltSync(10);
const hashedPassword = bcrypt.hashSync(password, salt);
```

### c) Système d'authentification JWT

**C'est quoi ?** Système qui génère un token JWT lors de la connexion, contenant les infos utilisateur.  
**À quoi ça sert ?** Permettre une authentification stateless, scalable et sécurisée.

**Pourquoi ?** Réduit la charge serveur, facilite la gestion des sessions, empêche la falsification.

```
// src/app/api/Login/route.js
const token = jwt.sign(
 payload,
 process.env.JWT_SECRET,
 // { expiresIn: '72h' }
);
```

J'ai opté pour un système d'authentification basé sur les tokens **JWT (JSON Web Tokens)** pour ses avantages en termes de scalabilité et de sécurité. Cette approche stateless permet une architecture distribuée tout en maintenant un niveau de sécurité élevé.

**Avantages :**

- Stateless : Aucun stockage de session côté serveur, réduisant la charge système
- Scalable : Compatible avec une architecture distribuée et des microservices
- Sécurisé : Signature cryptographique empêchant la falsification
- Standard : Technologie éprouvée et largement supportée

**C'est quoi ?** Contrôle qui impose des critères stricts lors de la création d'un mot de passe.

**À quoi ça sert ?** Renforcer la sécurité des comptes utilisateurs.

**Pourquoi ?** Limiter les risques de piratage par force brute ou mots de passe faibles.

**Contraintes appliquées :**

- Longueur minimale : 12 caractères
- Au moins une majuscule
- Au moins une minuscule
- Au moins un chiffre
- Au moins un caractère spécial parmi [@\$!%\*?&]

```

function isStrongPassword(password) {
 const lengthCheck = password.length >= 12;
 const lowercaseCheck = /[a-z]/.test(password);
 const uppercaseCheck = /[A-Z]/.test(password);
 const digitCheck = /\d/.test(password);
 const specialCharCheck = /[#$!%*?&]/.test(password);
 return {
 isValid: lengthCheck && lowercaseCheck && uppercaseCheck && digitCheck && specialCharCheck,
 lengthCheck,
 lowercaseCheck,
 uppercaseCheck,
 digitCheck,
 specialCharCheck
 };
}

```

## a) Helpers d'authentification

**C'est quoi ?** Fonctions qui vérifient l'état de connexion et le rôle de l'utilisateur côté client.

**À quoi ça sert ?** Afficher ou masquer dynamiquement des éléments de l'interface selon le statut utilisateur. **Pourquoi ?** Améliorer l'expérience utilisateur et éviter des appels API inutiles.

**La sécurité côté client repose sur la gestion de la session, la vérification du rôle et la protection de l'interface.**

```

// src/lib/auth.js
export function isLoggedIn() {
 const token = localStorage.getItem('token');
 // Vérification et décodage du token
 // ...
}
export function isAdmin() {
 // Décodage du token et vérification du rôle
 // ...
}

```

## b) Gestion de la session et des erreurs

**C'est quoi ?** Mécanismes pour gérer la persistance de la session, la déconnexion automatique et l'affichage des erreurs. À

**quoi ça sert ?** Sécuriser la navigation, informer l'utilisateur en cas de problème, éviter les accès non autorisés. **Pourquoi ?** Protéger l'application contre les sessions expirées, les tokens invalides et les attaques XSS.

```

// Exemple de gestion d'erreur côté client
if (!isLoggedIn()) {
 // Rediriger vers la page de connexion
}

```

# VIV. TESTS & VALIDATIONS

## 1. Tests API avec Postman – Connexion Utilisateur

Dans cette étape, j'ai utilisé **Postman** pour tester mon endpoint d'authentification.

**Endpoint testé :** **POST /api/login**

**Objectif :** Valider la connexion des utilisateurs à l'API en utilisant leur email et mot de passe.

**Déroulement :** J'ai configuré une requête POST dans Postman avec l'URL **http://localhost:3000/api/login** et un body contenant un email et un mot de passe. J'ai ensuite vérifié les différentes situations possibles :

The screenshot shows a Postman interface with a 'POST' request to 'http://localhost:3000/api/login'. The 'Params' tab is selected, showing a single parameter 'Key' with 'Value'. The 'Body' tab is selected, showing a JSON response with a message and a token. The response body is:

```
1 {
2 "message": "Connexion réussie",
3 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImNtY3g1eXB1YTAwMDBkaHprdjJzeHU1MzUiLCJlbWFpbCI6InRlc3RAZXhhbXBsZS5jb2giLCJpYXQiOjE3NTIxDU1NDMsImV4cCI6MTc1MjE0OTE0M30..v4ykmCj1065_cZdYCl_o2RSadvKNw_xlW2p_7AAjgZk"
4 }
```

The status bar at the bottom indicates a 200 OK response with 378 ms and 501 B.

- Connexion avec identifiants valides** : en envoyant un email et un mot de passe corrects, j'ai bien reçu une réponse 200 OK avec un message "*Connexion réussie*" et un token JWT.
- Rejet avec email incorrect** : j'ai volontairement utilisé une adresse email inexistante et l'API a répondu par un message d'erreur approprié.
- Rejet avec mot de passe incorrect** : même email valide mais mauvais mot de passe ; la connexion a été refusée.
- Vérification de la génération du token JWT** : j'ai contrôlé que la réponse contient bien le champ token et que celui-ci est un JWT.
- Structure de la réponse** : j'ai validé que la réponse renvoyée contient bien les informations prévues (message, token, et éventuellement les données utilisateur).

Ces tests m'ont permis de m'assurer que le processus d'authentification est fiable et sécurisé.

## 2. Tests unitaires du composant « auth »

Pour la partie front-end en Next.js, j'ai rédigé des tests unitaires avec **Jest** afin de valider le comportement du module d'authentification (**auth**). **Objectif** : M'assurer que la gestion de la connexion (`isLoggedIn`), des rôles utilisateurs (`isAdmin`, `isUser`) ainsi que le traitement des cas particuliers (cookies vides, tokens invalides, JSON mal formé) sont correctement pris en charge.



```
Lancement des Tests
Tests Unitaires

Installer les dépendances de test
npm install

Lancer tous les tests
npm test
```

**Déroulement :**

Dans le fichier `tests/lib/_tests__/auth.test.js`, j'ai écrit plusieurs tests unitaires pour vérifier les utilitaires d'authentification (`isLoggedIn`, `isAdmin`, `isUser`) ainsi que quelques cas particuliers. Quand j'ai lancé la commande `npm test auth.test.js`, j'ai obtenu le résultat

suivant (**17 tests réussis**) :

### `isLoggedIn`

- retourne false quand window est indéfini (cas SSR)
- retourne true quand un token existe dans les cookies
- retourne true quand un token existe dans le localStorage
- retourne false quand aucun token n'est trouvé
- retourne true quand un token est présent à la fois dans les cookies et dans le localStorage

### `isAdmin`

- retourne false quand window est indéfini (SSR)
- retourne false quand aucun token n'est dans le localStorage
- retourne true quand le token contient le rôle **admin**
- retourne false quand le token contient uniquement le rôle **user**
- retourne false quand le token est malformé
- gère correctement les erreurs de parsing JSON

### `isUser`

- retourne false quand l'utilisateur n'est pas connecté
- retourne false si l'utilisateur est admin
- retourne true si l'utilisateur est connecté et non admin

### Cas particuliers (edge cases)

- gère une chaîne de cookie vide
- gère plusieurs cookies sans token
- gère un token dans un cookie avec des espaces

Ces tests unitaires m'ont permis de valider le comportement de mes utilitaires d'authentification dans différents scénarios, notamment la gestion de la connexion via cookies et localStorage, la distinction entre les rôles **admin** et **user**, le traitement des erreurs de parsing ainsi que certains cas particuliers. En tant que développeur, cela me donne confiance dans le fait que l'authentification est correctement détectée, que les rôles sont bien différenciés et que le code reste robuste même lorsqu'il est confronté à des données malformées ou incomplètes.

valider le comportement de mes utilitaires

```
PS C:\Users\desig\Desktop\Next.Js - Projet TP DWWM\alexia-energies> npm test auth.test.js

> site-vitrine@0.1.0 test
> jest auth.test.js

PASS src/lib/_tests__auth.test.js
Auth Utilities
 isLoggedIn
 ✓ returns false when window is undefined (SSR) (3 ms)
 ✓ returns true when token cookie exists (1 ms)
 ✓ returns true when token in localStorage exists (1 ms)
 ✓ returns false when no token found (1 ms)
 ✓ returns true when both cookie and localStorage have tokens (1 ms)
 isAdmin
 ✓ returns false when window is undefined (SSR) (1 ms)
 ✓ returns false when no token in localStorage
 ✓ returns true when token contains admin role (2 ms)
 ✓ returns false when token contains user role (2 ms)
 ✓ returns false when token is malformed (3 ms)
 ✓ handles JSON parsing errors gracefully (2 ms)
 isUser
 ✓ returns false when not logged in (1 ms)
 ✓ returns false when user is admin
 ✓ returns true when user is logged in and not admin
 Edge cases
 ✓ handles empty cookie string (1 ms)
 ✓ handles multiple cookies without token
 ✓ handles token cookie with spaces

Test Suites: 1 passed, 1 total
Tests: 17 passed, 17 total
Snapshots: 0 total
Time: 8.661 s
Ran all test suites matching /auth.test.js/i.
```

## X. DEPLOIEMENT

Pour ce projet, mis en place une intégration continue grâce à Vercel. Cela m'a permis de voir en temps réel le rendu final de mon application et de tester rapidement chaque fonctionnalité. J'ai utilisé **Vercel**, la plateforme d'hébergement conçue pour Next.js, pour déployer mon application.

### 1. Préparation du projet

J'ai configuré mes scripts dans le fichier **package.json** pour que le build et le démarrage soient corrects :

```
package.json > ...
You, il y a 6 jours | 2 authors (saez.fanny.63@gmail.com and one other)
1 { "name": "site-vitrine",
2 "version": "0.1.0",
3 "private": true,
4 "scripts": {
5 "dev": "next dev",
6 "build": "prisma generate && next build",
7 "start": "next start",
8 "lint": "next lint",
9 "test": "jest",
10 "test:watch": "jest --watch",
11 "test:coverage": "jest --coverage"
12 },
13 "dependencies": {
14 "@emailjs/browser": "^4.4.1",
15 "@prisma/client": "^6.13.0",
16 "bcryptjs": "^3.0.2",
17 "cloudinary": "^2.7.0",
18 "embla-carousel-react": "^8.6.0",
19 "jsonwebtoken": "9.0.2",
20 "next": "^15.5.3",
21 "react": "19.0.0",
22 "react-calendly": "4.4.0",
23 "react-dom": "19.0.0",
24 "react-icons": "5.5.0",
25 "resend": "6.0.1"
26 },
27 "devDependencies": {
28 "eslint/eslintrc": "3",
29 "@testing-library/jest-dom": "6.1.0",
30 "@testing-library/react": "14.0.0",
31 "@testing-library/user-event": "14.5.0",
32 "eslint": "8",
33 "eslint-config-next": "15.3.3",
34 "jest": "29.7.0",
35 "jest-environment-jsdom": "29.7.0",
36 "prisma": "6.13.0"
37 }
38}
39}
40}
```

### Installation et Build

```
Installation des dépendances
npm install

Génération du client Prisma
npx prisma generate

Migration de la base de données
npx prisma migrate deploy

Build de production
npm run build
```

Ces scripts me permettent : de lancer le projet en local avec **npm run dev**

- de construire le projet avec **npm run build**
- d'exécuter les migrations Prisma avant chaque build (**prisma generate** et **prisma migrate deploy**)
- d'accéder à Prisma Studio pour inspecter la base de données.

## 2. Déploiement sur Vercel

Pour déployer, j'ai simplement connecté mon dépôt GitHub à Vercel. Ensuite :

- Vercel a automatiquement détecté mon projet Next.js (**version 14**)
- J'ai choisi la branche principale (main) comme branche de production
- Vercel a lancé le build en utilisant le script "build" défini dans **package.json**
- Une fois le build terminé, l'application a été mise en ligne automatiquement

### Prérequis

- Node.js 18+
- NPM ou Yarn
- Compte Vercel (déploiement)
- Base de données PostgreSQL (Neon)

### Variables d'Environnement

Créer un fichier **.env** avec :

```
Base de données
DATABASE_URL="postgresql://username:password@host:port/database"

Authentification
JWT_SECRET="your-super-secret-jwt-key-minimum-32-characters"

Cloudinary (images)
CLOUDINARY_CLOUD_NAME="your-cloud-name"
CLOUDINARY_API_KEY="your-api-key"
CLOUDINARY_API_SECRET="your-api-secret"

Email (Resend)
RESEND_API_KEY="re_your-resend-api-key"

NextAuth (optionnel)
NEXTAUTH_SECRET="your-nextauth-secret"
NEXTAUTH_URL="https://your-domain.com"
```

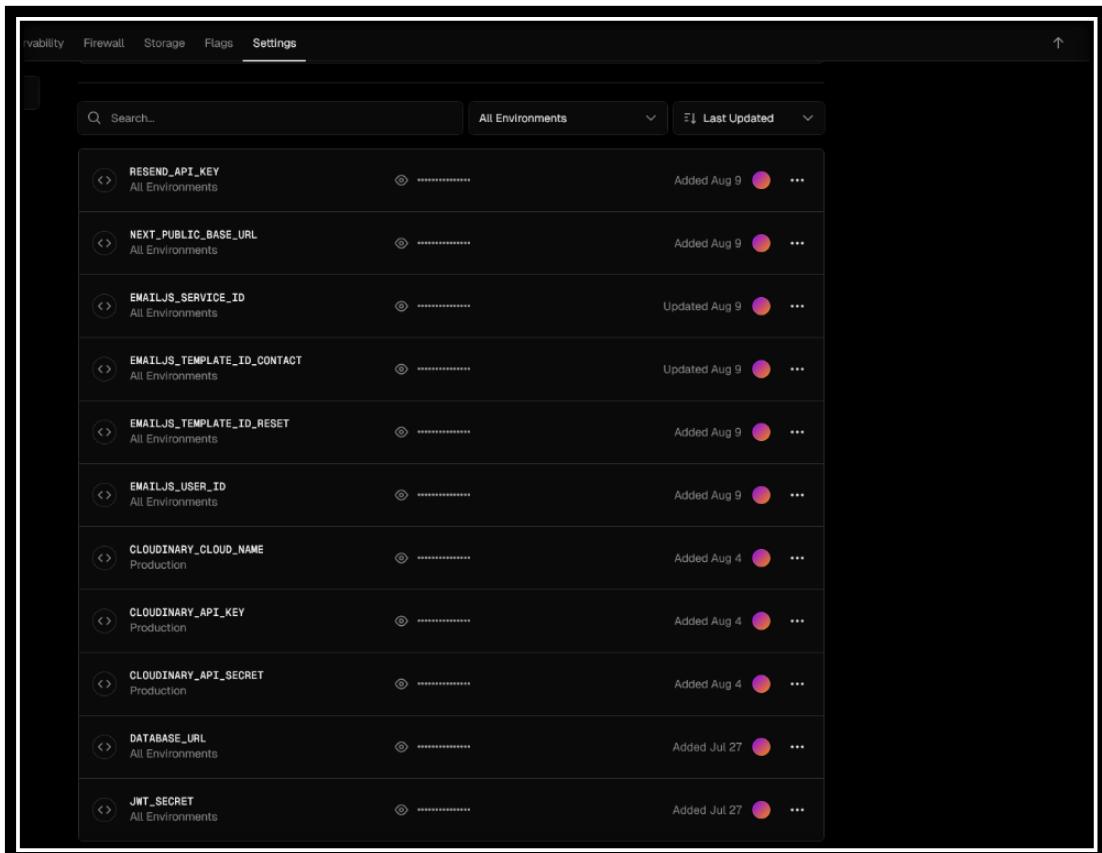
J'ai aussi configuré mes **variables d'environnement** directement dans l'interface Vercel (menu **Settings > Environment Variables**) pour que la base de données, Cloudinary, JWT et EmailJS soient accessibles côté serveur.

### 3. Résultat

À chaque fois que je pousse un commit sur la branche main, Vercel<sup>12</sup> reconstruit et redéploie automatiquement l'application. Le projet est désormais accessible publiquement à l'adresse :

<https://alexia-energies.vercel.app/>

Cette méthode m'a permis d'avoir un déploiement<sup>13</sup> continu simple, rapide et intégré avec mon flux de travail Git.



The screenshot shows the Vercel Settings interface. At the top, there are tabs for Visibility, Firewall, Storage, Flags, and Settings, with 'Settings' being the active tab. Below the tabs is a search bar and dropdown menus for 'All Environments' and 'Last Updated'. The main area displays a list of environment variables:

Name	Environment	Last Updated
RESEND_API_KEY	All Environments	Added Aug 9
NEXT_PUBLIC_BASE_URL	All Environments	Added Aug 9
EMAILJS_SERVICE_ID	All Environments	Updated Aug 9
EMAILJS_TEMPLATE_ID_CONTACT	All Environments	Updated Aug 9
EMAILJS_TEMPLATE_ID_RESET	All Environments	Added Aug 9
EMAILJS_USER_ID	All Environments	Added Aug 9
CLOUDINARY_CLOUD_NAME	Production	Added Aug 4
CLOUDINARY_API_KEY	Production	Added Aug 4
CLOUDINARY_API_SECRET	Production	Added Aug 4
DATABASE_URL	All Environments	Added Jul 27
JWT_SECRET	All Environments	Added Jul 27

<sup>12</sup> **Vercel** : Plateforme qui héberge et déploie automatiquement les applications web, notamment celles faites avec Next.js.

<sup>13</sup> **Déploiement** : Mise en ligne d'une application pour la rendre accessible aux utilisateurs.

## XI. CONCLUSION

Ce projet *Alexia Énergies* marque l'aboutissement de ma formation de développeuse web et web mobile, mais aussi une étape clé dans mon parcours global d'apprentissage. Lorsque j'ai débuté, mes connaissances étaient limitées et chaque nouvelle notion représentait un véritable défi. Aujourd'hui, je mesure pleinement le chemin parcouru, fruit d'un travail régulier, d'un accompagnement de qualité et de mes propres recherches et expérimentations personnelles. Je tiens à remercier **Simplon** et l'ensemble des formateurs pour leur accompagnement et leur soutien tout au long de cette aventure.

Mon parcours est également marqué par mon **profil autodidacte**. Depuis plusieurs années, j'ai pris l'habitude d'apprendre par moi-même, en explorant des plateformes d'apprentissage en ligne, en testant de nouvelles technologies et en menant des projets personnels. Cette curiosité et cette autonomie m'ont permis de progresser de manière continue et d'acquérir une réelle capacité à me former rapidement et efficacement. J'aime relever des défis, sortir de ma zone de confort et aller plus loin que ce qui m'est proposé en formation, car pour moi, apprendre est avant tout une passion.

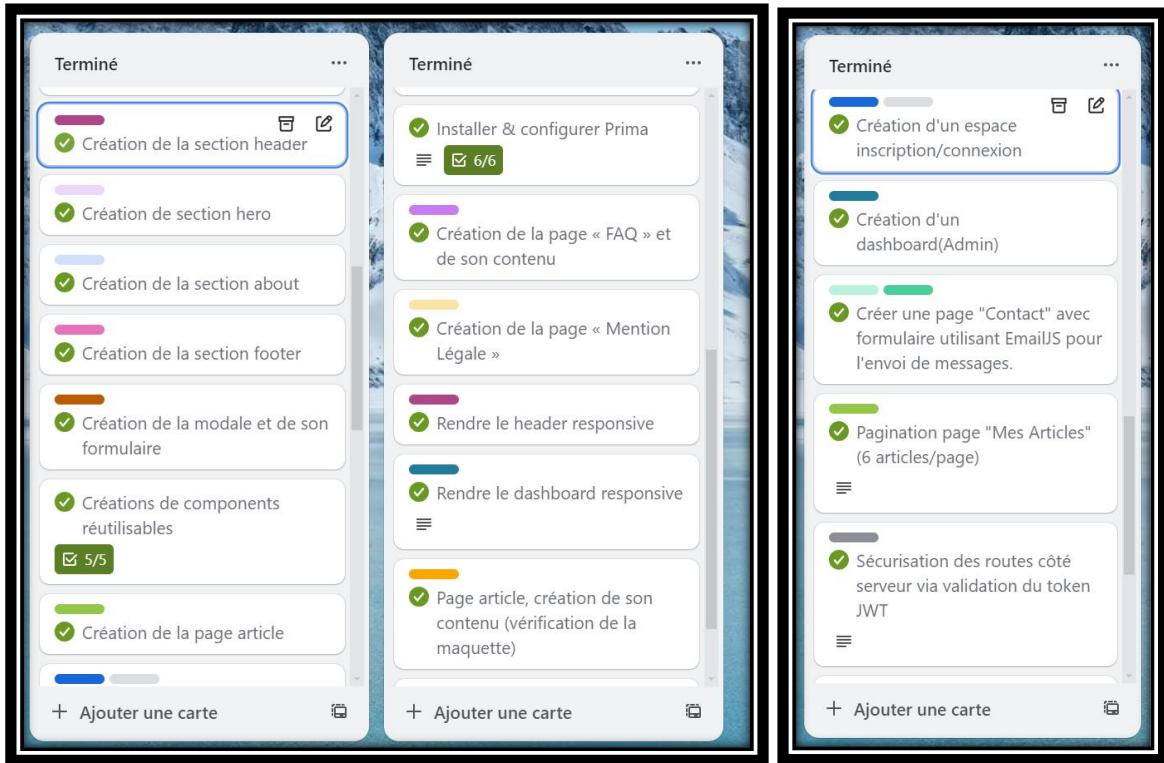
Travailler sur ce projet en conditions réelles, dans le cadre d'un stage en freelance, a été une expérience particulièrement enrichissante. Il ne s'agissait pas uniquement de mettre en pratique des compétences techniques, mais aussi de gérer un projet de A à Z, d'apprendre à organiser mon travail, de respecter des contraintes professionnelles et de transformer une idée en un produit concret et fonctionnel. Concevoir une première application web et mobile complète a renforcé ma confiance et confirmé mon envie de poursuivre dans ce métier exigeant et stimulant.

Je suis consciente que dans le développement web et mobile, l'apprentissage est permanent et ne s'arrête jamais. Cette dynamique d'évolution constante est pour moi une véritable source de motivation. Je continuerai à développer mes compétences et à m'investir dans une démarche d'amélioration continue, convaincue que chaque projet est une opportunité de progresser et de grandir, tant sur le plan technique que personnel.

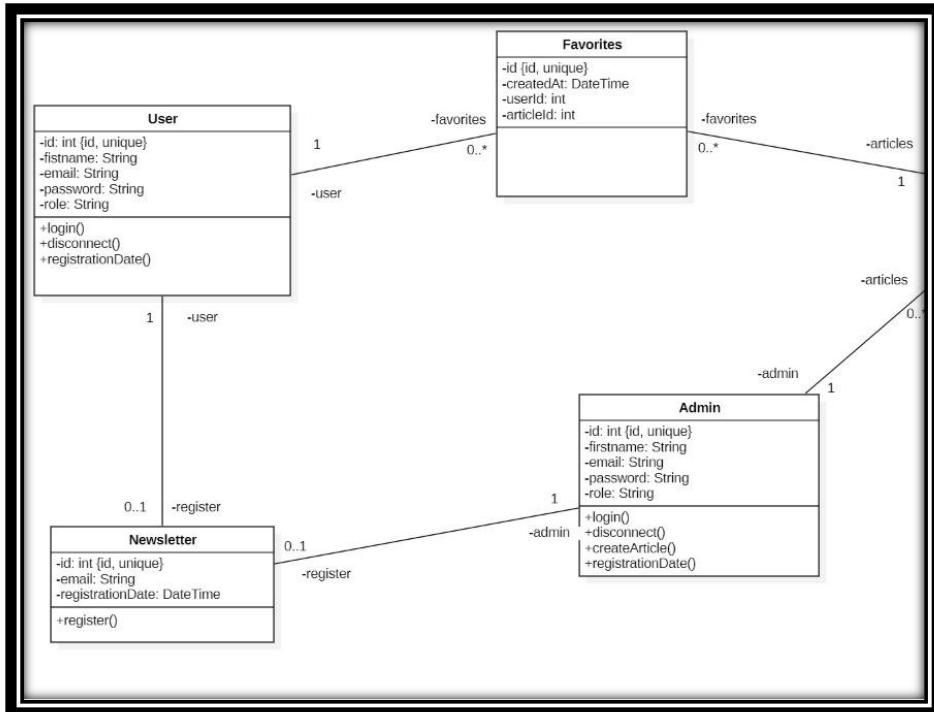
Ce projet constitue une étape fondatrice de mon parcours professionnel, et je suis heureuse de l'avoir mené à bien en conditions réelles. Je regarde désormais vers l'avenir avec enthousiasme et ambition, prête à relever de nouveaux défis dans le cadre de ma prochaine formation **Concepteur Développeur d'Applications & DevOps**, qui me permettra de consolider mes acquis, d'approfondir mes compétences et de participer activement à des projets innovants et porteurs de sens.

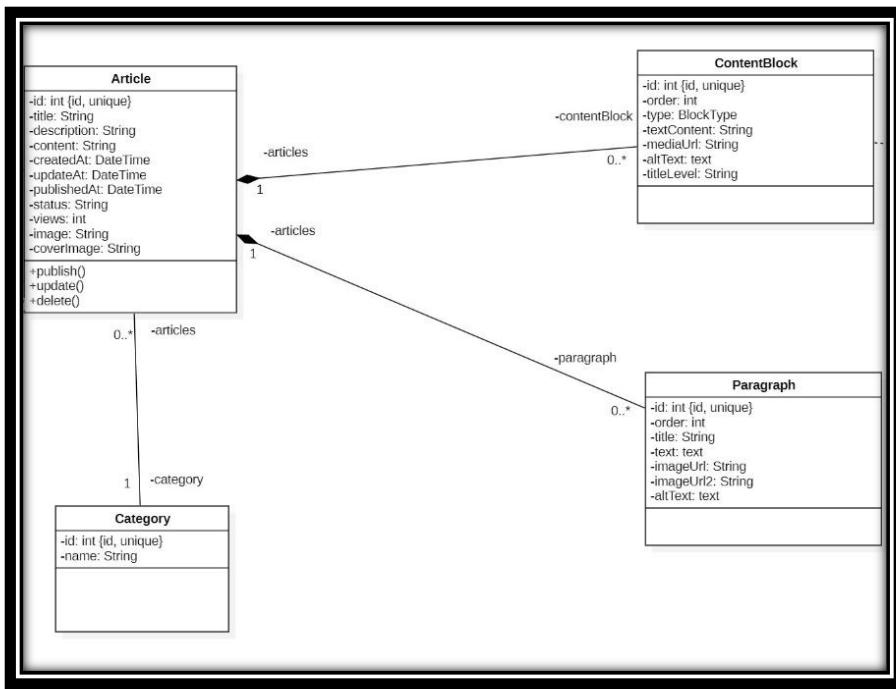
# ANNEXES

## Annexe 1 : « Organisation des tâches avec Trello »

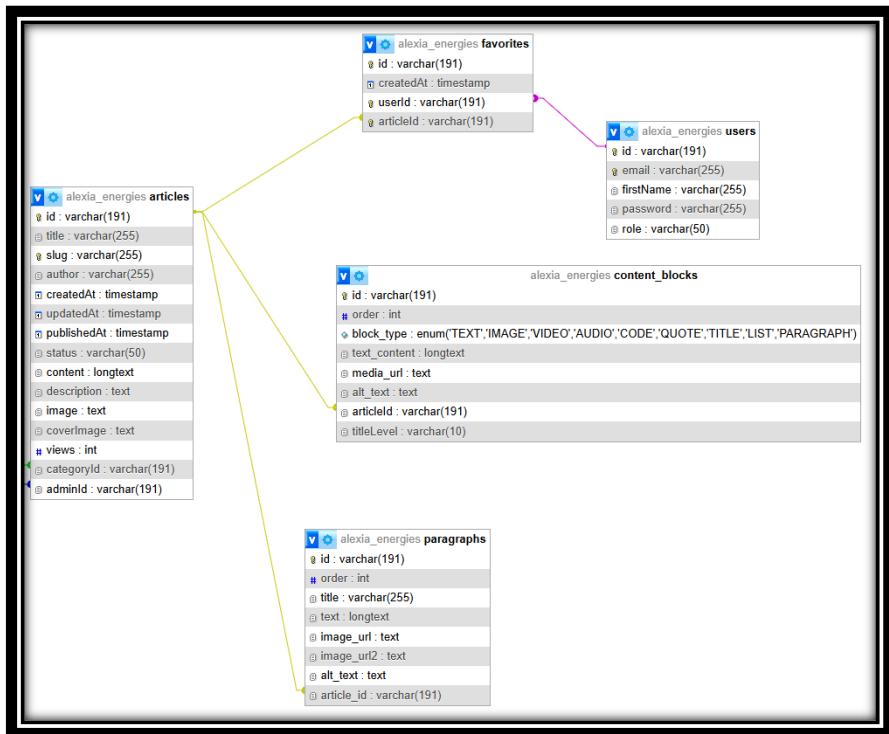


## Annexe 2 : « Diagramme UML »

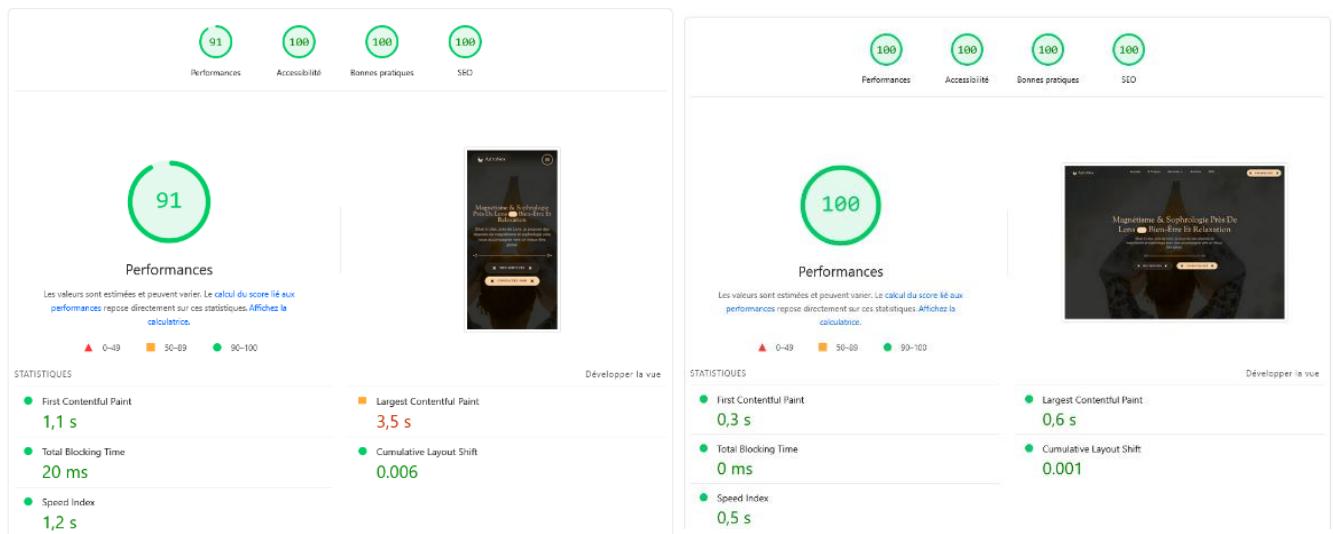




### Annexe 3 : « Schéma de bases de données PhpMyAdmin »



## Annexe 4 : Conception graphique « Lighthouse » - Desktop & Mobile



## Annexe 5 : Cahier des charges – App bien-être & formation

**Cahier des Charges — Application Web Bien-Etre & Formations**

**Objectif général**  
Développer une application web comportant :

1. Un site vitrine présentant les services de bien-être : **magnétisme, sophrologie, human design**.
2. Une prise de rendez-vous via Calendly, avec choix du mode (présentiel/distanciel).
3. Un espace de formation en ligne, avec accès sécurisé pour les participants.
4. Une interface d'administration pour gérer les utilisateurs, articles, et contenu de formation.

**Structure du site**

1. Site vitrine (public)
  - Accueil
  - Qui suis-je ?
  - Services
  - Articles
2. Espace formation en ligne (privé après paiement)
  - Fonctionnalités :
  - Espace personnel :
3. Espace d'administration (privé)

Accessible uniquement à l'administrateur du site.  
Fonctionnalités :

- Gestion des Utilisateurs (CRUD)
- Gestion des chapitres de formation (CRUD)
  - Titre
  - Contenu : texte, audio, vidéo, fichiers.
- Gestion des articles (CRUD)

**Stack technique**

Élement	Technologie
Framework	Next.js
Base de données	Neon (PostgreSQL)
ORM	Prisma
Authentification	BcryptJs + JsonWebToken
Paiement	Stripe
Emails/Resend	Email.js (coté client) / Resend (coté serveur)
Icons	React Icons
Formulaires	React Hook Form + Zod
Déploiement	Vercel

**Contraintes & exigences techniques**

- Site responsive (desktop / mobile)
- SEO friendly (balises meta, titres, performances)
- Sécurisation des pages privées.

Utilisation de Stripe Webhooks pour débloquer l'accès formation

• Expérience utilisateur fluide, interface apaisante, moderne et lisible