

R-TRAINING

Application web et web mobile de musculation et de fitness permettant la gestion de programmes d'entraînements et le suivi des performances.



*Projet réalisé en autonomie dans le cadre de la présentation au
Titre professionnel Développeur web et web mobile*

présenté par

Ryan LADMIA
La Plateforme_ - formation CDPI

SOMMAIRE :

INTRODUCTION

LISTE DES COMPÉTENCES COUVERTES PAR LE PROJET

- I - Développer la partie front-end d'une application web ou web mobile sécurisée
 - A. Installer et configurer son environnement de travail en fonction du projet web ou web mobile
 - B. Maquetter des interfaces utilisateur web ou web mobile
 - C. Réaliser des interfaces utilisateur statiques web ou web mobile
 - D. Développer la partie dynamique des interfaces utilisateur web ou web mobile
- II - Développer la partie back-end d'une application web et web mobile sécurisée
 - A. Mettre en place une base de donnée relationnelle
 - B. Développer des composants d'accès aux données SQL et NoSQL
 - C. Développer des composants métiers côté serveur
 - D. Documenter le déploiement d'une application dynamique web ou web mobile

RESUMÉ DU PROJET

CAHIER DES CHARGES

- I - Besoins et objectifs de l'application
 - A. Besoins
 - B. Objectifs
 - C. Cibles
- II - Users stories
- III - Arborescence
- IV - MVP
 - A. Administrateurs
 - B. Utilisateurs
- V. Fonctionnalités détaillées des pages
- VI. Évolutions potentielles
 - A. Administrateurs
 - B. Utilisateurs
 - C. Entraîneurs
- VII. Wireframes
- VIII. Charte graphique et logo
- IX. Exemple de maquette

SPÉCIFICATION TECHNIQUES

- I - Technologies
- II - Navigateurs compatibles
- III - Possibilité de déploiement
- IV - Création de la base de données
 - A. MCD
 - B. MLD
- V - Routes front et back
 - A. Back-end
 - B. Front-end

RÉALISATIONS PERSONNELLES

- I - Suivi d'entraînement
 - A. Back-end
 - B. Front-end
- II - Création du programme d'entraînement public
 - A. Back-end
 - B. Front-end

TEST UNITAIRES

VULNÉRABILITÉ DE SÉCURITÉ ET VEILLE

- I - Veille technologique
 - A. Prisma
 - B. Cloudinary
 - C. Nodemailer et SMTP
- II - Veille de sécurité
 - A. JWT et cryptage du mot de passe utilisateur
 - B. Sécurisation des requêtes
 - C. Validation des entrées
 - D. Routes protégées
- III - Processus de recherche et traduction

CONCLUSION

INTRODUCTION :

Depuis toujours, j'ai été attiré par la logique, les systèmes structurés et l'envie de comprendre comment les choses fonctionnent. Pourtant, mon parcours scolaire m'a d'abord orienté vers un baccalauréat littéraire, avec une spécialisation en mathématique et une option en latin, avant de poursuivre des études à l'université de droit et de sciences politiques de Aix-en-Provence. Bien que ces filières m'aient apporté rigueur, méthode et capacité d'analyse, je ressentais un certain éloignement par rapport à mes intérêts d'enfance : l'utilisation de l'ordinateur et du web.

C'est en explorant moi-même le monde du web et de la programmation, suite aux conseils d'un proche, que j'ai redécouvert cette passion initiale. Petit à petit, j'ai commencé à apprendre les bases du développement web en autodidacte, à travers des ressources en ligne et des tutoriels sur des plateformes comme OpenClassroom ou Youtube. Ces découvertes m'ont rapidement convaincu de réorienter mon parcours de façon plus concrète et alignée avec mes aspirations profondes.

Souhaitant structurer mes connaissances et acquérir une véritable expérience, j'ai décidé d'intégrer une formation de développeur web et web mobile. Ce choix m'a permis non seulement d'approfondir mes compétences techniques, mais aussi de valider mes acquis à travers des projets concrets, réalisés en groupe dans un cadre professionnel. Toutes ces expériences ont été des étapes essentielles, tant sur le plan technique et professionnel que sur le plan humain.

Le projet que je vais vous présenté est le fruit de toutes ces découvertes et expériences et constitue une mise en application de mes compétences.

Cette formation et ce projet m'ont permis de gagner en assurance, en autonomie et de confirmer mon choix de carrière dans un domaine qui me passionne.

LISTE DES COMPÉTENCES COUVERTES PAR LE PROJET :

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

A. Installer et configurer son environnement de travail en fonction du projet web ou web mobile

Afin de mener à bien mon projet, j'ai adapté mon environnement de travail afin de répondre a des besoins spécifiques. Si j'ai pour habitude d'utiliser des bases de données en SQL via PHPmyadmin ou SQLite avec Dbeaver, j'ai cette fois fais le choix d'installer et de configurer PostgreSQL associé à l'ORM Prisma pour la gestion de la base de données.

Ce changement m'a permis de me familiariser avec des technologies plus modernes et adapté à un environnement Node.js, tout en découvrant les avantages d'un ORM tel que Prisma.

B. Maquetter des interfaces utilisateur web ou web mobile

Pour de partir sur de bonne bases j'ai décider de commencer par travailler sur la phase de préparation. Celle-ci consistait, tout d'abord, par réaliser des users stories pour définir ce que les différents utilisateurs pourront faire sur mon application. J'ai ensuite réaliser quelques wireframes pour avoir une idée globale de ce à quoi allait ressembler mon application et quelle serait la structure des pages principales. Ensuite j'ai établi la charte graphique et en me basant sur celle-ci, j'ai réaliser une maquette de mon application.

C. Réaliser des interfaces utilisateur statiques web ou web mobile

En m'appuyant sur les users stories, les wireframes et la maquette, j'ai conçu les interfaces statiques du site web et web mobile. J'ai ainsi réalisé les principales pages : la page d'accueil, la page qui liste les exercices, la page connexion, la page inscription, la page mot de passe oublié, la page 404 et la page accès non autorisé.

Ces pages statiques constituent le socle de l'application. Elles présentent le même contenu et la même structure pour tous les visiteurs, indépendamment de leur statut de connexion. Bien qu'elles soient qualifiées de « statiques » en terme de personnalisation, elles intègrent des éléments interactifs et des comportements asynchrones pour permettre l'échange de données (comme la récupération des images des exercices) sans recharge complet de la page, offrant une meilleure fluidité à l'utilisateur.

D. Développer la partie dynamique des interfaces utilisateur web ou web mobile

Après avoir conçu les interfaces statiques, j'ai développé leur version les pages dynamiques en les connectant à la logique métier et aux données du backend. Grâce à une architecture réactive, les différentes pages (dashboard, profil, suivi d'entraînement...) se mettent à jour avec les données personnelles de l'utilisateur connecté, sans recharge complet de la page.

Les interactions sont gérées de manière asynchrone via React et des appels API, permettant par exemple de récupérer les exercices en temps réel, de valider un formulaire ou vérifier les droits d'accès d'un utilisateur. Ce fonctionnement assure une expérience fluide et responsive tant sur web que sur mobile.

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

A. Mettre en place une base de donnée relationnelle

Lors de la phase de préparation, j'ai également structuré la manière dont les données allaient être stockées. Pour cela, j'ai réalisé un MCD (Modèle Conceptuel de Données) puis un MLD (Modèle Logique de Données), ce qui m'a permis de définir précisément les entités, leurs attributs et les relations entre elles.

Afin d'assurer la cohérence, l'intégrité et la non-redondance des données, j'ai appliqué le processus de normalisation jusqu'à la troisième forme normale (3NF). Ce choix garantit une meilleure organisation des données relationnelles, les futures évolutions du modèle ainsi que l'optimisation des performances.

Pour la gestion de la base de données, j'ai choisi PostgreSQL, un système relationnel robuste, moderne et performant associé à l'ORM Prisma afin de faciliter les interactions entre le serveur et la base de données de manière typée et sécurisée.

B. Développer des composants d'accès aux données SQL et NoSQL

Pour interagir avec les données stockées dans la base relationnelle (PostgreSQL), j'ai utilisé Prisma Client, un ORM moderne offrant une interface intuitive et type-safe pour écrire des requêtes SQL de manière fiable et sécurisée.

J'ai mis en place des requêtes optimisées à l'aide des options include et select, afin de limiter les champs retournés et réduire les appels inutiles à la base de données, ce qui améliore significativement les performances.

Des transactions ont également été utilisées pour garantir la cohérence des données lors d'opérations complexes (comme une inscription avec création de profil ou la gestion des historiques des entraînements).

Pour la gestion des erreurs, j'ai intégré un middleware centralisé capable de capturer et traiter les exceptions liées aux requêtes SQL, assurant ainsi une remontée d'erreurs claire et un comportement cohérent de l'application.

C. Développer des composants métier côté serveur

Pour structurer la logique applicative, j'ai adopté une architecture en couches respectant le principe de séparation des responsabilités. Cette approche garantit la maintenabilité, la testabilité et la réutilisation de code métier.

J'ai organisé le back-end selon le pattern MVC (Model, View, Controller) adapté, avec une séparation claire entre les routes, les controllers, les services et les middlewares.

Les services encapsulent des algorithmes métier spécifiques au domaine sportif (durée des entraînement, statistique de progression, gestion des programmes sportifs, suivi d'entraînement...)

J'ai aussi implémenté un système de contrôle d'accès en fonction des rôles de l'utilisateur : utilisateur, entraîneur, administrateur.

Les middleware d'autorisation vérifient dynamiquement, les rôles via des requêtes optimisées à la base de données, permettant une gestion flexible des permissions sans duplication de code.

Chaque route est protégée par des middleware en cascade (vérification et renouvellement des tokens JWT, contrôle des permissions et schéma zod).

D. Documenter le déploiement d'une application dynamique web ou web mobile

Le déploiement de l'application a été documenté étape par étape afin d'en assurer la reproductibilité. Cela inclut la configuration de l'environnement distant, l'installation des dépendances, le déploiement du front-end et du back-end, ainsi que la connexion à la base de donnée. Chaque étape a été consignée pour faciliter la maintenance, la mise à jour ou une éventuelle migration de l'application.

RÉSUMÉ DU PROJET :

R-Training est une plateforme web complète offrant aux utilisateurs la possibilité de créer, gérer et suivre leurs programmes d'entraînement sportifs. Ils peuvent consulter les fiches des différents programmes et exercices pour connaître leurs spécificités: groupes musculaires sollicités, conseils d'exécution, indications techniques.

L'application propose trois niveaux d'accès :

- Utilisateurs : création de programme personnels, suivi des séances en temps réel, consultations de statistiques de progression
- Entraîneurs : ajout d'exercices, création de programmes publics, gestion de contenu
- Administrateur : gestion globale des utilisateurs, supervision de la plateforme, accès à l'ensemble des fonctionnalités

Ce projet s'adresse à tous les profils, qu'ils soient débutants ou confirmés et leur donne la possibilité de se renseigner rapidement et efficacement sur les bonnes pratiques en musculation. Chaque utilisateur peut suivre et noter sa séance, enregistrer ses paramètres et ses performances puis les visualiser sous forme de graphiques clairs et détaillés.

Ainsi R-Training aide à mieux s'informer et organiser ses séances, à suivre sa progression de manière concrète et structurée.

CAHIER DES CHARGES :

I - Besoins et objectifs de l'application

A. Besoins

Avant de concevoir l'application, je me suis posé une question simple mais essentielle : de quoi aurait besoin un pratiquant de musculation, qu'il soit débutant ou expérimenté, pour organiser sa séance dans les meilleures conditions possibles ?

L'objectif était de répondre à ces besoins de manière claire, sécurisée et motivante.

Tout d'abord, un pratiquant a besoin d'une liste d'exercices fiables et validés, soigneusement sélectionnés. Les exercices dangereux, approximatifs ou à la mode sur les réseaux sociaux, qui n'apportent aucun bénéfice réel voire peuvent provoquer des blessures, doivent être exclus.

Chaque exercice doit être accompagné d'une fiche descriptive indiquant :

- les groupes musculaires sollicités
- les consignes d'exécution technique
- les précautions à prendre, afin d'assurer une pratique efficace et sans risque

Ensuite, un système de suivi des séances est indispensable.

L'utilisateur doit pouvoir noter ses performances à chaque entraînement, en enregistrant des paramètres comme :

- le nombre de séries effectuées
- le nombre de répétition effectués
- les charges utilisées (le cas échéant)
- les temps de repos

Enfin, il est essentiel de maintenir la motivation des utilisateurs. Pour cela j'ai choisi d'intégrer une visualisation graphique de leur progression, afin de pouvoir suivre ses résultats au fil du temps, de manière concrète et visuelle, de les inciter à s'améliorer et limiter les risques de découragement ou d'abandon.

L'application vise donc à offrir un accompagnement complet, structuré et motivant à tous les pratiquants pour qu'ils puissent progresser en toute sécurité.

B. Objectifs

L'objectif principal de l'application R-Training est de fournir un outil simple, complet et fiable pour aider les pratiquants de musculation à organiser, suivre et optimiser leurs entraînements.

Plus précisément, l'application vise à :

- centraliser l'information utile pour l'entraînement : proposer un catalogue d'exercices fiables, bien décrits, classés par groupes musculaire, avec des conseils techniques pour favoriser une exécution correcte et sécurisée.
- accompagner l'utilisateur dans son parcours sportif : permettre de planifier et personnaliser ses séances, en fonction de ses objectifs et de son niveau
- favoriser le suivi de progression : offrir un système de journal d'entraînements où l'utilisateur peut enregistrer ses performances à chaque séance, consulter ses historiques, et visualiser ses progrès sous forme de graphiques.
- stimuler la motivation : encourager les bonnes habitudes sportives par une expérience utilisateur engageante, et par la valorisation des efforts réalisés (progression visible, sentiment d'accomplissement...)
- offrir une expérience sécurisée et personnalisée : mettre en place un système d'authentification avec rôles pour adapter les droits et les accès aux fonctionnalités
- garantie la possibilité et l'accessibilité : l'application est responsive et adaptée à une utilisation web comme mobile afin que l'utilisateur puisse suivre ses séances facilement, où qu'il soit.

C. Cibles

Mon application web s'adresse à un large public, avec une attention particulière portée à la personnification de l'expérience utilisateur selon le profil et les besoins.

Les pratiquants débutants ont souvent besoin d'un cadre clair, de conseils fiables et d'un accompagnement pédagogique. R-Training leur permet de découvrir les exercices de base validés, de comprendre comment les exécuter correctement, de consulter des programmes préétablis avant de constituer leurs propre programme et de suivre leur évolution au fil des séances.

Les pratiquants intermédiaires ou confirmés qui souhaiteraient optimiser leurs performances et suivre leurs progression pourront créer leur propre séance sur mesure, affiner leurs charge, séries, répétitions et temps de repos et analyser leurs performances sur la durée grâce aux graphiques.

Les entraîneurs disposent d'un accès spécifique leur permettant d'ajouter de nouveaux exercices, de créer des programmes publics (qui contrairement aux programmes des utilisateurs peuvent être consultés par tous le monde), et donc de proposer du contenu encadré et qualitatif.

Les administrateurs auront accès à toutes les précédentes fonctionnalités, en plus de superviser l'ensemble des utilisateurs, gérer les rôles et contrôler le bon fonctionnement global de la plateforme.

II - Users Stories

Parcours utilisateur :

Les rôles :

- non connecté : accès limité au site,
 - consulter page d'accueil,
 - liste des exercices
 - inscription
 - connexion
 - mot de passe oublié
 - page 404
 - page accès non autorisé
- user : rôle attribué par défaut
accès à toutes les fonctionnalités du site sauf ajout d'exercices ou ajout de programme pré-établis (publics)
 - page consultation de la page détails des exercices
 - page de la liste des programmes pré-établis
 - page ajout de programme perso
 - page mes programmes perso
 - page mon suivi de séance (cocher des exercices réalisés, entrer ses charges et reps, voir des stats de progression (graphes etc))
 - dashboard (graphiques des performances)
 - profil
- trainer : accès aux fonctionnalité de user et plus :
Rôle attribué par l'admin
 - ajout d'exercice
 - ajout de programme/séance
- admin : accès à toute l'appli et au back office :
 - page gestion des utilisateurs

III - Arborescence des fichiers



IV - MVP

Au début du projet, j'avais envisagé deux rôles essentiels.

A. Les administrateurs

Les administrateurs, pour la supervision de la plateforme :

- authentification sécurisée (avec rôles et tokens JWT)
- gestion complète des utilisateurs
- accès en lecture seules aux fonctionnalités des utilisateurs
- dashboard d'administrateur affichant des statistiques globales : nombre d'utilisateurs actifs, programmes créés, nombre d'exercices...

B. Les utilisateurs

Les utilisateurs disposent de fonctionnalités clés pour un usage autonome et motivant :

- inscription et connexion sécurisée, avec vérification par email
- consultation des programmes préétablis (publics)
- consultation des exercices
- création de programme personnalisé (non visible par les autres utilisateurs)
- Suivi d'entraînement en temps réel : saisie des performances (séries, répétitions, charges, temps de repos)
- consultation de l'historique des entraînements
- consultation des performances sous forme de graphiques

V - Fonctionnalités détaillées des pages

Commençons par voir les pages auquel un utilisateur non authentifié (ou non connecté) aura accès.

La page d'accueil :

Il s'agit de la page de présentation de l'application avec le logo en grand et centré.

La page listant les exercices :

Chaque exercice est contenu dans une carte au fond blanc contenant une image de l'exercice en question, son nom, une description, le groupe musculaire sollicité (ex les jambes) et les muscles travaillés durant l'exercice (ex quadriceps). Les utilisateurs ont également la possibilité de filtrer les exercices par groupe musculaire (pectoraux, dos, jambes...)

La page d'inscription :

Cette page contient un formulaire afin qu'un utilisateur puisse s'inscrire dans la base de donnée pour avoir accès aux fonctionnalités de l'application. Les utilisateurs doivent obligatoirement renseigner leur nom, leur prénom, leur adresse email, leur mot de passe et confirmer leur mot de passe une seconde fois afin d'éviter d'éventuelles erreurs. Ils ont la possibilité de renseigner leur date de naissance et leur numéro de téléphone mais ces champs sont optionnels. Cette page contient également un lien vers la page de connexion.

La page connexion :

Les utilisateurs peuvent se connecter en renseignant leur email et leur mot de passe. S'ils sont corrects, ils sont redirigés vers leur dashboard. Sinon un message d'erreur apparaît « Email ou mot de passe incorrect » sans préciser lequel pour plus de sécurité.

Cette page contient également un lien vers la page d'inscription et en cas de mot de passe oublié.

La page mot de passe oublié :

Cette page permet l'envoie d'un lien pour réinitialiser le mot de passe via un email fourni par l'utilisateur au moment de son inscription. Le lien est vérifier via un token stocké dans la base de données.

La page vérification de l'adresse email :

Lors de l'inscription, dans la base de données, le champs isVerified est par défaut false. Pour se connecter, l'utilisateur doit vérifier son adresse email via un token stocké en base de données, et le champs passe alors à true. L'utilisateur peut alors se connecter.

La page 404 et la page non autorisé :

Si un utilisateur entre une mauvaise URL, il est redirigé vers la page 404, sur laquelle il a la possibilité de revenir à la page précédente ou de revenir à la page d'accueil. Si il essaye d'accéder à une page pour laquelle il n'a pas les autorisations, comme une page admin, il est redirigé vers la page non autorisé.

Voyons maintenant les pages auquel un utilisateur au rôle basique (user) a accès en se connectant avec ses identifiants.

La page dashboard :

Lorsqu'un utilisateur se connecte, il est redirigé vers son dashboard. Sur cette page, il a accès au liens rapide vers les pages suivie d'entraînement, historique des entraînements, la liste des programmes publics, la liste des exercices, la création de programme personnel, et la liste de ses programmes personnels.

De plus il peut également consulter ses statistiques d'entraînement comme le total des entraînements effectués, la dernière séance, le nombre de séances réalisées durant la semaine et le nombre d'exercices exécutés dans la semaine.

Puis dans la partie statistiques, il peut voir dans un diagramme circulaire (ou diagramme camembert) qui représente le pourcentage des muscles travaillés . Dans une autre carte on peut voir des graphiques représentant la progression du pratiquant selon l'exercice et la charge utilisée. Ces graphiques peuvent être filtrés par groupes musculaires et par exercice.

La page profil :

Sur sa page profil, un utilisateur peut consulter et modifier ses informations personnelles comme le nom, le prénom, l'adresse email, le rôle, le numéro de téléphone et la date de naissance et le mot de passe. Ce dernier n'est pas visible mais peut être modifié en entrant un nouveau mot de passe et en le confirmant dans un second input.

L'utilisateur n'a pas la possibilité de modifier son rôle. Seul l'admin le peut dans la page de gestion des utilisateurs.

On peut également ajouter une photo de profil si on le désire.

La page historique des entraînements :

Cette page liste tous les entraînements effectués par le pratiquant. Il peut faire une recherche par mot clé comme le nom du programme de la séance, ou de l'exercice. Il peut également filtrer les entraînements selon la date (année, mois ou semaine).

Ces séances comportent une date à laquelle elle a été effectuée, le programme lié, le nom de la séance, le nombre d'exercices, de séries et sa durée approximative. En cliquant sur la carte d'un entraînement, il a accès aux détails comme les noms des exercices, le nombre de séries, de répétition, la charge utilisée ou le temps de repos.

La page du suivi des entraînements :

C'est sur cette page que l'utilisateur va choisir le programme qu'il voudra suivre. Après avoir choisi son programme, il sélectionne la séance voulue.

À ce moment il peut commencer son entraînement, en cliquant sur le bouton. La page se met à jour de façon asynchrone permettant de voir les exercices à

effectuées ainsi que les paramètres contenus dans le programme. L'utilisateur peut alors choisir par quel exercice il veut commencer, selon son envie ou la disponibilité des équipements de sa salle de sport. Il renseigne alors le nombre de séries, de répétitions, la charge utilisée (s'il y en a une) et le temps de repos. Il peut ensuite démarrer un compte à rebours durant lequel il va se reposer. Lorsqu'il a fini l'exercice, il lui reste à cliquer sur le bouton « Terminer l'exercice » et passer suivant.

Lorsqu'il a fini sa séance en ayant réalisé, ou pas, tous les exercices, ses performances sont enregistrées. Il peut également annuler le suivi en cliquant sur le bouton annuler.

La page détails des exercices :

Tout utilisateur connecté ou pas peut accéder à la liste des exercices. Mais pour pouvoir voir les détails, comme les consignes techniques, il doit être authentifié.

La page programmes :

Cette page liste les programmes préétabli ou publics accessibles à tous les utilisateurs authentifiés. Ces programmes sont créés par les utilisateurs au rôle entraîneur ou admin.

La page mes programmes :

Le pratiquant peut voir et modifier les programmes qu'il a lui-même créé. Afin d'éviter de communiquer des informations erronées, seul l'utilisateur ayant créé un programme personnel aura accès à celui-ci.

La page création de programme personnel :

Les utilisateurs authentifiés au rôle utilisateur (user) auront accès à cette page sur laquelle ils peuvent créer leur programme personnel visible uniquement par eux-mêmes.

Les utilisateurs au rôle entraîneur auront accès aux fonctionnalités précédentes mais auront des prérogatives supplémentaires.

La page création d'exercice :

Un entraîneur pourra créer un nouvel exercice qui s'ajoutera à la liste visible par tous les utilisateurs de la plateforme. Les paramètres de ces exercices sont le nom, le groupe musculaire, les muscles sollicités, la description, l'exécution technique et une image de l'exercice en question.

La page création de programme public :

Ces programmes visibles par tous les utilisateurs ne peuvent qu'être créés que par un entraîneur ou un administrateur pour éviter la propagation de mauvaise pratiques sportives.

Ces programmes comportent un nom, une description et différents nombre de séance constitués d'exercices aux différents paramètres selon l'objectif visé par le programme ou le pratiquant.

La page dashboard :

Le dashboard de l'entraîneur est légèrement différent de celui de l'utilisateur ordinaire. S'il dispose des même fonctionnalités, il a des informations supplémentaires qui lui sont affichées.

Ces informations sont le nombre de programme et d'exercices édité par lui-même.

Le dernier rôle est celui d'administrateur.

La page dashboard :

L'administrateur dispose de plus d'informations que les autres utilisateur. Il peut consulter le nombre total d'utilisateurs, ceux actifs, les nouveaux inscrits (durant la dernière semaine), et le nombre d'utilisateur en fonction des rôles.

La page gestion des utilisateurs :

C'est à partir de cette page que l'admin peut consulter les fiches de tous les utilisateurs comportant leur photo de profil, leur nom, prénom, adresse email, rôle et s'ils sont actifs ou non. En cliquant sur la carte d'un utilisateur, l'admin peut modifier toutes ses information sauf son mot de passe. Il peut supprimer sa photo de profil mais pas en mettre une autre. Bien sur l'admin peut également changer le rôle, rendre le compte inactif ou le supprimer.

VI - Évolutions potentielles

A. Administrateur :

Les évolutions de mon application comprennent une boutique en ligne pour que les pratiquants puisse commander du matériel sportif ou des compléments alimentaire.

L'administrateur aura une seconde page dans le back office pour la gestion des produits et des stockes.

Les administrateurs assureront également la modération du contenu.

B. Utilisateurs

Les utilisateur pourraient disposer d'un chat pour communiquer en temps réel avec les autres utilisateurs ou les entraîneurs. De plus ils pourraient bénéficier d'un système de gamification afin de parfaire l'expérience utilisateur et augmenter la motivation.

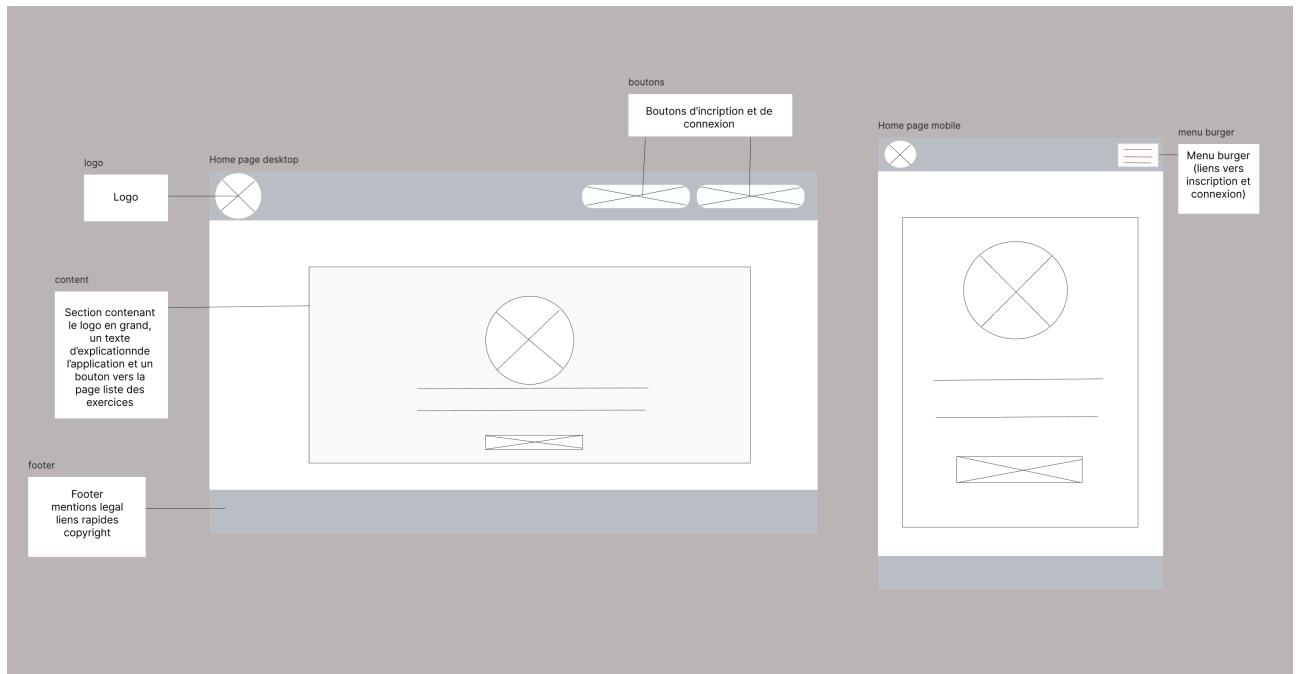
Une boutique en ligne et un panier accessible à tout utilisateur authentifié sera à leur disposition pour commander du matériel sportif ou des compléments alimentaires.

C. Entraîneurs

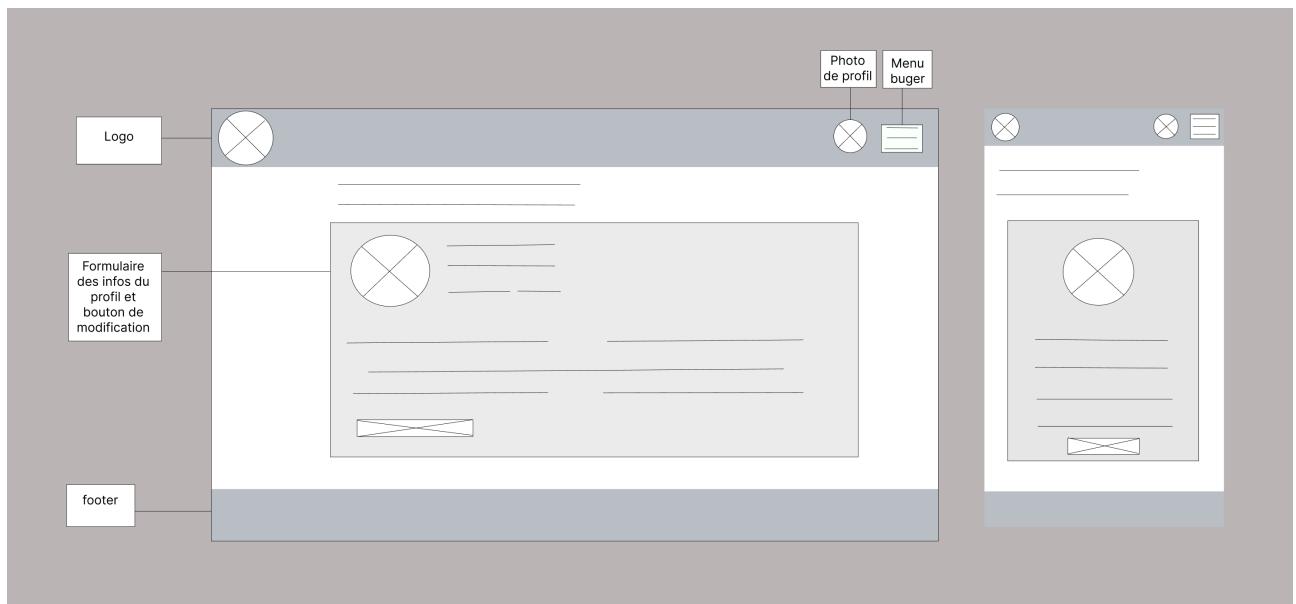
Les entraîneurs auront la possibilité de proposer leurs services et leurs conseils aux utilisateur. Ils pourront aussi proposer de nouveaux programmes personnalisés et adaptés à leurs clients.

VII - Wireframes

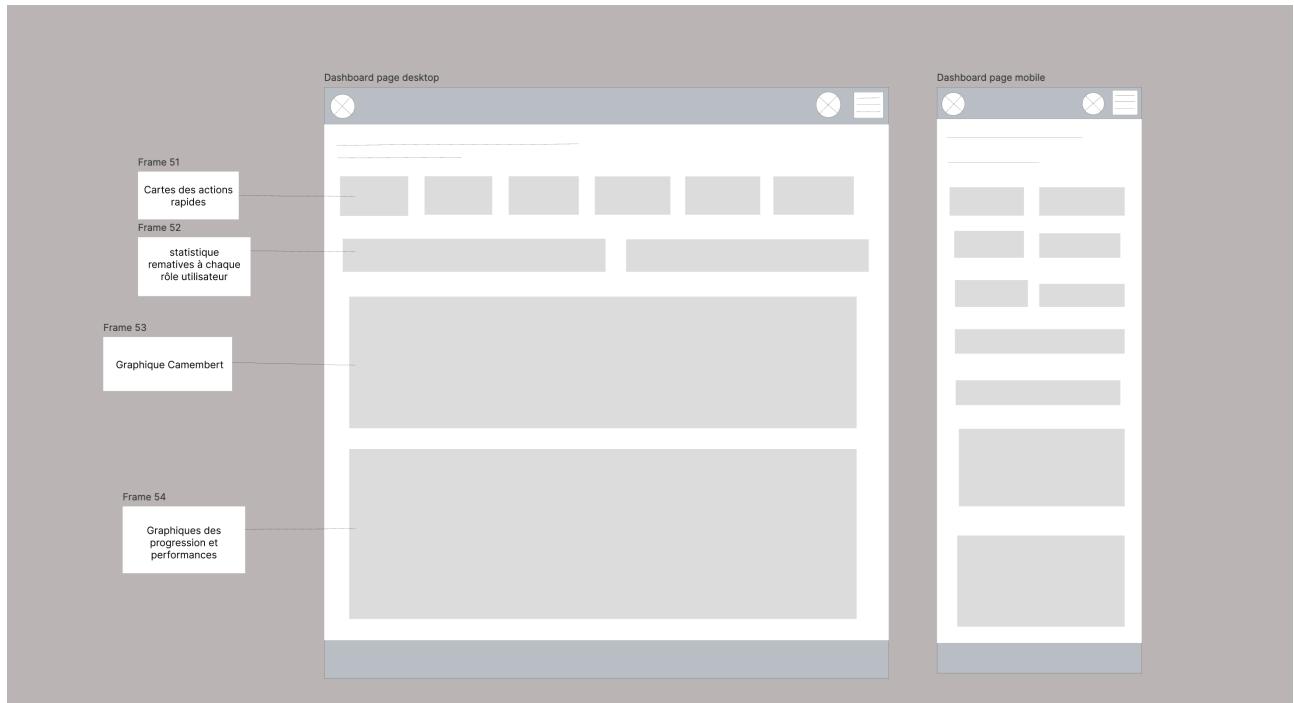
Page d'accueil :



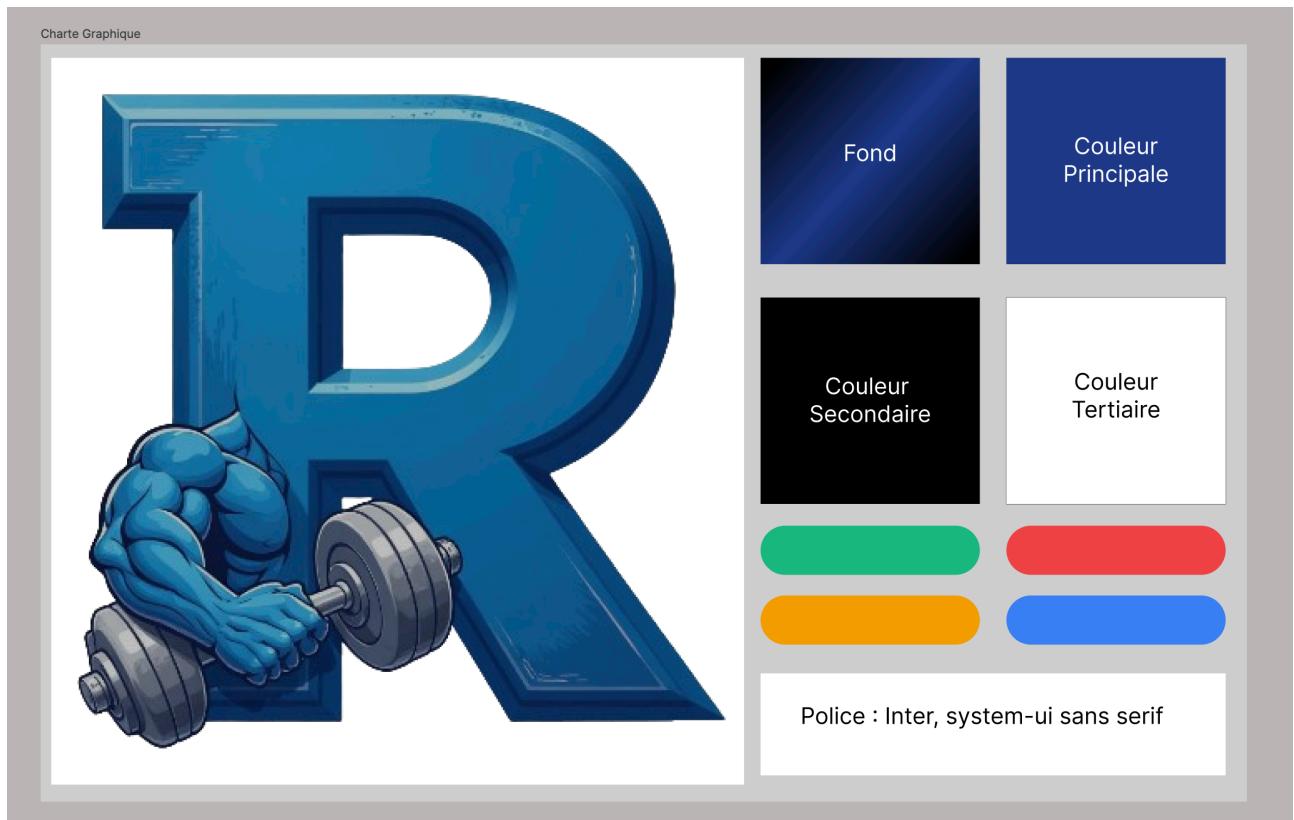
Page profil :



Page dashboard :



VIII - Charte graphique et logo



IX - Exemple de maquette

Page d'accueil :

The image shows two side-by-side screenshots of the R-Training website's home page, one for desktop and one for mobile.

Home page desktop: The background is dark blue with a large white circle containing a stylized blue 'R' logo. Below the logo, the text "R-Training" is written in a bold, sans-serif font. Underneath it, the tagline "Transformer votre entraînement" is displayed. A blue button labeled "Découvrez nos exercices" is centered at the bottom. At the very bottom of the page, there is a footer bar with links: "Accueil", "Exercices", "Créer un compte", "Se connecter", "support@r-training.com France", and a copyright notice "@2025 R-Training. Tous droits réservés".

Home page mobile: The layout is similar but adapted for a smaller screen. The "R-Training" logo and tagline are present. A blue button labeled "Découvrez nos exercices" is at the bottom. The footer bar includes links: "Accueil", "Exercices", "Créer un compte", "Se connecter", "support@r-training.com France", and a copyright notice "@2025 R-Training. Tous droits réservés".

Page profil :

The image shows two side-by-side screenshots of the R-Training user profile page, one for desktop and one for mobile.

Profile page desktop: The title is "Profil administrateur". It displays a user profile picture of a cartoon character. Below the profile picture, there is a form for updating personal information: "Prénom Nom", "adresse email", "Rôle", "Compte créé le:", "Prénom *", "Nom *", "Email *", "Numéro de téléphone", "Date de naissance", and a "Modifier le profil" button. At the bottom of the page, there is a footer bar with links: "Accueil", "Exercices", "Créer un compte", "Se connecter", "support@r-training.com France", and a copyright notice "@2025 R-Training. Tous droits réservés".

Profile page mobile: The title is "Profil administrateur". It shows a similar profile view and form for updating personal information. The footer bar includes links: "Accueil", "Exercices", "Créer un compte", "Se connecter", "support@r-training.com France", and a copyright notice "@2025 R-Training. Tous droits réservés".

Page liste des exercices :

Exercises list page desktop

Nos exercices :
Filtrer par groupe musculaire
[Tous les exercices](#)

Face pull
Description
Groupe musculaire
Dos
Muscles ciblés
Muscles dos Muscles épaules Muscles bras
Crée par

Squat
Description
Groupe musculaire
Jambes
Muscles ciblés
Muscles jambes Muscles fessiers
Crée par

Soulevé de terre jambes tendues
Description
Groupe musculaire
Jambes
Muscles ciblés
Muscles jambes Muscles fessiers
Crée par

La plateforme complète pour transformer votre approche du fitness et de la musculation avec des outils pratiques et intuitifs.
©2025 R-Training. Tous droits réservés

Accueil Exercices Crée un compte Se connecter support@r-training.com France

Exercises list page mobile

Nos exercices :
Filtrer par groupe musculaire
[Tous les exercices](#)

Face pull
Description
Groupe musculaire
Dos
Muscles ciblés
Muscles dos Muscles épaules Muscles bras
Crée par

Squat
Description
Groupe musculaire
Jambes
Muscles ciblés
Muscles jambes Muscles fessiers
Crée par

Soulevé de terre jambes tendues
Description
Groupe musculaire
Jambes
Muscles ciblés
Muscles jambes Muscles fessiers
Crée par

La plateforme complète pour transformer votre approche du fitness et de la musculation avec des outils pratiques et intuitifs.
©2025 R-Training. Tous droits réservés

Accueil Exercices Crée un compte Se connecter support@r-training.com France

SPÉCIFICATIONS TECHNIQUES :

I - Technologies

Front-end :

L'interface de l'application a été développée avec React 22.15.0, un framework JavaScript moderne, associé à Vite 6.3.1 pour un environnement de développement rapide et optimisé.

La mise en forme repose sur Tailwind CSS 3.4.17 (framework utilitaire) et Radio UI 1.1 pour des composants accessibles.

Le routage client est géré avec React Router DOM v7, tandis que les appels API sont effectués via Axios 1.9.0.

Les icônes sont intégrées grâce à Lucide React.

Back-end :

Le serveur repose sur Node.js 22.15.0 et le framework Hono 4.7.8, choisi pour sa légèreté et ses performances. Les données sont gérées via PostgreSQL, avec Prisma 6.7.0 comme ORM pour faciliter les requêtes.

L'authentification est sécurisée grâce à JsonWebtoken 9.0.2 (JWT) et les mots de passe sont protégés avec Bcrypt 6.0.0. La validation des données côté serveur est assurée par Zod 3.24.3.

Outils et services

- Nodemailer : envoi d'email (ex : confirmation, réinitialisation du mot de passe)
- Cloudinary : gestion et hébergement des images
- Vitest : tests unitaires et de composants
- ESLint : assurance qualité du code source

II - Navigateurs compatibles

Les navigateurs les plus utilisés sont Google Chrome, Safari, Microsoft Edge et Mozilla Firefox.

Mon application est compatible avec ces navigateurs.

III - Possibilité de déploiement

J'ai choisi de déployer mon application web et web mobile sur Render. Pour cela j'ai d'abord créer une base de donnée en PostgreSQL 16.

Puis j'ai crée un Web Service pour mon back-end. J'ai sélectionner mon repository sur GitHub et la branche que je voulais lier. J'y ai mis les commandes pour lancer le serveur et installer les dépendances. J'ai ensuite entrer mes variables d'environnement.

Et j'ai lancer le service.

J'ai ensuite procéder à la configuration du Static Site, qui correspond au front-end. La procédure a été la même que celle du back-end.

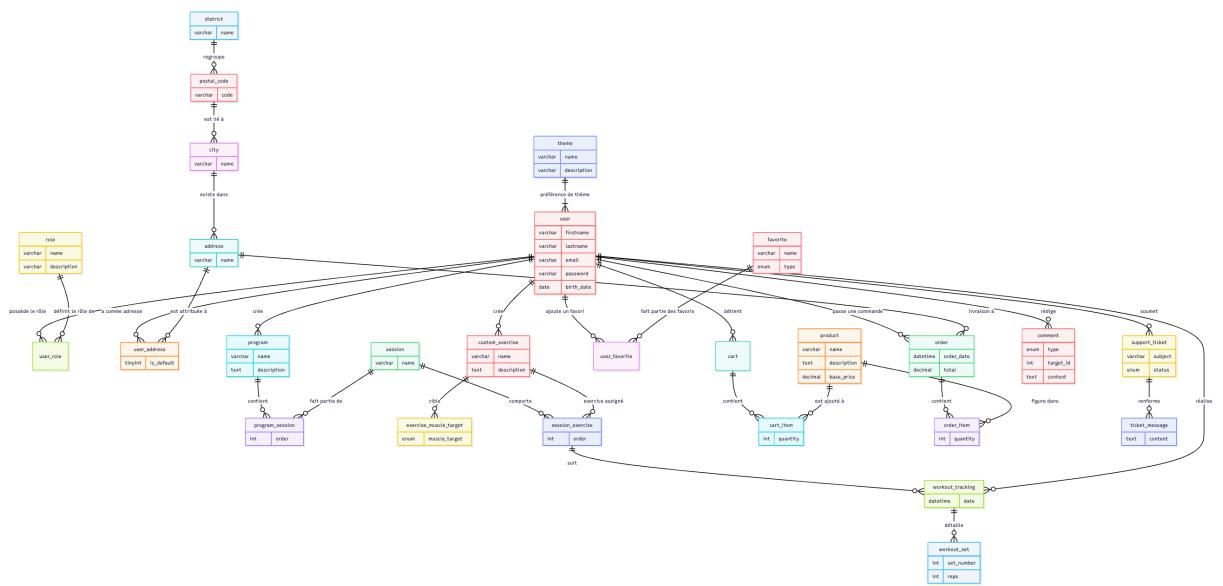
J'ai ensuite lancé mon application web.

IV - Création de la base de données

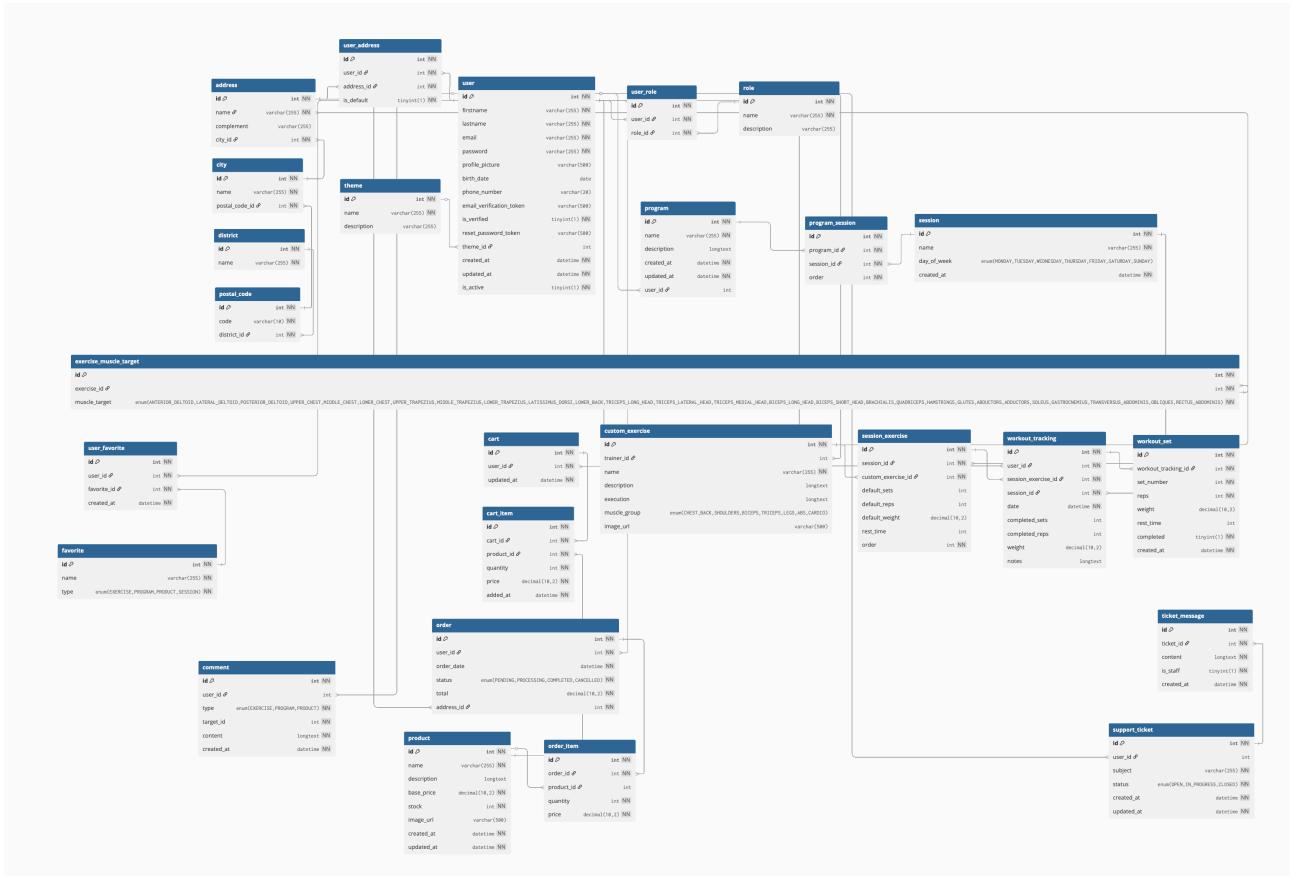
La base de données a été créée selon le processus de normalisation au format 3NF. De cette façon la base de données est bien structurée, facile à maintenir et cohérente même après des mises à jour fréquentes. Cela permet d'éviter la redondance des données, d'en assurer une meilleure intégrité et d'en optimiser les performances.

L'inconvénient est que les requêtes sont plus complexes même avec Prisma.

A. MCD



B. MLD



V - Routes front et back

A. Back-end

Auth :

```
authRoutes.post("/register", register);
authRoutes.post("/login", login);
authRoutes.post("/logout", authGuard(), logout);
authRoutes.post("/forgot-password", forgotPassword);
authRoutes.post("/reset-password", resetPassword);
authRoutes.get("/verify-email/:token", verifyEmail);
authRoutes.post("/resend-verification", resendVerificationEmail);
```

Admin :

```
// Routes protégées par authentification et rôle admin
adminRoutes.use('/admin/*', authGuard(), adminGuard);

// Routes du dashboard
adminRoutes.get("/admin/dashboard/stats", getDashboardStats);
adminRoutes.get("/admin/dashboard/muscle-groups", getMuscleGroupStats);
adminRoutes.get("/admin/dashboard/progression", getProgressionStats);
adminRoutes.get("/admin/dashboard/exercises", getExercisesForStats);

// Nouvelles routes pour les statistiques d'entraînement des utilisateurs
adminRoutes.get("/admin/dashboard/users/muscle-groups", getUsersMuscleGroupStats);
adminRoutes.get("/admin/dashboard/users/progression", getUsersProgressionStatsByMuscleGroup);

// Route pour les statistiques d'entraînement globales
adminRoutes.get("/admin/dashboard/global-training-stats", getGlobalTrainingStats);

// Routes de profil admin
adminRoutes.get("/admin/profile/:userId", getAdminProfileById);
adminRoutes.put("/admin/profile/:userId", zValidator('json', updateAdminProfileSchema), updateAdminProfile);
adminRoutes.put("/admin/profile/:userId/upload", updateAdminProfile);
adminRoutes.delete("/admin/profile/:userId/picture", removeAdminProfilePicture);

// Routes de gestion des utilisateurs
adminRoutes.get("/admin/users", getAllUsers);
adminRoutes.get("/admin/users/:userId", getUserById);
adminRoutes.put("/admin/users/:userId", zValidator('json', updateUserSchema), updateUser);
adminRoutes.delete("/admin/users/:userId", deleteUser);
adminRoutes.delete("/admin/users/:userId/picture", removeUserProfilePicture);
```

Trainer :

```
// Routes de profil utilisateur (pages personnelles)
trainerRoutes.get('/trainer/profile/:userId', authGuard(), trainerPersonalGuard, getTrainerProfileById);
trainerRoutes.put('/trainer/profile/:userId', authGuard(), trainerPersonalGuard, zValidator('json', updateUserProfileSchema), updateTrainerProfile);
trainerRoutes.put('/trainer/profile/:userId/upload', authGuard(), trainerPersonalGuard, updateTrainerProfile);
trainerRoutes.delete('/trainer/profile/:userId/picture', authGuard(), trainerPersonalGuard, removeTrainerProfilePicture);

// Routes pour la création d'exercice (fonctionnalités générales)
trainerRoutes.post('/trainer/add-exercise', authGuard(), trainerGuard, zValidator('json', createExerciseSchema), createExercise);
trainerRoutes.post('/trainer/add-exercise/upload', authGuard(), trainerGuard, createExercise);

// Route pour la suppression d'exercice (fonctionnalités générales)
trainerRoutes.delete('/trainer/exercise/:exerciseId', authGuard(), trainerGuard, deleteExercise);

// Routes pour la mise à jour d'exercice
trainerRoutes.put('/trainer/exercise/:exerciseId', authGuard(), trainerGuard, updateExercise);
trainerRoutes.put('/trainer/exercise/:exerciseId/upload', authGuard(), trainerGuard, updateExercise);

// Routes pour la gestion des programmes d'entraînement
trainerRoutes.get('/trainer/programs', authGuard(), trainerGuard, getAllPrograms);
trainerRoutes.get('/trainer/program/:programId', authGuard(), trainerGuard, getProgramById);
trainerRoutes.post('/trainer/add-program', authGuard(), trainerGuard, zValidator('json', createProgramSchema), createProgram);
trainerRoutes.put('/trainer/program/:programId', authGuard(), trainerGuard, updateProgram);
trainerRoutes.delete('/trainer/program/:programId', authGuard(), trainerGuard, deleteProgram);

// Routes pour le dashboard trainer
trainerRoutes.get('/trainer/dashboard/stats', authGuard(), trainerGuard, getTrainerDashboardStats);
trainerRoutes.get('/trainer/dashboard/muscle-groups', authGuard(), trainerGuard, getTrainerMuscleGroupStats);
trainerRoutes.get('/trainer/dashboard/progression-by-muscle-group', authGuard(), trainerGuard, getTrainerProgressionStatsByMuscleGroup);
trainerRoutes.get('/trainer/dashboard/exercises', authGuard(), trainerGuard, getTrainerExercises);
trainerRoutes.get('/trainer/dashboard/global-training-stats', authGuard(), trainerGuard, getTrainerGlobalTrainingStats);
trainerRoutes.get('/trainer/dashboard/clients', authGuard(), trainerGuard, getTrainerClients);
```

User :

```
// Routes protégées par authentification et rôle user/trainer/admin (fonctionnalités générales)
userRoutes.use('/user/exercises*', authGuard(), userGuard);

// Routes protégées par authentification et rôle user uniquement (pages personnelles)
userRoutes.use('/user/profile*', authGuard(), userPersonalGuard);
userRoutes.use('/user/dashboard*', authGuard(), userGuard);

// Routes du dashboard utilisateur - utiliser userGuard au lieu de userPersonalGuard
userRoutes.get('/user/dashboard/stats', authGuard(), userGuard, getUserDashboardStats);
userRoutes.get('/user/dashboard/muscle-groups', authGuard(), userGuard, getUserMuscleGroupStats);
userRoutes.get('/user/dashboard/progression', authGuard(), userGuard, getUserProgressionStatsByMuscleGroup);

// Routes de profil utilisateur (pages personnelles)
userRoutes.get('/user/profile/:userId', getUserProfileById);
userRoutes.put('/user/profile/:userId', zValidator('json', updateUserProfileSchema), updateUserProfile);
userRoutes.put('/user/profile/:userId/upload', updateUserProfile);
userRoutes.delete('/user/profile/:userId/picture', removeUserProfilePicture);

// Routes pour la consultation des exercices (accessibles à tous les utilisateurs connectés)
userRoutes.get('/user/exercises', getAllExercises);
userRoutes.get('/user/exercises/:exerciseId', getExerciseById);

// Routes pour la consultation des programmes (accessibles à tous les utilisateurs connectés)
userRoutes.get('/programs', authGuard(), userGuard, getAllPrograms);
userRoutes.get('/programs/:programId', authGuard(), userGuard, getProgramById);

// Routes pour la gestion des programmes personnels des utilisateurs
userRoutes.post('/personal-programs', authGuard(), userGuard, zValidator('json', programSchema), createPersonalProgram);
userRoutes.put('/personal-programs/:programId', authGuard(), userGuard, zValidator('json', programSchema), updatePersonalProgram);
userRoutes.delete('/personal-programs/:programId', authGuard(), userGuard, deletePersonalProgram);
userRoutes.get('/personal-programs', authGuard(), userGuard, getPersonalPrograms);
userRoutes.get('/personal-programs/:programId', authGuard(), userGuard, getPersonalProgramById);

// Route pour récupérer tous les programmes disponibles
userRoutes.get('/available-programs', authGuard(), userGuard, getAllAvailablePrograms);

// Routes pour le suivi d'entraînement
userRoutes.post('/workout-tracking', authGuard(), userGuard, zValidator('json', workoutTrackingSchema), createWorkoutTracking);
userRoutes.put('/workout-tracking/:trackingId', authGuard(), userGuard, zValidator('json', workoutTrackingSchema.partial()), updateWorkoutTracking);
userRoutes.get('/workout-tracking', authGuard(), userGuard, getWorkoutTrackings);
userRoutes.delete('/workout-tracking/:trackingId', authGuard(), userGuard, deleteWorkoutTracking);
```

B. Front-end

```
'/',
'/exercises-list',
'/muscles-list',
'/muscle/:id',
'/login',
'/register',
'/forgot-password',
'/reset-password',
'/verify-email/:token',
'/cookies-policy',
'/legal-mentions',
'/about',
'/privacy-policy',
'/terms-of-service',
'/unauthorized'

'/admin/profile',
'/admin/dashboard',
'/admin/users'

'/trainer/dashboard',
'/trainer/profile',
'/trainer/add-exercise',
'/trainer/add-program',
'/trainer/edit-program/:programId'

'/user/dashboard',
'/user/profile'
'/programs',
'/programs/:programId',
'/exercises',
'/exercise/:id',
'/workout-tracking',
'/my-training',
'/personal-programs',
'/add-personal-program',
'/personal-program/:id',
'/edit-personal-program/:id'

'*' // 404
```

RÉALISATION PERSONNELLES :

I - Suivi d'entraînement

A. Back-end

L'architecture de données repose sur trois entités principales interconnectées qui forment un système cohérent et évolutif :

WorkoutTracking : représente le suivi global d'un exercice lors d'une séance.

WorkoutSet : détaille chaque série individuelle réalisée pendant l'exercice avec son nombre de répétitions, les charges utilisées et les temps de repos.

SessionExercice : sert de template définissant les paramètres recommandés pour chaque exercice dans un programme.

Logique métier :

Le service de suivi d'entraînement implémente une logique métier qui automatise les calculs et garantit la cohérence des données.

Lorsqu'un utilisateur termine un exercice, le système calcule les métriques globales à partir des séries individuelles. Il détermine le nombre total de séries réalisées, additionne toutes les répétitions et calcule la modèle pondérée des poids utilisés. L'automatisation évite les erreurs et assure la cohérence.

```
const workoutTracking = await prisma.$transaction(async (prisma) => {
  const tracking = await prisma.workoutTracking.create({
    data: {
      userId: userId,
      sessionExerciseId: parseInt(trackingData.sessionExerciseId),
      sessionId: sessionExercise.sessionId,
      date: workoutDate,
      completedSets: completedSets,
      completedReps: completedReps,
      weight: weight,
      notes: trackingData.notes || null
    }
  });

  if (trackingData.sets && trackingData.sets.length > 0) {
    const setPromises = trackingData.sets.map((set, index) => {
      return prisma.workoutSet.create({
        data: {
          workoutTrackingId: tracking.id,
          setNumber: index + 1,
          reps: set.reps || 0,
          weight: set.weight ? parseFloat(set.weight) : null,
          restTime: set.restTime || null,
          completed: set.completed !== undefined ? set.completed : true
        }
      });
    });
    await Promise.all(setPromises);
  }
  return tracking;
});
```

L'enregistrement d'un suivi utilise les transactions Prisma pour garantir l'intégrité des données. Si l'enregistrement du suivi principal échoue, les séries détaillées ne sont pas créées, évitant ainsi les états incohérents.

Chaque opération vérifie rigoureusement les permissions. Un utilisateur ne peut créer modifier ou consulter que ses propres suivis. Le système empêche l'accès aux données d'autres utilisateurs, même en cas de manipulation malveillante des requêtes.

Chaque endpoint a une responsabilité claire. La création, modification et suppression des suivis sont séparées avec des permissions appropriées. Les paramètres de filtrage (par session ou date) permettent des requêtes ciblées.

La validation s'effectue sur trois niveaux; côté client pour l'expérience utilisateur, au niveau API avec Zod pour la sécurité, et en base de données avec les contraintes Prisma.

```
const workoutTrackingSchema = z.object({
    sessionExerciseId: z.number().int().positive("L'ID de l'exercice de séance est requis"),
    date: z.string().optional().nullable(),
    completedSets: z.number().int().min(0, "Le nombre de séries ne peut pas être négatif").optional(),
    completedReps: z.number().int().min(0, "Le nombre de répétitions ne peut pas être négatif").optional(),
    weight: z.number().min(0, "Le poids ne peut pas être négatif").optional().nullable(),
    notes: z.string().optional().nullable()
});
```

Le système distingue les erreurs utilisateurs (données invalides ou permissions insuffisantes) des erreurs système (base de données indisponible, problème réseau) pour fournir des messages d'erreur appropriés.

B. Front-end

Le suivi d'entraînement priviliege la simplicité d'utilisation en contexte sportif. L'utilisateur peut naviguer entièrement à une main sur mobile.

L'interface guide naturellement l'utilisateur à travers les étapes, sélection du programme, choix de la séance, puis progression exercice par exercice. Cette approche linéaire évite la confusion et maintient la concentration.

Chaque action produit un résultat immédiat. Les exercices terminés sont marqués en verts, l'exercice en cours en bleu, les exercices en attente restent neutres.

Pendant l'entraînement, seuls les informations essentielles sont affichées. Les détails techniques apparaissent uniquement quand c'est nécessaire.

Après chaque série, un minuteur se déclenche automatiquement selon les paramètres saisies par l'utilisateur sur la série accomplie. L'utilisateur peut passer le temps de repos s'il le souhaite.

```
useEffect(() => {
  let interval = null;
  if (isResting && restTimer > 0) {
    interval = setInterval(() => {
      setRestTimer(timer => {
        if (timer <= 1) {
          setIsResting(false);
          return 0;
        }
        return timer - 1;
      });
    }, 1000);
  } else if (!isResting) {
    clearInterval(interval);
  }
  return () => clearInterval(interval);
}, [isResting, restTimer]);
```

Si l'utilisateur quitte momentanément l'application, l'état de l'entraînement est préservé et il peut reprendre où il s'était arrêté.

```
useEffect(() => {
  const interval = setInterval(() => {
    if (!isAuthenticated || !user) {
      console.log('Dashboard User - Utilisateur non authentifié, arrêt du rechargeement automatique');
      clearInterval(interval);
      return;
    }

    get_user_dashboard_stats()
      .then(response => {
        if (response && response.success) {
          setStats(response.data);
          setLastUpdated(new Date());
        }
      })
      .catch(error => {
        if (error.response?.status === 401) {
          clearInterval(interval);
        }
      });
  }, 30000);

  return () => clearInterval(interval);
}, [user, isAuthenticated, navigate]);
```

II - Crédit du programme d'entraînement

A. Back

La fonctionnalité de création de programme public permet aux entraîneurs de créer des programmes d'entraînement structurés que les utilisateurs peuvent découvrir et adopter. Cette fonctionnalité constitue la bibliothèque de programme partagés de l'application R-Training.

L'architecture s'appuie sur un modèle relationnel sophistiqué permettant la création de programme hiérarchiques :

Program : entité principale contenant les métadonnées (nom, description, créateur)

ProgramSession : table de liaison ordonnée entre programmes et sessions

Session : séances d'entraînement avec jour de la semaine et liste d'exercices

SessionExercice : exercices configuré avec les paramètre par défaut (séries...)

Le système vérifie l'unicité du nom de programme par entraîneur, l'existence de tous les exercices référencés et la cohérence des données avant création.

```
const exerciseIds = programData.sessions.flatMap(session =>
  session.exercises.map(ex => ex.customExerciseId)
).filter(id => id !== null && id !== undefined);

if (exerciseIds.length === 0) {
  throw new Error('Aucun exercice valide fourni');
}

const exercises = await prisma.customExercise.findMany({
  where: {
    id: {
      in: exerciseIds
    }
  }
});

if (exercises.length !== exerciseIds.length) {
  throw new Error('Un ou plusieurs exercices n\'existent pas');
}
```

L'utilisation de Prisma avec relation imbriquées garantit l'atomicité. Si une partie de la création échoue, l'ensemble est annulé.

Seuls les entraîneurs et les admins peuvent créer des programmes publiques. Les admins peuvent modifier tous les programmes, les entraîneurs seulement les leurs.

B. Front

L'interface guide l'entraîneur à travers un processus de création étape par étape, en évitant d'être trop complexe.

Il peut ajouter des sessions une par une, chacune ayant son propre nom, jour de la semaine et sa liste d'exercices. L'écran s'adapte automatiquement au contenu, avec défilement intelligent vers les nouvelles sections créées.

```
const addSession = () => {
  const newSessionIndex = program.sessions.length;
  setProgram(prev => ({
    ...prev,
    sessions: [...prev.sessions, {
      name: '',
      day_of_week: 'MONDAY',
      exercises: []
    }]
  }));
  setTimeout(() => {
    if (sessionsRef.current[newSessionIndex]) {
      sessionsRef.current[newSessionIndex].scrollIntoView({
        behavior: 'smooth',
        block: 'center'
      });
    }
  }, 100);
};
```

La plateforme intègre un système avancé de sélection d'exercices reposant sur un modal dédié permettant de parcourir efficacement les mouvements disponibles, en les filtrant en temps réel par nom ou groupe musculaire. Lors de l'ajout d'un exercice, l'utilisateur peut configurer immédiatement les paramètres essentiels (séries, répétitions...) ce qui rend l'expérience fluide et sans interruption.

L'interface adopte une approche mobile-first pensée pour les smartphone et les tablettes. Côté performance, le système repose sur des mécanisme de chargement asynchrone, évitant de bloquer l'interface lors de la navigation.

La création de programme se fait de manière intuitive et simple. Elle ne nécessite pas ou peu de temps d'adaptation.

TESTS UNITAIRES :

Afin d'assurer la fiabilité et la sécurité du système d'authentification, des tests unitaires ont été mis en place à l'aide de Vitest, un framework moderne adapté à un environnement JavaScript / Node.js.

Les tests que je vais vous présenté portent sur le contrôleur d'authentification, en simulant les appels et en vérifiant les retours attendus dans différents cas d'usage.

Ces test ont pour but de valider la connexion réussie avec token JWT et retour utilisateur, gérer les erreurs courantes (compte non vérifié, identifiants invalides), et vérifier que la déconnexion réinitialise bien la session utilisateur.

Les fonctions login et logout sont testées dans différents scénarios à l'aide de mocks du service d'authentification.

```
describe('Auth Controller Tests', () => {
  beforeEach(() => {
    vi.clearAllMocks();
    mockContext = { req: { valid: vi.fn() }, ... };
  });

  describe('login', () => {
    it('should successfully login and set cookies', async () => { ... });
    it('should handle unverified email with 400 status', async () => { ... });
    it('should handle invalid credentials', async () => { ... });
  });

  describe('logout', () => {
    it('should clear cookies and logout successfully', async () => { ... });
  });
});
```

Exemple connexion réussie :

```
expect(mockContext.header).toHaveBeenCalledWith(
  'Set-Cookie',
  expect.stringContaining('accessToken=mock-jwt-token')
);
expect(mockContext.json).toHaveBeenCalledWith({
  message: 'Connexion réussie',
  user: mockResult.user
});
```

Ce test garantit que le cookieJWT est bien défini et que l'utilisateur reçoit une réponse structurée.

Exemple email non vérifié :

L'authentification retourne une erreur 400 avec un message clair.

```
expect(mockContext.json).toHaveBeenCalledWith(mockResult, 400);
```

Exemple identifiants invalides :

```
expect(mockContext.json).toHaveBeenCalledWith({
  error: 'Email ou mot de passe incorrect'
}, 400);
```

Exemple déconnexion :

```
expect(mockContext.header).toHaveBeenCalledWith(
  'Set-Cookie',
  expect.stringContaining('Max-Age=0')
);
expect(mockContext.json).toHaveBeenCalledWith(
  message: 'Déconnexion réussie'
);
```

Réponse dans le terminal de test positifs :

```
Mac:back ryan$ npm test auth.test.js
> back@1.0.0 test
> vitest auth.test.js

[DEV] v3.1.2 /Applications/MAMP/htdocs/R-Training/back

✓ src/test/auth.test.js (4 tests) 3ms
  ✓ Auth Controller Tests > login > should successfully login and set cookies 2ms
  ✓ Auth Controller Tests > login > should handle unverified email with 400 status 0ms
  ✓ Auth Controller Tests > login > should handle invalid credentials 0ms
  ✓ Auth Controller Tests > logout > should clear cookies and logout successfully 0ms

Test Files 1 passed (1)
Tests 4 passed (4)
Start at 21:55:30
Duration 1.27s (transform 48ms, setup 7ms, collect 124ms, tests 3ms, environment 0ms, prepare 36ms)
```

Ces tests garantissent un comportementalisme fiable de l'authentification utilisateur, une gestion d'erreurs les plus courantes, une expérience fluide sécurisée, essentielle pour toute application web.

VULNÉRABILITÉ DE SÉCURITÉ ET VEILLE :

I - Veille technologique

A. Prisma

J'ai choisi PostgreSQL comme système de gestion de base de données (SGBD) avec Prisma. L'avantage de PostgreSQL repose sur sa puissance, sa conformité aux standards SQL et ses fonctionnalités avancées:

Le support des types complexes, des hautes performances (gestion des index, transactions et jointures complexes), fiabilité et sécurité (transition ACID, gestion fine des rôles et permissions), large compatibilité notamment avec Prisma mais aussi avec les outils SQL. PostgreSQL convient autant aux petits projets qu'aux projets complexes.

Prisma représente un choix technologique en phase avec les standards actuels du développement back-end JavaScript / TypeScript. Prisma permet de générer un schéma clair et version de la base de données. Il permet d'interroger la base avec une syntaxe claire et plus maintenable que des requêtes SQL. L'intégration avec d'autres outils moderne comme Hono et Zod se fait facilement.

B. Cloudinary

Cloudinary est un service cloud qui permet de stocker, transformer et diffuser des images de manière performante. Il n'est plus besoin de stocker les fichiers localement. Cela améliore les performances de l'application tout en simplifiant la gestion des médias sans surcharger le back-end.

```
import fs from 'fs/promises';
import path from 'path';
import { v2 as cloudinary } from 'cloudinary';

// Configuration du stockage
const storageConfig = {
  type: 'cloudinary', // 'local', 'cloudinary' ou 'aws'
  localPath: '../front/public/images/profiles',
  allowedTypes: [
    'image/jpeg',
    'image/jpg',
    'image/png',
    'image/gif',
    'image/webp',
    'image/avif'
  ],
  maxSize: 5 * 1024 * 1024 // 5MB
};

class StorageService {
  constructor(config = storageConfig) {
    this.config = config;
  }

  async saveProfilePicture(file, userId) {
    switch (this.config.type) {
      case 'local':
        return this.saveLocal(file, userId);
      case 'cloudinary':
        return this.saveCloudinary(file, userId);
      case 'aws':
        return this.saveAWS(file, userId);
      default:
        throw new Error('Type de stockage non supporté');
    }
  }

  async saveLocal(file, userId) {
    try {
      console.log('Début du traitement local pour userId:', userId);
      console.log('Fichier reçu:', {
        type: file.type,
        size: file.size,
        name: file.name,
        buffer: file.buffer ? 'Buffer présent' : 'Buffer manquant'
      });
    }
  }
}
```

C. Nodemailer et SMTP

Nodemailer est une bibliothèque Node.js permettant d'envoyer des emails via le protocole SMTP.

Les fonctionnalités clés sont l'envoi d'email de confirmation, la récupération de mot de passe, notifications...

La compatibilité SMTP se fait avec tous les fournisseurs (Gmail, Mailjet, OVH...). La configuration est simple. Il faut une adresse email avec un mot de passe, un port et un serveur suffisent.

Tout cela permet à l'application de communiquer avec les utilisateurs de façon sécurisée et automatisée.

II - Veille de sécurité

A. JWT et cryptage du mot de passe

JsonWebToken est un format de jeton sécurisé utiliser pour authentifier les utilisateurs via un token signé, stocké dans un cookie HTTPOnly pour éviter les attaques XSS.

Bcrypt est une fonction permettant de hacher les mots de passe avant de les stocker en base de données pour éviter toute récupération en clair, même en cas de faille.

B. Sécurisation des requêtes

Les requêtes ont été sécurisées de plusieurs façons. Tout d'abord avec un middleware AuthGuard qui vérifie la présence et la validité des tokens JWT ainsi que l'identité des utilisateurs. De plus, un RoleMiddleware contrôle l'accès en se basant sur les rôles. Enfin j'ai mis en place une protection CORS a été configurée pour empêcher les requêtes cross-origin, venant de domaines externes.

C. Validation des entrées

La validation des entrées se fait avec Zod. Il s'intègre parfaitement avec le framework Hono.

```
z.object({
    email: z.string().email("Adresse e-mail invalide"),
    password: z.string()
        .min(10, "Le mot de passe doit contenir au moins 10 caractères")
        .regex(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$/, "Le mot de passe doit contenir une majuscule, une minuscule et un chiffre."),
    phoneNumber: z.string()
        .regex(/^(\d{2} [0-9]{2}){2} (\d{2}){4}/, "Format de numéro invalide")
        .optional()
})
```

D. Routes protégées

Les routes sont protégées avec une architecture à trois niveaux.

AuthGuard : les routes avec une authentification obligatoire.

RoleGuard : vérifie les rôles permettant d'accéder aux différentes routes.

PersonnalGuard : Accès aux données personnelles uniquement (profil, dashboard)

Modalités d'accès :

Les admins ont accès à l'ensemble des pages.

Les entraîneurs ont accès à toutes les pages sauf celles réservées aux admin, comme la gestion des utilisateurs.

Les utilisateurs ont accès au reste des pages sauf celles des admins et des entraîneurs comme la création d'exercices et des programmes publics.

III - Processus de recherche et traduction

Pour garantir la fiabilité des données et la qualité de l'application, un travail de recherche technique a été mené à chaque étape du développement.

Cela a inclus la consultation de la documentation officielle (Prisma, Hono, JWT...), la lecture d'articles ainsi que la consultations de forum et de tutoriels (LinkedIn, GitHub, Youtube, Stack Overflow).

Dde plus certains contenus et erreurs étant en anglais, un travail de traduction et de compréhension technique a été nécessaire, notamment pour les messages d'erreur ou les concepts avancés.

Cette étape a permis d'adapter des solutions à mon propre projet, tout en approfondissant ma compréhension des technologies utilisées.

CONCLUSION :

En somme, ce projet m'a permis de relever plusieurs défis techniques tout en consolidant mes compétences en développement web et web mobile.

L'exploration et la découverte de nouvelles technologies comme PostgreSQL, Prisma ou encore Hono m'a offert une expérience enrichissante et concrète, bien au-delà de la théorie.

Grâce à une préparation rigoureuse en amont et à une organisation progressive, adapté à un travail seul et en équipe de façon professionnelle et cohérente, j'ai pu apprendre rapidement et apporter des solutions fonctionnelles aux différentes problématiques rencontrées.

Ce projet constitue ainsi une étape clé dans mon parcours, et une base solide pour de futures évolutions ou projets plus complexes.