**Analyse du besoin**

Objectif(s) : L'objectif de cette application est de permettre aux utilisateurs de réserver des séances de sport dans différents établissements d'une chaîne de salles de sport. Pour cela, l'application doit fournir les fonctionnalités suivantes :

1. Authentification :
   * Les utilisateurs doivent pouvoir créer un compte et se connecter à l'application.
2. Consultation des séances de sport :
   * Les utilisateurs doivent pouvoir consulter les séances de sport disponibles.
   * Les séances de sport doivent être affichées par établissement, thème, date et heure.
   * Les utilisateurs doivent pouvoir filtrer les séances de sport en fonction de critères tels que la localisation, le thème et la date.
3. Réservation de séances de sport :
   * Les utilisateurs authentifiés doivent pouvoir réserver une séance de sport disponible.
   * En cas de conflit de réservation pour un même créneau horaire, l'utilisateur doit pouvoir choisir parmi les établissements disponibles.
   * Les utilisateurs doivent pouvoir visualiser et gérer leurs réservations existantes.

Décomposition de l'application :

Modèles de données :

* User : représente les données d'un utilisateur (id, nom d'utilisateur, email, mot de passe, etc.).
* Session : représente les données d'une séance de sport (id, établissement, thème, date, heure, etc.).
* Reservation : représente les données d'une réservation (id, id de la séance réservée, id de l'utilisateur, établissement, thème, date, heure, etc.).

Découpage en modules :

* Authentication Module : contiendra les fonctionnalités d'inscription, de connexion et de déconnexion des utilisateurs.
* Sessions Module : gérera l'affichage des séances de sport, le filtrage et la recherche.
* Reservations Module : permettra aux utilisateurs authentifiés d'effectuer des réservations et de gérer leurs réservations existantes.

Routes de l'application :

* /login : page de connexion
* /signup : page d'inscription
* /sessions : page de consultation des séances de sport
* /reservations : page de réservation de séances de sport et gestion des réservations existantes

Hiérarchie de composants (Smart/Dumb Pattern) :

* Smart Components : ces composants sont responsables de la logique métier, de la récupération des données et de la communication avec les services. Ils contiennent généralement peu de logique d'affichage.
  + LoginComponent (Smart) : gère la logique de connexion.
  + SignupComponent (Smart) : gère la logique d'inscription.
  + SessionListComponent (Smart) : gère la logique d'affichage et de filtrage des séances de sport.
  + ReservationComponent (Smart) : gère la logique de réservation de séances de sport.
  + MyReservationsComponent (Smart) : gère la logique d'affichage et de gestion des réservations existantes.
* Dumb Components : ces composants sont principalement responsables de l'affichage des données et de la réception des actions de l'utilisateur. Ils ne contiennent généralement pas de logique métier complexe.
  + SessionCardComponent (Dumb) : affiche les détails d'une séance de sport dans une carte.
  + ReservationCardComponent (Dumb) : affiche les détails d'une réservation dans une carte.
* En adoptant cette décomposition et cette hiérarchie de composants, notre application sera organisée de manière cohérente, offrant ainsi une meilleure maintenabilité et extensibilité. Chaque module aura une responsabilité bien définie, ce qui permettra de diviser les tâches en fonctionnalités distinctes, favorisant ainsi la réutilisation des composants. Les Smart Components se concentreront sur la gestion de la logique métier, tandis que les Dumb Components se concentreront sur la présentation et l'affichage des données.
* L'utilisation de services pour gérer les opérations liées aux données, comme l'authentification et la récupération des séances de sport et des réservations, contribuera à garder une séparation claire entre la couche de présentation et la couche de données. Cela permettra également de réduire la complexité des composants en déléguant la gestion des requêtes et des opérations asynchrones aux services.
* En adoptant cette approche modulaire et en suivant le Smart/Dumb Pattern, notre application bénéficiera d'une structure claire, facilitant ainsi la collaboration entre les développeurs et la maintenance du code. De plus, cette conception facilitera également la mise en œuvre de nouvelles fonctionnalités et l'ajout de nouvelles pages sans perturber l'ensemble de l'application.
* En somme, cette approche de conception bien réfléchie nous permettra de créer une application Angular robuste et conviviale, répondant aux besoins de nos utilisateurs et offrant une expérience utilisateur optimale tout en garantissant une base solide pour un développement futur sans encombre.