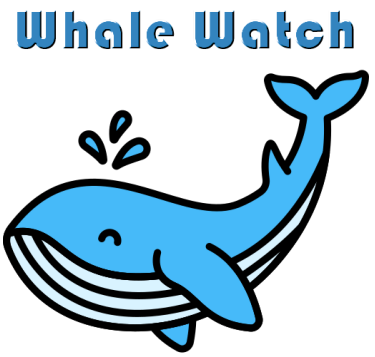


Rapport du projet React Native

Introduction

Vue d'ensemble de l'application

L'application mobile est un compteur de calories et un planificateur de repas. Les utilisateurs pourront mettre en place des objectifs pour leur santé en choisissant de la nourriture parmis une base de données. Cette base de données garde une trace des calories gagnées à chaque repas.

Objectifs du projet

Les objectifs de ce projet sont pour nous de mettre en pratique les notions apprises en TD pour faire une première application. Pour l'utilisateur, l'objectif est de perdre du poids.

Configuration du projet

- Cloner le répertoire git

```
git clone https://github.com/catherinegaochangdffeef/React_Projet.git
```

- Ouvrir un terminal
- Changer de répertoire

```
cd my-app
```

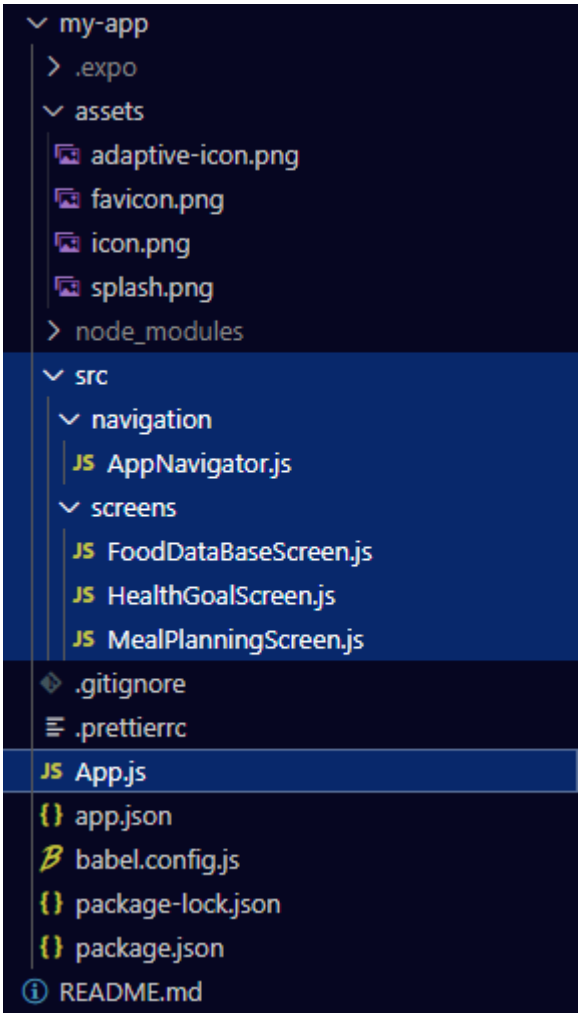
- Installer les dépendances

```
npm install @react-native-material/core
npm install @react-native-picker/picker
npm install @rneui/themed @rneui/base
npx expo install @react-native-async-storage/async-storage@1.17.11
```

- Lancer expo

```
npx expo start
```

Architecture



AppNavigator.js

La vue retourne un composant NavigationContainer qui enveloppe les écrans de l'application. À l'intérieur de NavigationContainer, il y a un composant Tab.Navigator qui gère la navigation par onglets. Les onglets sont définis avec les noms "HealthGoal", "FoodDataBase" et "MealPlanning".

Chaque onglet est représenté par un composant Tab.Screen. Le premier onglet est associé à l'écran HealthGoalScreen, le deuxième onglet est associé à l'écran FoodDataBaseScreen, et le troisième onglet est associé à l'écran MealPlanningScreen.

L'écran FoodDataBaseScreen est rendu en utilisant une fonction fléchée avec des props ({...props}) et les variables d'état mealPlan et setMealPlan sont

passées en tant que props supplémentaires. Cela permet à l'écran FoodDataBaseScreen d'accéder à mealPlan et de le mettre à jour via setMealPlan.

L'écran MealPlanningScreen est rendu de manière similaire, avec les props et mealPlan transmis à travers la fonction fléchée.

En résumé, on configure la navigation de l'application mobile en utilisant react-navigation avec trois onglets. Il utilise également le hook useState pour gérer l'état du plan de repas (mealPlan) et le partager entre les écrans associés aux onglets.

HealthGoalScreen.js

Ce composant représente l'écran où les utilisateurs peuvent définir leurs objectifs de santé, tels que leur âge, leur genre, leur taille, leur poids, leur niveau d'activité et leur objectif de santé. Il utilise plusieurs hooks useState pour gérer l'état des différentes variables.

Le composant commence par déclarer plusieurs variables d'état à l'aide de useState. Ces variables d'état comprennent age, gender, height, weight, activityLevel, healthGoal et bMR. Chacune de ces variables d'état est initialisée avec une valeur par défaut.

Le composant définit également une fonction calculateBMR qui calcule le métabolisme de base (BMR) de l'utilisateur en fonction des informations fournies, telles que l'âge, le genre, la taille, le poids et le niveau d'activité. La formule de calcul utilisée est l'équation de Harris-Benedict. Le BMR calculé est ensuite ajusté en fonction de l'objectif de santé spécifié.

La fonction handleSave est appelée lorsque l'utilisateur appuie sur le bouton "Save" pour enregistrer ses paramètres. Elle vérifie si toutes les variables d'état nécessaires sont remplies et, le cas échéant, affiche une alerte d'erreur. Sinon, elle appelle la fonction calculateBMR pour effectuer le calcul du BMR.

Le rendu du composant HealthGoalScreen consiste en un conteneur View qui enveloppe tout le contenu. L'intérieur du conteneur est une ScrollView avec un arrière-plan de couleur verte. Dans cette ScrollView, les différents éléments d'interface utilisateur sont affichés, tels que les champs de saisie de l'âge, du genre, de la taille et du poids, ainsi que les menus déroulants pour le niveau d'activité et l'objectif de santé.

Lorsque l'utilisateur modifie les valeurs des champs de saisie ou des menus déroulants, les fonctions setAge, setGender, setHeight, setWeight, setActivityLevel et setHealthGoal sont appelées pour mettre à jour les variables d'état correspondantes.

Enfin, il y a un bouton "Save" qui déclenche la fonction handleSave lorsque l'utilisateur appuie dessus. On définit aussi les styles CSS pour le composant tout en bas, tels que la mise en page, les marges, les couleurs, etc.

Ce composant est exporté en tant que composant par défaut, ce qui signifie qu'il peut être importé et utilisé dans d'autres parties de l'application.

FoodDataBaseScreen.js

On a défini une variable d'état appelée mealPlan. Cette variable d'état est initialisée avec un objet représentant un plan de repas. L'objet mealPlan contient les repas de chaque jour, tels que le petit-déjeuner, le déjeuner, la collation et le dîner. Chaque repas est initialement vide, mais peut être modifié ultérieurement.

Ce composant est utilisé pour chercher les nourritures et les ajouter dans le MealPlanning. Le composant a plusieurs parties importantes :

Le composant utilise le hook useState pour déclarer plusieurs états locaux, tels que search pour la valeur de recherche, foodData pour stocker les données d'aliments, selectedFood pour représenter l'aliment sélectionné, etc.

La fonction fetchFoodData est une fonction asynchrone qui effectue une requête à une API (utilisant axios) pour récupérer les résultats de recherche d'aliments correspondant à la valeur de recherche entrée par l'utilisateur. Les résultats sont ensuite stockés dans l'état foodData.

Les fonctions handleSearchSubmit, handleFoodSelection et handleAddToMealPlan sont des gestionnaires d'événements qui sont appelés respectivement lors de la soumission de la recherche, de la sélection d'un aliment et de l'ajout d'un aliment au plan de repas. Ces fonctions effectuent différentes actions telles que la récupération des données des aliments, la mise à jour des états, et l'affichage ou la suppression de la fenêtre modale.

La fonction renderFoodItem est utilisée pour afficher les éléments individuels de la liste d'aliments dans un composant FlatList. Chaque élément de la liste est représenté par une vue contenant les détails de l'aliment.

Le composant rend un ensemble de vues, y compris un composant SearchBar pour la saisie de la recherche, un bouton pour soumettre la recherche, une FlatList pour afficher les résultats de recherche, et une fenêtre modale pour afficher les détails d'un aliment sélectionné et permettre à l'utilisateur de spécifier la quantité et le type de repas.

La constante styles définit les styles CSS pour les éléments du composant, tels que la couleur de fond, les marges, les bordures, etc. Le composant FoodDataBaseScreen est exporté pour être utilisé ailleurs dans l'application.

MealPlanningScreen.js

Ce composant représente l'écran de planification des repas de l'application. Il reçoit une prop mealPlan qui contient les informations sur les repas planifiés pour chaque jour de la semaine.

Le composant utilise le hook useState pour gérer l'état de la variable fold, qui représente l'état de pli (ou d'ouverture) des jours de la semaine. Par défaut, tous les jours de la semaine sont initialisés à false, ce qui signifie qu'ils sont pliés.

La fonction inverseFold est appelée lorsque l'utilisateur appuie sur un jour de la semaine. Elle met à jour l'état de fold en inversant la valeur du jour spécifié. Cela permet d'ouvrir ou de plier la section correspondante.

Le rendu du composant MealPlanningScreen consiste en une View qui enveloppe tout le contenu. L'intérieur de la View contient une ScrollView avec un arrière-plan de couleur verte. À l'intérieur de la ScrollView, les jours de la semaine sont affichés sous forme de TouchableOpacity, ce qui permet à l'utilisateur de les sélectionner.

Lorsque l'utilisateur appuie sur un jour de la semaine, la fonction inverseFold est appelée avec le jour correspondant en tant qu'argument. En fonction de l'état de fold pour ce jour, la section des repas est affichée ou masquée.

Pour chaque jour de la semaine, s'il est ouvert (fold[day] === true), une View est affichée avec des Text indiquant les différents repas : petit-déjeuner, déjeuner, dîner et collation.

On a des possibilités d’ajouter ou supprimer les nourritures avec les icon fournies.

Chaque jour on calcule la calories total.

Les styles CSS sont définis à l'aide de StyleSheet.create pour définir le style du titre des jours de la semaine. Enfin, le composant MealPlanningScreen est exporté en tant que composant par défaut, ce qui permet de l'importer et de l'utiliser dans d'autres parties de l'application.

Développement de l'application

Versionning

Nous nous sommes servies de github pour partagé et versionné notre code.

Fonctions réalisés

On a réussi à réaliser tous les fonctions demandées.

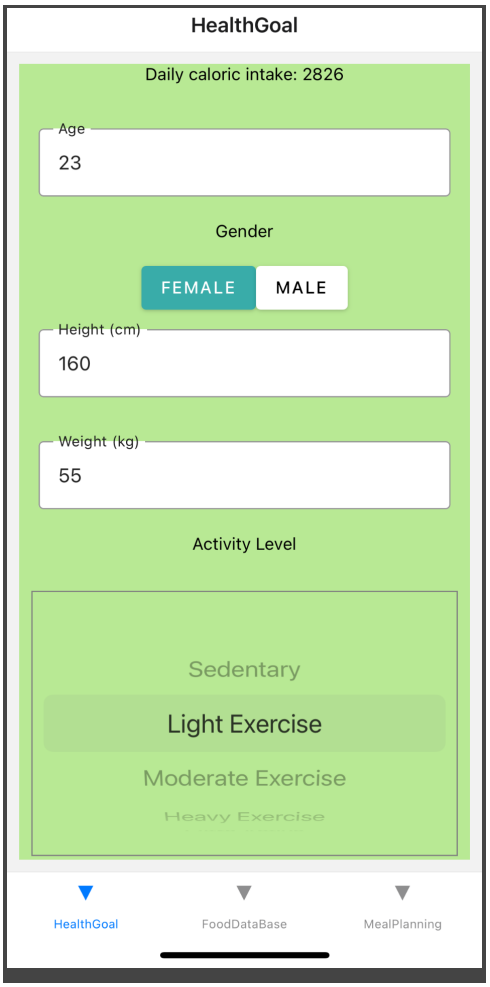
Difficultés rencontrés:

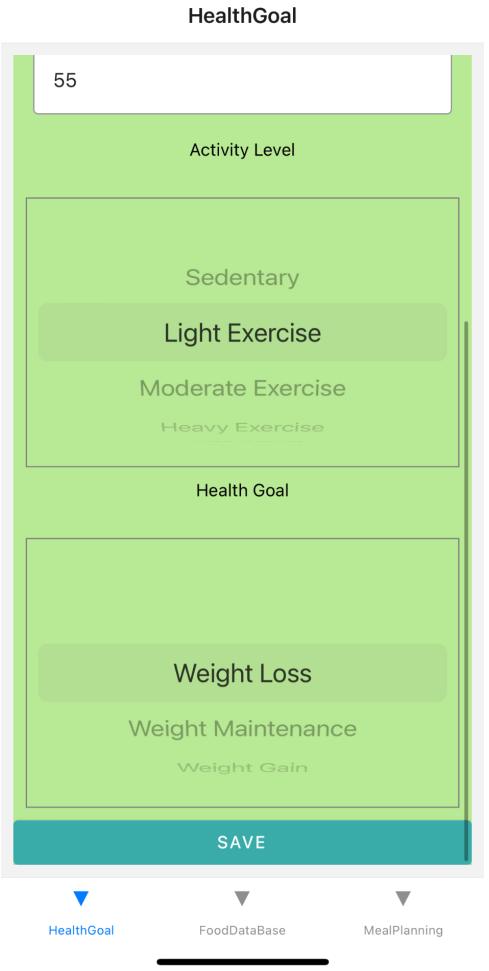
1. le temps insuffisant
   On est dans une période avec beaucoup de projets à réaliser. Cela nous donne moins de temps à travailler sur le projet. On a réussi à développer tous les fonctions demandées, on a appris beaucoup de connaissances via ce projet, si on a un peu plus de temps, on va continuer à améliorer le CSS and le User experience.

2. Traitement des variables de type objet
   Créer une variable const et useState est simple, mais si ce variable est un objet, il devient un peu plus compliqué à le manipuler. Utiliser le map pour parcourir l'objet, filtrer selon la condition, etc.

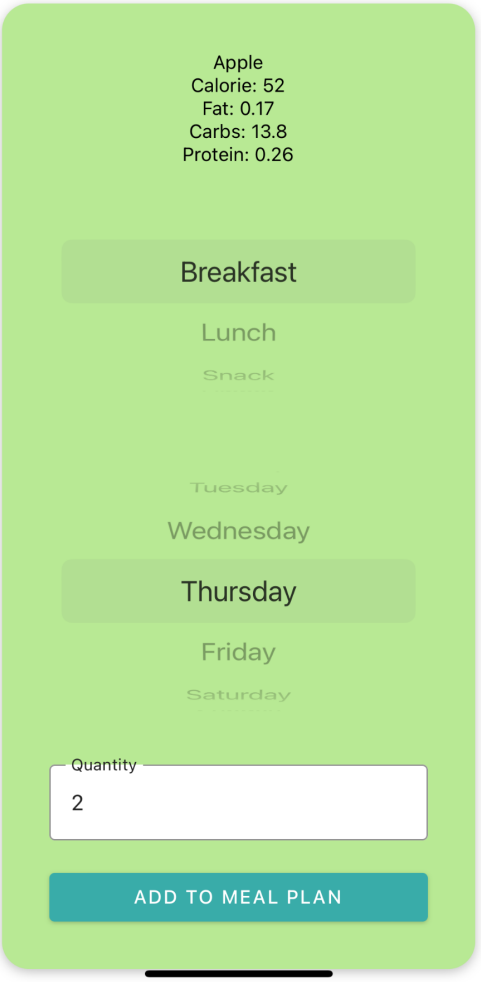
3. Passage de props
   React utilise des props pour passer des variables de une fonction à une autre. Savoir bien s'organiser les props n'est pas une chose évident.

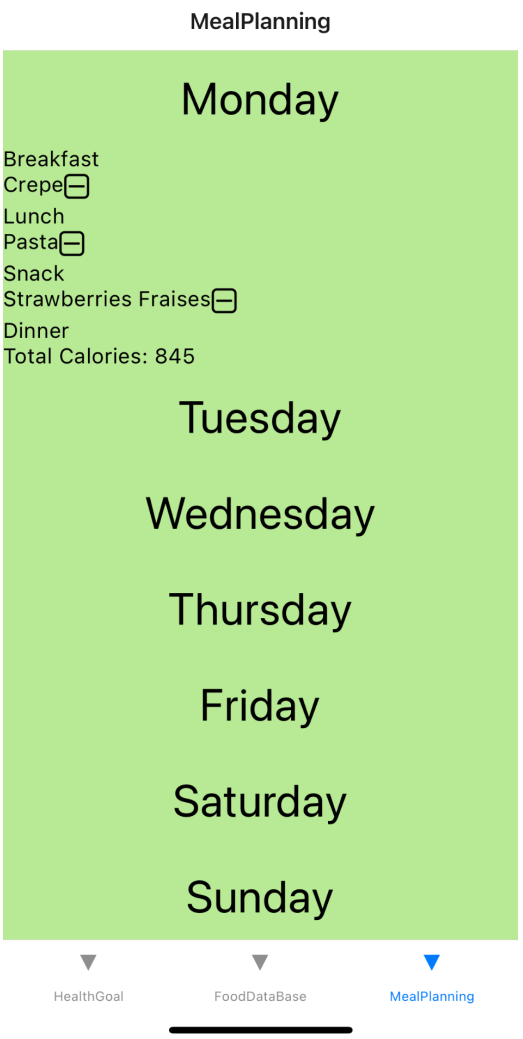
ScreenShots:






FoodDataBase
Apple
SUBMIT
Apple
Calorie: 52
Fat: 0.17
Carbs: 13.8
Protein: 0.26
Empire Apple
Calorie: 52
Fat: 0.17
Carbs: 13.8
Protein: 0.26
Sorb Apple
Calorie: 52
Fat: 0.17
Carbs: 13.8
Protein: 0.26
Fuji Apple
Calorie: 63
Fat: 0.18
Carbs: 15.2
Protein: 0.2
HealthGoal
FoodDataBase
MealPlanning


Apple
Calorie: 52
Fat: 0.17
Carbs: 13.8
Protein: 0.26
Breakfast
Lunch
Snack
Tuesday
Wednesday
Thursday
Friday
Saturday
Quantity
2
ADD TO MEAL PLAN



Conclusion

On a beaucoup apprécié ce projet React Native, cette expérience “fun” nous donner plus envie d’apprendre le React. Créer une application de 0 à 1, ça nous offre beaucoup de satisfaction et nous motiver à aller plus loins.