Dossier de Conception

1. Introduction

« Le bien-être n'est pas une 'solution médicale' mais un mode de vie… une approche de la vie que nous concevons chacun pour atteindre notre plus haut potentiel de bien-être maintenant et pour toujours. » – Greg Anderson

L'application **Santé & Fitness** s'inscrit dans cette vision. Son objectif est de proposer un outil simple et accessible qui accompagne l'utilisateur dans la gestion de ses activités physiques et de son alimentation, tout en lui offrant une visualisation claire de ses progrès.

Ce document présente le plan d'action technique, traduisant les exigences du DSF en choix concrets pour le développement. Le Front-end sera réalisé avec React pour une interface réactive et ergonomique, tandis que le Back-end utilisera Laravel pour gérer l'API, la logique métier et la sécurité des données.

2. Architecture Générale

L'application de suivi santé et fitness sera conçue selon une architecture Client-Serveur avec une Single Page Application (SPA) côté front-end et une API RESTful côté back-end. Cette approche garantit modularité, maintenabilité et évolutivité.

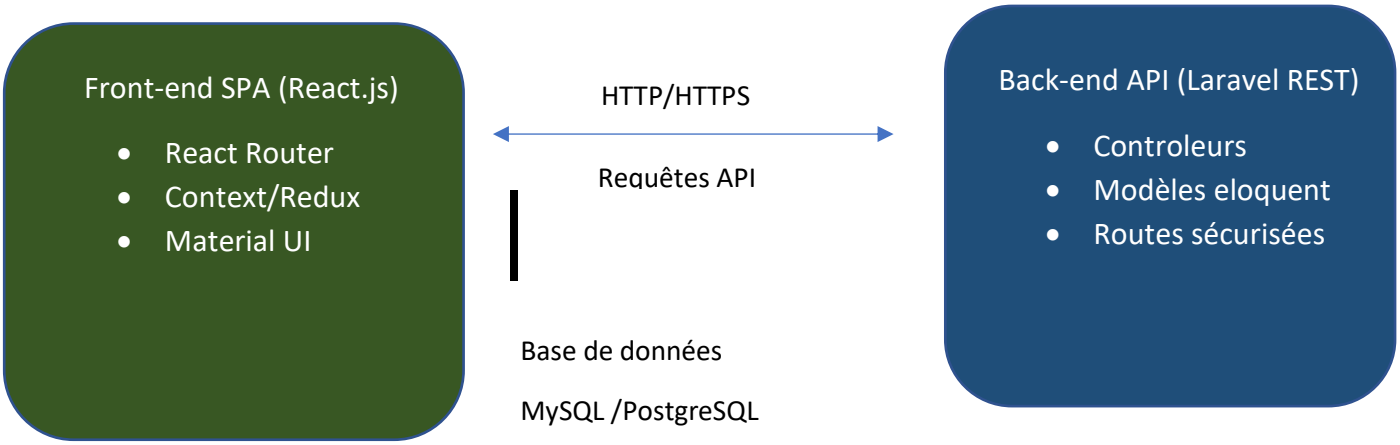
Le front-end, développé avec React.js, fournira une interface utilisateur dynamique et réactive. Il permettra à l'utilisateur de naviguer entre les différentes vues (tableau de bord, activités, repas, profil, notifications) sans recharger la page grâce à React Router. La gestion de l'état global (données utilisateur, activités, repas) sera assurée par Context API ou Redux, et les composants UI réutilisables seront conçus avec une UI Library telle que Material UI pour garantir cohérence et ergonomie.

Le back-end, développé avec Laravel, fournira une API REST organisée autour des ressources principales : /users, /activities, /meals et /notifications. Chaque ressource proposera les opérations CRUD correspondantes, sécurisées par authentification et gestion des rôles. Les données seront stockées dans une base de données MySQL (ou PostgreSQL), et toutes les communications front-end/back-end se feront via HTTP/HTTPS, avec des échanges au format JSON.

Un exemple de flux typique : un utilisateur saisit une activité dans React, Axios envoie une requête POST vers /api/activities, Laravel traite la requête, enregistre l'activité en base de données, renvoie un JSON confirmant la création, et React met automatiquement à jour le tableau de bord.

L’environnement de développement comprendra Node.js avec Create React App ou Vite pour le front-end, un serveur Laravel local via Artisan ou Docker pour le back-end, et une base de données locale via XAMPP, MAMP ou Docker. Les outils complémentaires incluent VS Code comme IDE, Postman pour tester l’API, et Git/GitHub pour la gestion de version.

Schéma simplifié :



Cette architecture permet une séparation claire des responsabilités : le front-end gère l’interface et l’expérience utilisateur, tandis que le back-end gère la logique métier, les données et la sécurité. Elle facilite également l’évolution future de l’application, l’ajout de nouvelles fonctionnalités ou l’intégration avec d’autres services.

3. Modèle de Données (Structure de la Base de Données)

Table : users

- id (INT, PK)
- name (VARCHAR)
- email (VARCHAR, UNIQUE)
- password (VARCHAR)
- age (INT)
- poids (FLOAT)
- taille (FLOAT)
- objectif_poids (FLOAT, nullable)
- role (VARCHAR, ex: "utilisateur" ou "administrateur")

- created_at (DATETIME)
- updated_at (DATETIME)

Table : activities

- id (INT, PK)
- user_id (INT, FK vers users)
- type (VARCHAR, ex: marche, course, musculation)
- duree (INT, minutes)
- calories_brulees (FLOAT)
- date_activite (DATE)
- created_at (DATETIME)
- updated_at (DATETIME)

Table : meals

- id (INT, PK)
- user_id (INT, FK vers users)
- description (TEXT)
- calories (FLOAT)
- date_repas (DATE)
- created_at (DATETIME)
- updated_at (DATETIME)

Table : notifications (optionnelle pour cette version)

- id (INT, PK)
- user_id (INT, FK vers users)
- type (VARCHAR, ex: rappel_activite, rappel_repas)
- message (TEXT)
- etat (BOOLEAN, lu/non lu)
- date_envoi (DATETIME)

Relations entre les tables

- **User hasMany Activities**
- **Activity belongsTo User**

- **User hasMany Meals**
- **Meal belongsTo User**
- **User hasMany Notifications**
- **Notification belongsTo User**

Cette structure permet de gérer efficacement les données utilisateurs, les activités et repas, tout en facilitant la création des migrations Laravel pour l'implémentation de la base de données.

4. Conception de l'API Back-end (Laravel)

Authentification et gestion des utilisateurs

- **POST /api/register** : Crée un nouvel utilisateur. (Authentification non requise)
- **POST /api/login** : Authentifie un utilisateur et retourne un token/session. (Authentification non requise)
- **POST /api/logout** : Déconnecte l'utilisateur. (Authentification requise)
- **GET /api/user** : Récupère les informations du profil connecté. (Authentification requise)
- **PUT /api/user** : Met à jour les informations du profil connecté. (Authentification requise)

Gestion des activités physiques

- **GET /api/activities** : Récupère la liste des activités de l'utilisateur connecté. (Authentification requise)
- **GET /api/activities/{id}** : Récupère les détails d'une activité spécifique. (Authentification requise)
- **POST /api/activities** : Crée une nouvelle activité. (Authentification requise)
- **PUT /api/activities/{id}** : Met à jour une activité existante. (Authentification requise, propriétaire ou admin)
- **DELETE /api/activities/{id}** : Supprime une activité. (Authentification requise, propriétaire ou admin)

Gestion des repas

- **GET /api/meals** : Récupère la liste des repas de l'utilisateur connecté. (Authentification requise)

- **GET /api/meals/{id}** : Récupère les détails d'un repas spécifique. (Authentification requise)
- **POST /api/meals** : Ajoute un nouveau repas. (Authentification requise)
- **PUT /api/meals/{id}** : Met à jour un repas existant. (Authentification requise, propriétaire ou admin)
- **DELETE /api/meals/{id}** : Supprime un repas. (Authentification requise, propriétaire ou admin)

Gestion des notifications (optionnel)

- **GET /api/notifications** : Récupère toutes les notifications de l'utilisateur. (Authentification requise)
- **POST /api/notifications** : Crée une nouvelle notification. (Authentification requise, généralement admin)
- **PUT /api/notifications/{id}** : Marque une notification comme lue ou modifie son état. (Authentification requise)
- **DELETE /api/notifications/{id}** : Supprime une notification. (Authentification requise, admin ou propriétaire)

5. Structure du Front-end (React)

L'application front-end sera organisée en **composants modulaires et réutilisables**, facilitant la maintenance et l'évolution du projet. La structure principale comprendra des **pages** correspondant aux différentes routes de l'application et des **composants UI réutilisables** pour standardiser l'affichage des informations et les interactions utilisateur.

Les **pages principales** incluent :

- Page d'accueil / tableau de bord avec résumé des activités et repas récents, graphiques de progression et accès rapide aux actions principales.
- login : Page de connexion pour les utilisateurs existants.
- register : Page d'inscription pour les nouveaux utilisateurs.
- profile : Page permettant de consulter et modifier les informations personnelles et objectifs de santé.
- activities : Page pour ajouter, modifier ou consulter l'historique des activités physiques.
- meals : Page pour ajouter, modifier ou consulter l'historique des repas.
- notifications : Page listant les notifications et rappels de l'utilisateur.

Les **composants réutilisables** incluront :

- Header et Footer pour l'interface globale.
- ActivityCard et MealCard pour l'affichage des activités et repas.
- ActivityForm et MealForm pour la saisie ou modification des données.
- DashboardStats pour les graphiques et statistiques de progression.
- NotificationAlert pour les rappels et messages.
- Button, Input et autres composants UI standardisés pour garantir cohérence et ergonomie.

La **gestion de l'état global** sera assurée par **Context API** ou éventuellement **Redux** si le projet nécessite une meilleure organisation des données utilisateur, activités et repas. Les états locaux seront gérés via useState pour les composants spécifiques.

La **navigation** sera gérée avec **react-router-dom**, permettant de passer d'une page à l'autre sans recharger la SPA et de protéger certaines routes nécessitant une authentification. Cette organisation assure une interface réactive, cohérente et facile à maintenir.

6. Réflexions sur les Exigences Non Fonctionnelles Clés

Pour l'application Suivi Santé et Fitness, les exigences non fonctionnelles seront intégrées dès le développement et vérifiées tout au long des tests afin d'assurer performance, sécurité et responsabilité environnementale.

Sécurité : L'authentification sera gérée via Laravel avec des sessions sécurisées, éventuellement extensibles vers JWT si nécessaire pour une API REST externe. L'autorisation sera contrôlée grâce aux rôles définis (Utilisateur, Administrateur), limitant l'accès aux actions sensibles. Les données reçues par l'API seront systématiquement validées côté back-end à l'aide des fonctionnalités de validation de Laravel, tandis que le front-end effectuera un sanitizing et une vérification des inputs pour prévenir les failles XSS. L'ORM Eloquent et les requêtes préparées assureront une protection contre les injections SQL, et les protections CSRF seront activées pour tous les formulaires.

Performance : Pour éviter des temps de chargement excessifs, l'application optimisera les requêtes vers la base de données avec des index sur les colonnes fréquemment recherchées et l'Eager Loading pour limiter le nombre de requêtes. Les données volumineuses seront paginées et les calculs de statistiques seront optimisés pour réduire la charge serveur. La réactivité de l'interface React sera testée régulièrement, en veillant à ce que les composants se mettent à jour efficacement sans provoquer de rendus inutiles.

Éco-responsabilité : L'application adoptera des pratiques visant à réduire l'impact environnemental et la consommation de ressources. Les images et assets seront compressés et optimisés pour limiter la taille des fichiers. Les appels API seront minimisés et mis en cache

lorsque possible, et le chargement paresseux (lazy loading) sera utilisé pour les composants non essentiels afin de réduire la consommation de bande passante et la charge du serveur.

Cette approche intégrée garantit que l'application reste sécurisée, performante et responsable, tout en offrant une expérience utilisateur fluide et fiable.