Maxime Grillet

O'clock - Promo ELFE
2023 - 2025

Dossier de projet

IRON PAL

OCLOCK
Concepteur Développeur d'Applications

02/2025

SOMMAIRE

1. Remerciements

Je tiens tout d'abord à exprimer ma gratitude envers la promotion ELFE et mes camarades de classe. Ce fut un véritable plaisir de collaborer avec eux, d'évoluer ensemble et d'apprendre les uns des autres. Nos projets de groupe ont vu le succès grâce à une excellente cohésion d'équipe.

Je remercie également les intervenants pour leur capacité à transmettre leurs compétences avec efficacité et bienveillance dans un délai parfois très court.

Un immense merci à nos tuteurs pédagogiques, Gauthier, puis Chris, qui nous ont accompagnés avec rigueur et attention tout au long de cette année, rendant cette aventure encore plus enrichissante.

Je souhaite également remercier l'école O'clock pour la qualité de leur formation, qui m'a permis de développer des compétences solides et variées.

Enfin, je tiens à exprimer toute ma reconnaissance à l'agence Elixir, mon entreprise d'alternance, pour la confiance qu'ils m'ont accordée, l'autonomie qu'ils m'ont laissée, et les nombreux projets professionnels qui ont contribué à mon développement personnel et professionnel.

Un merci tout particulier à Valérian, Soukina et Thibaut, membres de l'équipe Ironpal, qui ont permis la concrétisation de ce projet ainsi que de ce dossier de présentation.

2. Introduction

a. Abstract

Ironpal addresses a common challenge faced by athletes and coaches: the difficulty and inconvenience of effectively planning and monitoring weightlifting sessions. While many currently use tools like Google Sheets to organize their training, these solutions often lack the clarity and usability required, especially on mobile devices.

To overcome these limitations, Ironpal was developed as a user-friendly, mobile-optimized application designed to simplify the planning and tracking of strength training. With an intuitive interface and a redefined user experience, Ironpal offers a modern and efficient alternative to traditional tools, making it easier to manage workouts and stay on track.

Ironpal is a web application tailored for weightlifting enthusiasts, enabling users to create, track, and customize their training programs. It features a comprehensive exercise library, allowing users to organize exercises into sessions and schedule them throughout the week. The platform's primary aim is to provide a straightforward and practical solution to optimize training routines while offering tools to monitor progress and adapt plans to evolving needs.

Ironpal's Objectives :

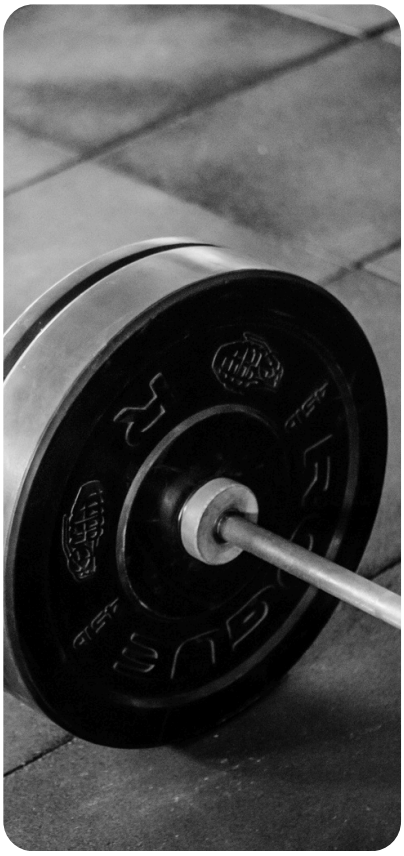
- Allow users to design and personalize training sessions that align with their fitness goals.
- Provide a versatile platform offering a diverse selection of exercises for various muscle groups.
- Facilitate the creation of custom sessions and their scheduling to suit individual preferences.
- Ensure a smooth and accessible experience across all devices, including desktop and mobile.

2. Introduction

b. Contexte et origine du projet

Ironpal a vu le jour en réponse à une problématique rencontrée par de nombreux sportifs et coachs : la difficulté de planifier et de suivre efficacement les séances de musculation. Aujourd'hui, beaucoup se tournent vers des outils comme Google Sheets pour structurer leurs entraînements. Bien que pratiques, ces tableurs, initialement conçus pour d'autres usages, se révèlent peu adaptés aux besoins spécifiques de la musculation. Leur manque d'ergonomie, surtout sur mobile, et leur complexité d'utilisation en font des outils limités pour gérer des programmes sportifs.

Dans un contexte où la mobilité et l'efficacité sont devenues essentielles, il était évident qu'une solution moderne et dédiée était nécessaire. C'est ainsi qu'est née l'idée d'Ironpal : une application intuitive et accessible, spécifiquement conçue pour répondre aux exigences des sportifs et des coachs d’aujourd’hui.



3. Compétences du référentiel couvertes par le projet

a. Les principales compétences

- **AT1** : Développer une application sécurisée
  - **CP1** : Installer et configurer son environnement de travail en fonction du projet
  - **CP2** : Développer une interface utilisateur
  - **CP3** : Développer des composants métier
  - **CP4** : Contribuer à la gestion d'un projet informatique

- **AT2** : Concevoir et développer une application sécurisée organisée en couches
  - **CP5** : Analyser les besoins et maquetter une application
  - **CP6** : Définir l'architecture logicielle d'une application
  - **CP7** : Concevoir et mettre en place une base de données relationnelle
  - **CP8** : Développer des composants d'accès aux données SQL et NoSQL

- **AT3** : Préparer le déploiement d'une application sécurisée
  - **CP9** : Préparer et exécuter les plans de tests d'une application

b. Les compétences évoquées

- **AT3** : Préparer le déploiement d'une application sécurisée
  - **CP10** : Préparer et documenter le déploiement d'une application
  - **CP11** : Contribuer à la mise en production dans une démarche DevOps

4. Sprint 0 - Conception

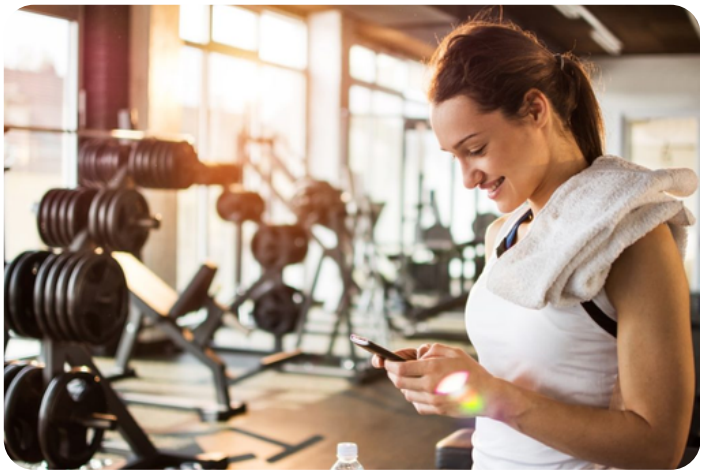
a. Cahier des charges

i. Présentation du projet

Ironpal est une application web innovante dédiée à la musculation. Elle permet aux utilisateurs de planifier, suivre et personnaliser leurs entraînements de manière simple et efficace. Avec une interface optimisée pour les formats mobiles et une expérience utilisateur fluide, Ironpal se positionne comme une véritable alternative aux outils traditionnels.

L'application propose une vaste bibliothèque d'exercices adaptés à différents groupes musculaires, que les utilisateurs peuvent organiser en séances personnalisées et programmer sur une semaine. Elle centralise toutes les informations nécessaires, facilitant ainsi la gestion et le suivi des progrès sportifs.

L'objectif principal d'Ironpal est d'offrir une solution intuitive et moderne qui aide les utilisateurs à optimiser leurs entraînements tout en leur permettant d'ajuster leurs routines en fonction de leurs besoins et objectifs. Avec Ironpal, la gestion des entraînements devient plus simple, engageante et adaptée aux exigences des sportifs modernes.



ii. Définition des besoins

- **Planification personnalisée des séances :** Permettre aux utilisateurs de planifier, suivre et personnaliser leurs séances d'entraînement en fonction de leurs objectifs, niveaux de compétence, et contraintes personnelles.

- **Plateforme en ligne ergonomique :** Offrir une plateforme intuitive et accessible qui simplifie l'organisation des entraînements grâce à une bibliothèque d'exercices variés, adaptés à différents groupes musculaires et types de séances (force, hypertrophie, endurance, etc.).

- **Regroupement et planification des exercices :** Donner aux utilisateurs la possibilité de regrouper les exercices en séances structurées, de les programmer facilement sur une semaine ou plus, et de les réajuster à tout moment.

- **Expérience utilisateur optimale :** Proposer une interface moderne avec une expérience utilisateur (UX/UI) optimisée, garantissant une navigation fluide sur tout type d'appareil (responsive : desktop, tablette, mobile).

- **Personnalisation avancée :** Permettre aux utilisateurs d'ajouter leurs propres exercices ou de personnaliser les paramètres (temps de repos, poids, répétitions) en fonction de leurs besoins spécifiques.

- **Sécurité et confidentialité :** Garantir la sécurité des données des utilisateurs (Authentification, RGPD, Sessions...) grâce à une infrastructure robuste et conforme aux réglementations.

- **En constante évolution :** Permettre l'évolutivité de l'application grâce a un choix de technos modernes modulables et évoluable.

iii. Fonctionnalités du projet

CP5

1. MVP

- **Inscription, connexion et profil utilisateur :**
  - Fonctionnalités de base : Inscription et connexion via email/mot de passe (authentification simple avec JWT token).
  - Profil utilisateur : Permettre aux utilisateurs de saisir des informations de base (nom, mail, âge...).

- **Bibliothèque d’exercices :**
  - Catalogue d’exercices : Intégrer un seeding avec une cinquantaine d'exercices couvrant les principaux groupes musculaires (bras, jambes, dos, etc.).
  - Fiches d'exercices : Chaque exercice devrait inclure une description basique, des instructions sur l'exécution et le groupe musculaire ciblé.

- **Création et planification de séances d’entraînement :**
  - Création de séances personnalisées : Les utilisateurs peuvent sélectionner des exercices et les regrouper en séances (avec nombre de séries, répétitions et poids, temps de repos).
  - Planification mensuel : Un simple calendrier où l’utilisateur peut créer ses séances pour les planifier.

- **Interface utilisateur simple et responsive :**
  - Design : Interface propre et facile à utiliser sur desktop et mobile, avec une navigation simple pour accéder aux exercices, aux séances.

- **Sécurité des données :**
  - Gestion de la confidentialité : Protection des informations utilisateurs avec un chiffrement des mots de passe et des mesures de protection des routes api pour sécuriser les données.

- **Suivi des performances :**
  - Enregistrement des performances : Après une séance, les utilisateurs peuvent enregistrer les poids soulevés, la difficulté, le nombre de répétitions et de séries réalisées.

iii. Fonctionnalités du projet

CP5

2. Ouverture et évolutions

Les opportunités de développement pour Ironpal sont nombreuses. Nous souhaitons faire évoluer la plateforme en fonction des retours et des besoins de notre communauté d'utilisateurs. Nous nous appuierons sur l'engagement et les suggestions de nos utilisateurs les plus actifs pour guider notre roadmap future. Voici quelques axes de développement envisagés :

- **Intégration de fonctionnalités sociales :** Permettre aux utilisateurs de suivre leurs amis ou d'autres athlètes, de partager leurs séances d’entraînement, de publier leurs progrès ou des photos/vidéos, et de commenter les publications des autres. Un système de likes et de commentaires pourrait encourager l'interaction au sein de la communauté.

- **Suivi de performance enrichi :** Développer des outils plus avancés pour suivre la progression des utilisateurs, y compris des analyses détaillées sur les séries, les répétitions, et les poids soulevés. Des graphiques comparatifs et des tableaux de bord de performances pourraient permettre aux utilisateurs de mieux visualiser leurs progrès.

- **Programmes d'entraînement personnalisés :** Intégrer des algorithmes qui génèrent des programmes d'entraînement personnalisés basés sur les objectifs, le niveau et les préférences des utilisateurs. Des ajustements automatiques pourraient être proposés en fonction des progrès réalisés ou des périodes de récupération nécessaires.

- **Ajout d'un mode "coach personnel"** : Proposer un service premium où des entraîneurs professionnels peuvent interagir avec les utilisateurs pour fournir des conseils personnalisés, des ajustements de programme, ou des recommandations en direct lors de leurs séances d'entraînement.

- **Fonctionnalités de gamification :** Encourager l'engagement en ajoutant des badges, des défis hebdomadaires, et un système de progression par niveaux pour récompenser les utilisateurs assidus et les motiver à atteindre de nouveaux objectifs.

- **Partenariats avec des marques de fitness :** Offrir des réductions exclusives sur des équipements ou des produits de nutrition via des partenariats avec des marques de fitness, pour créer une valeur ajoutée pour les utilisateurs tout en générant des opportunités de monétisation.

- **Statistiques basiques** : Un tableau simple ou un graphique affichant les progrès sur un exercice spécifique (ex. : poids soulevé au fil des semaines).

- **Faciliter les échanges** : Gérer les communications entre les membres inscrits pour pour que ces derniers puissent échanger sur leurs progrès

- **Notifications basiques** : Envoi de rappels pour les séances planifiées via une notification ou un email.

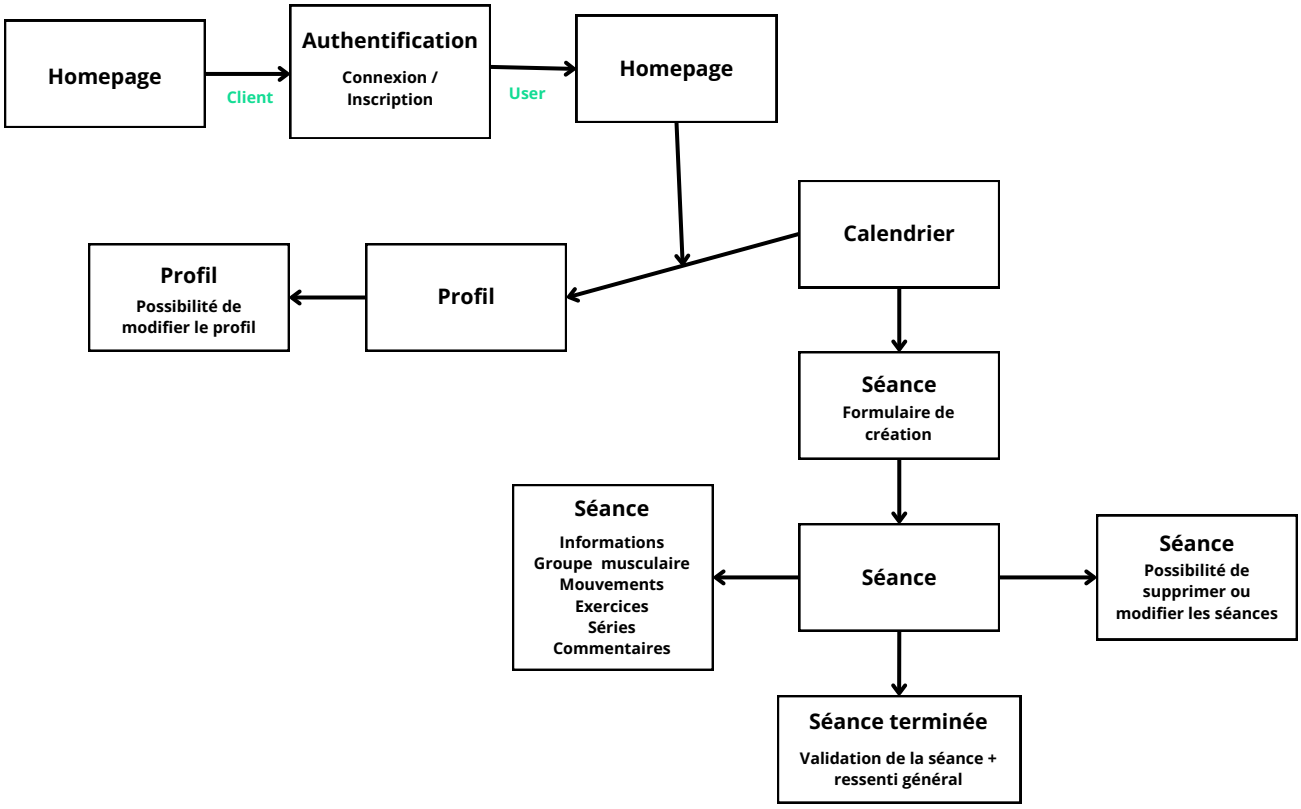
CP5

iv. Client et public visé

Le public cible pour Ironpal comprend toutes les personnes passionnées par la musculation qui souhaitent organiser, planifier et suivre efficacement leurs entraînements. L'application s'adresse particulièrement aux utilisateurs qui cherchent à optimiser leurs séances de musculation, à suivre leur progression et à atteindre leurs objectifs personnels en matière de condition physique. Le public de Ironpal est large et comprend des personnes de tous âges, de différents niveaux de pratique, et aux motivations variées en matière de musculation.

CP5 CP6

v. Arborescence



CP5

vi. Users Stories

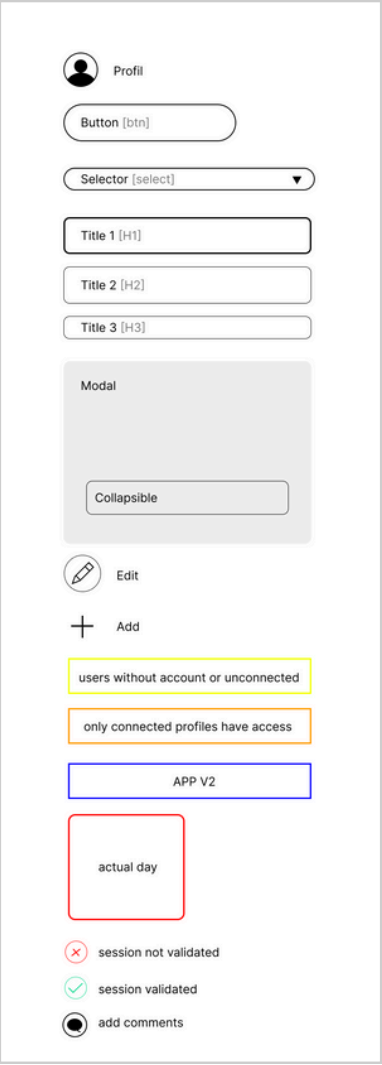
En tant que	Je veux
Visiteur	Pouvoir creer un compte avec mes credentials
Visiteur	Pouvoir me connecter si j'ai déjà un compte avec des credentials
Utilisateur	Pouvoir me deconnecter lorsque je suis deja connecté
Utilisateur	Pouvoir naviguer entre les différentes semaines d'un mois et avoir la visu sur les séances que j'ai déjà effectuées et celle que j'ai déjà enregistrés a venir
Utilisateur	Pouvoir créer des séances sur les différentes journées d'une semaine
Utilisateur	Pouvoir ajouter un titre a une séance que je suis en train de créer
Utilisateur	Pouvoir choisir les groupe musculaires travaillés dans la séance en choisissant parmi la liste des groupes musculaires

Utilisateur	Pouvoir ajouter un ou plusieurs exercices à une séance en choisissant parmi une liste d'exercices en fonction proposé en fonction des groupes musculaires choisi pour cette séance (ou libre si rien choisi)
Utilisateur	Pouvoir ajouter une charge sur l'exos en KG ou poids de corps
Utilisateur	Pouvoir ajouter des commentaires sur l'exercice
Utilisateur	Pouvoir ajouter un temps de repos a la fin de l'exercice
Utilisateur	Pouvoir ajouter une ou plusieurs séries au sein d'un exercice
Utilisateur	Pouvoir ajouter des temps de repos entre les séries
Utilisateur	Pouvoir ajouter une difficulté ressentie sur une série
Utilisateur	Pouvoir avoir le détail lorsque je clique sur une séance passée
Utilisateur	Pouvoir éditer lorsque je suis dans le détail de la séance passée

CP5

vii. Wireframes

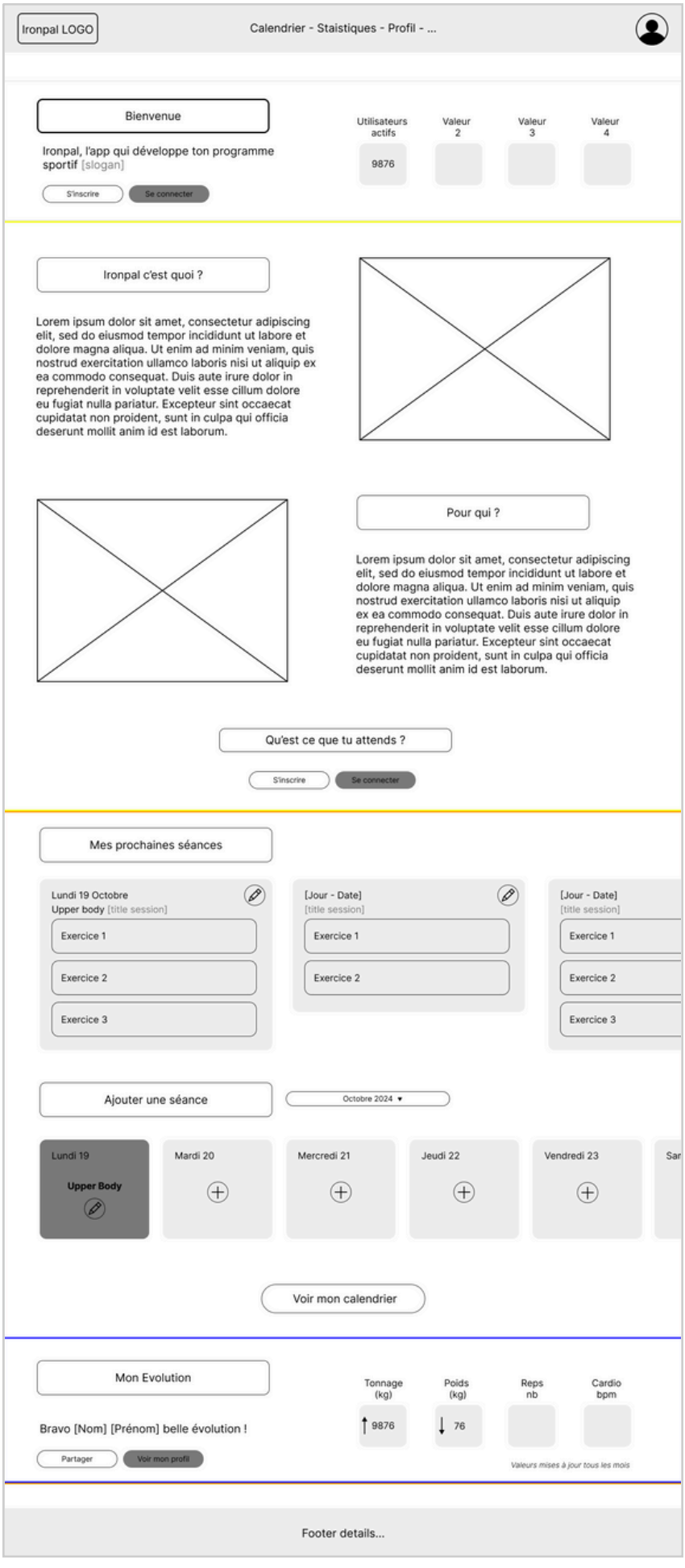
Wireframes System



Design System UI

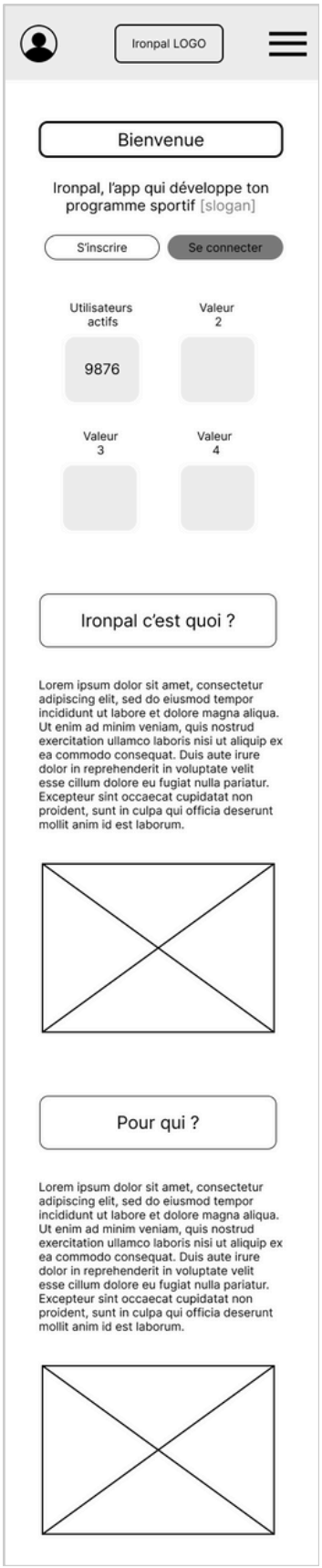
Wireframes Desktop

Home

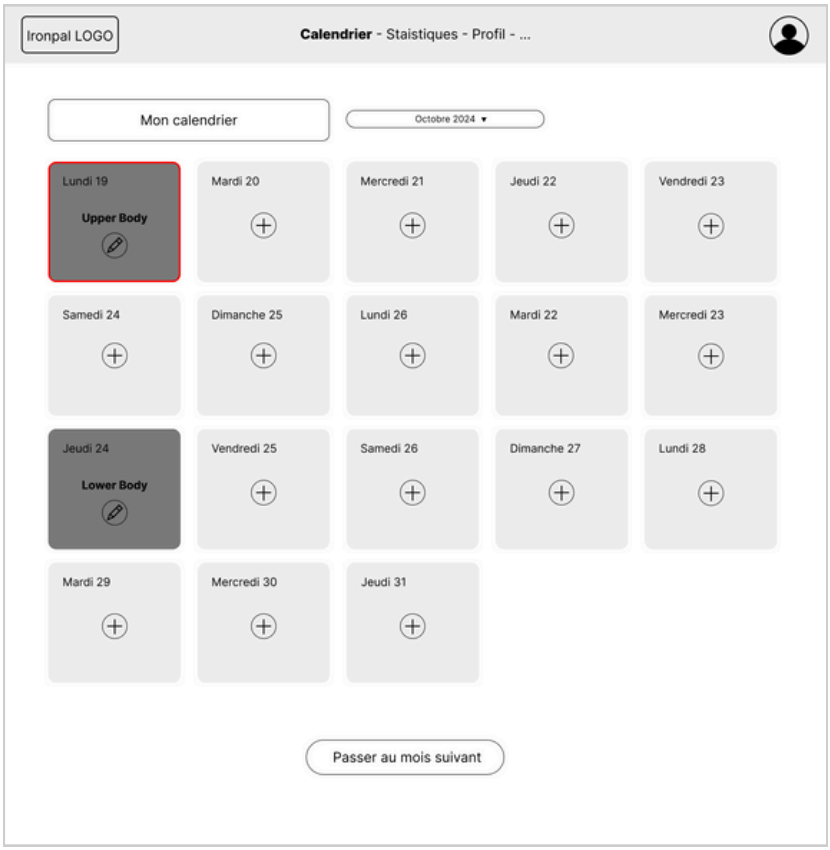


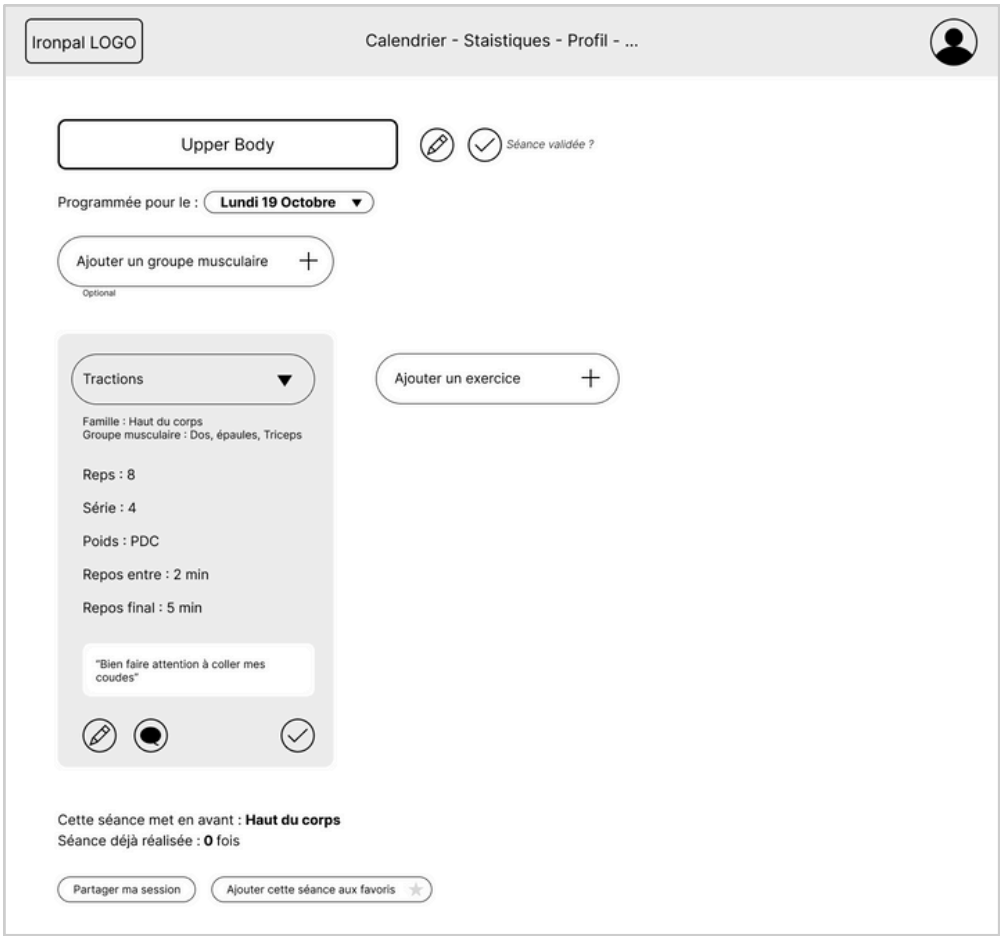
Wireframes Mobile

Home



Wireframes Desktop
Calendar





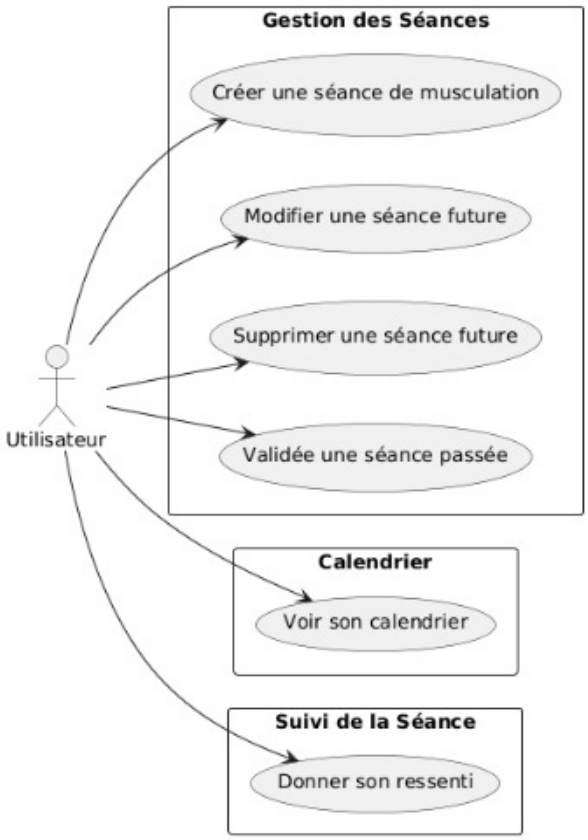
Wireframes Desktop
Session

*Suite a ces wireframes, il ya le développement de la **maquette** pour préparer au mieux le développement frontend (voir annexe)*

viii. Les spécifications techniques du projet

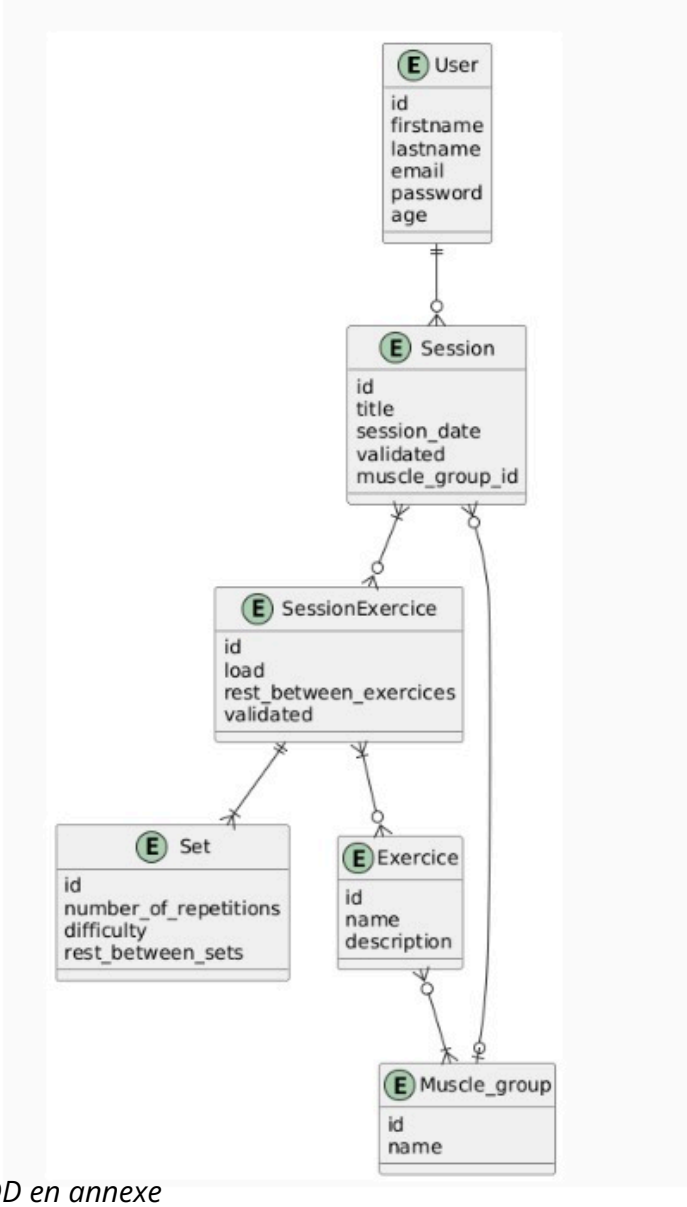
CP1 CP6

1. Use-cases



2. Documents relatifs à la BDD

a. Diagramme entité-relation



Retrouvez le dictionnaire de données de la BDD en annexe

3. Analyse des risques

- **Sécurité des données utilisateurs :** Il y a un risque de vol des données personnelles ou de failles de sécurité. Les attaques peuvent être de type phishing ou injection SQL. Si les données des utilisateurs sont compromises alors cela pourrait entraîner une mauvaise image de l'application, une perte de confiance des utilisateurs et des conséquences légales.
  - Solution : Utiliser des techniques de chiffrement avancé pour stocker les mot de passe (hashing, salage) ou encore contre les injections SQL, utilisation d'instructions préparées et de requêtes paramétrées ou l'utilisation d'un ORM.

- **Problèmes de performance et de scalabilité :** Dans le cas où il y aurait une augmentation du nombre d'utilisateurs et des données, cela pourrait impacter l'application qui pourrait devenir lente ou instable. Si des problèmes de performance ou scalabilité impact notre application cela pourrait réduire l'engagement des utilisateurs, entraîner des interruptions de service et impacter l'expérience utilisateur.
  - Solution : Ne pas hésiter à faire des tests de charges ainsi que des optimisations de performance grâce à la mise en cache, à la gestion asynchrone des requêtes, mise en place de pagination, faire de la compression de requête ou encore du monitoring.

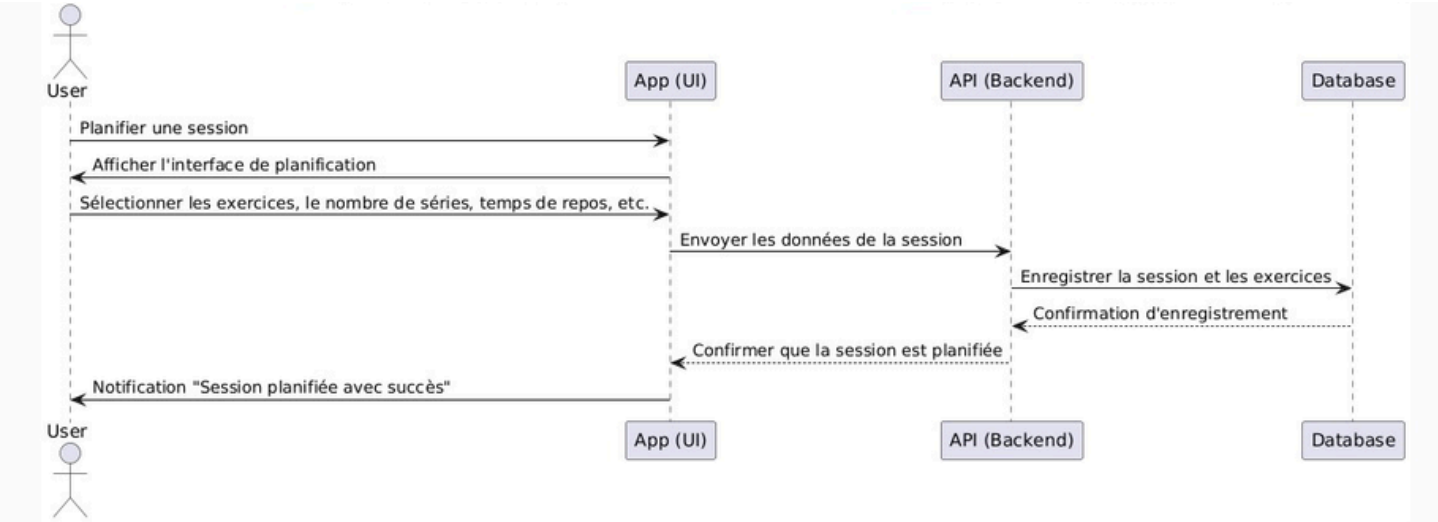
- **Problèmes d'ergonomie et d'expérience utilisateur (UX) :** Si l'application a une interface complexe ou une mauvaise conception UX cela pourrait décourager les utilisateurs. Les utilisateurs impactés peuvent abandonner l'application en raison de difficultés à naviguer.
  - Solution : Réalisation de tests utilisateurs récurrents pour ainsi optimiser l'interface et garantir une expérience fiable et fluide. L'application doit être responsive (mobile, desktop).

- **RGPD (Règlement Général sur la Protection des Données) :** Le risque de ne pas respecter les règles de protection des données personnelles imposées par la RGPD pourrait entraîner des amendes importantes et des sanctions légales, ainsi qu'une perte de confiance des utilisateurs.
  - Solution : Implémenter des politiques de gestion des données conformes à la RGPD, informer les utilisateurs de la collecte des données, et offrir des options pour supprimer leurs informations personnelles.

- **Dépendance technologique** : Dépendre fortement d'une technologie ou d'un service tiers (par exemple pour l'authentification) pourrait devenir obsolète ou coûteux. Cela pourrait entraîner des interruptions de service, une augmentation des coûts ou des complications liées à la maintenance.
  - Solution : Utiliser des technologies standardisées et largement adoptées.

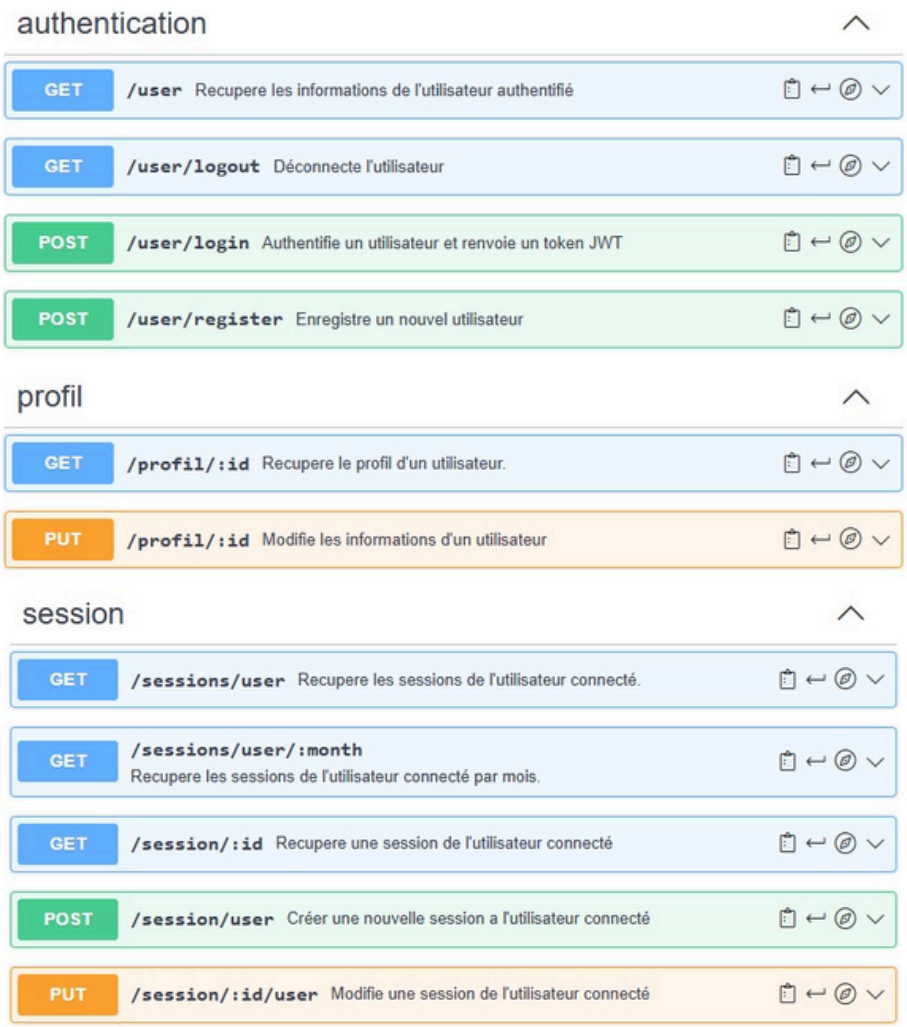
CP1 CP6

4. Diagramme de séquence



CP1 CP6

5. Liste des routes (API)



swagger

6. Liste des technos utilisées

- **React :** Bibliothèque JavaScript pour créer des interfaces utilisateur. Les composants vont nous permettre de créer des UI modulaires, réutilisables et maintenables.

- **Node.js :** Environnement d'exécution pour exécuter du JavaScript côté serveur. Node.js est particulièrement utile pour unifier le langage utilisé côté frontend et backend.

- **Jest** : Pour les tests côté backend.

- **React-testing-library** : Pour les tests côté React.

- **PostgreSQL** : Base de données relationnelle utilisée pour stocker les informations utilisateurs et données d'entraînement.

- **TypeScript** : Superset de JavaScript avec typage statique, utilisé pour un développement plus sûr et maintenable. TypeScript est un sous-ensemble de JavaScript qui ajoute le typage statique, ce qui permet de détecter plus facilement les erreurs lors du développement.

- **Prisma** : ORM pour simplifier l'interaction entre l'application et la base de données. Prisma est un ORM (Object-Relational Mapping) moderne et performant qui facilite les interactions avec la base de données PostgreSQL. Il permet de simplifier les requêtes SQL complexes et d'automatiser certaines tâches liées à la gestion des données.

- **Docker** : Organiser les conteneurs et gérer le projet front/back. Docker permet de conteneuriser les applications, garantissant qu'elles fonctionnent de manière cohérente sur différents environnements. Cela facilite le déploiement et la gestion de l'infrastructure front/backend.

7. Outil de travail en commun

Nous avons utilisé GitHub, un outil essentiel pour le travail collaboratif, offrant de nombreux outils pour le travail d'équipe mais aussi le développement de l'application en bon uniforme.

1. Gestion du Code Source avec Git

- Versioning efficace : GitHub permet de suivre l'historique des modifications du code. Chaque développeur peut voir qui a fait quoi, quand et pourquoi, grâce à des commits détaillés.
- Restauration facile : En cas de bug ou d'erreur, on peut revenir à une version antérieure du code.

2. Travail en Collaboration

- Branches pour les fonctionnalités : Chaque développeur peut travailler sur une branche indépendante (par exemple, une branche pour une nouvelle fonctionnalité ou une correction de bug). Cela évite les conflits et permet de tester avant de fusionner dans la branche principale.
- Pull Requests (PR) : Les développeurs soumettent leurs modifications via des PR. Cela permet aux autres membres de l'équipe de revoir le code, de commenter et de valider avant de l'intégrer.
- Résolution des conflits : En cas de modifications simultanées sur une même partie du code, GitHub aide à résoudre les conflits en affichant les différences.

3. Communication et Organisation

- Issues et Discussions : Les "issues" permettent de signaler des bugs, de planifier des tâches ou de proposer des idées. Les discussions autour des issues assurent une communication claire entre les membres de l'équipe.
- Projets et tableaux Kanban : GitHub offre des outils pour organiser les tâches sous forme de tableaux, ce qui facilite le suivi de l'avancement du projet.

8. Choix des librairies utilisées

LIBRAIRIES FRONTEND

- **Material-UI (MUI)** : Librairie de composants d'interface utilisateur, basée sur le design. MUI est une bibliothèque de composants React qui met en œuvre les principes du design Material. Cela permet d'avoir des composants pré-construits, le respect des normes UX/UI, et une personnalisation facile.

- **React-Router-Dom** : Librairie pour gérer le routage et la navigation dans l'application React. React-Router-Dom gère le routage dans les applications React. Il permet de naviguer entre différentes pages sans recharger la page complète, ce qui est crucial pour une expérience utilisateur fluide.

- **Axios** : Pour faciliter la gestion des calls API. Axios est une librairie JavaScript pour effectuer des requêtes HTTP vers des API. Elle est plus flexible et plus puissante que l'API native fetch de JavaScript, offrant un meilleur contrôle des requêtes et des réponses.

- **Dayjs** : Pour faciliter la manipulation des dates. Day.js est une bibliothèque JavaScript légère et performante conçue pour faciliter la manipulation, le traitement et l'affichage des dates et des heures. C'est une alternative minimaliste aux bibliothèques plus lourdes comme Moment.js.

LIBRAIRIES BACKEND

- **ExpressJS** : Framework minimaliste pour créer des APIs côté serveur. ExpressJS est utilisé comme framework côté serveur pour créer et gérer les APIs.

- **BcryptJS** : Librairie pour le hashage des mots de passe et la sécurité des données utilisateur. Dans notre application, elle nous permettra de hasher les mots de passe des utilisateurs.

- **JsonWebToken (JWT)** : Librairie pour gérer l'authentification basée sur des tokens. Les tokens JWT permettent de s'assurer que les utilisateurs sont authentifiés à chaque requête sans avoir à stocker d'informations sensibles sur le serveur.

- **Cors :** Middleware Express pour activer les requêtes cross-origin (CORS) entre le frontend et le backend. Nous utilisons le middleware Cors pour autoriser les requêtes entre le frontend (hébergé sur un domaine ou un port différent) et le backend.

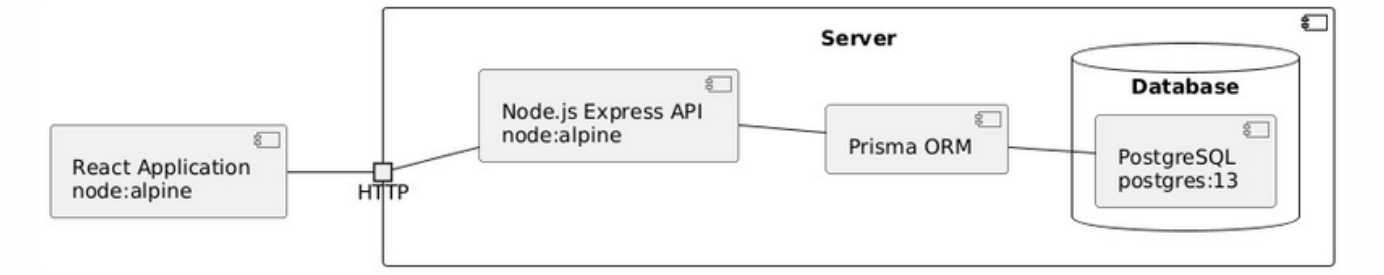
- **Dotenv** : Librairie pour gérer les variables d'environnement de manière sécurisée. Dotenv est utilisé pour gérer les variables d'environnement, telles que les clés d'API, les informations de connexion à la base de données, et d'autres configurations sensibles.

- **Swagger** : Swagger génère une documentation interactive qui décrit chaque endpoint de l'API, les paramètres attendus, et les réponses possibles. Cela facilite à la fois le développement et l'intégration avec d'autres systèmes, en fournissant une interface pour tester et comprendre l'API et établir le CRUD.

- **Joi** : Outils pour gérer les contraintes sur les champs des données. Par exemple, dans les champs du formulaire d'inscription, elle a permis de mettre en place des contraintes de champs particulièrement pour les e-mails et mots de passe.

CP1 CP6

9. Architecture



b. Gestion du projet

CP4

i. L'équipe

Le projet Ironpal a vu le jour grâce à une équipe de 4 étudiants de la promotion Elfe, qui ont su unir leurs forces pour donner vie à une idée ambitieuse. Nous avons travaillé en collaboration, en organisant efficacement nos tâches et en répartissant les rôles de manière claire pour respecter le délai imparti.

Voici la répartition des rôles au sein de l'équipe :

- Backend - **Product Owner** : **Valérian**
- Backend - **Scrum Master** : **Soukina**
- Frontend - **Référent Technique** : **Thibaut**
- Frontend - **Git Master** : **Maxime**

Malgré ces "casquettes" attitrées, nous avons travaillés en équipe et touché à tout les domaines afin de suivre et comprendre au mieux l'avancé de l'application.

CP4

ii. Organisation et méthodologie de travail

Nous avons rejoint un salon Discord dédié pour des réunions quotidiennes, où nous définissions les objectifs et tâches de la journée. Des salons spécifiques pour le frontend et le backend ont été créés afin d'optimiser les échanges et la coordination.

Lors du premier sprint, nous avons collaboré sur le cahier des charges pour clarifier les tâches de chaque équipe et les interactions entre le frontend et le backend. Ensuite, deux équipes distinctes ont été formées, avec une réunion chaque soir pour faire le point sur les avancées.

GitHub a été notre principal outil de gestion, avec un tableau Kanban pour organiser les tâches et un processus rigoureux de versionnement. Nous avons adopté les Conventional Commits et une convention de nommage des branches (ex. feat/login). Chaque Pull Request devait être approuvée par un membre avant d'être fusionnée, et des vérifications automatiques via GitHub Actions garantissaient la qualité du code.

Cette organisation nous a permis de mener à bien le projet dans les délais grâce à une communication efficace et un bon travail d'équipe.

5. Sprint 1 - Développement

CP1

a. Initialisation du projet

Suite au sprint de conception et à la finalisation du cahier des charges, nous avons entamé la phase de développement. Comme indiqué dans les technologies utilisées, nous avons installé le framework React en TypeScript. Pour ce faire, nous avons utilisé le bundler Vite, qui offre un environnement rapide et optimisé pour le développement. L'ensemble des fichiers du frontend a été structuré dans le dossier frontend.

```
npm create vite@latest frontend -- --template react-ts
```

Du côté backend, les fichiers ont été créés manuellement, en commençant par le fichier principal index.ts, qui sert de point d'entrée. Les autres fichiers sources ont été placés à la racine du dossier backend. Nous avons intégré un dossier prisma pour gérer l'ORM, notamment la définition des modèles, les migrations et la gestion des requêtes à la base de données.

Un fichier init.sql a été mis en place pour initialiser les tables de base et insérer les données nécessaires (seeding) lors du premier démarrage (plus de détail par la suite du dossier). Ce fichier est exécuté automatiquement par PostgreSQL au démarrage du conteneur. Une fois conteneurisé avec Docker, PostgreSQL prendra en charge la gestion complète des données.

Nous avons ensuite créé un fichier docker-compose.yml pour conteneuriser l'ensemble du projet, en s'appuyant sur les Dockerfiles configurés respectivement pour le frontend et le backend. Ce fichier permet de lancer les différents services nécessaires, notamment le frontend, le backend, et la base de données. Les conteneurs sont configurés pour communiquer entre eux, et les ports sont mappés pour garantir un accès fluide. La commande sudo docker-compose up permet de démarrer tous les services simultanément.

docker-compose :

```yaml
version: '3.8'

services:
  backend:
    container_name: backend
    build:
      context: ./backend
      dockerfile: Dockerfile
    volumes:
      - ./backend:/app/backend
      - /app/backend/node_modules
    ports:
      - 3000:3000
      - 5555:5555 # Port used by Prisma Studio
    working_dir: /app/backend
    depends_on:
      - database

  database:
    container_name: database
    image: postgis/postgis
    restart: on-failure
    env_file:
      - docker.env
    ports:
      - "5432:5432"
    volumes:
      - ./backend/init.sql:/docker-entrypoint-initdb.d/init.sql
      - postgresql-data:/var/lib/postgresql/data
  frontend:
    container_name: frontend
    build:
      context: ./frontend
      dockerfile: Dockerfile
    volumes:
      - ./frontend:/app/frontend
      - /app/frontend/node_modules
    ports:
      - 5173:5173
    working_dir: /app/frontend
    depends_on:
      - backend
      - database

  adminer:
    container_name: adminer
    image: adminer
    restart: always
    ports:
      - 8080:8080

volumes:
  postgresql-data:
```

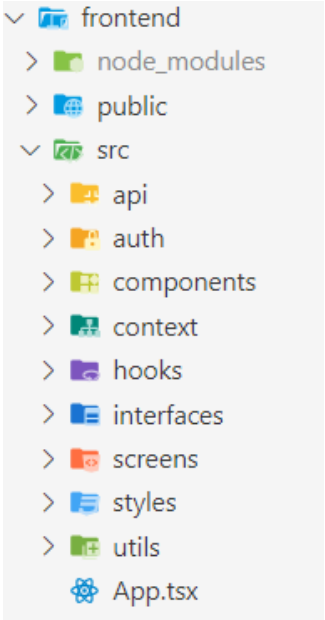
Désormais, tous les conteneurs se lancent correctement sur les bons ports. On utilise les images de Docker dans les Dockerfiles du frontend et du backend.

- Le frontend est accessible via le port 5173.
- Le backend est exposé sur le port 3000 pour l'API et 5555 pour Prisma Studio.
- La base de données PostgreSQL utilise le port 5432.

b. Architecture fichiers

Une fois les principaux dossiers créés, il est essentiel de respecter une architecture propre tout au long du développement pour assurer la maintenabilité et la clarté du projet. Pour le frontend, nous avons structuré le projet en plusieurs dossiers dédiés :

FRONTEND



- **API** : Ce dossier contient la configuration d'Axios pour gérer les requêtes HTTP avec Vite, ainsi que les définitions des routes (CRUD : CREATE, PUT, DELETE, etc.) et les services associés.

- **Components** : Ce dossier organise tous les composants réutilisables de l'application, tels que les modals, les cards, les layouts...

- **Screens** : Ce dossier regroupe les différentes pages de l'application, chaque page étant composée de plusieurs composants.

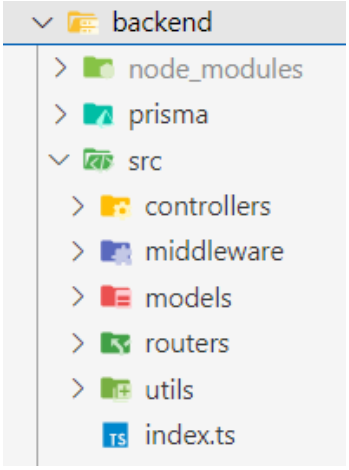
- **Context** : Ce dossier gère les différents Providers nécessaires à l'application, permettant de partager des états ou des fonctionnalités globales (comme l'authentification ou les notifications Snackbar) entre les composants.

- **Interfaces** : Ce dossier contient des interfaces TypeScript, qui permettent de définir des structures pour les données et les props des composants. Par exemple, une interface décrit précisément la forme d'un objet ou les types de valeurs attendus, assurant un développement plus robuste et sécurisé.

- **Styles** : Ce dossier comprend des fichiers liés au design de l'application, notamment le fichier theme.ts, qui configure et personnalise les styles Material-UI (MUI) en fonction du thème global.

- **Utils** : Ce dossier regroupe des fonctions utilitaires réutilisables dans plusieurs parties de l'application. Ces fonctions simplifient certaines opérations ou calculs spécifiques comme par exemple transformer le format du temps de repos entre les séries (minutes en secondes).

BACKEND



- **Prisma** : Ce dossier contient le fichier schema.prisma, qui définit les modèles et les relations de la base de données. Par défaut, Prisma génère ici le client et les fichiers de migration nécessaires pour synchroniser le schéma avec la base de données.

- **Controllers** : Ce dossier regroupe tous les contrôleurs, comme AuthController. Les contrôleurs gèrent la logique métier et orchestrent les appels entre les services, la base de données et les routes.

- **Middleware** : Ce dossier est dédié aux middlewares, qui sont des fonctions exécutées avant ou après les contrôleurs. Ils permettent de gérer des tâches transversales comme l'authentification, la validation des données, ou la gestion des erreurs.

- **Models** : Ce dossier contient les modèles spécifiques au backend. Ces modèles définissent les structures de données utilisées au sein des contrôleurs, en se basant souvent sur les définitions Prisma ou d'autres besoins métier.

- **Routers** : Ce dossier contient toutes les routes de l'application, qui définissent les endpoints exposés par l'API. Chaque fichier de ce dossier correspond à une fonctionnalité ou à une ressource spécifique.

- **Utils** : Ce dossier regroupe des fonctions utilitaires génériques, réutilisables dans plusieurs parties du backend. Ces fonctions servent à simplifier des opérations courantes, comme des transformations de données ou même la génération du token avec jwt.

c. Conception d'une base de données relationnelle

La conception de notre base de données relationnelle a été réalisée en suivant une méthodologie structurée. Nous avons utilisé des outils et des approches classiques, comme Merise, pour modéliser nos données :

- **Modèle Conceptuel de Données (MCD)** :
  - Identification des entités principales (utilisateurs, exercices, sessions, etc.).
  - Définition des relations entre les entités, telles que les associations entre les utilisateurs et leurs sessions d'entraînement ou les exercices liés aux sessions.
  - Précision des cardinalités pour garantir l'intégrité relationnelle (par exemple, un utilisateur peut avoir plusieurs sessions, mais chaque session appartient à un seul utilisateur).
- **Modèle Logique de Données (MLD)** :
  - Transformation des entités et relations du MCD en tables, avec des clés primaires, des clés étrangères, et des contraintes pour représenter les relations et assurer l'intégrité référentielle.
- **Entity-Relationship Diagram (ERD)** :
  - Création d'un schéma visuel illustrant les relations entre les tables de la base de données, facilitant la compréhension globale et la collaboration avec l'équipe.

Comme évoqué précédemment, notre base de données relationnelle développée avec PostgreSQL utilise le fichier init.sql pour initialiser la structure (tables, contraintes, relations) et insérer des données de seeding essentielles au fonctionnement initial de l'application.

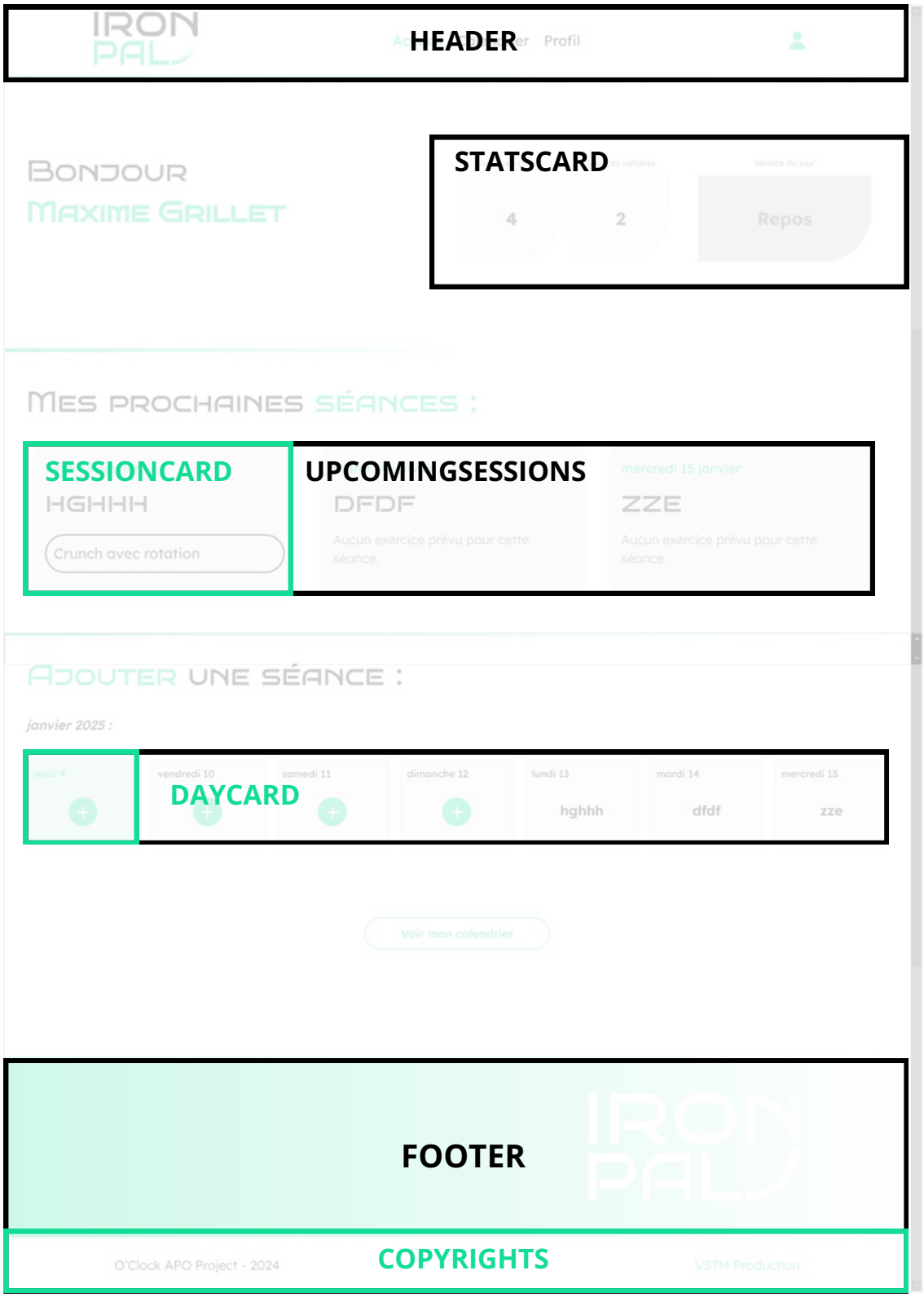
- **Tables** : Structures principales pour stocker les données (ex. user, session, exercice, etc.).
- **Relations** : Liens entre les tables via des clés étrangères, comme user_id dans session, qui relie une session à un utilisateur.
- **Contraintes** : Règles pour garantir l'intégrité des données, comme les clés primaires (id) et les valeurs par défaut (ex. validated BOOLEAN DEFAULT false).
- **Données d'exemple** : Commandes INSERT INTO pour initialiser les tables avec des utilisateurs, groupes musculaires, exercices, etc.
- **Hiérarchie des données** : Organisation claire : un utilisateur → plusieurs sessions → exercices → séries.

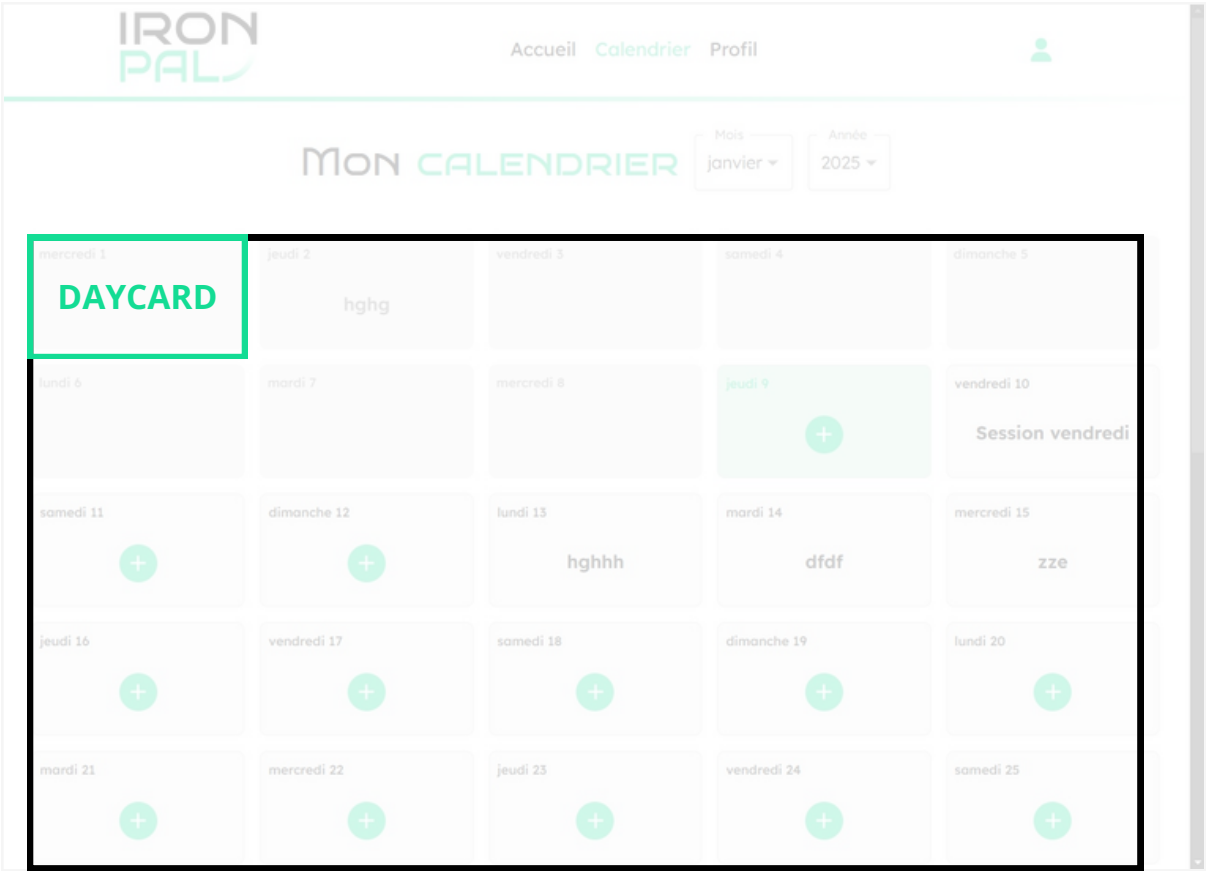
Grâce à l'intégration avec Docker et l'image de PostgreSQL, l'initialisation de la base de données est automatisée lors du démarrage du conteneur. Cette approche offre de nombreux avantages pour le développement mais aussi le déploiement et assure la fiabilité, la flexibilité et l'évolutivité de la base de données.

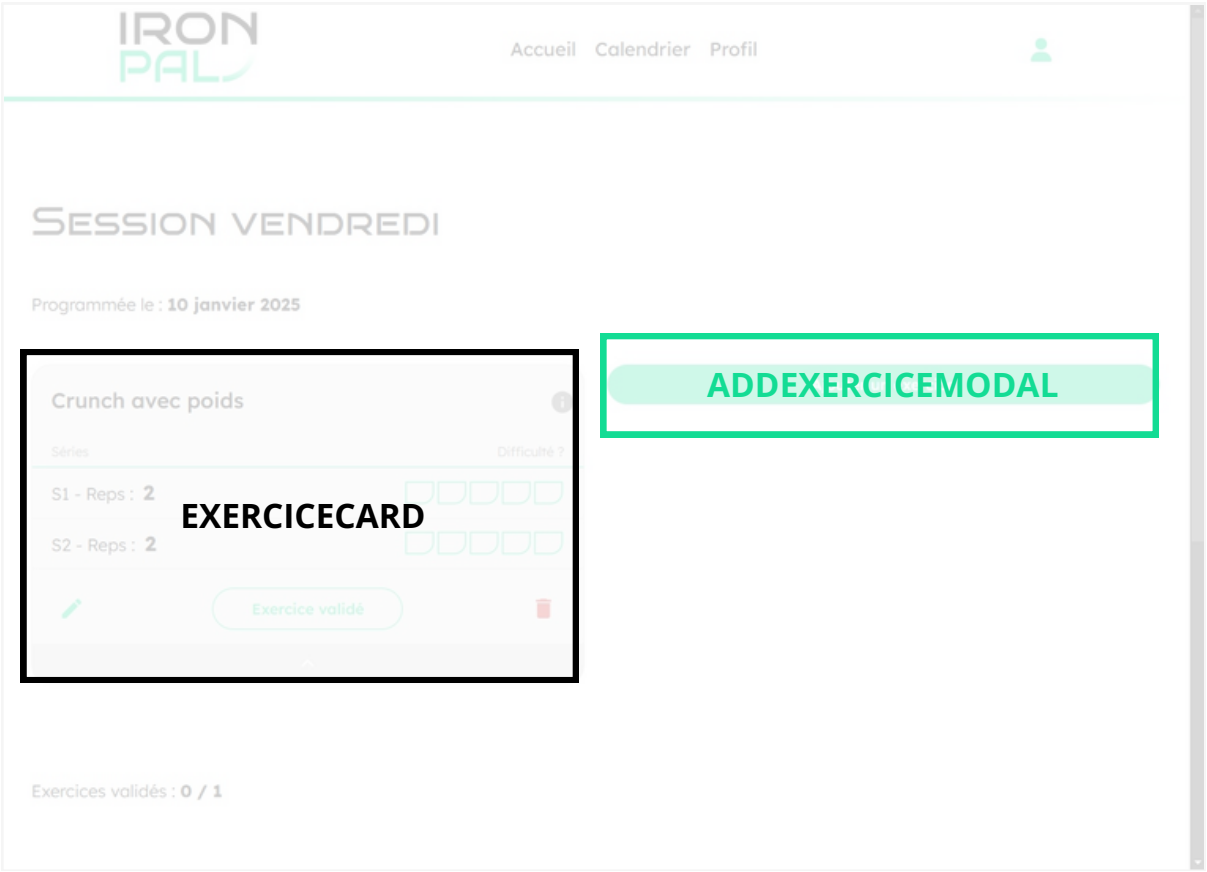
Init.sql :

```sql
-- CreateTable
CREATE TABLE "user" (
    "id" SERIAL NOT NULL,
    "firstname" VARCHAR NOT NULL,
    "lastname" VARCHAR NOT NULL,
    "email" VARCHAR NOT NULL,
    "password" VARCHAR NOT NULL,
    "birthdate" VARCHAR,

    CONSTRAINT "user_pkey" PRIMARY KEY ("id")
);
-- CreateTable
CREATE TABLE "exercise" (
    "id" SERIAL NOT NULL,
    "name" VARCHAR NOT NULL,
    "description" VARCHAR,

    CONSTRAINT "exercise_pkey" PRIMARY KEY ("id")
);
-- CreateTable
CREATE TABLE "muscleGroup" (
    "id" SERIAL NOT NULL,
    "name" VARCHAR NOT NULL,

    CONSTRAINT "muscleGroup_pkey" PRIMARY KEY ("id")
);

-- ... other tables

-- Insérer des données dans la table "Utilisateur"
INSERT INTO "user" (firstname, lastname, email, password, birthdate) VALUES
('Jean', 'Dupont', 'jean.dupont@exemple.fr', 'motDePasse123', '1985-10-28'),

-- Insérer des données dans la table "GroupeMusculaire"
INSERT INTO "muscleGroup" (name) VALUES
('Trapezes'),
-- ... other muscleGroups

-- Insérer des données dans la table "exercise"
INSERT INTO "exercise" (name, description) VALUES

-- Trapèze - 1
('Shrugs avec haltères', 'Tenez un haltère dans chaque main, bras le long du corps. Haussez les
épaules aussi haut que possible, maintenez la contraction un instant, puis relâchez lentement.'),
-- ...

-- Insérer des données dans la table "exerciseMusculaire"
INSERT INTO "exerciseMuscleGroup" (exercise_id, muscle_group_id) VALUES

-- Trapèze - 1
(1, 1),
-- ...

-- Insérer les données dans la table "session"
INSERT INTO "session" (title, session_date, validated, user_id, muscle_group_id) VALUES
('Entraînement trapèze', '2023-10-01 08:00:00', true, 1, 1),

-- Insérer les données dans la table "sessionExercice"
INSERT INTO "sessionExercise" (load, rest_between_exercises, validated, comment, session_id,
exercise_id) VALUES
(80, 180, true,'Mettre des coudieres', 1, 1),

-- Insert data into "set" table
INSERT INTO "set" (number_of_repetitions, difficulty, rest_between_sets, session_exercise_id) VALUES
(12, 2, 120, 1) -- séries d'exercices pour les trapèzes
```

d. Les composants

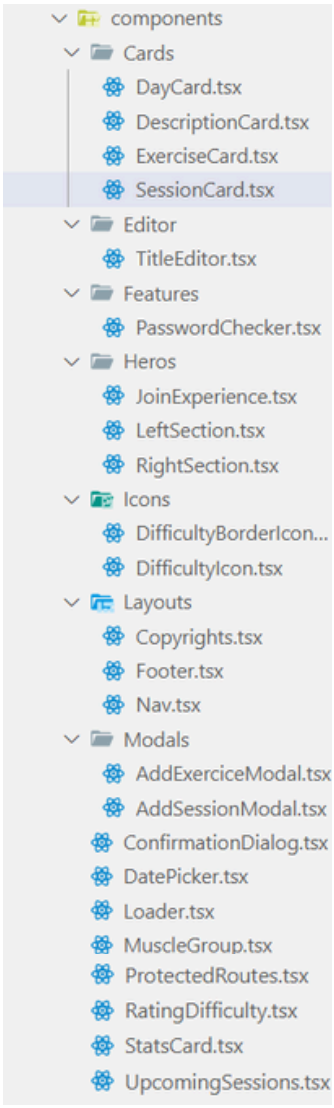



IRON PAL
Accueil Calendrier Profil
Mon calendrier
Mois
janvier
Année
2025
DAYCARD
mercredi 1
jeudi 2
hghg
vendredi 3
samedi 4
dimanche 5
lundi 6
mardi 7
mercredi 8
jeudi 9
vendredi 10
Session vendredi
samedi 11
dimanche 12
lundi 13
hghhh
mardi 14
dfdf
mercredi 15
zze
jeudi 16
vendredi 17
samedi 18
dimanche 19
lundi 20
mardi 21
mercredi 22
jeudi 23
vendredi 24
samedi 25

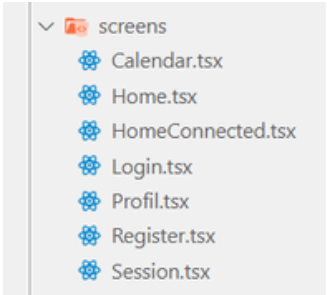

IRON PAL
Accueil Calendrier Profil
Session vendredi
Programmée le : 10 janvier 2025
ADDEXERCICEMODAL
Crunch avec poids
Séries
Difficulté ?
S1 - Reps : 2
S2 - Reps : 2
EXERCICECARD
Exercice validé
Exercices validés : 0 / 1

FRONTEND

Composants :

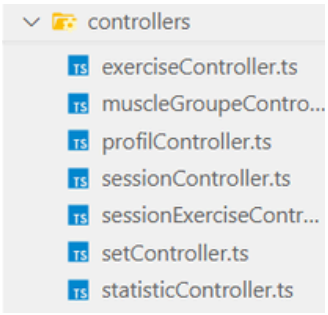


Screens :

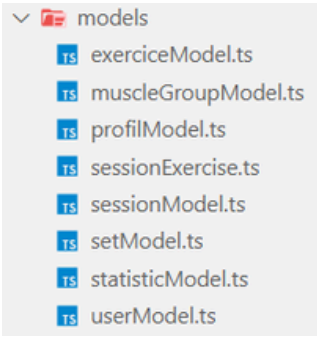


BACKEND

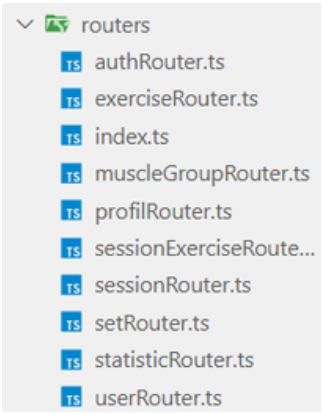
Controllers :



Models :



Routers :



e. Composant métier en détail : addExerciceModal

CP3

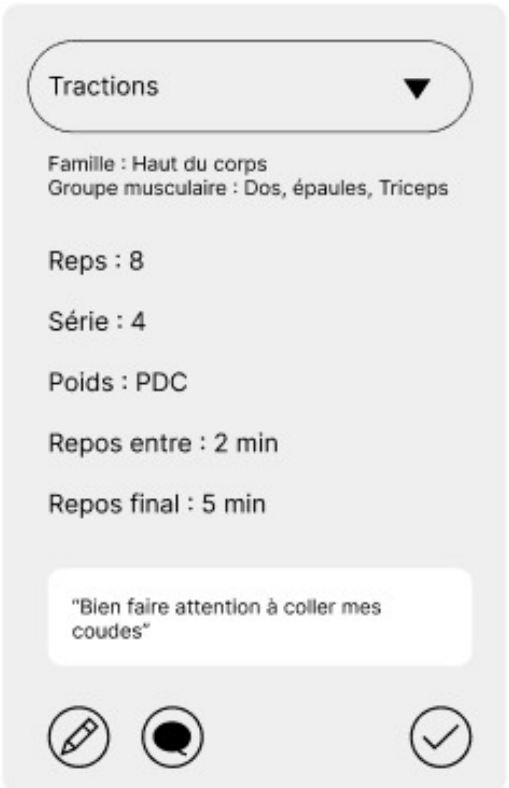
i. Conceptualisation et schéma de conception

Le composant AddExerciceModal , comme son nom l'indique, est une modale permettant d'ajouter un exercice. C'est un élément central de notre projet, car il permet de créer des exercices et de personnaliser ainsi ses sessions d'entraînement.

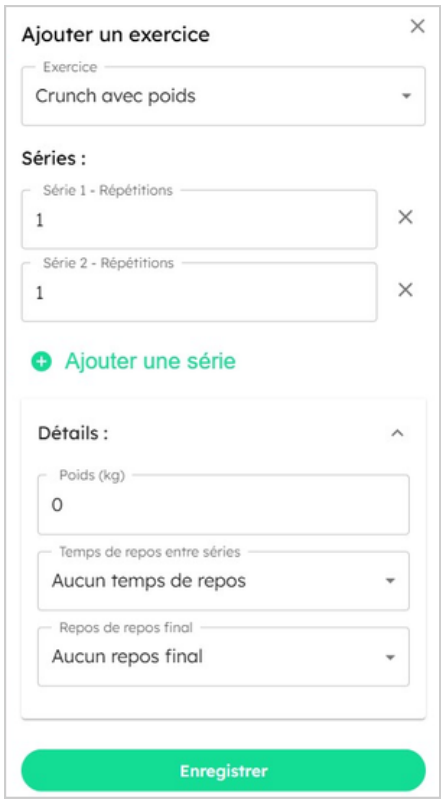
Cette modale présente plusieurs fonctionnalités : elle affiche un sélecteur d'exercice parmi ceux définis dans le fichier init.sql comme données de seeding. Ensuite, elle permet d'ajouter des séries (sets) avec un nombre personnalisé de répétitions pour chaque série. Un accordéon est également présent pour ajouter des détails supplémentaires, tels que le poids de l'exercice, ainsi que le temps de repos entre les séries et le repos final après l'exercice.

L'ensemble de la modale AddExerciceModal fonctionne comme un formulaire de type CRUD (CREATE, READ, UPDATE, DELETE). Elle permet une gestion complète des exercices associés à une session d'entraînement, directement depuis l'interface utilisateur, avec des interactions en temps réel avec la base de données.

Maquette du composant AddExerciceModal de base :



Composant AddExerciceModal final :



FRONTEND

La balise Modale de MUI avec ses composants dynamiques permettant en front la gestion de l'ajout et de l'édition des exercices dans une session

```jsx
<Modal open={open} onClose={onClose}>
  <Box>
    {/** Bouton pour fermer la modal */}
    <IconButton onClick={onClose}>
      <CloseIcon />
    </IconButton>

    {/** Titre de la modal */}
    <Typography variant="h6">Ajouter un exercice</Typography>

    {/** Champ de sélection pour choisir un exercice */}
    <FormControl fullWidth>
      <InputLabel>Exercice</InputLabel>
      <Select
        value={selectedExercise}
        onChange={handleChangeExercise}
        MenuProps={{ PaperProps: { sx: { maxHeight: 400 } } }}
      >
        {/** Les exercices sont triés alphabétiquement */}
        {exercises.sort((a, b) => a.name.localeCompare(b.name)).map((exercise) => (
          <MenuItem key={exercise.id} value={exercise.id}>
            {exercise.name}
          </MenuItem>
        ))}
      </Select>
    </FormControl>

    {/** Section pour gérer les séries (sets) */}
    {selectedExercise && (
      <Box>
        <Typography variant="subtitle1">Séries :</Typography>
        {sets.map((serie, index) => (
          <Box key={index}>
            {/** Champ pour le nombre de répétitions */}
            <TextField
              label={`Série ${index + 1} - Répétitions`}
              type="number"
              value={serie.number_of_repetitions}
              onChange={(e) => handleChangeSets(index, Number(e.target.value))}
              fullWidth
            />
            {/** Bouton pour supprimer une série */}
            <IconButton onClick={() => handleDeleteSet(index)}>
              <CloseIcon />
            </IconButton>
          </Box>
        ))}
        {/** Bouton pour ajouter une nouvelle série */}
        <IconButton onClick={handleAddSet}>
          <AddCircleIcon /> Ajouter une série
        </IconButton>
      </Box>
    )}
```

```
          {/** Champ pour le poids */}
          <TextField
            label="Poids (kg)"
            type="number"
            value={load}
            onChange={handleChangeLoad}
            fullWidth
            inputProps={{ min: 0 }}
          />

          {/** Temps de repos entre séries */}
          <FormControl fullWidth>
            <InputLabel>Temps de repos entre séries</InputLabel>
            <Select
              value={restBetweenSets}
              onChange={handleChangeRestBetweenSets}
              MenuProps={{ PaperProps: { style: { maxHeight: 200 } } }}
            >
              <MenuItem value="0">Aucun temps de repos</MenuItem>
              {timeOptions.map((option) => (
                <MenuItem key={option} value={option}>
                  {convertSecondsToRest(option)}
                </MenuItem>
              ))}
            </Select>
          </FormControl>

          {/** Temps de repos final */}
          <FormControl fullWidth>
            <InputLabel>Repos final</InputLabel>
            <Select
              value={restBetweenExercises}
              onChange={handleChangeBetweenExercises}
              MenuProps={{ PaperProps: { style: { maxHeight: 200 } } }}
            >
              <MenuItem value="0">Aucun repos final</MenuItem>
              {timeOptions.map((option) => (
                <MenuItem key={option} value={option}>
                  {convertSecondsToRest(option)}
                </MenuItem>
              ))}
            </Select>
          </FormControl>
        </AccordionDetails>
      </Accordion>
    )}

    {/** Bouton pour enregistrer les données de l'exercice */}
    <Button
      variant="contained"
      color="primary"
      onClick={handleSubmit}
      fullWidth
      disabled={isSaveDisabled}
    >
      Enregistrer
    </Button>
  </Box>
</Modal>
```

Fonctions de soumission et gestion des opérations CRUD pour les exercices de la session :

```
/** Fonction pour gérer la soumission des données */
const handleSubmit = async () => {
  const payload = {
    session_id: parseInt(id),
    exercise_id: parseInt(selectedExercise),
    load,
    rest_between_exercises: String(restBetweenExercises),
    sets: sets.map((set) => ({
      ...set,
      rest_between_sets: String(restBetweenSets),
    })),
  };

  try {
    if (isEmptyObject(sessionExercise)) {
      /** Si la session exercice n'existe pas encore, on crée un nouvel exercice */
      const createResponse = await CREATEsessionExercise(payload);
      if (createResponse.status === 200) {
        handleAddSessionExercise(createResponse.data);
        onClose();
      }
    } else {
      /** Si la session exercice existe, on la met à jour */
      const updateResponse = await PUTsessionExercise(sessionExercise.id, payload);
      if (updateResponse.status === 200) {
        handleUpdateSessionExercise(updateResponse.data);
        onClose();

    }
  } catch (error) {
    console.error("Erreur lors de la soumission de l'exercice :", error);
  } finally {
    /** Réinitialise les états après soumission */
    resetForm();
    setSessionExerciseToEdit({} as SessionExerciseWithExerciseAndSets);
  }
};
```

```
/** Méthodes pour créer et mettre à jour les exercices */
export const CREATEsessionExercise = async (payload: CreateSessionExercise) => {
  try {
    return await axios.post(SESSION_EXERCISE.CREATE, payload);
  } catch (error: any) {
    console.error("Erreur lors de la création :", error);
    throw error.response?.data || error.message;
  }
};

export const PUTsessionExercise = async (id: number, payload: UpdateSessionExercise) => {
  try {
    return await axios.put(`${SESSION_EXERCISE.UPDATE}/${id}`, payload);
  } catch (error: any) {
    console.error("Erreur lors de la mise à jour :", error);
    throw error.response?.data || error.message;
  }
};
```

BACKEND

Nous allons voir désormais coté backend la structure de notre projet, principalement concernant la structure des routes pour la notre méthode CREATE pour ajouter notre session_exercice

```
//  Ce fichier sert de point d'entrée pour organiser les sous-routers  :
router.use('/sessionExercise', sessionExerciseRouter);
```

```
//  C'est ici que la route est définie pour créer une session exercise :
router.post('/', catchErrors(sessionExerciseController.create))
```

```
// Le controller sessionExerciseController.create contient la logique métier pour la création d'une
session exercise :
async create(req: Request, res: Response, next: NextFunction) {
    const { body } = req
    const createdSessionExercise = await sessionExerciseModel.create(body);
    if(!createdSessionExercise) {
      const err = new ApiError(`Can not create session exercise`, 400);
      return next(err);
    };
    res.status(200).json(createdSessionExercise);
  }
// qui appel mon model qui va contenir les interactions avec la base de données via Prisma :
  async create(data: CreateExerciseSessionDTO) {
    const { session_id, exercise_id, load, comment, rest_between_exercises, sets } = data
    const createdSessionExercise = await prisma.sessionExercise.create({
        data: {
            load: Number(load),
            session_id: Number(session_id),
            exercise_id: Number(exercise_id),
            rest_between_exercises: Number(rest_between_exercises),
            validated: false,
            comment: comment ?? '',
        },
    });
    return this.getOneWithExerciseAndSets(createdSessionExercise.id)
  }
```

Tout ceci est le chemin (les routes, les méthodes et les requête avec Prisma) qui nous permet dans le front de faire les requêtes à notre API.

En résumé, ce code permet d'ajouter ou de modifier des exercices associés à une session, avec gestion dynamique des séries, des détails comme le poids ou les temps de repos, et enregistrement des données via des appels API. Côté backend, une structure claire avec routes, contrôleurs, et modèles Prisma assure la création ou la mise à jour des exercices en base de données.

f. Éléments de sécurité de l'application

Les éléments de sécurité sont essentiels pour garantir la protection des données et la fiabilité de notre application. Ils couvrent plusieurs aspects cruciaux, notamment la sécurisation des données utilisateur, l'authentification, et la validation des données pour prévenir les abus ou attaques potentielles. Voici une description détaillée des mesures de sécurité mises en place :

1. Sécurisation des données utilisateur
La confidentialité et l'intégrité des données utilisateur sont des priorités absolues. Pour garantir cette protection :

- **Chiffrement des données sensibles** : Toutes les données sensibles, comme les mots de passe, sont hachées avant d'être stockées dans la base de données. Nous utilisons des algorithmes de hachage robustes (comme bcrypt) pour protéger les informations personnelles.

```
const hashedPassword = await bcrypt.hash(password, 10);
```

- **Transmissions sécurisées** : Toutes les communications entre le client et le serveur sont protégées par le protocole HTTPS, garantissant que les données ne peuvent pas être interceptées ou altérées en transit.

- **Permissions et contrôle d'accès** : Les utilisateurs ne peuvent accéder qu'aux données qui leur sont strictement autorisées grâce à une combinaison de mesures techniques dans notre backend. Les modèles Prisma définissent les relations entre les entités et servent de base pour restreindre les données récupérées. Les contrôleurs appliquent des filtres supplémentaires pour s'assurer que seules les données autorisées sont transmises lors des requêtes. Enfin, les middlewares jouent un rôle crucial en interceptant et vérifiant l'authenticité et les permissions des utilisateurs avant d'exécuter une action, garantissant ainsi une gestion rigoureuse des rôles et des droits d'accès.

2. Authentification avec tokens (JWT)
L'authentification des utilisateurs repose sur l'utilisation de JSON Web Tokens (JWT), un standard sécurisé et performant. Voici comment il est mis en œuvre :

- **Génération de tokens** : Lorsqu'un utilisateur se connecte, un token JWT unique est généré, contenant les informations nécessaires à son authentification (par exemple, son identifiant ou son rôle).

```
login: (async (req: Request, res: Response, next: NextFunction) => {
    const { email, password } = req.body;

      // Check if user exists
      const user = await userModel.findUserByEmail(email);

      if (!user) {
        const err = new ApiError(`Invalid email or password`, 401);
        return next(err);
      }

      // Compare password
      const isPasswordValid = await bcrypt.compare(password, user.password);

      if (!isPasswordValid) {
        const err = new ApiError(`Invalid email or password`, 401);
        return next(err);
      }

      const userPayload = {
        id: user.id,
        firstname: user.firstname,
        lastname: user.lastname,
        email: user.email,
        birthdate: user.birthdate,
      };

      // Create the token for the next protected API calls, authenticity of the token will be checked in
security middleware
      const newToken = createOrRefreshToken(user.id);
      res.cookie('token', newToken, { httpOnly: true, expires: new Date(Date.now() +
TOKEN_EXPIRATION_TIME) });

      // Authentication successful
      res.status(200).json({ message: "Logged in successfully", user: userPayload });

  }) as RequestHandler,
```

- **Stockage sécurisé** : Les tokens sont stockés côté client dans des cookies sécurisés.

```
import jwt from 'jsonwebtoken';
const { sign, verify } = jwt;

const JWT_SECRET = process.env.JWT_SECRET as string
//2 Hours token duration
export const TOKEN_EXPIRATION_TIME = 1000 * 60 * 60 * 2

export const createOrRefreshToken = (userId: number) => {
    return sign({ id: userId }, JWT_SECRET, { expiresIn: TOKEN_EXPIRATION_TIME });
}

export const checkToken = (token: string) => {
    return verify(token, JWT_SECRET);
}
```

- **Validation des tokens** : À chaque requête nécessitant une authentification, le token est vérifié par le serveur pour s'assurer de son authenticité et de sa validité.

```
const authMiddleware = async (req: ReqWithUser, res: Response, next: NextFunction): Promise<void> => {
  const token = req.cookies.token;

  if (!token) {
    res.status(401).json({ message: 'Access denied. No token provided.' });
    return;
  }

  try {
    const decoded = checkToken(token)
    if (typeof decoded === "string") {
      res.status(403).json({ message: 'Invalid token' });
      return;
    }

    const user = await userModel.findUniqueWithoutPassword(decoded.id);
    if (!user) {
      res.status(403).json({ message: 'User not found.' });
      return;
    }

    const newToken = createOrRefreshToken(decoded.id)
    res.cookie('token', newToken, { httpOnly: true, expires: new Date(Date.now() +
TOKEN_EXPIRATION_TIME) });
    req.user = user;

    next();
  } catch (error) {
    res.status(403).json({ message: 'Invalid token' });
  }
};

export default authMiddleware;
```

- **Expiration et renouvellement** : Les tokens ont une durée de vie limitée (2h) pour réduire les risques liés à leur compromission. Un mécanisme de rafraîchissement permet aux utilisateurs de rester connectés sans compromettre la sécurité grâce aux cookies.

```
import jwt from 'jsonwebtoken';
const { sign, verify } = jwt;

const JWT_SECRET = process.env.JWT_SECRET as string
//2 Hours token duration
export const TOKEN_EXPIRATION_TIME = 1000 * 60 * 60 * 2

export const createOrRefreshToken = (userId: number) => {
    return sign({ id: userId }, JWT_SECRET, { expiresIn: TOKEN_EXPIRATION_TIME });
}

export const checkToken = (token: string) => {
    return verify(token, JWT_SECRET);
}
```

3. Validation des données avec la bibliothèque Joi

La bibliothèque Joi est utilisée comme un outil central pour valider les données soumises par les utilisateurs, garantissant qu'elles respectent les contraintes définies. Cela permet d'assurer la cohérence des informations et d'éviter toute donnée malformée ou malveillante. Voici comment elle est mise en œuvre :

- **Schémas de validation** : Pour chaque formulaire ou donnée attendue, nous définissons un schéma de validation spécifique. Par exemple :
  - Un champ email doit correspondre au format standard des adresses électroniques (avec Regex*).
  - Les mots de passe doivent respecter des critères de complexité (longueur minimale, présence de caractères spéciaux, etc.) (avec Regex*).
  - Vérification des types : Joi s'assure que chaque donnée correspond bien au type attendu (string, number, boolean, etc.). Cela évite les erreurs inattendues ou les comportements imprévisibles dans le système.

**Regex : Une regex (expression régulière) est un motif de caractères utilisé pour décrire et rechercher des correspondances dans un texte à l'aide d'un moteur d'interprétation.*

```
post: Joi.object({
        //data validation for route register user
        firstname: Joi.string().required(), //first name is required and it must be a character string
        lastname: Joi.string().required(), //last name is required and it must be a character string
        email: Joi.string().pattern(/^([a-zA-Z0-9.\-_]{1,64}@[a-zA-Z0-9]+\.[a-zA-Z]{1,63})
{1,255}$/).required(), //email is required, must be a character string and must respect: - between 1 and
64 characters can include numbers, as well as dashes - must include an @ - must include numbers and/or
letters - must include a period and end with letters between 1 and 64 characters and all this must not
exceed 255 characters
        password: Joi.string().pattern(/^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).
{12,}$/).required(), //password must include: an uppercase letter, a lowercase letter, a special
character, a number and all this must include 8 characters
        repeat_password: Joi.ref('password'), //repeat password
        birthdate: Joi.string().pattern(/^([0-9]{4})-(0[1-9]|1[0-2])-(0[1-9]|[1-2][0-9]|3[0-
1])$/).message('require this format: aaaa-mm-jj').optional(), //birthday is optional but must respect a
specific format as a string: (2012/12/12)
    }).required(),
```

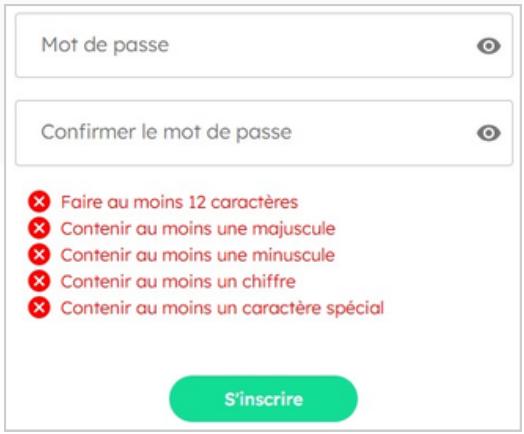
- **Validation côté backend** : Bien que des validations puissent être effectuées dans le frontend, Joi renforce cette validation au niveau du serveur, garantissant que seules des données conformes sont enregistrées dans la base.

```
import { NextFunction, Response, Request } from "express";
const Joi = require('joi');

type DataSource = "body" | "query" | "params" | "headers";

export default function validate(schema: typeof Joi.schema, dataSource: DataSource) {
    return async (req: Request, res: Response, next: NextFunction) => {
        try {
            await schema.validateAsync(req[dataSource]);
            next();
        } catch (err: any) {
            if (err instanceof Joi.ValidationError) {
                // Handling a Joi validation error
                res.status(400).json({
                    message: "Validation failed",
                    details: err.details[0],
                });
            } else {
                // Handling another error
                console.error(err);
                res.status(500).json({
                    message: "An unexpected error occurred.",
                });
            }
        }
    }
};
```

- **Messages d'erreur personnalisés** :

Lorsqu'une validation échoue, nous indiquons des messages d'erreur clairs et explicites à l'utilisateur en relation avec la validation Joi grâce à react-criteria, l'aidant à corriger ses saisies.

```
  const criterias = [
    { text: 'Faire au moins 12 caractères', isValid: isValidLength },
    { text: 'Contenir au moins une majuscule', isValid: hasUppercase },
    { text: 'Contenir au moins une minuscule', isValid: hasLowercase },
    { text: 'Contenir au moins un chiffre', isValid: hasDigit },
    { text: 'Contenir au moins un caractère spécial', isValid: hasSpecialChar },
  ];

{criterias.map((criterion, index) => (
          <Box key={index} display='flex' alignItems='center' sx={{ marginLeft:"10px"}}>
            {criterion.isValid ? (
              <CheckCircle className={styles.iconValid} sx={{fontSize: "1rem"}} />
            ) : (
              <Cancel className={styles.iconError} sx={{fontSize: "1rem"}}/>
            )}
            <Typography
              className={criterion.isValid ? styles.valid : styles.error}
              variant='body1'
              sx={{ fontSize: "1rem"}}
            >
              {criterion.text}
            </Typography>
          </Box>
        ))}
```



g. Présentation du plan de tests

Nous avons structuré les tests dans des dossiers dédiés, situés à proximité des composants ou fonctionnalités qu'ils concernent. Cette organisation facilite leur gestion et leur exécution.

- **Écriture des tests unitaires** : Les composants clés ont été testés de manière isolée pour vérifier qu'ils fonctionnent correctement indépendamment des autres parties du projet.

```
import request from 'supertest';  // Importation de la bibliothèque pour tester les requêtes HTTP
import express from 'express';    // Importation d'Express pour créer l'application serveur
import exerciseController from '../src/controllers/exerciseController'; // Importation du contrôleur
pour gérer les exercices
import exerciseModel from '../src/models/exerciceModel'; // Importation du modèle pour interagir avec la
base de données des exercices

// Mock de la méthode 'getAllExercices' du modèle 'exerciseModel' pour isoler le test
jest.mock('../src/models/exerciceModel');

// Création de l'application Express pour effectuer des tests
const app = express();
app.use(express.json());  // Middleware pour gérer les requêtes JSON
app.get('/exercise', exerciseController.getAllExercices);  // Définition de la route pour obtenir tous
les exercices, utilisant le contrôleur

// Début de la description du groupe de tests pour le contrôleur 'Exercise'
describe('Exercise Controller', () => {

  // Début du test pour vérifier le retour de tous les exercices
  it('should return all exercises', async () => {

    // On simule la réponse de 'exerciseModel.getAllExercices' pour renvoyer une liste d'exercices
fictive
    (exerciseModel.getAllExercices as jest.Mock).mockResolvedValue([
      {
        id: 1,
        name: 'Développé couché à la barre',
        description: 'Allongé sur un banc...'  // Description d'un exercice fictif
      },
      {
        id: 2,
        name: 'Squat avec barre derrière la nuque',
        description: 'Debout, l'écartement des pieds...'  // Description d'un autre exercice fictif
      },
    ]);

    // Envoi d'une requête HTTP GET à l'endpoint '/exercise' pour obtenir la liste des exercices
    const response = await request(app)
      .get('/exercise');  // Envoi de la requête GET

    // Vérification de la réponse de l'API pour s'assurer qu'elle retourne bien un statut HTTP 200 (OK)
    expect(response.status).toBe(200);  // Le code de statut HTTP doit être 200 pour indiquer que la
requête a réussi
    // Vérification que la réponse contient bien les exercices attendus dans le corps de la réponse
    expect(response.body).toEqual([
      {
        id: 1,
        name: 'Développé couché à la barre',
        description: 'Allongé sur un banc...'
      },
      {
        id: 2,
        name: 'Squat avec barre derrière la nuque',
        description: 'Debout, l'écartement des pieds...'
      },
    ]);
  });
});
```

- **Tests d'intégration** : Ces tests permettent de valider les interactions entre plusieurs composants ou fonctionnalités, en simulant des scénarios d'utilisation plus complexes.

```
import request from 'supertest'; // Supertest pour effectuer des requêtes HTTP simulées
import express from 'express'; // Express pour configurer une application de test
import sessionModel from '../src/models/sessionModel'; // Modèle des sessions (mocké pour les tests)
import sessionController from '../src/controllers/sessionController'; // Contrôleur des sessions

// Mock du modèle sessionModel pour éviter les interactions réelles avec la base de données
jest.mock('../src/models/sessionModel');

// Création d'une application Express minimaliste pour tester un endpoint spécifique
const app = express();
app.use(express.json()); // Middleware pour gérer les requêtes avec un corps JSON
app.get('/session/:id', sessionController.getOne); // Route testée, qui utilise la méthode `getOne` du
contrôleur

// Début de la suite de tests pour le Session Controller
describe('Session Controller', () => {
  // Test spécifique pour l'endpoint GET /session/:id
  describe('GET /session/:id', () => {
    it('should return a session when it exists', async () => {
      // Données fictives représentant une session existante
      const mockSession = {
        id: 1,
        title: 'Session jambes',
        session_date: '2024-11-12 08:00:00',
        validated: false,
      };

      // Simulation du comportement de sessionModel.findUnique pour retourner les données fictives
      (sessionModel.findUnique as jest.Mock).mockResolvedValue(mockSession);

      // Effectuer une requête GET sur l'endpoint avec l'ID 1
      const response = await request(app).get('/session/1');

      // Vérifier que la réponse HTTP est un statut 200
      expect(response.status).toBe(200);

      // Vérifier que le corps de la réponse contient les données de la session fictive
      expect(response.body).toEqual(mockSession);

      // Vérifier que la méthode findUnique du modèle a été appelée avec l'ID correct
      expect(sessionModel.findUnique).toHaveBeenCalledWith(1);
    });
  });
});
```

- **Tests des fonctionnalités métier** : Nous avons également testé des cas d'utilisation spécifiques à notre domaine, comme la gestion des exercices dans une session d'entraînement, afin de nous assurer que toutes les règles métier sont respectées.

```
import request from 'supertest';
import express from 'express';
import sessionController from '../src/controllers/sessionController';
import sessionModel from '../src/models/sessionModel';
import exerciseModel from '../src/models/exerciceModel'; //  Modèle des exercices

jest.mock('../src/models/sessionModel');
jest.mock('../src/models/exerciceModel');

const app = express();
app.use(express.json());
app.post('/session/:id/exercise', sessionController.addExerciseToSession);

describe('Session Controller - Add Exercise', () => {

  it('should add an exercise to a session', async () => {
    const mockSession = {
      id: 1,
      title: 'Session jambes',
      session_date: '2024-11-12 08:00:00',
      validated: false,
      exercises: [],
    };

    const mockExercise = {
      id: 1,
      name: 'Squat',
      description: 'Squat with barbell',
    };

    // Mock des appels aux modèles
    (sessionModel.findUnique as jest.Mock).mockResolvedValue(mockSession);
    (exerciseModel.findUnique as jest.Mock).mockResolvedValue(mockExercise);

    // Simuler l'ajout d'un exercice à la session
    const response = await request(app)
      .post('/session/1/exercise')
      .send({ exerciseId: 1, sets: 3, reps: 10 });

    // Vérifier que l'exercice a bien été ajouté à la session
    expect(response.status).toBe(200);
    expect(response.body.exercises).toContainEqual({
      id: 1,
      name: 'Squat',
      sets: 3,
      reps: 10,
    });

    // Vérifier que les règles métier (par exemple, validité du nombre de sets et reps) sont respectées
    expect(response.body.exercises[0].sets).toBeGreaterThan(0);
    expect(response.body.exercises[0].reps).toBeGreaterThan(0);
  });
});
```

- **Exécution régulière des tests** : Les tests sont exécutés automatiquement avant chaque déploiement grâce à une intégration continue (CI). Cela garantit qu'aucune régression n'est introduite dans le projet.

Les tests nous permettent d'identifier et de corriger les erreurs dès les premières étapes du développement, réduisant ainsi le temps consacré au débogage. Grâce à Jest, nous assurons une meilleure qualité de l'application et une expérience utilisateur optimale, même lors de son évolution.

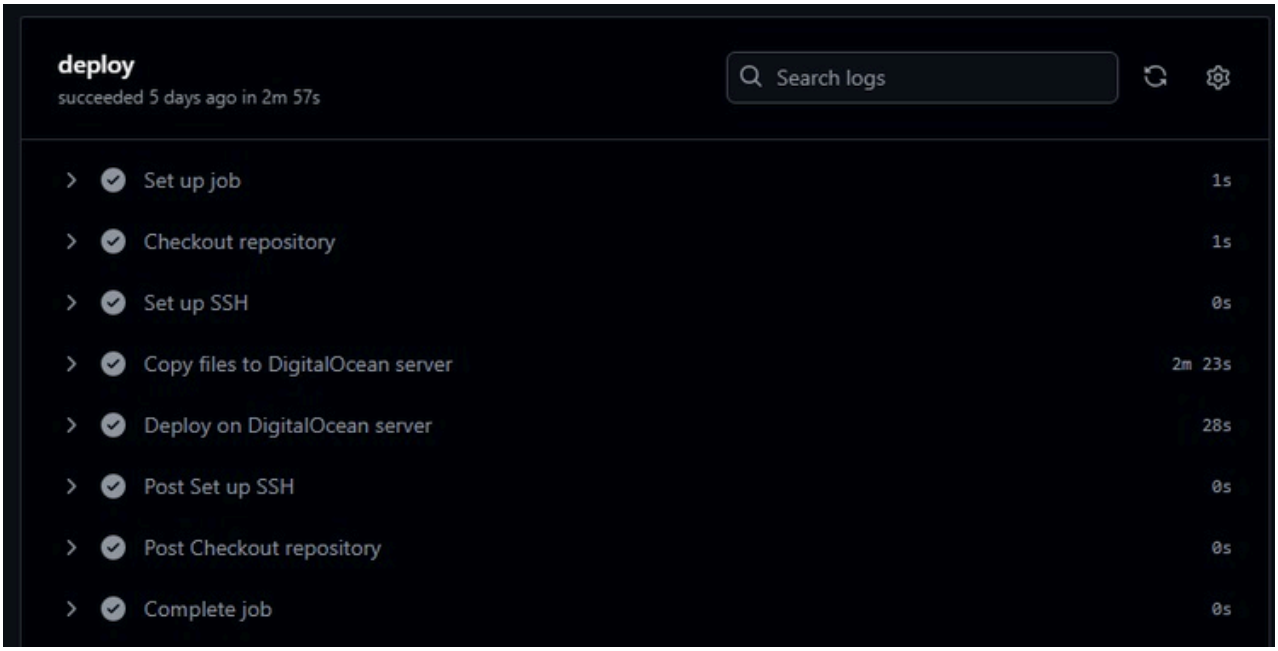
6. Sprint 2 - Recette

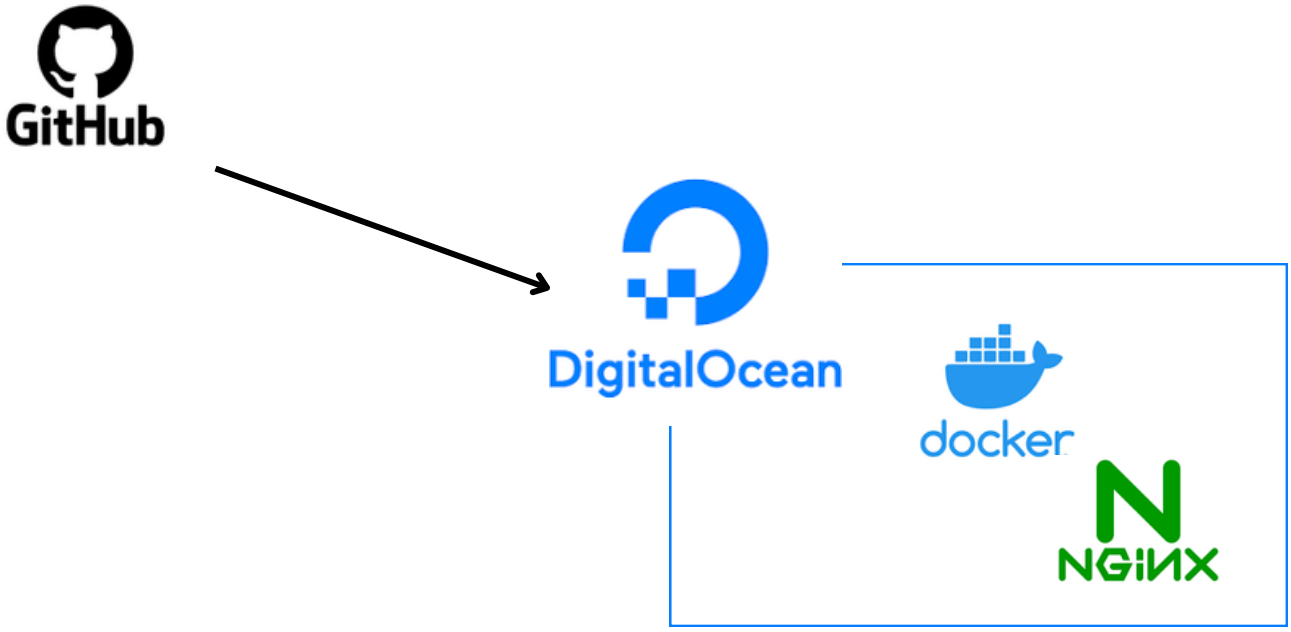
CP10 CP11

a. Déploiement

Pour le déploiement, j'ai mis en place un workflow automatisé avec GitHub Actions qui déploie l'application sur un VPS DigitalOcean à chaque push sur la branche main. Le fichier deploy.yml gère la connexion sécurisée au serveur via SSH, transfère le code source, puis met à jour et redémarre les conteneurs Docker avec docker compose up -d --build. Les identifiants sensibles sont protégés grâce aux secrets GitHub Actions, et un nettoyage des ressources inutilisées est effectué pour optimiser l'espace. Ce processus garantit un déploiement rapide, sécurisé et évolutif, en maintenant l'application toujours à jour avec une gestion efficace des conteneurs. Cette action automatise ainsi la mise ne production sur l'url https://ironpal.fr/

```yaml
name: Deploy to DigitalOcean

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      # 1. Check-out your repository
      - name: Checkout repository
        uses: actions/checkout@v4

      # 2. Set up SSH access to the DigitalOcean server
      - name: Set up SSH
        uses: webfactory/ssh-agent@v0.5.3
        with:
          ssh-private-key: ${{ secrets.SSH_DIGITALOCEAN_PRIVATE_KEY }}

      # 3. Copy project files to the server
      - name: Copy files to DigitalOcean server
        run: |
          scp -o StrictHostKeyChecking=no -r . root@${{ secrets.DIGITALOCEAN_IP }}:/root/ironpal

      # 4. Connect to the server and deploy the application
      - name: Deploy on DigitalOcean server
        run: |
          ssh -o StrictHostKeyChecking=no root@${{ secrets.DIGITALOCEAN_IP }} << 'EOF'
            # Navigate to the project directory
            cd /root/ironpal

            # Stop and remove existing Docker containers (if any)
            docker compose down

            # Build and start the Docker containers
            docker compose up -d --build

            # Clean up unused Docker resources
            docker system prune -f
          EOF
```





J'ai dû configurer le serveur pour assurer le bon fonctionnement du site en installant Docker ainsi que toutes les dépendances nécessaires. Ensuite, j'ai mis en place un serveur NGINX qui joue le rôle de reverse proxy, permettant de gérer les requêtes entrantes et de rediriger le trafic vers les bons services en fonction des besoins.

NGINX facilite également la gestion des certificats SSL pour sécuriser les connexions HTTPS et optimise les performances en servant certains fichiers statiques directement. Cette configuration garantit une meilleure scalabilité, une sécurisation des échanges et une gestion efficace des conteneurs Docker sur le serveur.

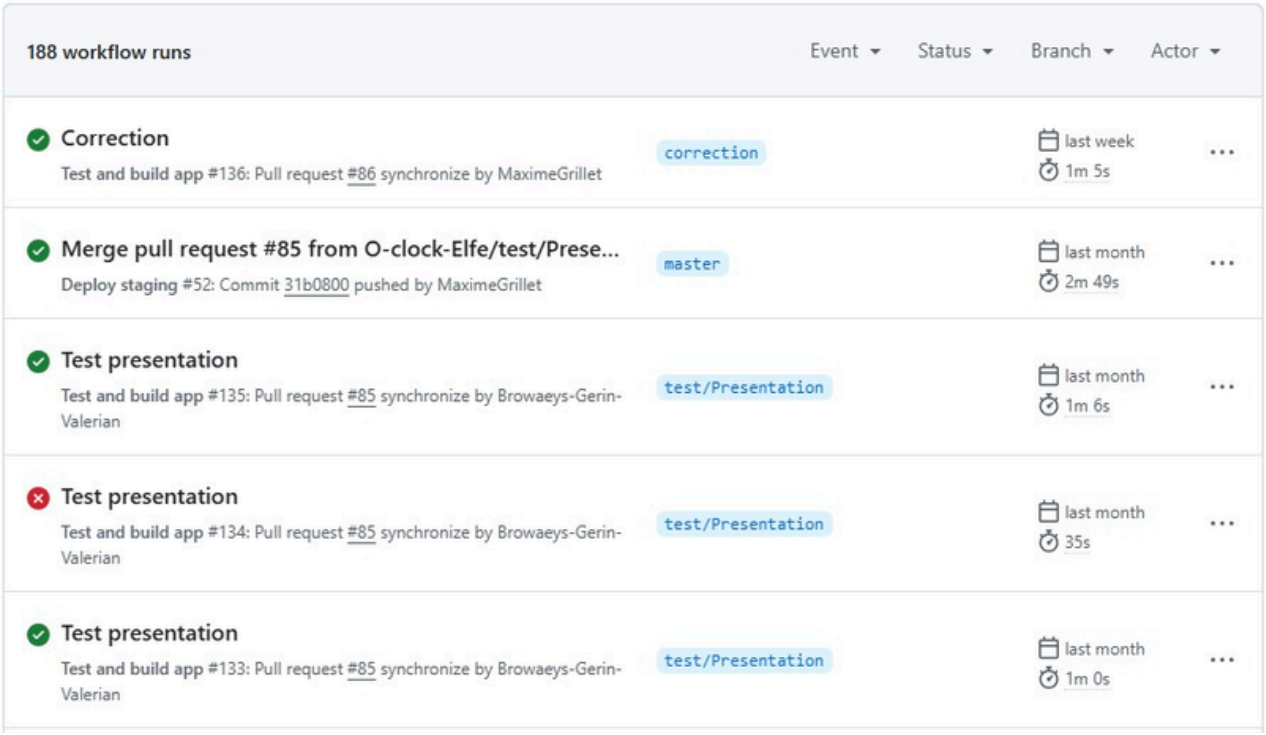
b. Gestion des erreurs

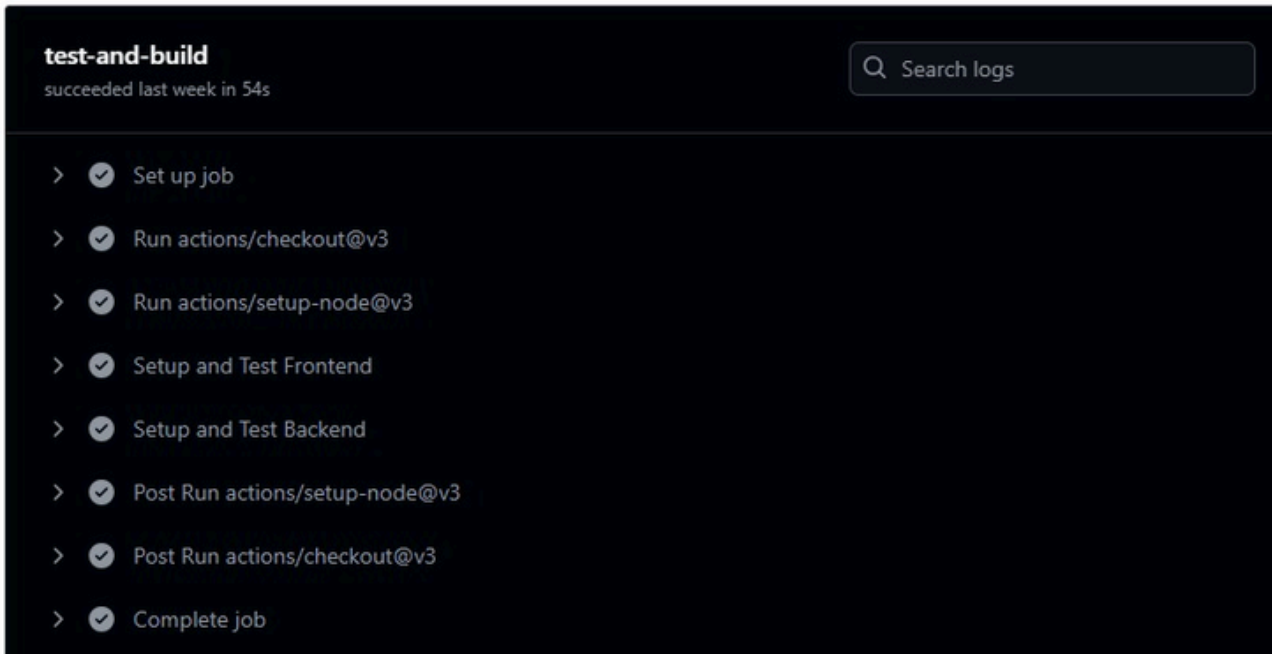
Dans les GitHub Actions, nous avons mis en place un workflow test-and-build.yml qui vérifie que les modifications proposées dans une Pull Request sont fonctionnelles avant leur intégration à la branche en production. Ce workflow, déclenché automatiquement lors d'une PR, récupère d'abord le dépôt avec actions/checkout, configure l'environnement Node.js avec la version 20 via actions/setup-node, puis exécute des étapes distinctes pour le frontend et le backend.

```yaml
name: Test and build app

on: pull_request

jobs:
  test-and-build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '20.x'

      # Installer les dépendances et tester le frontend
      - name: Setup and Test Frontend
        working-directory: ./frontend
        run: |
          npm ci
          npm run test:ci
          npm run build

      # Installer les dépendances et tester le backend
      - name: Setup and Test Backend
        working-directory: ./backend
        run: |
          npm ci
          npm run test:ci
          npm run build
```

Pour le frontend, il installe les dépendances du dossier frontend, exécute les tests (npm run test:ci), vérifie l'absence d'erreurs TypeScript et de code inutilisé, puis construit l'application (npm run build). De manière similaire pour le backend, il installe les dépendances du dossier backend, exécute les tests, vérifie le code TypeScript et supprime le code inutilisé avant de construire le projet.

Ce workflow garantit que le code est testé, optimisé, et construit correctement sur les deux parties du projet, minimisant ainsi les risques d'erreurs ou de mauvaises pratiques en production.


188 workflow runs
Event
Status
Branch
Actor
Correction
Test and build app #136: Pull request #86 synchronize by MaximeGrillet
correction
last week
1m 5s
Merge pull request #85 from O-clock-Elfe/test/Prese...
Deploy staging #52: Commit 31b0800 pushed by MaximeGrillet
master
last month
2m 49s
Test presentation
Test and build app #135: Pull request #85 synchronize by Browaeys-Gerin-Valerian
test/Presentation
last month
1m 6s
Test presentation
Test and build app #134: Pull request #85 synchronize by Browaeys-Gerin-Valerian
test/Presentation
last month
35s
Test presentation
Test and build app #133: Pull request #85 synchronize by Browaeys-Gerin-Valerian
test/Presentation
last month
1m 0s


test-and-build
succeeded last week in 54s
Search logs
Set up job
Run actions/checkout@v3
Run actions/setup-node@v3
Setup and Test Frontend
Setup and Test Backend
Post Run actions/setup-node@v3
Post Run actions/checkout@v3
Complete job

7. Veille technique

a. MUI

Tout au long du projet, j'ai dû me documenter et me former sur les technologies utilisées, en particulier sur le frontend. L'une des principales librairies que j'ai adoptées est MUI (Material UI), qui permet de créer une interface utilisateur moderne et cohérente.

J'ai créé un thème unique et ajustable grâce au fichier theme.ts, où j'ai configuré les couleurs, typographies, espacements et autres paramètres visuels afin d'assurer une uniformité dans l'application. J'ai utilisé les composants MUI comme le Grid pour gérer le responsive design et adapter l'interface à différents formats d'écran. En intégrant des éléments tels que les icônes MUI et les boutons, j'ai pu garantir un code plus propre, concis et optimisé, tout en respectant les bonnes pratiques de développement.

b. Hooks React

Dans le cadre de l'application, j'ai exploré en profondeur les hooks de React, qui sont essentiels pour rendre l'interface réactive et performante. J'ai particulièrement utilisé useState pour gérer l'état local des composants, useEffect pour gérer les effets de bord et les appels API de manière optimisée, et useMemo pour mémoriser les valeurs calculées afin de réduire les recalculs inutiles. Ces hooks m'ont permis de rendre l'interface plus fluide et de garantir que les composants se mettent à jour de manière efficace, en réduisant le nombre de rendus et en améliorant la réactivité de l'application.

c. Optimisation et performances (RGAA & SEO)

L'optimisation des performances a été un point clé tout au long du projet, avec une attention particulière portée à l'adoption des meilleures pratiques de développement, tout en veillant à respecter les critères RGAA et SEO. L'objectif était de garantir un site rapide et efficace, en optimisant la structure des composants et en assurant une bonne organisation du code pour une meilleure performance globale.

Cependant, nous sommes conscients que les technologies et les pratiques d'optimisation évoluent continuellement, et il sera nécessaire de maintenir une vigilance constante pour mettre à jour le code afin de suivre les nouvelles recommandations et garantir une performance optimale à long terme. L'optimisation des performances est donc un processus continu, nécessitant des ajustements réguliers et une adaptation aux nouvelles normes.

8. Difficultés rencontrées

a. Transfert de données asynchrones

L'utilisation de TypeScript permet d'améliorer le code JavaScript en le rendant plus clair et structuré. Cependant, pour que cela fonctionne bien, le framework utilisé et l'architecture du code doivent être bien pensés, notamment dans la gestion des fonctions asynchrones.

N'ayant pas encore une grande maîtrise du framework, j'ai parfois eu des difficultés avec les fonctions asynchrones, ce qui a causé des problèmes d'organisation des données et des traitements. Les fonctions ne s'exécutaient pas toujours dans le bon ordre, ce qui créait des bugs et des comportements inattendus. Pour éviter cela, il a fallu revoir la structure du projet et organiser le code de manière plus logique afin d'optimiser les échanges de données, tout en respectant les délais imposés.

b. MUI

Nous avons choisi d'utiliser Material-UI (MUI) pour gagner du temps grâce à ses nombreux composants déjà prêts à l'emploi. Toutefois, son intégration n'a pas été immédiate, car je ne l'avais jamais utilisé auparavant. Même si MUI propose un système de thèmes permettant de personnaliser facilement les styles, il a fallu du temps pour bien l'adapter à la charte graphique de l'application. Certains ajustements ont été compliqués à mettre en place, et certaines parties de la documentation faisaient référence à des composants qui n'étaient plus à jour. Heureusement, la documentation restait globalement claire, ce qui m'a permis de comprendre rapidement le fonctionnement de la librairie et d'optimiser son utilisation.

c. Prisma

Pour le backend, nous avons utilisé Prisma, un ORM très puissant qui simplifie la gestion des bases de données. Cependant, au début, j'ai eu du mal à comprendre comment il fonctionnait, notamment au niveau des routes qu'il génère automatiquement. Contrairement à d'autres outils plus classiques, Prisma gère certaines parties du backend de manière interne, ce qui a rendu son apprentissage un peu plus complexe.

J'ai dû approfondir mes connaissances, apprendre à créer des routes, gérer les modèles de données et utiliser les middlewares pour sécuriser les requêtes. Finalement, cette phase d'apprentissage m'a permis de comprendre correctement le fonctionnement du backend et des interactions entre l'API et la base de données.

d. DevOps / Déploiement

Le déploiement de l'application a été l'un des aspects compliqués du projet, principalement parce que je manquais d'expérience dans ce domaine. De plus, ce sujet n'avait pas été abordé en détail pendant ma formation, ce qui m'a obligé à faire de nombreuses recherches pour comprendre comment héberger et faire tourner l'application en production.

J'ai exploré différentes solutions comme Heroku, AWS et DigitalOcean afin de trouver la meilleure option pour héberger un serveur optimisé et gérer Docker ainsi que ses conteneurs. La configuration du serveur, la mise en place des conteneurs et la gestion des accès, la config NGINX ont nécessité plusieurs ajustements avant d'atteindre un résultat stable. Malgré ces défis, cette expérience m'a permis d'acquérir de solides connaissances sur le déploiement et l'hébergement d'une application web. Aujourd'hui, grâce à GitHub Actions, je sais automatiser la mise en production d'une app.

9. Pour le futur

a. User stories v2

En tant que	Je veux
Utilisateur	Pouvoir ajouter des séances en favoris
Utilisateur	Pouvoir supprimer mon compte
Utilisateur	Pouvoir mettre un mode opaque sur l'application
Utilisateur	Pouvoir se connecter en SSO
Utilisateur	Pouvoir consulter une vidéo d'explication en plus de la description de l'exercice
Utilisateur	Ajouter une heure de démarrage a ma séance lors sa création
Utilisateur	Avoir un rappel X minutes avant ma séance
Utilisateur	Avoir ma séance pré remplie lorsque je la crée basé sur les données de la séance de la semaine passée
Utilisateur	Pouvoir avoir une écran de dashboard ou j'ai la visu sur ma progression sur les différents exercices
Utilisateur	Pouvoir ajouter des personnes en amis sur l'application
Utilisateur	Pouvoir discuter avec mes amis sur l'application
Utilisateur	Avoir la possibilité d'activer des notifications lorsque l'un de mes amis progresse sur une séance et/ou un exercice
Utilisateur	Gagner des badges en fonction des objectifs que j'ai atteint
Utilisateur	Pouvoir comparer mes badges avec mes amis
Coach	Pouvoir avoir la visu sur mes coachés
Coach	Avoir la visu sur les séances de mes coachés
Coach	Pouvoir ajouter des séances a mes coachés
Coach	Pouvoir suivre la progression de mes coachés
Coach	Pouvoir discuter avec mes coachés

b. MVP v2

Comme évoqué au début du dossier dans les ouvertures du projet, il faudra mettre en place pour le futur ces points d'évolutions de l'app :

- Intégration de fonctionnalités sociales
- Suivi des performances
- Programmes d'entraînement personnalisés
- Ajout d'un mode "coach personnel"
- Fonctionnalités de gamification
- Partenariats avec des marques de fitness
- Statistiques basiques
- Faciliter les échanges
- Notifications

c. Refacto / optimisation

La refactorisation du code est une étape essentielle pour tout développeur souhaitant garantir un code propre, lisible et maintenable. Bien que nous ayons optimisé le projet dans les délais impartis, certaines parties du code comme les fonctions JavaScript et la gestion asynchrone, pourraient encore être améliorées. Ce travail d'optimisation permettra non seulement d'améliorer les performances de l'application, mais aussi de réduire la taille des fichiers chargés, contribuant ainsi à une approche plus écoresponsable et à une meilleure expérience utilisateur.

d. Tests et avis utilisateurs

Avant de mettre le site officiellement en ligne, nous pourrons, sur une URL de préprod, réaliser des tests utilisateurs et demander à des personnes de confiance, apte à utiliser l'app, de nous faire des retours sur leur expérience utilisateur, et aussi noter leurs avis pour que lors de sa sortie, Ironpal réponde aux meilleurs exigences

e. Indexation et référencement

En complément des bonnes pratiques d'accessibilité (RGAA), l'optimisation du référencement naturel (SEO) devra être une priorité. React étant une technologie SPA (Single Page Application) qui génère son contenu côté client, il n'est pas optimisé pour le référencement sur les moteurs de recherche. Pour améliorer la visibilité de l'application, il faudra sûrement envisager l'intégration de Next.js, qui permet un rendu côté serveur (SSR), et/ou configurer un reverse proxy avec NGINX afin de mieux structurer et indexer le site sur les moteurs de recherche.

f. Application mobile

L'objectif ultime serait d'adapter cette web app React en React Native afin de pouvoir la développer en tant qu'application mobile. Cette transition permettrait de rendre l'application accessible sur les plateformes iOS et Android, tout en réutilisant une grande partie du code existant. Ce projet pourrait inclure des fonctionnalités spécifiques aux appareils mobiles, comme les notifications push, l'accès aux capteurs ou la gestion offline, afin d'offrir une expérience fluide et adaptée à un usage mobile.

De plus, grâce à Docker, nos conteneurs étant bien séparés, la base de données en ligne pourrait servir à la fois l'application mobile et le site web. Cette architecture permettrait de maintenir une gestion centralisée des données, offrant ainsi une scalabilité optimale grâce à l'API, accessible sur toute les plateformes sans duplications ou incohérences.

10. Conclusion / Bilan

Ironpal, c'est le résultat de plus d'une année de formation, avec un large panel de compétences évoquées et mises en place tout au long de l'année, que vous retrouverez dans ce dossier. C'est aussi le résultat d'un travail de groupe, avec des éléments complémentaires qui ont su dans un délai imparti lancer les débuts d'Ironpal.

En travaillant sur Ironpal, j'ai appris à concevoir et développer une application sécurisée, en installant et configurant les environnements nécessaires, tout en prenant soin de respecter les meilleures pratiques de sécurité. J'ai également conçu des interfaces utilisateur intuitives et des composants métiers performants, tout en mettant en place une base de données relationnelle adaptée au projet.

Ce parcours m'a permis de maîtriser des outils et frameworks essentiels, tels que React, React Router et Prisma, tout en utilisant des langages comme TypeScript pour garantir la robustesse du code JavaScript, qui est l'un des langages les plus utilisés et les plus performants aujourd'hui. J'ai ainsi pu aborder l'ensemble du processus, des tests jusqu'au déploiement, en passant par la gestion de la partie DevOps.

Parallèlement, j'ai acquis des compétences en gestion de projet, en communication efficace au sein d'une équipe ainsi qu'en méthode d'organisation et de travail.

Aujourd'hui, en plus de l'utiliser pour mes séances de sport, je souhaite continuer à développer Ironpal, en l'enrichissant et en l'adaptant aux besoins de ses utilisateurs, avec l'ambition de faire évoluer l'application pour qu'elle devienne un outil performant, évolutif et largement adopté par le public cible.

11. Annexes

Repo du projet :
https://github.com/O-clock-Elfe/projet-4-ironpal

URL en ligne :
https://ironpal.fr/

readme.md / Documentation du projet :

```
## Ironpal overview:

Ironpal is a web application for weight training that allows users to plan, track, and customize their
workouts.

It focuses on organizing effective training sessions by offering a vast library of exercises, which can
be grouped into sessions and scheduled over a week.

The goal is to provide a simple and intuitive tool to help users optimize their workouts while offering
features to track their progress and adjust their routines.


## Set up:

1. At the root of the project copy `docker.env.example` to `docker.env` and add your local values.
2. Go to /backend directory and copy `.env.example` to `.env` and add your local values.
3. Return to the root of the project and execute the following command `docker compose up`

### Docker Configuration

- To start all containers: `docker compose up` or `docker compose up -d`
- To stop all containers: `docker compose down`

The docker-compose.yml file contains configurations for the following services:

- backend: The main server of the application
- database: The PostgreSQL database
- adminer: The Adminer web interface for database management

### Swagger

Once container are start to check API documentation open your browser and go to the URL:
http://localhost:3000/api-docs/

### Connecting to Adminer

Once the database container has started, open a browser and go to the URL: http://localhost:8080 You
will see the Adminer interface.

Fill in the fields with the following information:

- System: PostgreSQL
- Server: database
- Username: the username you specified in docker.env
- Password: the password you specified in docker.env
- Database: the value specified in docker.env

Click "Connect."
```

Once connected via Adminer, you can:

Browse tables and their data
Execute SQL queries
Create, modify, or delete tables and data

Connecting to Prisma

If you feel the need to check data in prisma studio you have to follow these steps:

1. Use this commande to `docker compose exec backend sh`
2. Once in the container's shell exec `npx prisma studio` command
3. Open prisma studio on `http://localhost:5555/`

Naming Conventions

Commits

We use [Conventional Commits](https://www.conventionalcommits.org/en/v1.0.0-beta.2/). Each commit message must be formatted with the following convention:

```md
<type>[optional scope]: <description>

[optional body]

[optional footer]
```

Examples:

- feat: add contact form
- fix(lang): fix language typos
- chore(templating): add pull request template

Branches

We use something similar to our commit naming convention :

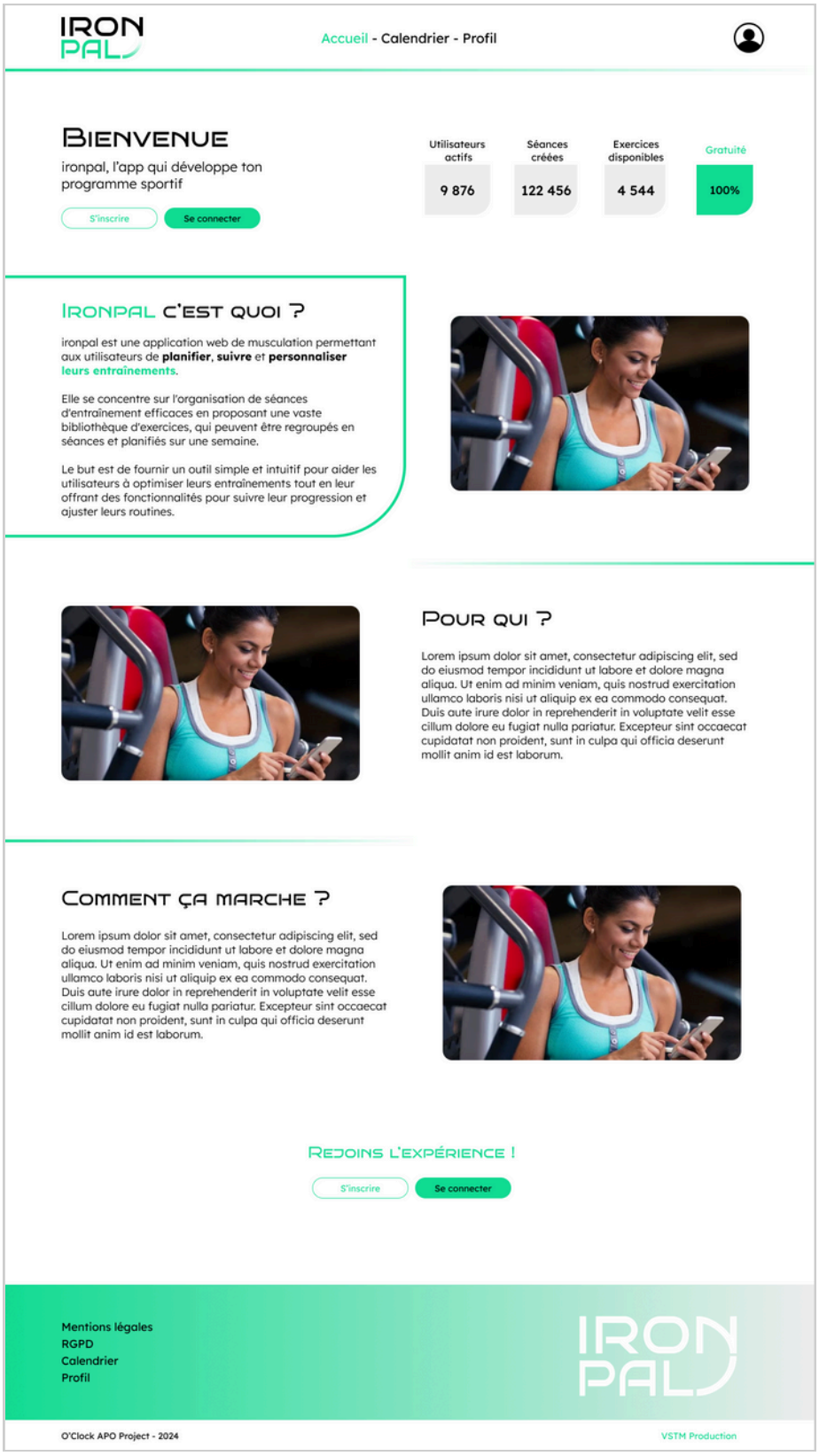
```md
<type>/<description>
```

Examples:

- feat/contact-form
- fix/language-typos
- chore/pr-template

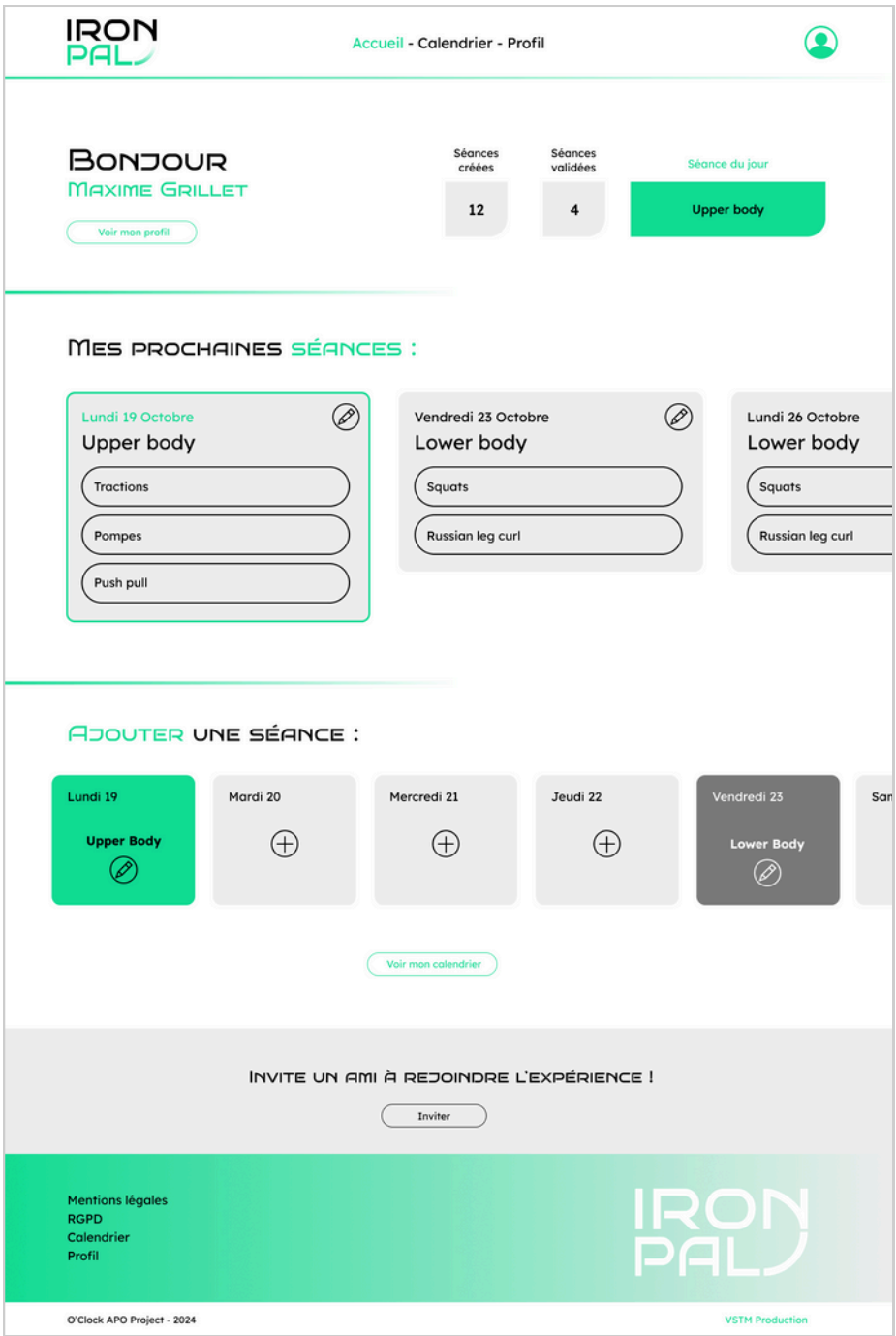
Maquette

Home disconnected



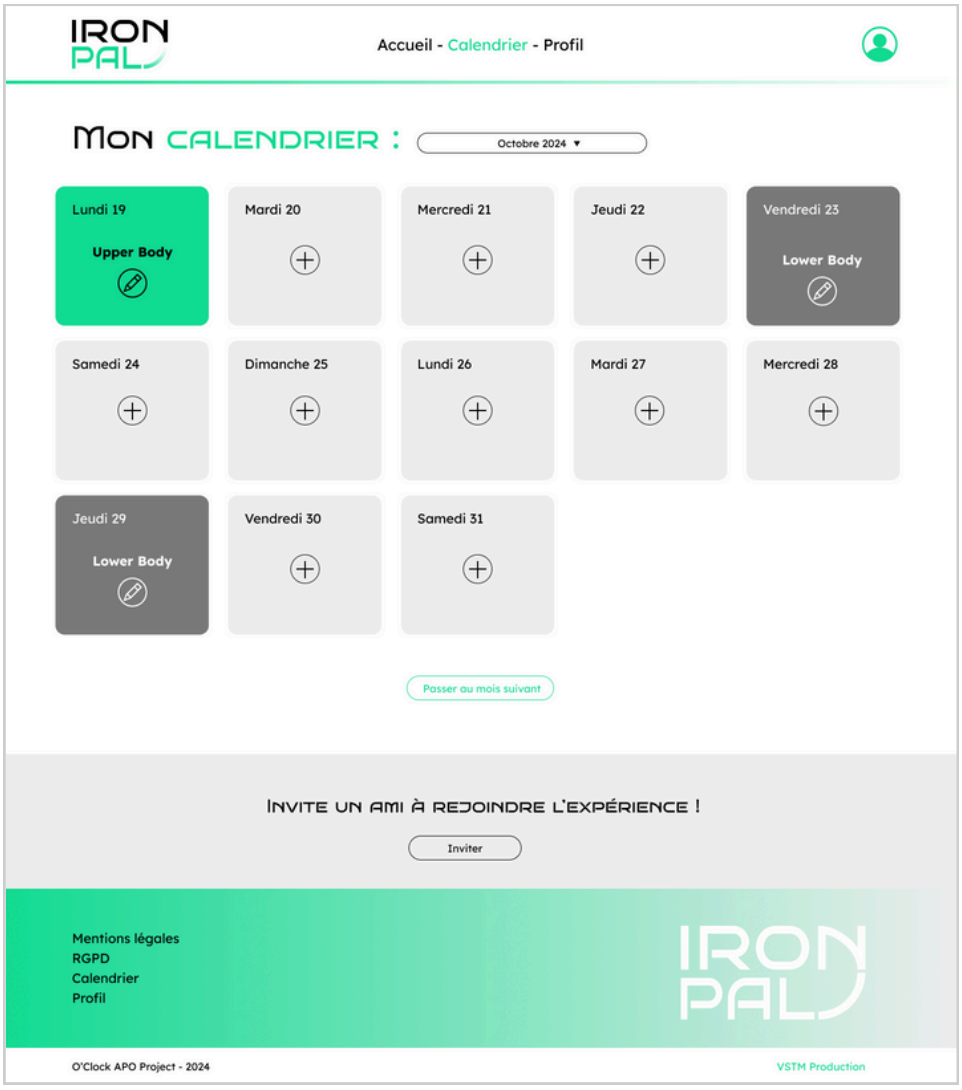
Maquette

Home connected

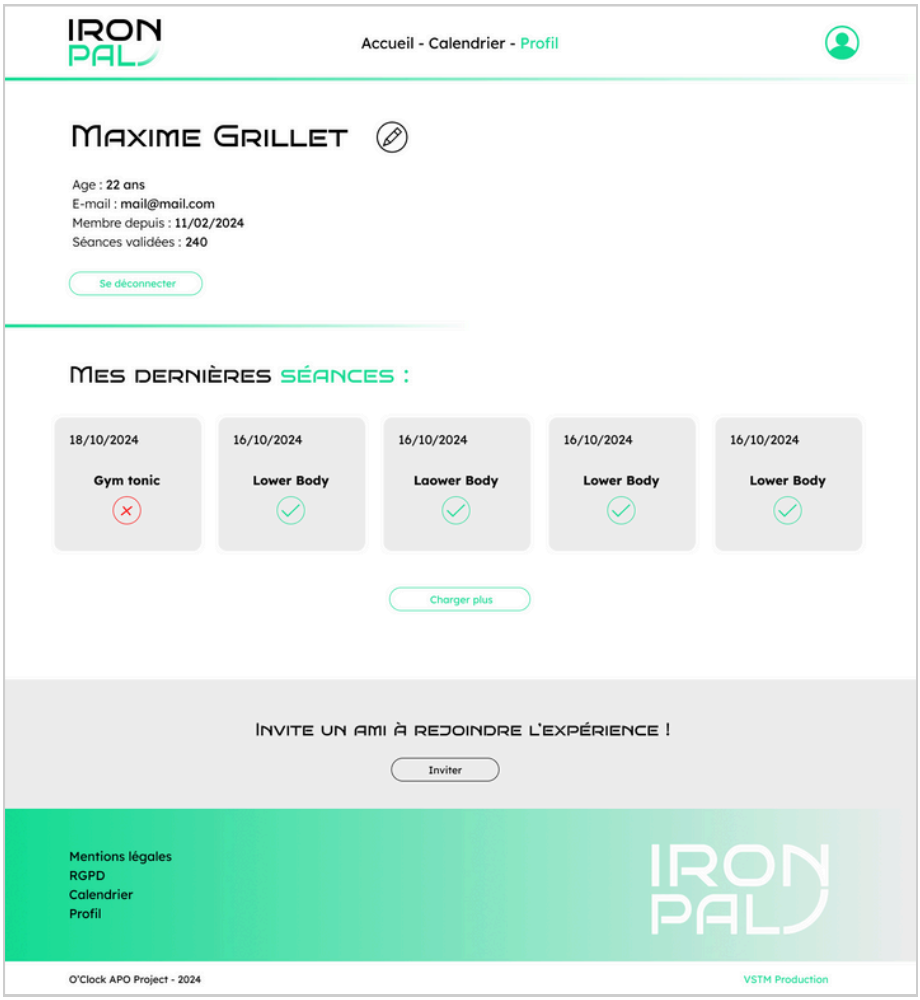


Maquette

Calendar

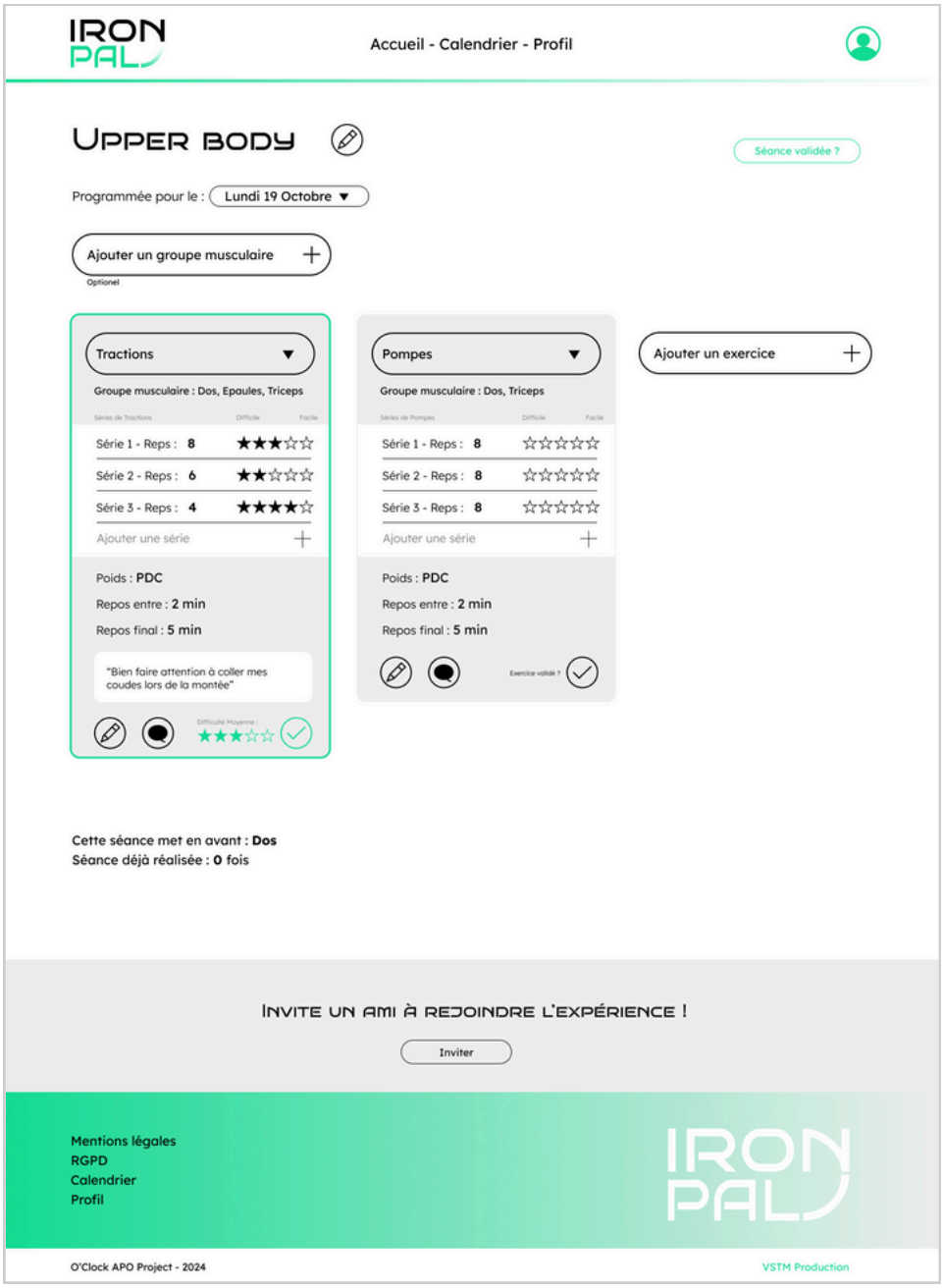


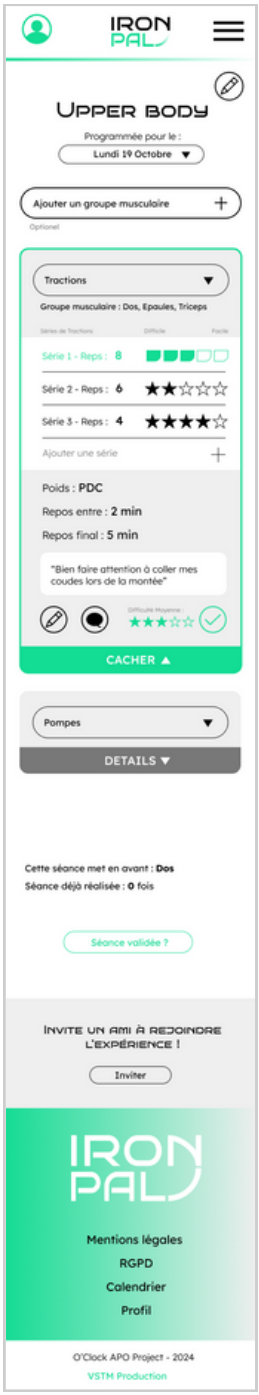

IRON PAL
MAXIME GRILLET
Age : 22 ans
E-mail : mail@mail.com
Membre depuis : 11/02/2024
Séances validées : 240
Se déconnecter
MES DERNIÈRES SÉANCES :
18/10/2024
Gym tonic
16/10/2024
Lower body
16/10/2024
Lower body
16/10/2024
Lower body
16/10/2024
Lower body
16/10/2024
Lower body
Charger plus
INVITE UN AMI À REJOINDRE L'EXPÉRIENCE !
Inviter
IRON PAL
Mentions légales
RGPD
Calendrier
Profil
O'Clock APO Project - 2024
VSTM Production


IRON PAL
Accueil - Calendrier - Profil
MAXIME GRILLET
Age : 22 ans
E-mail : mail@mail.com
Membre depuis : 11/02/2024
Séances validées : 240
Se déconnecter
MES DERNIÈRES SÉANCES :
18/10/2024
Gym tonic
16/10/2024
Lower Body
16/10/2024
Laower Body
16/10/2024
Lower Body
16/10/2024
Lower Body
Charger plus
INVITE UN AMI À REJOINDRE L'EXPÉRIENCE !
Inviter
Mentions légales
RGPD
Calendrier
Profil
IRON PAL
O'Clock APO Project - 2024
VSTM Production

Maquette

Session





Dictionnaire de données

Table user

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique de l'utilisateur
firstname	varchar	NOT NULL	Prénom de l'utilisateur
lastname	varchar	NOT NULL	Nom de l'utilisateur
email	varchar	NOT NULL	Adresse email de l'utilisateur
password	varchar	NOT NULL	Mot de passe de l'utilisateur
age	int	NULLABLE	Age de l'utilisateur

Table session

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique de la session
user_id	int	FOREIGN KEY REFERENCES user(id) ON DELETE CASCADE	identifiant de l'utilisateur
title	varchar	NOT NULL	Titre de la session
session_date	date	NOT NULL	Date de la session
validated	boolean	DEFAULT FALSE	Indique si la session a été validée
muscle_group_id	int	FOREIGN KEY REFERENCES muscle_group(id) NULLABLE	Groupe musculaire associé à la session

Table session_exercice

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique de l'exercice
session_id	int	FOREIGN KEY REFERENCES session(id) ON DELETE CASCADE	identifiant de la session
exercise_id	int	FOREIGN KEY REFERENCES exercice(id)	identifiant de l'exercice
load	decimal	NOT NULL	Charge utilisée pour l'exercice
rest_between_exercices	interval	NOT NULL	Temps de repos entre les exercices
validated	boolean	DEFAULT FALSE	Indique si l'exercice a été validé
coment	string	NULLABLE	commentaire sur l'exercice

Table set

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique de la série
session_exercice_id	int	FOREIGN KEY REFERENCES Session_exercice(id) ON DELETE CASCADE	identifiant de l'exercice auquel appartient la série
number_of_repetitions	int	NOT NULL	Nombre de répétitions dans la série
difficulty	int	CHECK (difficulty BETWEEN 1 AND 5)	Niveau de difficulté (1 = facile, 5 = difficile)
rest_between_sets	interval	NOT NULL	Temps de repos entre les séries

Table exercice

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique de exercice
name	varchar	NOT NULL	Nom de l'exercice
description	varchar	NULLABLE	Description de l'exercice(facultatif)

Table exercice_muscle_group

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique des exercices rattachés aux groupes musculaires
exercice_id	int	FOREIGN KEY REFERENCES exercice(id)	identifiant unique de l'exercice
muscle_group_id	int	FOREIGN KEY REFERENCES muscle_group(id)	identifiant unique du groupe musculaire

Table muscle_group

Champ	Type	Spécificités	Description
id	int	GENERATED ALWAYS AS IDENTITY PRIMARY KEY	identifiant unique du groupe musculaire
name	varchar	NOT NULL	Nom du groupe musculaire