React Native

Projet développement mobile

PARTIE 2: Page de profil:

Formateur : Djemai Samy

1/ Image de profil:

Nous allons afficher dans la page de profil (**./components/page/Profil/Profil.jsx**), l'image avatar de l'utilisateur, ainsi que la possibilité de la changer.

SI l'utilisateur n'a pas encore déterminé son image de profil, nous afficherons une image par défaut depuis notre dossier **./assets/default_avatar.png**

Nous offrirons la possibilité à l'utilisateur de changer son image de profil depuis la librairie de téléphone, ou en prenant une photo avec l'appareil photo de son téléphone.

1.1/Afficher une image:

En React Native, pour afficher un image, on utilise la balise **<Image/>:**

Par défaut, le composant **image** de **react-native** a une hauteur et une largeur de 0.

Nous allons utiliser un hook fourni par React Native **useWindowDimensions()** pour connaître la taille de la fenêtre, et spécifier la taille de l'image par rapport à la fenêtre.

Import:

```
import { useWindowDimensions, Image } from "react-native";
```

Utilisation dans un constante:

```
const size = useWindowDimensions();
```

Composant Image:

```
<View>
   <Image
     style={[
             styles.image,
           {
             width: size.width,
             height: size.width,
             maxWidth: 250,
             maxHeight: 250,
           },
         ]}
     source={user.avatar ? user.avatar : defaultAvatar}
   />
</View>
```

Style:

```
image: {
    borderRadius: 250,
  },
```

1.2/ Boutons pour modifier l'image:

Nous allons ajouter deux boutons (**TouchableOpacity**) sous forme d'icônes, pour permettre à l'utilisateur de changer son image de profil.

Expo nous fournit nativement des icônes (lien vers docs).

Import:

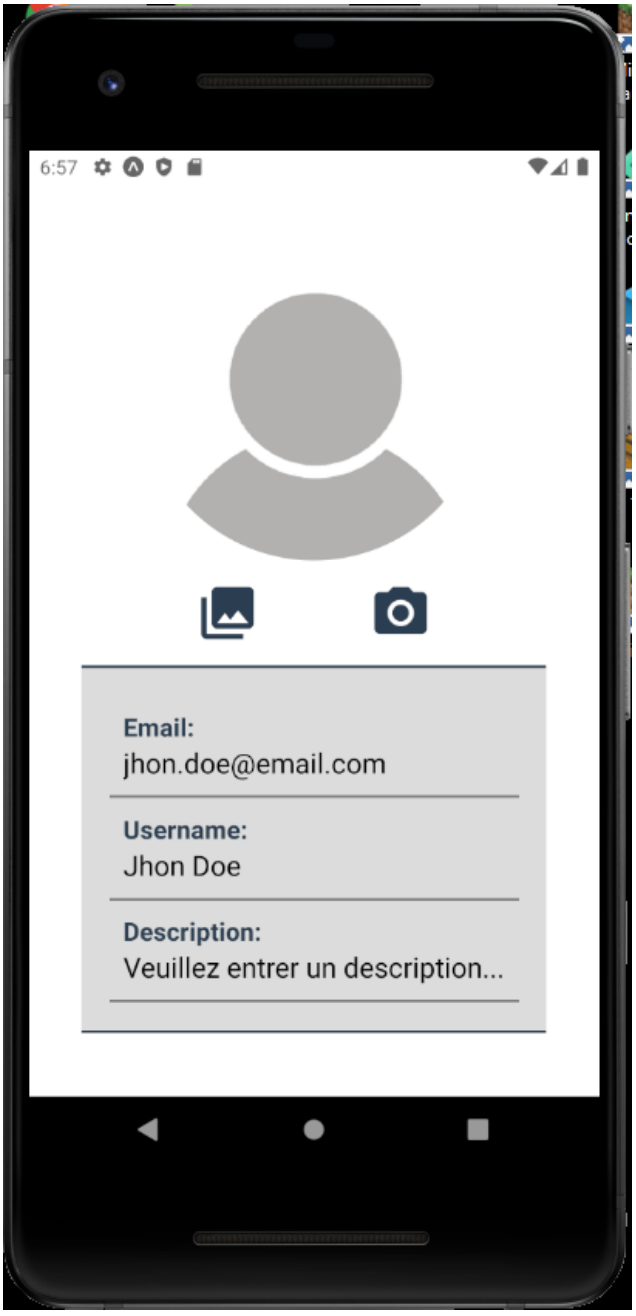
```
import { MaterialIcons } from "@expo/vector-icons";
```

Composants:

```
<View style={styles.iconsContainer}>
    <TouchableOpacity>
        <MaterialIcons
              name='photo-library'
              size={45}
              color={"#2C3E50"}
         />
    </TouchableOpacity>
    <TouchableOpacity>
       <MaterialIcons
              name='camera-alt'
              size={45}
              color={"#2C3E50"} />
    </TouchableOpacity>
</View>
```

Style:

```
iconsContainer: {
    display: "flex",
    flexDirection: "row",
    justifyContent: "space-around",
    paddingVertical: 15,
    maxWidth: 300,
  },
```



1.3/ Choisir une image depuis la librairie:

Installation du package:

Pour pouvoir permettre à l'utilisateur de choisir un image depuis la librairie du téléphone, nous devons installer un package depuis expo:

```
expo install expo-image-picker
```

import de l'objet ImagePicker:

```
import * as ImagePicker from 'expo-image-picker';
```

Fonction pickImage:

Nous allons créer la fonction **async pickImage()** qui s'exécute quand on clique sur le bouton, cette fonction s'occupe :

- D'exécuter la fonction **launchImageLibraryAsync().**

- Puis de modifier le contexte en ajoutant à l'objet user, la clé avatar, contenant l'image retourner par la fonction **launchImageLibraryAsync()**, si l'utilisateur n'a pas annulé.

```
const pickImage = async () => {
    let pickedImage = await ImagePicker.launchImageLibraryAsync();

    if (!pickedImage.cancelled) {
      setUser({ ...user, avatar: pickedImage });
    }
  };
```

Lier la TouchableOpacity a la fonction pickImage():

```
<TouchableOpacity onPress={pickImage}>
    <MaterialIcons
       name='photo-library'
       size={45}
       color={"#2C3E50"}
/>
</TouchableOpacity>
```

1.4/ Navigation en Stack:

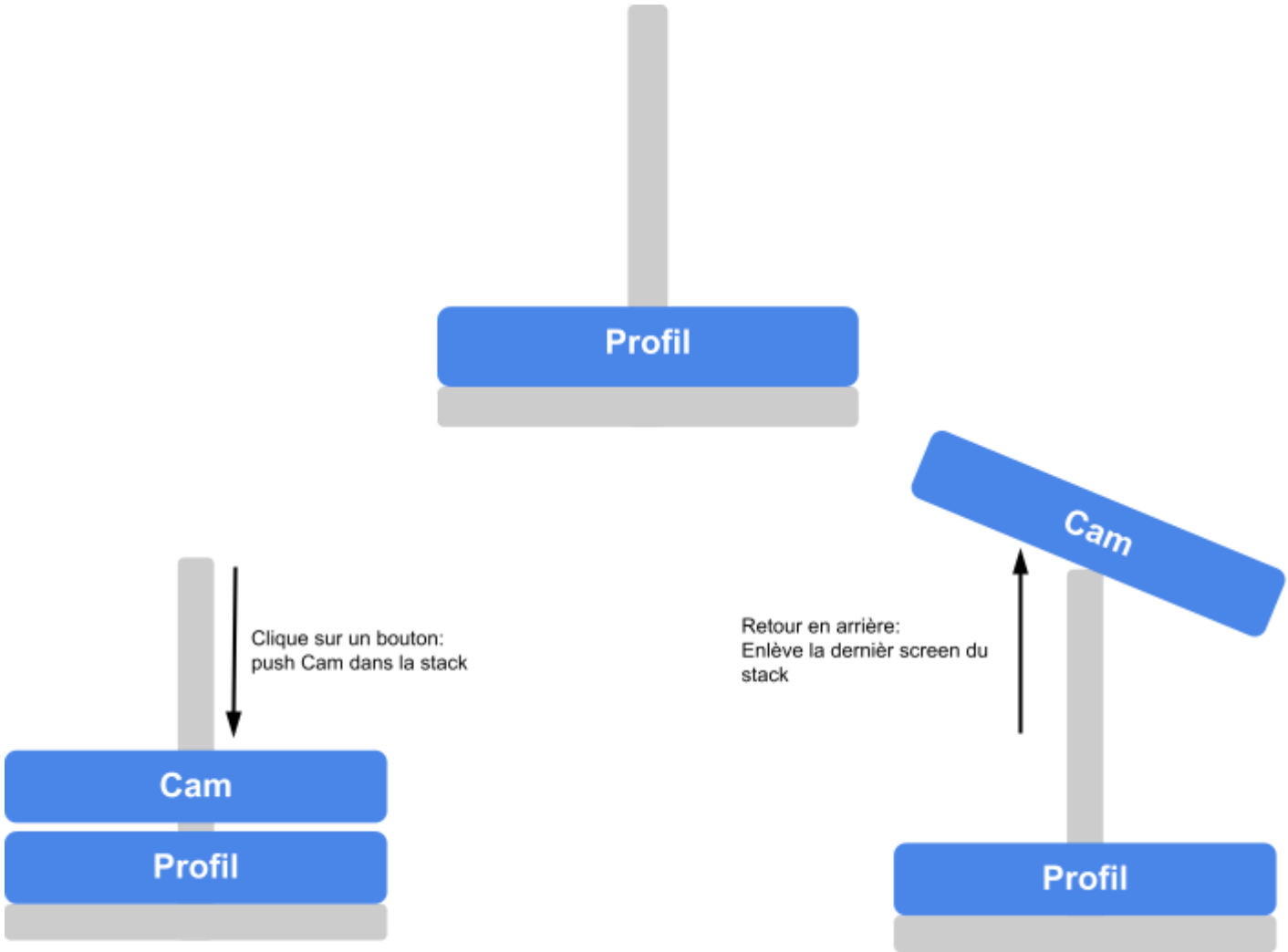
Maintenant nous devons mettre en place une manière de naviguer entre nos pages, pour afficher la caméra. Nous allons utiliser la navigation en pile.

Stack Navigation:

Le principe est simple, imaginez une série d'écrans empilés les uns sur les autres.

Pour naviguer de l'écran actuel vers un autre, il faut ajouter l'écran de destination en haut de la pile d'écran.

Et il faut supprimer l'écran en haut de la pile pour pouvoir revenir au précédent.



Installation des packages nécessaire:

```
expo install @react-navigation/native
expo install @react-navigation/stack
expo install react-native-screens
expo install react-native-reanimated
expo install react-native-gesture-handler
expo install react-native-safe-area-context
```

Création du Stack pour le Profil:

Un Stack est un composant HOC qui possède des composants enfants sous forme de screen.

Ce composant s'initialise avec les écrans qu'il va contenir. Cela signifie que vous devez renseigner la vue principale, mais aussi les vues qui vont être poussées et affichées.

React Navigation associe ensuite des noms à vos vues et est capable d'appeler une vue, de revenir en arrière, etc

Nous allons créer notre premier Stack, il possédera la composant Profil et Cam.

Créer un composant basique: **./components/page/Cam/Cam.jsx**

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

const Cam= () => {
  return (
    <View>
      <Text>Cam</Text>
    </View>
  );
};

export default Cam;
```

Puis le composant: **./components/stacks/ProfilStack.jsx:**

```
import { createStackNavigator } from "@react-navigation/stack";
import Cam from "../page/Cam/Cam";
import Profil from "../page/Profil/Profil";

const Stack = createStackNavigator();
export default function ProfilStack() {
  return (
    <Stack.Navigator
      screenOptions={{
        headerStyle: {
          backgroundColor: 'royalblue'
        },
        headerTitleStyle: {
          color: 'whitesmoke',
        },
      }}
    >
      <Stack.Screen
        name='profil'
        component={Profil}
        options={{
          title: "Page de profil"
        }}
      />
      <Stack.Screen
        name='camera'
        component={Cam}
        options={{
          title: "Prendre une photo",
        }}
      />
    </Stack.Navigator>
  );
}
```

- **screenOptions**: Nous permet de configurer et styliser la barre de navigation ajoutée automatiquement par react.
- **options**: Nous permet de spécifier une configuration pas écran.

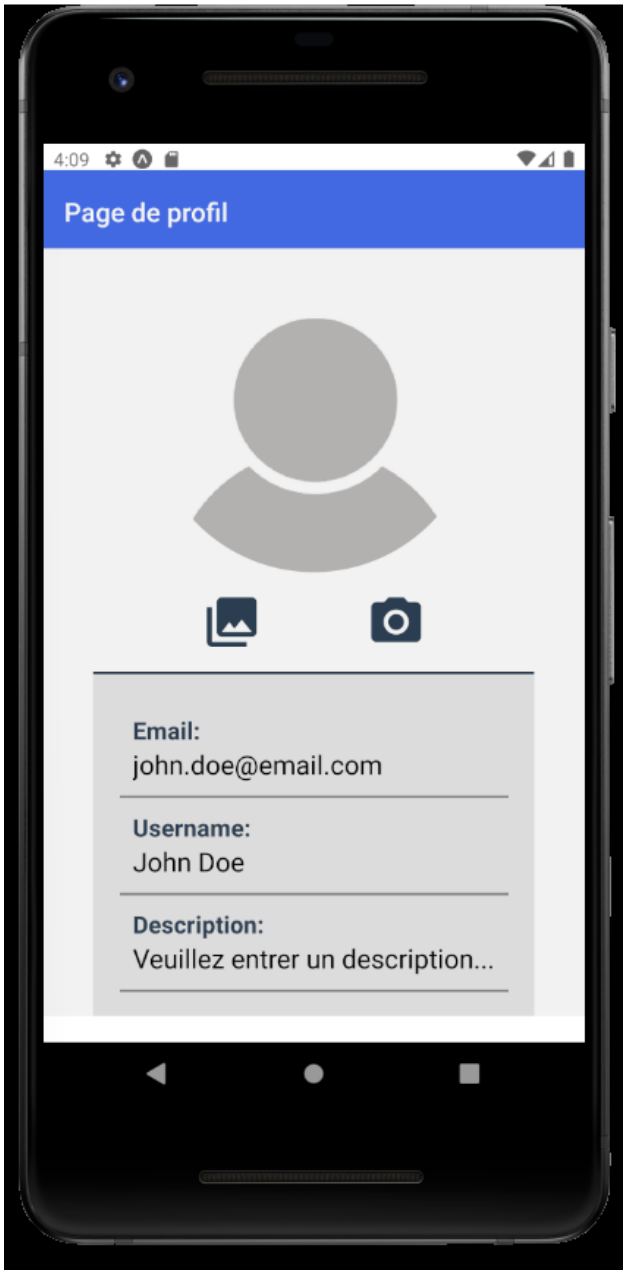
Remplacer Profil par ProfilStack:

Maintenant que notre composant stack est créé, nous allons l'afficher dans **App.js**, a la place du composant **Profil**:

```
import { useState } from "react";
import { StyleSheet, View } from "react-native";
import Auth from "./components/page/Auth/Auth";
import ProfilStack from "./components/stacks/ProfilStack";
import { UserContext } from "./contexts/UserContext";
import { NavigationContainer } from "@react-navigation/native";
export default function App() {
  const fakeUser = { email: "john.doe@email.com", username: "John Doe" };
  const [user, setUser] = useState(fakeUser);
  return (
    <UserContext.Provider value={{ user, setUser }}>
      <View style={styles.container}>
        <NavigationContainer>
          {user ? <ProfilStack /> : <Auth />}
        </NavigationContainer>
      </View>
    </UserContext.Provider>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
});
```

Pour utiliser un composant Navigator, nous devons l'englober dans un composant **NavigationContainer** importé depuis: `@react-navigation/native.`



Naviguer entre les écrans:

Les composants ajoutés à un StackNavigator, reçoivent dans leurs props l'objet navigation.

Cet objet contient plusieurs fonctions nous permettant de naviguer entre les écrans.

Nous pouvons déclarer **props** dans les paramètres de la fonction **Profil** et utiliser la fonction **push**, pour ajouter le **screen** camera au stack et ainsi l'afficher:

./component/Page/Profil/Profil.jsx:

```
<TouchableOpacity onPress={() => navigation.push("camera")}>
   <MaterialIcons name='camera-alt' size={45} color={"#2C3E50"} />
</TouchableOpacity>
```

1.5/ La caméra:

Liens de la documentation.

Installation du package:

```
expo install expo-camera
```

Demander la permission:

Pour pouvoir utiliser la caméra du téléphone, nous devons d'abord demander l'autorisation à l'utilisateur.

- Nous allons utiliser le hook **useEffect()**, afin de lancer la demande de permission quand le composant est chargé.
- La fonction asynchrone qui permet de demander la permission est importé depuis expo-camera: requestCameraPermissionsAsync()
- Elle retournera un objet contenant la clé **granted**, que nous mettrons dans une **variable d'état**, ce qui nous permettra de savoir si la demande a été acceptée ou refusée.

./compoenent/Page/Cam/Cam.jsx:

```jsx
import React, { useEffect, useState } from "react";
import { View, Text, StyleSheet } from "react-native";
import { Camera } from "expo-camera";

const Cam = ({ navigation }) => {
  const [cameraPermission, setCameraPermission] = useState(null);

  useEffect(() => {
    (async () => {
      let permission = await Camera.requestCameraPermissionsAsync();
      setCameraPermission(permission.granted);
    })();
  }, []);

  if (cameraPermission === false) {
    return (
      <View style={styles.container}>
        <Text>Permission refusée...</Text>
      </View>
    );
  }

  if (cameraPermission === null) {
    return <View style={styles.container}>Chargement...</View>;
  }

  return (
    <View style={styles.container}>
      <Camera style={{ width: "100%", height: "100%" }}></Camera>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
});

export default Cam;
```

Changer le type de caméra (avant/arrière):

- Il nous faut une variables d'états, camera Type, pour mémoriser le choix de l'utilisateur.
- Et une fonction permettant de basculer entre les types de caméras

```
const [cameraType, setCameraType] = useState(Camera.Constants.Type.back);

function toggleCamera() {
  setCameraType(
    cameraType === Camera.Constants.Type.back
      ? Camera.Constants.Type.front
      : Camera.Constants.Type.back
  );
}
```

```
<View style={styles.iconsContainer}>
   <TouchableOpacity onPress={toggleCamera}>
      <MaterialIcons
         name='flip-camera-android'
         size={50}
         color='green'
       />
   </TouchableOpacity>
 </View>
```

```
iconsContainer: {
    display: "flex",
    flexDirection: "row",
    justifyContent: "space-around",
    position: "absolute",
    bottom: 75,
    width: "100%",
},
```

Prendre une photo:

- Pour utiliser la fonction **takePictureAsync()**, il nous faut une **référence** au composant **Camera**. Nous pouvons utiliser le **hook** **useRef()**.

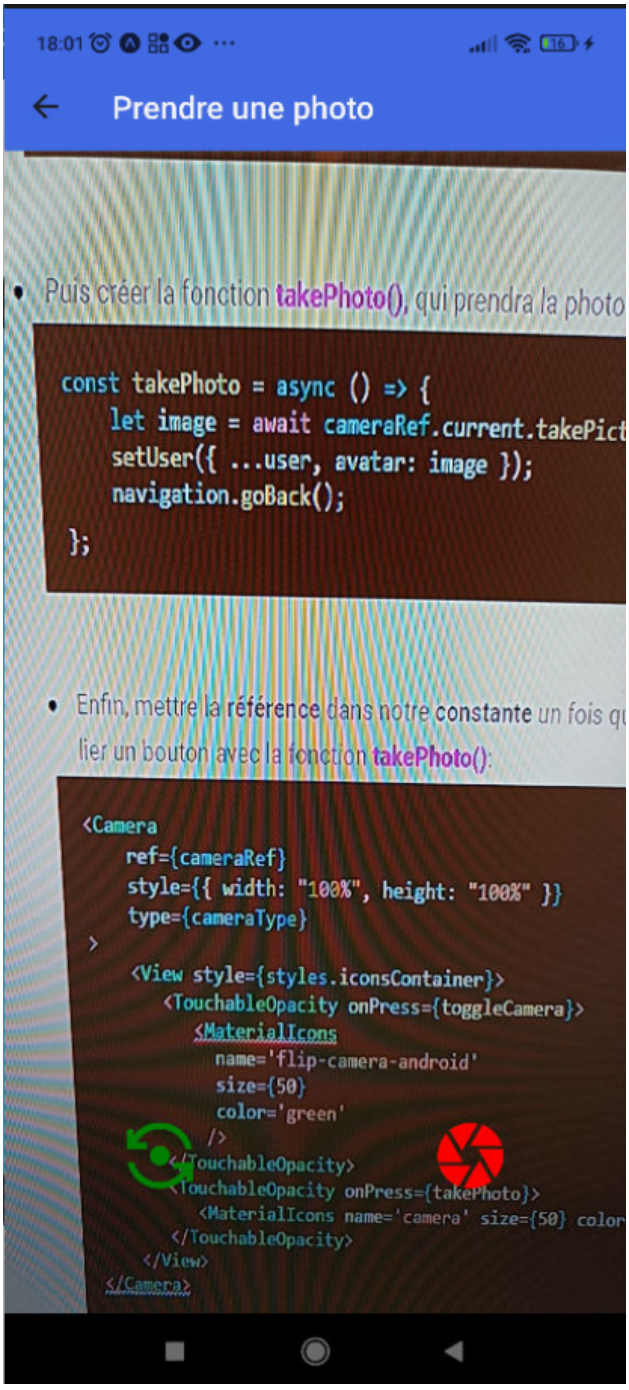
```
const cameraRef = useRef();
```

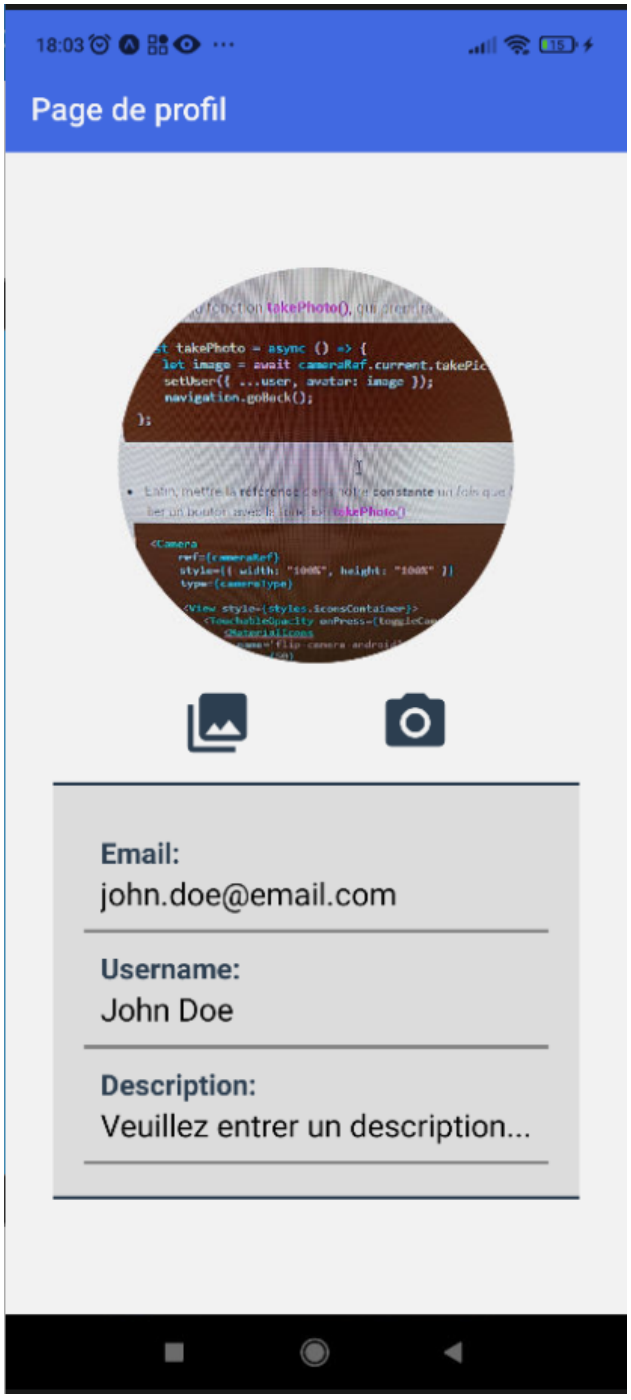
- Puis créer la fonction **takePhoto(),** qui prendra la photo et met à jour **user**:

```
const takePhoto = async () => {
    let image = await cameraRef.current.takePictureAsync();
    setUser({ ...user, avatar: image });
    navigation.goBack();
};
```

- Enfin, mettre la **référence** dans notre **constante** un fois que le composant est rendu, et lier un bouton avec la fonction **takePhoto()**:

```
<Camera
    ref={cameraRef}
    style={{ width: "100%", height: "100%" }}
    type={cameraType}
>
    <View style={styles.iconsContainer}>
       <TouchableOpacity onPress={toggleCamera}>
          <MaterialIcons
            name='flip-camera-android'
            size={50}
            color='green'
           />
       </TouchableOpacity>
       <TouchableOpacity onPress={takePhoto}>
          <MaterialIcons name='camera' size={50} color='red' />
       </TouchableOpacity>
    </View>
</Camera>
```


18:01
Prendre une photo
Puis créer la fonction takePhoto(), qui prendra la photo
const takePhoto = async () => {
let image = await cameraRef.current.takePict
setUser({ ...user, avatar: image });
navigation.goBack();
};
Enfin, mettre la référence dans notre constante un fois q
lier un bouton avec la fonction takePhoto():
<Camera
ref={cameraRef}
style={{ width: "100%", height: "100%" }}
type={cameraType}
>
<View style={styles.iconsContainer}>
<TouchableOpacity onPress={toggleCamera}>
<MaterialIcons
name='flip-camera-android'
size={50}
color='green'
/>
</TouchableOpacity>
<TouchableOpacity onPress={takePhoto}>
<MaterialIcons name='camera' size={50} color
</TouchableOpacity>
</View>
</Camera>

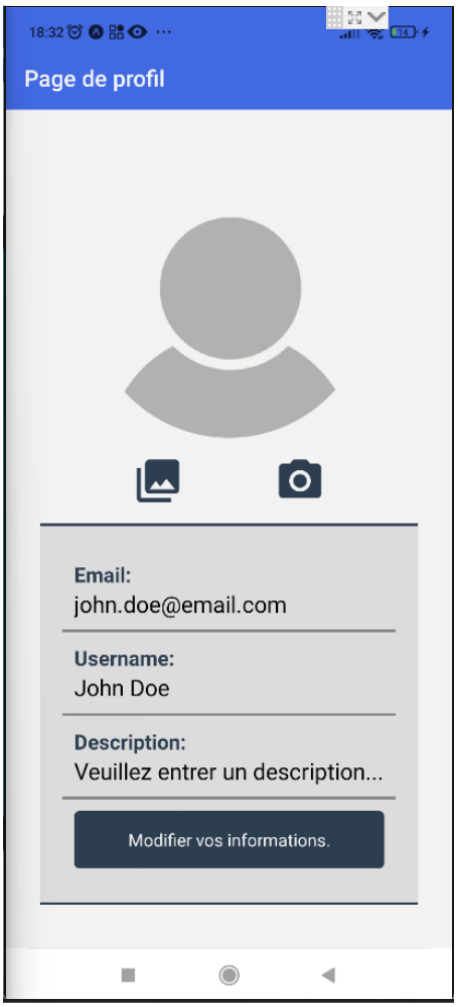

18:03
Page de profil
Email:
john.doe@email.com
Username:
John Doe
Description:
Veuillez entrer un description...

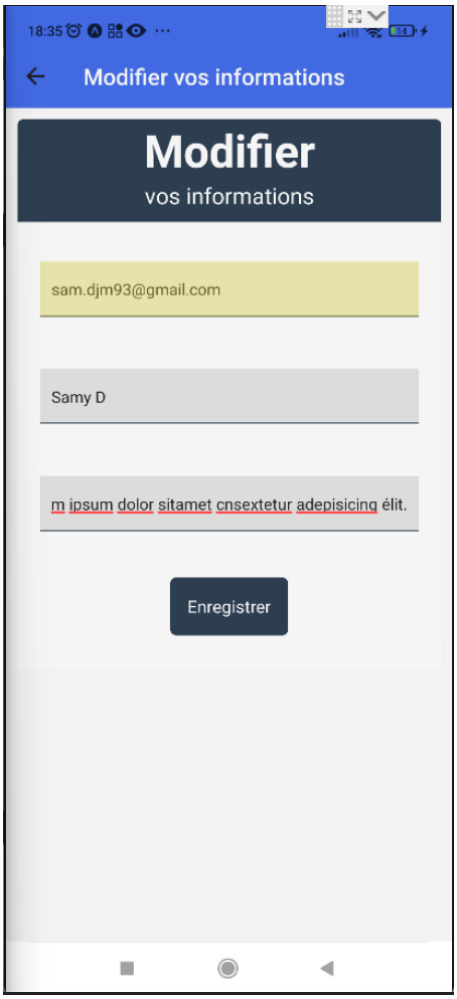
2/ Modification des informations de l'utilisateur:

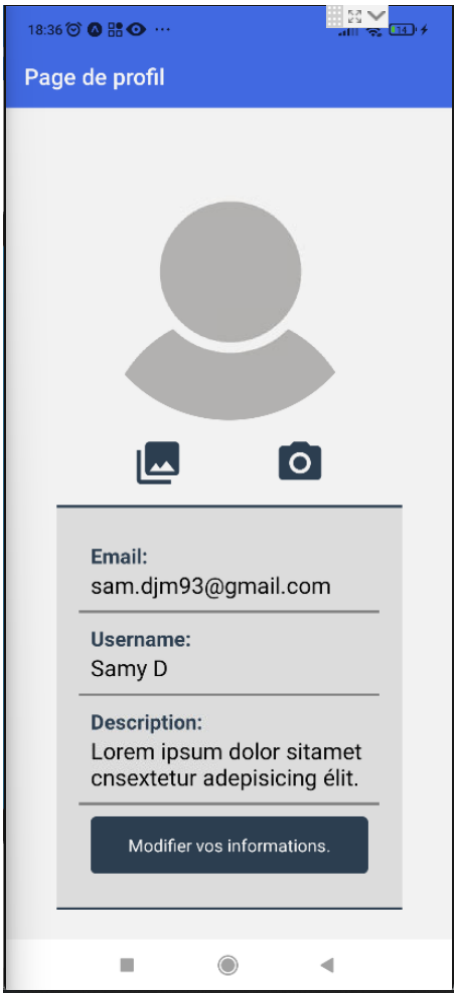
2.1/Exercice: Modification des informations de l'utilisateur:

Mettre en place un nouvel écran permettant à l'utilisateur de modifier ses informations personnelles (email, username et description).

- L'utilisateur peut naviguer vers cet écran depuis le Profil.
- Les entrées seront testées lors de la validation du formulaire.
- Une fois l'objet user mis à jour, l'utilisateur sera redirigé vers la page de profil automatiquement.
- Envoyer: **Profil.jsx, ProfilStack.jsx** et **EditInfos.jsx**







2.2/Solution: Modification des information de l'utilisateur:

```
import { useContext, useState } from "react";
import { View, Text, StyleSheet } from "react-native";
import Button from "../../ui/Button/Button";
import InputWithError from "../../ui/InputWithError/InputWithError";
import { UserContext } from "../../../contexts/UserContext";
import Card from "../../hoc/Card/Card";

const EditInfosForm = ({ navigation }) => {
  const { user, setUser } = useContext(UserContext);

  const [emailInput, setEmailInput] = useState(user.email);
  const [emailError, setEmailError] = useState("");

  const [usernameInput, setUsernameInput] = useState(user.username);
  const [usernameError, setUsernameError] = useState("");

  const [descriptionInput, setDescriptionInput] = useState(
    user.descitpion ? user.descitpion : ""
  );
  const [descriptionError, setDescriptionError] = useState("");

  function handleEmail(event) {
    setEmailInput(event);
    setEmailError("");
  }

  function handleUsername(event) {
    setUsernameInput(event);
    setUsernameError("");
  }

  function handleDescription(event) {
    setDescriptionInput(event);
    setDescriptionError("");
  }
```

```
function editInfos() {
    if (
      emailInput.includes("@") &&
      usernameInput.length >= 3 &&
      usernameInput.length < 12 &&
      descriptionInput.length <= 120
    ) {
      setUser({
        ...user,
        email: emailInput,
        username: usernameInput,
        description: descriptionInput,
      });
      navigation.goBack();
    } else {
      setEmailError(!emailInput.includes("@") ? "Email incorrect!" : "");
      setUsernameError(
        usernameInput.length < 3
          ? "Username trop court!"
          : usernameInput.length >= 12
          ? "Username trop long"
          : ""
      );
       setDescriptionError(descriptionInput.length > 120 ?
          "Description trop longue!" : "");
    }
  }

  return (
    <Card title='Modifier' content='vos informations'>
      <View style={styles.container}>
        <InputWithError
          holder='Email'
          valeur={emailInput}
          action={handleEmail}
          errorMessage={emailError}
          type='email-address'
        />

        <InputWithError
          holder='Username'
          valeur={usernameInput}
          action={handleUsername}
          errorMessage={usernameError}
          type='default'
        />
```

```
        <InputWithError
           holder='Description'
           valeur={descriptionInput}
           action={handleDescription}
           errorMessage={descriptionError}
           type='default'
         />

         <Button label='Enregistrer' action={editInfos}></Button>
       </View>
     </Card>
  );
};

const styles = StyleSheet.create({
  container: { justifyContent: "center", alignItems: "center" },
});

export default EditInfosForm;
```