REACT NATIVE ENDPOINT 🚀

📚 1 - INSTALATION

Nous utiliserons la dépendance "Expo" pour installer notre application React Native.

"Expo" est une plateforme de développement destinée à React Native, qui offre un environnement permettant de simplifier la configuration initiale de votre projet React Native. Cela favorise un développement rapide.

Pour commencer, installez le package "npx" ainsi que la CLI d'Expo. Cela nous permettra d'installer notre application par la suite.

PS : Si vous avez déjà installé Expo sur votre machine, n'exécutez pas la commande "npm i -g expo-cli".

```
npm i -g npx
npm i -g expo-cli
```

Pour installer votre application, exécutez la commande ci-dessous.

```
npx create-expo-app --template
```

Ensuite, sélectionnez l'option "Blank".

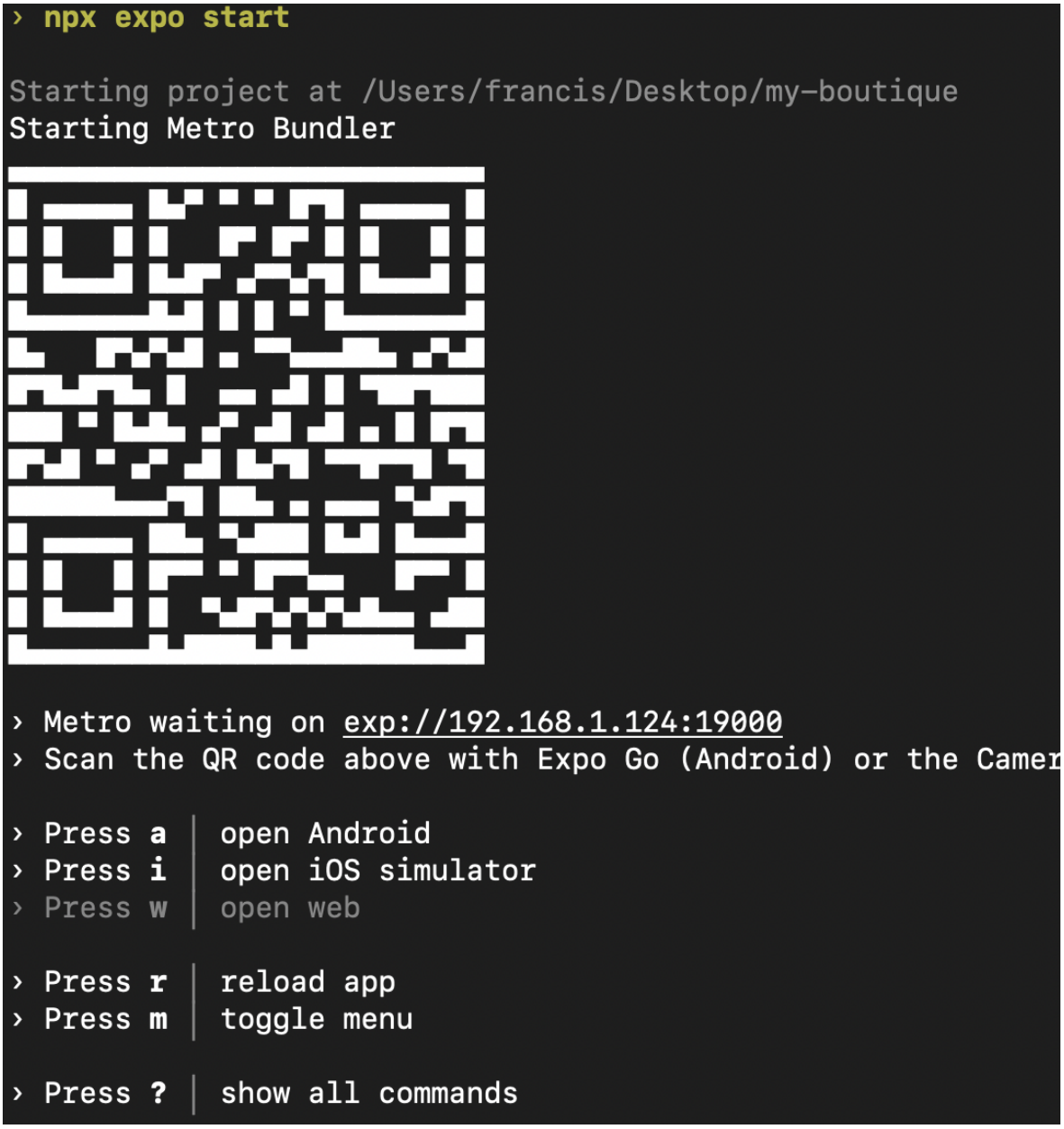
```
? Choose a template: › - Use arrow-keys. Return to submit.
❯   Blank - a minimal app as clean as an empty canvas
    Blank (TypeScript)
    Navigation (TypeScript)
    Blank (Bare)
```

Définissez un nom pour votre application.

```
✔ Choose a template: › Blank
? What is your app named? › my-boutique
```

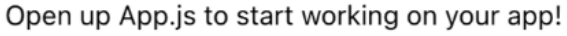
Une fois que vous êtes dans votre projet via le terminal, exécutez la commande suivante :

```
expo start
```



*Lorsque le code **QR** apparaît, appuyez sur "w" pour ouvrir l'application mobile côté web. Si un message apparaît, écrivez "y" pour installer les packages nécessaires permettant de lancer votre application mobile côté web.*

Une page de votre navigateur s'ouvrira, et vous y trouverez le résultat suivant :



"Open up App.js to start working on your app!" provient du fichier "App.js" situé à la racine de votre projet.

App.js > App

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});

```

📓 2 - STRUCTURATION

Ensuite, créez les dossiers suivants :

> **screen** (Ce dossier contiendra tous nos écrans)

> **components** (*Ce dossier contiendra tous nos composants qui seront susceptibles d'être appelés dans plusieurs écrans*)

> **constants** (Ce dossier contiendra toutes les contantes de notre application)

> **navigation** (*Ce dossier contiendra la navigation de notre application*)

Ensuite, ajoutez les fichiers suivants dans les dossiers que vous venez de créer :

'api.js' → constants

'router.js' → navigation

'home.js' → screen

Dans le fichier "api.js", ajoutez les URL de votre API, par exemple :

Le port de votre API sera certainement 8000, soyez vigilant avec ça.

```
constants > JS api.js > URL
1   export const URL = {
2     FETCH_PLAYERS: 'http://localhost:8080/api/player/all',
3     POST_PLAYER: 'http://localhost:8080/api/player/signup'
4   }
```

Dans le fichier home.js

Ajoutez cette structure de base qui se modifiera par la suite dans le fichier "home.js".

```
screen > JS home.js > ...
1   import { StyleSheet, Text, View } from 'react-native'
2   import React from 'react'
3
4   export default function Home() {
5     return (
6       <View>
7         <Text>home</Text>
8       </View>
9     )
10  }
11
12  const styles = StyleSheet.create({})
```

Dans le fichier app.js

Dans votre fichier ‘app.js’ qui ressemble à l’image ci-dessous.

App.js > ...

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

1 - Importer le composent Home

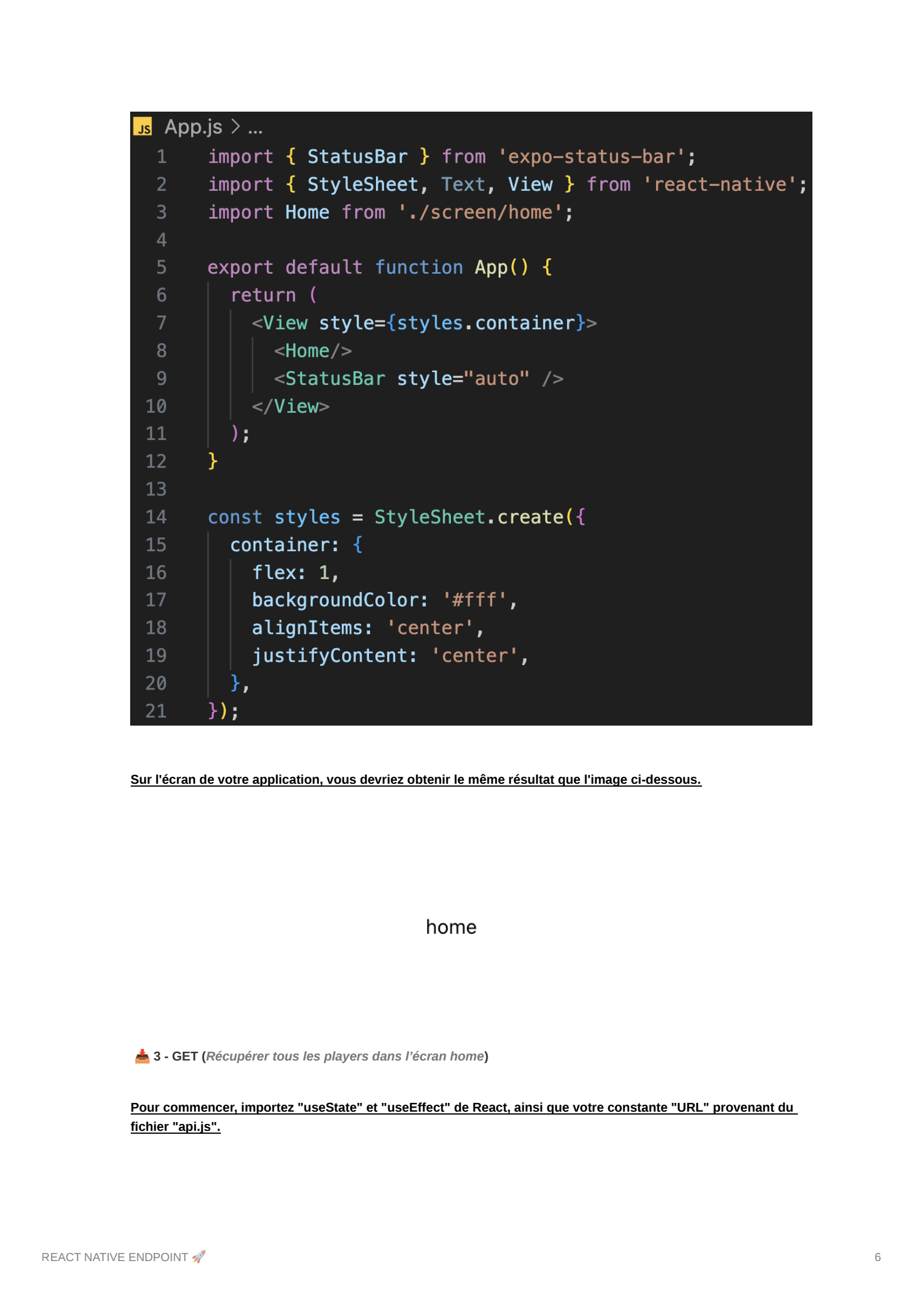
```
import Home from './screen/home';
```

2 - Remplacer la composant **‘Text’** par **‘Home’**

App.js > ...

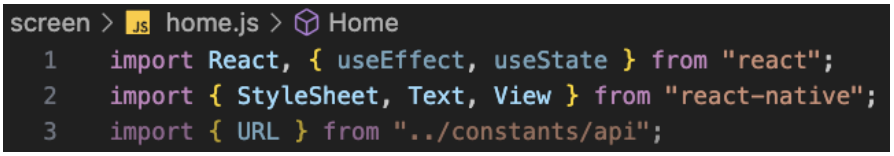
```js
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';
import Home from './screen/home';

export default function App() {
  return (
    <View style={styles.container}>
      <Home/>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Sur l'écran de votre application, vous devriez obtenir le même résultat que l'image ci-dessous.



📥 **3 - GET (*Récupérer tous les players dans l'écran home*)**

Pour commencer, importez "useState" et "useEffect" de React, ainsi que votre constante "URL" provenant du fichier "api.js".



Créez un état (state) qui nous permettra de stocker les données provenant de notre API.

```
const [players, setPlayers] = useState([]);
```

Vous devez installer Axios, qui nous permettra de faire des requêtes vers notre API.

À la racine de votre projet, exécutez la commande suivante.

```
npm i axios
```

Importez Axios dans le fichier "home.js".

```
import axios from 'axios'
```

Ensuite, mettez en place le hook useEffect qui se chargera de faire une requête.

```
useEffect(() => {
  // Cette fonction asynchrone fetchPlayers est appelée
  // lorsqu'un composant est monté.
  const fetchPlayers = async () => {
    try {
      // Effectue une requête GET à l'URL spécifiée (URL.FETCH_PLAYERS)
      // en utilisant Axios.
      const { data } = await axios.get(URL.FETCH_PLAYERS);

      // Vérifie si le statut de la réponse est égal à 200 (succès).
      if (status == 200) {
        // Si le statut est 200, affiche un message de succès dans la console.
        console.log('Succès de la requête');
      }

      // Met à jour l'état "players" du composant avec les données
      // récupérées de la requête.
      setPlayers(data);
    } catch (error) {
      // En cas d'erreur lors de la requête, lance une exception
      // avec le message d'erreur.
      throw error.message;
    }
  };

  // Appelle la fonction fetchPlayers lorsque le composant est monté
  // (avec un tableau de dépendances vide).
  fetchPlayers();
}, []);
```

Ensuite, itérez sur notre état "players" avec le composant FlatList de React Native. N'oubliez pas de l'importer.

```
<FlatList
  // Les données que vous souhaitez afficher dans la liste.
  data={players}
```

```
  // Une fonction qui extrait une clé unique pour chaque élément de données.
  // on utilise l'ID de chaque élément comme clé unique.
  keyExtractor={item => item._id}

  // Une fonction qui rend chaque élément de données de la liste.
  renderItem={renderItem}
/>
```

Mettez en place la fonction "renderItem".

```
const renderItem = ({ item }) => {
  // Destructure les propriétés de l'élément de données pour un accès
  // plus facile.
  const { _id, pseudo, email } = item;

  return (
      <View>
        {/*
          Affiche le pseudo et l'e-mail de l'élément de données
          dans un composant Text.
        */}
        <Text>{pseudo} : {email}</Text>
      </View>
  );
};
```

À la suite de cela, tous les joueurs de votre API devraient apparaître sur votre écran.

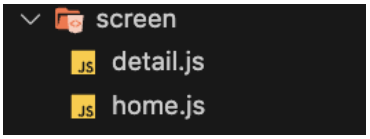
📖 4 - GET BY ID (*Récupérer un élément par son id*)

Il faut rendre notre liste d'utilisateurs cliquable.

Nous allons utiliser le composant Pressable, n'oubliez pas de l'importer.

```
    <Pressable>
        <View>
          {/*
            Affiche le pseudo et l'e-mail de l'élément de données
            dans un composant Text.
          */}
          <Text>{pseudo} : {email}</Text>
        </View>
      </Pressable>
```

Pour le moment, rien ne se passe. Il faut créer un nouvel écran nommé "detail.js" dans le dossier "screen".



Avec le snippet 'rnfs', mettez en place rapidement une structure. Assurez-vous que le nom de votre composant commence par une majuscule.

```
import React from 'react'
import { StyleSheet, Text, View } from 'react-native'

export default function Detail() {
  return (
    <View>
      <Text>detail</Text>
```

```
    </View>
  )
}

const styles = StyleSheet.create({})
```

Ensuite, l'objectif est que lorsque l'on clique sur un joueur, on soit redirigé vers "detail.js". Pour cela, il faudra mettre en place un système de navigation.

Pour commencer, installer ces trois dépendances à la racine de votre application.

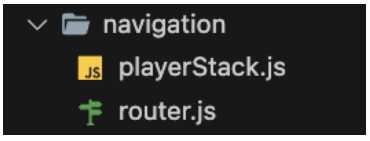
```
npm i @react-navigation/native @react-navigation/native-stack
```

```
npm i @react-navigation/bottom-tabs
```

Ensuite, importez ces trois dépendances dans le fichier "router.js", comme ceci.

```
import React from 'react'

import { NavigationContainer } from '@react-navigation/native'
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs'
```

Créez ensuite un fichier "PlayerStack.js" dans le dossier "navigation".



Dans le fichier "PlayerStack.js", nous allons importer le package "stack".

```
import React from 'react'
import { createNativeStackNavigator } from '@react-navigation/native-stack'
```

Il faudra également importer les écrans liés aux joueurs (home, detail, etc.).

```
import Home from '../screen/home'
import Detail from '../screen/detail'
```

Ensuite, créez une constante "Stack" qui aura pour valeur la fonction "createNativeStackNavigator()".

```
// Import de la fonction de création de stack de navigation
// à partir de la bibliothèque React Navigation.
const Stack = createNativeStackNavigator();
```

Ajoutez une fonction "PlayerStack" qui doit être exportée, et cette fonction contiendra la navigation liée aux Players.

```
// Cette fonction déclare le composant "PlayerStack"
// qui est un composant de navigation.
export default function PlayerStack() {
  return (
    // Utilisation de la composante de navigation Stack.Navigator
    // pour gérer la navigation entre les écrans.
    <Stack.Navigator>
      {/*
        Écran nommé "Home" qui affiche le composant "Home"
        lorsque la navigation atteint cet écran.
      */}
      <Stack.Screen name='List' component={Home} />
      {/*
        Écran nommé "Detail" qui affiche le composant "Detail"
        lorsque la navigation atteint cet écran.
      */}
      <Stack.Screen name='Detail' component={Detail} />
    </Stack.Navigator>
  );
}
```

Une fois cela terminé, importez "PlayerStack" dans le fichier "router.js".

```
import PlayerStack from './playerStack'
```

Créez une constante "Tabs" qui nous permettra d'avoir des liens directs dans notre application.

```
// Création d'une instance de tab navigation (navigation par onglets)
const Tabs = createBottomTabNavigator();
```

Ensuite, créez une fonction "AppNavigation" qui sera la fonction principale de votre routeur. Elle devra regrouper tous les écrans que vous souhaitez afficher.

```
// Cette fonction déclare le composant de navigation principal "AppNavigation"
export default function AppNavigation() {
  return (
    // Le composant de navigation global est enveloppé dans
    // "NavigationContainer".
    <NavigationContainer>
      {/* Configuration des onglets de navigation */}
      <Tabs.Navigator>
        {/*
            Onglet "Home" : Utilise le composant "PlayerStack"
            pour la navigation et cache la barre de navigation (header).
        */}
        <Tabs.Screen name='Home' component={PlayerStack} options={{ headerShown: false }} />

      </Tabs.Navigator>
    </NavigationContainer>
  );
}
```

L'option `options={{ headerShown: false }}` *retirera le titre "home" de l'en-tête de vos écrans.*

Pour finir, modifiez le fichier "App.js" en important "AppNavigation" et en supprimant le composant "Home" car il sera importé par notre navigation.

```
import { StatusBar } from "expo-status-bar";
import { StyleSheet, Text, View } from "react-native";
import AppNavigation from "./navigation/router";

export default function App() {
  return (
    <>
      <AppNavigation />
      <StatusBar style="auto" />
    </>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center",
  },
});
```

Maintenant que le système de navigation a été mis en place, revenons dans le fichier "Home.js" pour finaliser la redirection vers la page "detail" après un clic sur un Player.

Ajoutons la prop "navigation" à notre composant. N'oubliez pas de déstructurer cette prop, comme indiqué dans l'image ci-dessous.

```
export default function Home({ navigation }) {
```

Ensuite, nous allons envelopper les composants "View" (composants qui servent à afficher nos joueurs) avec le composant "Pressable", ce qui rendra chaque joueur cliquable.

N'oubliez pas d'importer le composant "Pressable".

```
const renderItem = ({ item }) => {
    const { _id, pseudo, email } = item

    return(
      <Pressable>
        <View>
          <Text>{pseudo} : {email} </Text>
        </View>
      </Pressable>
    )
  }
```

Ajoutons par la suite l'événement 'OnPress' (OnPress est l'équivalent de l'événement onClick en React). Cela nous permettra d'utiliser la prop "navigation".

La prop "navigation" contient une fonction "navigate", et c'est grâce à cette fonction que nous pourrons spécifier vers quel écran nous souhaitons être redirigés après le clic.

```
const renderItem = ({ item }) => {
    const { _id, pseudo, email } = item

    return(
<Pressable
  onPress={() => {
    // Lorsque l'élément Pressable est pressé,
    // déclenche la navigation vers l'écran "Detail"
    // avec un paramètre "id" égal à "_id".
```

```
    // ce qui nous permettra de récupérer l'id du player
    // dans l'ecran detail.
    navigation.navigate("Detail", {
      id: _id
    })
  }}
>
  {/* Votre contenu à l'intérieur du composant Pressable */}
  {/* Cela peut inclure du texte, des images, des icônes, etc. */}
</Pressable>
        <View>
          <Text>{pseudo} : {email} </Text>
        </View>
      </Pressable>
    )
  }
```

Maintenant, dans l'écran "détail", il nous faudra importer toutes les dépendances et constantes nécessaires.

```
import React, { useEffect, useState } from 'react'
import { StyleSheet, Text, View } from 'react-native'
import axios from 'axios'
import { URL } from '../constants/api'
```

Ensuite, ajoutez la prop "route" au composant.

```
export default function Detail({ route })
```

Par la suite, nous aurons besoin d'ajouter un état (state) qui nous permettra de stocker le résultat de la requête. Ensuite, récupérez l'ID qui est stocké dans l'objet "params" de notre prop "route".

```
// Déclaration d'une variable d'état "player"
// et d'une fonction "setPlayer" pour la mise à jour de l'état.
const [player, setPlayer] = useState([])

// Extraction de la valeur "id" à partir
// des paramètres de la route actuelle.
const { id } = route.params
```

Grâce à cet ID, nous pouvons effectuer une requête vers notre API pour récupérer uniquement les informations du joueur sélectionné.

N'oubliez pas d'ajouter l'URL nécessaire dans le fichier 'api.js'.

```
FETCH_PLAYER_BYID: 'http://localhost:8080/api/player/detail',
```

Maintenant, nous pouvons effectuer notre requête.

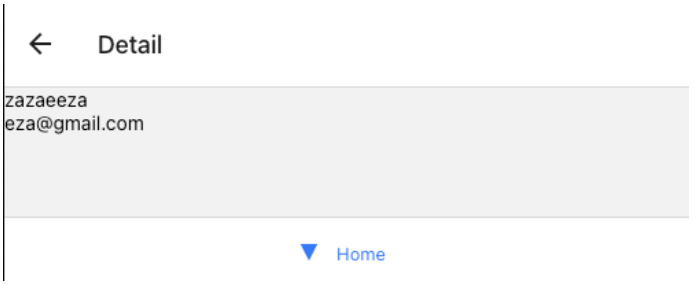
```
useEffect(() => {
    const fetchPlayer = async () => {
      try{
        const { data, status } = await axios.get(`${URL.FETCH_PLAYER_BYID}/${id}`)
        if(status == 200) {
          // Affiche un message de succès dans la console
          console.log("SUCCES GET");
        }
       setPlayer(data)
```

```
      }catch(error){
        throw error.message
      }
    }
    fetchPlayer()
  }, [])
```

Pour finir, affichons les données du player.

```
return (
    <View>
      <Text>{player.pseudo}</Text>
      <Text>{player.email}</Text>
    </View>
  )
```

À cette étape, vous pouvez sélectionner un joueur et être redirigé vers la page de détail, qui affichera le résultat suivant.



🗑️ 5 - SUPPRESION

Ajoutons à partir de l'écran de détail un bouton pour supprimer un joueur. Ce bouton appellera la fonction "deletePlayer". N'oubliez pas d'importer le composant Button.

Importer ces dépendances.

```
import { StyleSheet, Text, View, Button } from 'react-native'
```

Ajoutons la fonction "deletePlayer" et le composant Button à notre composant de la façon suivante.

```
const deletePlayer = () => {
    // ...
  }

  return (
    <View>
      <Text>{player.pseudo}</Text>
      <Text>{player.email}</Text>
      <Button
        title="suprimer"
        onPress={deletePlayer}
      />
```

```
    </View>
  )
```

Dans la fonction "delete", nous devons simplement faire une requête vers notre API pour supprimer le Player sélectionné grâce à son "ID" que nous connaissons.

N'oubliez pas d'ajouter l'URL de suppression dans le fichier 'api.js'.

```
DELETE_PLAYER: 'http://localhost:8080/api/player/delete'
```

Maintenant, passons à la requête.

```
// Définition d'une fonction asynchrone "deletePlayer"
  // qui permet de supprimer un joueur.
  const deletePlayer = async () => {
    try {
      // Tentative de faire une requête HTTP DELETE
      // pour supprimer un joueur en utilisant Axios.
      const { data, status } = await axios.delete(`${URL.DELETE_PLAYER}/${id}`);

      // Vérification du statut de la réponse du serveur
      // par exemple, statut 200 signifie que la suppression a réussi).
      if (status === 200) {
        // Si le statut est 200, cela signifie que la suppression
        // a réussi.
        // Affichage des données renvoyées par le serveur.
        console.log(data);
      }
    } catch (error) {
      // En cas d'erreur, la fonction "catch" est exécutée.
      // La variable "error" contient les détails de l'erreur.

      // Relance de l'erreur avec un objet personnalisé
      // contenant le message d'erreur et la réponse d'erreur.
      throw { message: error.message, response: error.response };
    }
  };
```

Pour finir, ajoutez le paramètre "navigation" à votre composant. Nous utiliserons cet objet pour revenir à l'écran précédent après une suppression.

```
export default function Detail({ route, navigation })
```

Ensuite, ajoutez `navigation.goBack()` *dans la condition "try" de votre requête Axios comme ceci.*

```
// Définition d'une fonction asynchrone "deletePlayer"
  // qui permet de supprimer un joueur.
  const deletePlayer = async () => {
    try {
      // Tentative de faire une requête HTTP DELETE
      // pour supprimer un joueur en utilisant Axios.
      const { data, status } = await axios.delete(`${URL.DELETE_PLAYER}/${id}`);

      // Vérification du statut de la réponse du serveur
      // par exemple, statut 200 signifie que la suppression a réussi).
      if (status === 200) {
        // Si le statut est 200, cela signifie que la suppression
        // a réussi.
        // Affichage des données renvoyées par le serveur.
        console.log(data);
        // Suppression réussie, utilisation de la navigation pour revenir
```

```
        // à l'écran précédent.
        navigation.goBack()
      }
    } catch (error) {
      // En cas d'erreur, la fonction "catch" est exécutée.
      // La variable "error" contient les détails de l'erreur.

      // Relance de l'erreur avec un objet personnalisé
      // contenant le message d'erreur et la réponse d'erreur.
      throw { message: error.message, response: error.response };
    }
  };
```

🥷 *6 - UPDATE (Modifier des données)*

Créez un fichier "update.js" en mettant en place une structure avec le snippet 'rnfs'. Assurez-vous que le nom du composant commence par une majuscule.

```
import { StyleSheet, Text, View } from 'react-native'
import React from 'react'

export default function Update() {
  return (
    <View>
      <Text>update</Text>
    </View>
  )
}

const styles = StyleSheet.create({})
```

Ensuite, ajoutez la route "update" dans vos routes de la stack.

```
import React from 'react'
import { createNativeStackNavigator } from '@react-navigation/native-stack'

import Home from '../screen/home'
import Detail from '../screen/detail'
import Update from '../screen/update'

const Stack = createNativeStackNavigator()

// Cette fonction déclare le composant "PlayerStack" qui est un composant de navigation.
export default function PlayerStack() {
  return (
    // Utilisation de la composante de navigation Stack.Navigator
    // pour gérer la navigation entre les écrans.
    <Stack.Navigator>
      {/*
        Écran nommé "Home" qui affiche le composant "Home"
        lorsque la navigation atteint cet écran.
      */}
      <Stack.Screen name='List' component={Home} />
      {/*
        Écran nommé "Detail" qui affiche le composant "Detail"
        lorsque la navigation atteint cet écran.
      */}
      <Stack.Screen name='Detail' component={Detail} />
        {/*
        Écran nommé "Update" qui affiche le composant "Detail"
        lorsque la navigation atteint cet écran.
      */}
      <Stack.Screen name='Update' component={Update} />
    </Stack.Navigator>
  );
}
```

Ajoutez l'URL qui permettra de faire la requête dans votre fichier "api.js".

```
UPDATE_PLAYER: 'http://localhost:8080/api/player/update'
```

Maintenant, ajoutons le Bouton "update" dans le fichier "detail.js" (à la suite du bouton "supprimer") qui nous permettra de nous rediriger vers l'écran "update".

```
return (
    <View>
      <Text>{player.pseudo}</Text>
      <Text>{player.email}</Text>
      <Button title="suprimer" onPress={deletePlayer} />
      <Button
        title="Modifier"
        onPress={() => {
          navigation.navigate('Update', {
            id: id
          })
        }}
      />
    </View>
  );
```

Dans le fichier "update.js", nous allons récupérer l'ID transmis à partir de l'écran "detail". Nous allons mettre en place un état qui se chargera de stocker le résultat de notre requête Axios.

Exactement comme nous l'avons fait pour "detail".

```
import React, { useEffect, useState } from "react";
import { StyleSheet, TextInput, Button } from "react-native";
import axios from "axios";
import { URL } from "../constants/api";

export default function Update({ route }) {
  const [player, setPlayer] = useState([]);

  const { id } = route.params;

  useEffect(() => {
    const fetchPlayer = async () => {
      try {
        const { data, status } = await axios.get(`${URL.FETCH_PLAYER_BYID}/${id}`);
        if (status == 200) {
          // Affiche un message de succès dans la console
          console.log("SUCCES GET BY ID");
        }
        setPlayer(data);
      } catch (error) {
        throw error.message;
      }
    };
    fetchPlayer();
  }, []);

  return (
    <View>
     <Text>Update</Text>
    </View>
  );
}

const styles = StyleSheet.create({});
```

Maintenant que nous sommes en mesure de récupérer le joueur qui subira la modification dans notre API, nous allons mettre en place un formulaire qui aura les informations du joueur pré-remplies par défaut.

```
return (
    <>
      {/* Champ de texte pour le pseudo du joueur */}
      <TextInput
        style={styles.textInput}
        defaultValue={player.pseudo}
        maxLength={20}
        onChangeText={(val) => _onChangeText("pseudo", val)}
      />
      {/* Champ de texte pour l'e-mail du joueur */}
      <TextInput
        style={styles.textInput}
        defaultValue={player.email}
        maxLength={20}
        onChangeText={(val) => _onChangeText("email", val)}
      />
      {/*
        Bouton "valider" qui déclenche la fonction _handleSubmit
        lorsqu'il est pressé
      */}
      <Button title="Valider" onPress={_handleSubmit} />
    </>
  );
```

Ps : La propriété "defaultValue" permet simplement de pré-remplir les champs.

Ajoutons maintenant la fonction qui nous permettra de mettre à jour notre état "Player" avec les nouvelles saisies.

```
const _onChangeText = (key, value) => {
  // Crée une nouvelle copie de l'objet "player" en utilisant la syntaxe
  // de décomposition ("spread operator")
  // Remplace la valeur de la clé spécifiée ("key")
  // par la nouvelle valeur ("value")
  setPlayer({ ...player, [key]: value });
}
```

Pour finir, il ne nous reste plus qu'à ajouter la fonction "_handleSubmit" qui se chargera d'envoyer les données à notre API pour effectuer la mise à jour.

```
const _handleSubmit = async () => {
    try {
      // Utilise une requête HTTP PUT pour mettre à jour le joueur en utilisant l'ID fourni
      const { data } = await axios.put(`${URL.UPDATE_PLAYER}/${id}`, player);

      // Vérifie si la réponse du serveur a un statut de 200 (succès)
      if (data.status === 200) {
        // Affiche un message de succès dans la console
        console.log("Player Updated !");
      }
    } catch (error) {
      // En cas d'erreur, lance une exception avec le message d'erreur
      throw error.message;
    }
  }
```

➕ 7 - POST (*signup → inscription*)

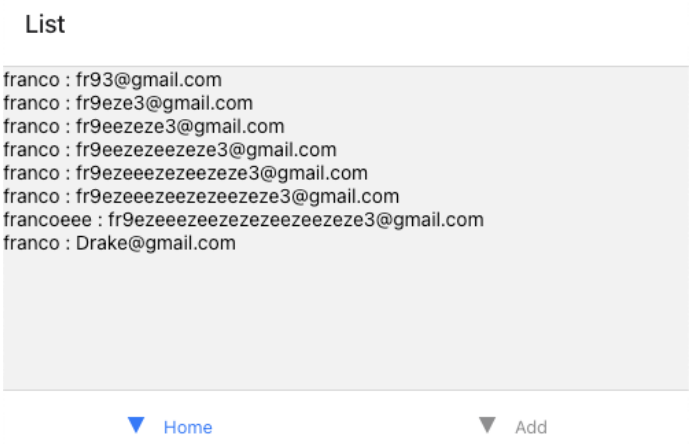
Créez un fichier "add.js" en mettant en place une structure avec le snippet 'rnfs'. Assurez-vous que le nom du composant commence par une majuscule.

```
import React from 'react'
import { StyleSheet, Text, View } from 'react-native'

export default function Add() {
  return (
    <View>
      <Text>add</Text>
    </View>
  )
}

const styles = StyleSheet.create({})
```

Ajoutez la route de cet écran directement dans "router.js", car nous devons y avoir un accès direct.

```
import React from 'react'

import { NavigationContainer } from '@react-navigation/native'
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs'

// STACK
import PlayerStack from './playerStack'
import Add from '../screen/add'

// Création d'une instance de tab navigation (navigation par onglets)
const Tabs = createBottomTabNavigator();

// Cette fonction déclare le composant de navigation principal "AppNavigation"
export default function AppNavigation() {
  return (
    // Le composant de navigation global est enveloppé dans "NavigationContainer".
    <NavigationContainer>
      {/* Configuration des onglets de navigation */}
      <Tabs.Navigator>
        {/* Onglet "Home" : Utilise le composant "PlayerStack" pour la navigation et cache la barre de navigation (header). */}
        <Tabs.Screen name='Home' component={PlayerStack} options={{ headerShown: false }} />
        <Tabs.Screen name='Add' component={Add} />
      </Tabs.Navigator>
    </NavigationContainer>
  );
}
```

Vérifiez que les deux onglets apparaissent correctement en bas de votre écran, comme indiqué dans l'image ci-dessous.



Dans le fichier "Add.js", nous savons que nous aurons besoin d'Axios, de la nouvelle URL pour effectuer une requête, ainsi que d'un formulaire. Nous allons mettre tout cela en place.

```
import React, {useState} from "react";
import { Button, StyleSheet, TextInput } from "react-native";
import axios from 'axios'

import { URL } from "../constants/api";

export default function Add() {
  return (
    <View>
      <Text>add</Text>
    </View>
  )
}

const styles = StyleSheet.create({})
```

Ajoutons un état (state) avec des valeurs vides par défaut, que nous mettrons à jour avec les saisies du formulaire.

```
const [player, setPlayer] = useState({
    pseudo: '',
    email: '',
    password: ''
  })
```

Ajoutons également la méthode qui nous permettra de mettre à jour cet état (state).

```
const _onChangeText = (key,value) => {
    setPlayer({...player, [key]: value})
  }
```

Mettons en place le formulaire.

```
return (
    <>
      <TextInput
        style={styles.textInput}
        placeholder="Pseudo"
        maxLength={20}
        onChangeText={(val) => _onChangeText("pseudo", val)}
      />
      <TextInput
        style={styles.textInput}
        placeholder="email"
        maxLength={20}
        onChangeText={(val) => _onChangeText("email", val)}
      />
      <TextInput
        style={styles.textInput}
        placeholder="password"
        secureTextEntry={true}
        maxLength={20}
        onChangeText={(val) => _onChangeText("password", val)}
      />
      <Button
        title="valider"
        onPress={_handleSubmit}
      />
    </>
  );
}
```

Dans le fichier "api.js", assurez-vous d'ajouter l'URL suivante.

```
POST_PLAYER: 'http://localhost:8080/api/player/signup'
```

Maintenant, il nous reste simplement à mettre en place la fonction "_handleSubmit", qui enverra les données saisies à notre API.

```
const _handleSubmit = async () => {
    try{
        // Envoie une requete POST à l'URL spécifié avec les donnée du joueur
        const data = await axios.post(URL.POST_PLAYER, player)
        // Vérifie si le statut de la réponse est 201 (création réussie)
        if(data.status == 201) {
          // Affiche un message de succès dans la console
          return console.log("SUCCES CREATION");
        }
    }catch(error){
      // En cas d'ereur, affiche le message d'erreur dans la console
      throw error.message
    }
  }
```