React Native

Projet développement mobile

PARTIE 2: Authentifications:

Formateur : Djemai Samy

1/ Changer le Splash Screen:

Le Splash screen, est l'image affichée en ouvrant notre application:

- La taille recommandée pour un Splash screen est de 1242 px x 2436 px.
- Il nous suffit juste de remplacer l'image nommée : splash.png dans le dossier assets.
- Pour changer la couleur d'arrière-plan (blanche par défaut), il faut nous rendre dans le fichier app.json, et changer la valeur de backgroundColor

2/ Changer l'icone:

Pour changer l'icône de notre application:

- La taille recommandée pour l'icône est de 1024 px x 1024 px.
- Il nous suffit juste de remplacer l'image nommée : icon.png dans le dossier assets.
- Et de remplacer l'image adaptive-icon.png, aussi dans le dossier assets

3/ Premiers composants:

3.1/Première structure de l'application:

Tout d'abord, nous allons organiser la structure de notre application, en gardant en tête que cette structure changera au fur et à mesure de l'avancement du cours.

Notre application affichera un système d'authentification quand l'utilisateur ne s'est pas encore euthentifié, ou une page de profil si c'est le cas.

Nous devons créer deux composants: **Auth.jsx** et **Profil.jsx**.

Création du composant: **./components/page/Auth/Auth.jsx**

```
import { View, Text, StyleSheet } from "react-native";
const Auth = () => {
  return (
    <View>
      <Text>Page d'authentification</Text>
    </View>
  );
};
export default Auth;
```

Création du composant: **./components/page/Profil/Profil.jsx**

```
import { View, Text, StyleSheet } from "react-native";
const Profil = () => {
  return (
    <View>
      <Text>Page de profil</Text>
    </View>
  );
};
export default Profil;
```

Puis nous allons afficher ces deux composants au niveau de **App.jsx:**

```
import { useState } from "react";
import { StyleSheet, ScrollView } from "react-native";
import Auth from "./components/page/Auth/Auth";
import Profil from "./components/page/Profil/Profil";

export default function App() {
  const [user, setUser] = useState(null);
  return (
    <ScrollView style={styles.container}>
      {user ? <Profil /> : <Auth />}
    </ScrollView>
  );
}

const styles = StyleSheet.create({
  container: {
    backgroundColor: "rgba(0, 0, 0, 0.8)",
  },
});
```

- Nous avons créé une variable d'état nommé **user**, initialisé à **null** représentant l'utilisateur, et la fonction **setUser** permettant de modifier cette variable.
- En utilisant un opérateur ternaire, nous affichons conditionnellement:
    - Le composant **Profil.jsx** si l'utilisateur est authentifié (**user !== null**)

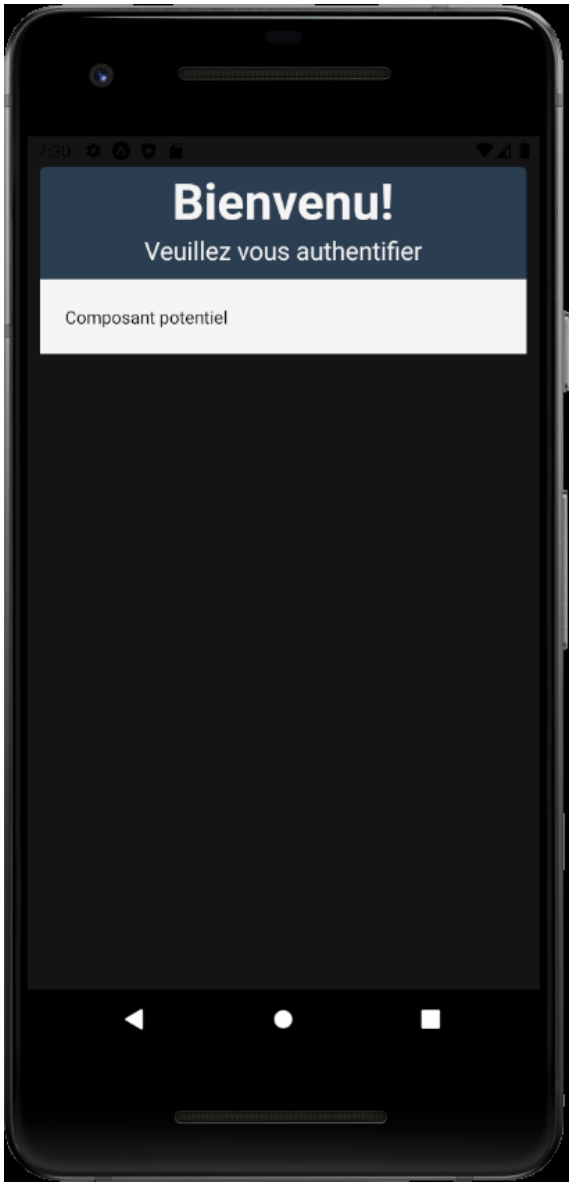
  ou,

    - Le composant **Auth.jsx** (**user === null**).

3.2/Exercice: Message de bienvenue:

Nous allons ajouter à notre code un nouveau composant, afin d'afficher un message de bienvenue aux utilisateurs.

1. Créer le composant fonctionnel : **./Components/HOC/Card/Card.jsx**
2. Le composant reçoit en paramètres : title, content, children.
3. Afficher title et content dans deux balises <Text>.
4. Afficher children dans une balise <View>.
5. Styliser le composant. (Exemple en dessous)
6. Afficher le composant **Card.jsx** dans **Auth.jsx.**



4/ Formulaires d'authentification:

4.1/Mise en place des composants:

Tout d'abord créer les deux composants:

Création du composant: **./components/container/SugnUpForm/SignUpForm.jsx**

```
import { View, Text } from "react-native";

const SignUpForm = () => {
  return (
    <View>
      <Text>SignUpForm</Text>
    </View>
  );
};
export default SignUpForm;
```

Création du composant: **./components/container/LoginForm/LoginForm.jsx**

```
import { View, Text } from "react-native";

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

export default LoginForm;
```

Puis dans le composant **Auth.jsx**:

```
import { useState } from "react";
import { StyleSheet, Text, TouchableOpacity, View } from "react-native";
import LoginForm from "../../containers/LoginForm/LoginForm";
import SignUpForm from "../../containers/SignUpForm/SignUpForm";
import Card from "../../hoc/Card/Card";

export default function Auth() {

  const [isLogin, setIsLogin] = useState(true);

  function toggleIsLogin() {
    setIsLogin(!isLogin);
  }

  return (
    <View style={styles.container}>
      <Card
        title='Bienvenue!'
        content={
          isLogin ? "Veuillez vous connecter." : "Veuillez vous inscrire."
        }
      >
        {isLogin ? <LoginForm /> : <SignUpForm />}
        <TouchableOpacity onPress={toggleIsLogin}>
          <Text>
            {isLogin
              ? "Pas encore membre? Inscrivez-vous!"
              : "Déjà membre? Connectez-vous!"}
          </Text>
        </TouchableOpacity>
      </Card>
    </View>
  );
}

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

Pour pouvoir afficher conditionnellement les formulaires:

- On utilise le hook **useState** pour déclarer une **variable d'état** nommée **isLogin** avec une valeur initiale a **true**, et la fonction **setIsLogin()**, permettant de modifier cette **variable.**
- On crée la fonction nommée **toggleIsLogin()**, pour pouvoir inverser la valeur de la **variable d'état isLogin**.
- On utilise l'**opérateur ternaire**, qui suivant l'état de la variable **isLogin**, affiche **LoginForm** ou **SignUpForm.**
- Nous avons fourni la fonction **toggleIsLogin()** a l'attribut **onPress** d'une TouchableOpacity, pour pouvoir passer d'un formulaire à l'autre.

4.2/Formulaire de connexion:

```
import React, { useState } from "react";
import {View, Text, StyleSheet, TextInput, TouchableOpacity}
from "react-native";

export default function LoginForm() {
  const [emailInput, setEmailInput] = useState("");
  const [emailError, setEmailError] = useState("");

  const [passwordInput, setPasswordInput] = useState("");
  const [passwordError, setPasswordError] = useState("");

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

  function handlePassword(event) {
    setPasswordInput(event);
    setPasswordError("");
  }

function login() {
    if (emailInput.includes("@") && passwordInput.length >= 6) {
      alert("Connexion avec succès!");
    } else {
      setEmailError(!emailInput.includes("@") ? "Email incorrect!" : "");
      setPasswordError(
        passwordInput.length < 6 ? "Mot de passe trop court!" : ""
      );
    }
  }
```

1- On déclare 4 variables d'état ainsi que leurs fonctions, pour mémoriser les entrées de l'utilisateur et les potentiels messages d'erreur à afficher.

2- On crée les fonctions permettant de mettre en mémoire les entrées de l'utilisateur dans nos variables d'états.

3- On crée la fonction **login()** pour valider le formulaire. Elle se chargera de tester les entrées de l'utilisateur et d'afficher les erreurs.

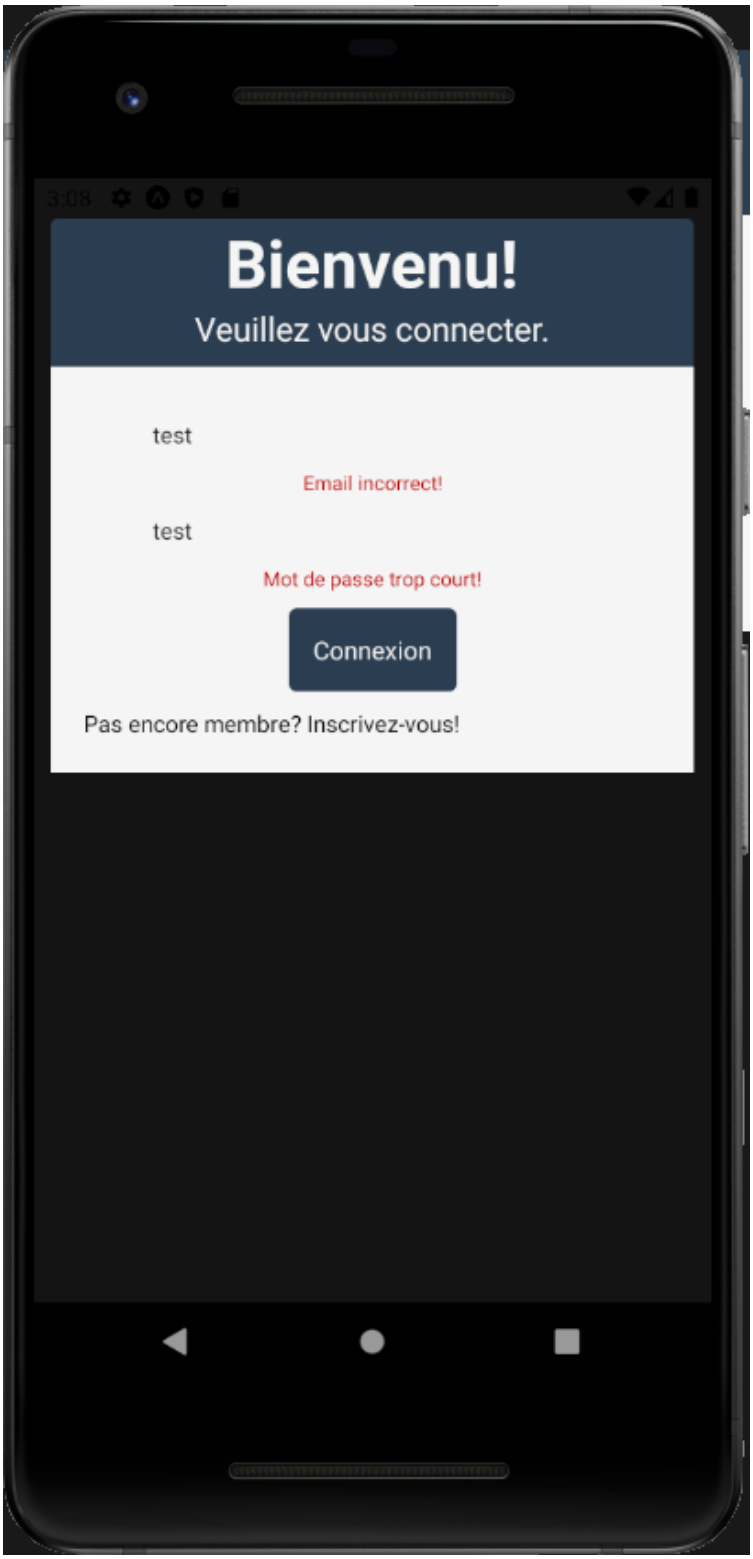
4- Dans le JSX, nous pouvons utiliser le composant **TextInput** pour afficher des entrées, et une **TouchableOpacity** pour valider le formulaire:

- **placeholder**: nous permet de spécifier du text quand le **TextInput** est vide
- **value**: nous permet de spécifier la valeur de l'entrée. (Entrée contrôlée)
- **onChageText:** nous permet de spécifier quelle fonction exécuter, quand l'utilisateur tape dans l'entrée.
- **secureTextEntry**: Nous permet de cacher le contenu de l'entrée (Pour cacher le mot de passe)

```
return (
    <View style={styles.container}>
      <TextInput
        style={styles.input}
        placeholder='Email'
        value={emailInput}
        onChangeText={handleEmail}
      />
      <Text style={styles.inputError}>{emailError}</Text>

      <TextInput
        style={styles.input}
        placeholder='Mot de passe'
        value={passwordInput}
        onChangeText={handlePassword}
      />
      <Text style={styles.inputError}>{passwordError}</Text>

      <TouchableOpacity style={styles.button} onPress={login}>
        <Text style={styles.labelButton}>Connexion</Text>
      </TouchableOpacity>
    </View>
  );
};
```

5- On ajoute du style aux composants:

```
const styles = StyleSheet.create({
  container: {
    justifyContent: "center",
    alignItems: "center",
  },
  input: {
    padding: 7,
    width: "80%",
  },
  inputError: {
    color: "rgb(200, 10, 10)",
    fontSize: 12,
  },
  button: {
    backgroundColor: "#2C3E50",
    padding: 10,
    margin: 10,
    borderRadius: 5,
    display: "flex",
    alignItems: "center",
    justifyContent: "center",
  },
  labelButton: {
    color: "whitesmoke",
    margin: 5,
    fontSize: 15,
  },
});
```


Bienvenu!
Veuillez vous connecter.
test
Email incorrect!
test
Mot de passe trop court!
Connexion
Pas encore membre? Inscrivez-vous!

4.3/ Composant Button:

Afin d'éviter la duplication de code dans notre application, nous pouvons créer nos propres composants réutilisables.

Nous allons implémenter le composant Button, qui recevra en propriété le texte a afficher dans le bouton dans l'attribut **label**, et la fonction à exécuter quand on clique dessus dans l'attribut **action**:

Dans le fichier: **./components/ui/Button/Button.jsx**

```
import { Text, StyleSheet, TouchableOpacity } from "react-native";

const Button = ({ label, action }) => {
  return (
    <TouchableOpacity style={styles.button} onPress={action}>
      <Text style={styles.labelButton}>{label}</Text>
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  button: {
    backgroundColor: "#2C3E50",
    padding: 10,
    margin: 10,
    borderRadius: 5,
    display: "flex",
    alignItems: "center",
    justifyContent: "center",
  },
  labelButton: {
    color: "whitesmoke",
    margin: 5,
    fontSize: 15,
  },
});

export default Button;
```

Dans le fichier: **./components/container/LoginForm/LoginForm.jsx**

```
return (
    <View style={styles.container}>
      <TextInput
        style={styles.input}
        placeholder='Email'
        value={emailInput}
        onChangeText={handleEmail}
      />
      <Text style={styles.inputError}>{emailError}</Text>

      <TextInput
        style={styles.input}
        placeholder='Mot de passe'
        value={passwordInput}
        onChangeText={handlePassword}
      />
      <Text style={styles.inputError}>{passwordError}</Text>

      <Button label={"Connexion"} action={login} />

    </View>
  );
};
```

4.4/ Exercice: Composant InputWithError:

Créer le composant fonctionnel : **./Components/UI/InputWithError/InputWithError.jsx**

Exemple d'utilisation dans LoginForm.jsx:

```
<InputWithError
   holder='Email'
   valeur={emailInput}
   action={handleEmail}
   errorMessage={emailError}
   type='email-address'
/>

<InputWithError
   holder='Mot de passe'
   valeur={passwordInput}
   action={handlePassword}
   errorMessage={passwordError}
   type='default'
   isPassword
/>
```

4.5/ Solution: Composant InputWithError:

```
import { Text, StyleSheet, TextInput } from "react-native";
export default functionInputWithError({
  holder,
  valeur,
  action,
  errorMessage,
  isPassword,
  type,
}){
  return (
    <>
      <TextInput
        style={styles.input}
        placeholder={holder}
        value={valeur}
        onChangeText={action}
        secureTextEntry={isPassword}
        keyboardType={type}
      />
      <Text style={styles.inputError}>{errorMessage}</Text>
    </>
  );
};
const styles = StyleSheet.create({
  input: {
    backgroundColor: "rgb(220,220,220)",
    padding: 10,
    width: "100%",
    margin: 15,
    backgroundColor: "rgb(220,220,220)",
    borderBottomColor: "#2C3E50",
    borderBottomWidth: 1,
  },
  inputError: {
    color: "rgb(200, 10, 10)",
    fontSize: 12,
  },
});
```

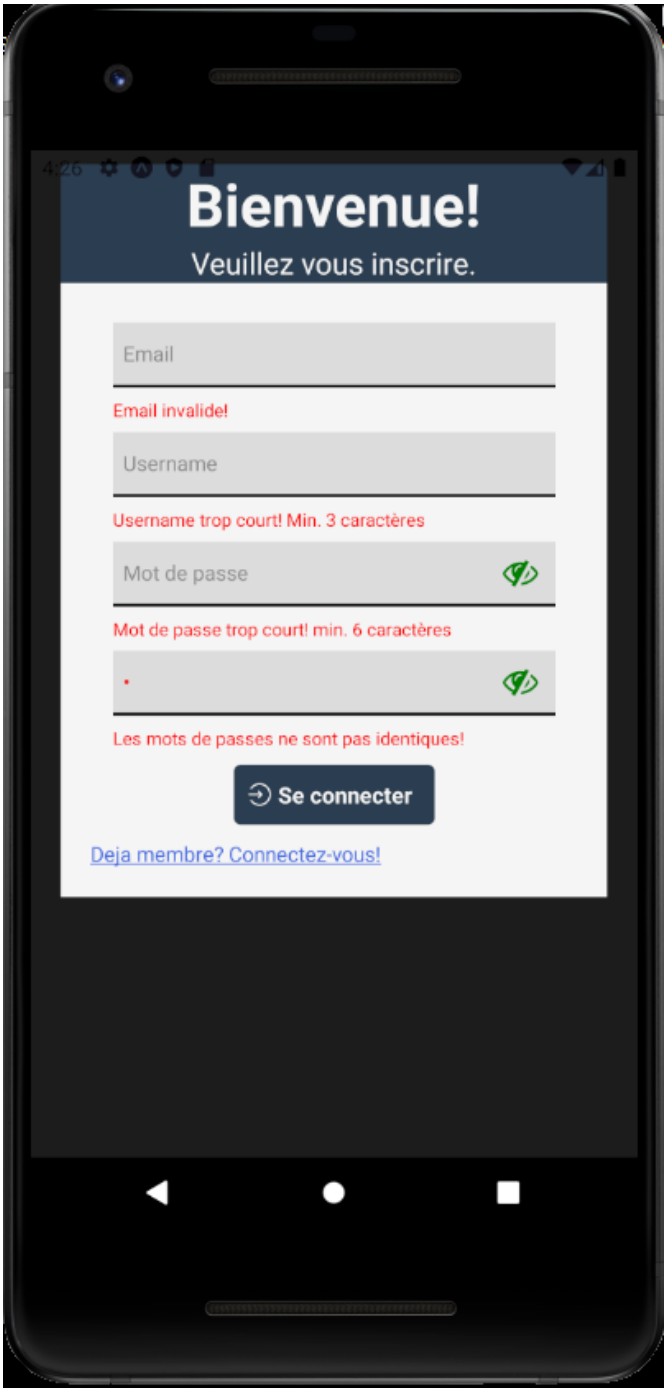
4.6/Exercice: Formulaire d'inscription:

Implémenter le formulaire d'inscription avec 4 inputs et un bouton :

- email: possède un @.
- username: minimum 3 caractères, et maximum 12 caractères
- password: minimum 6 caractères
- confirmPassword: identique à password (confirmPassword == password)

Le bouton exécute la fonction signup:

- teste les entrées de l'utilisateur.
- affiche les erreurs.



4.7/Solution: Formulaire d'inscription:

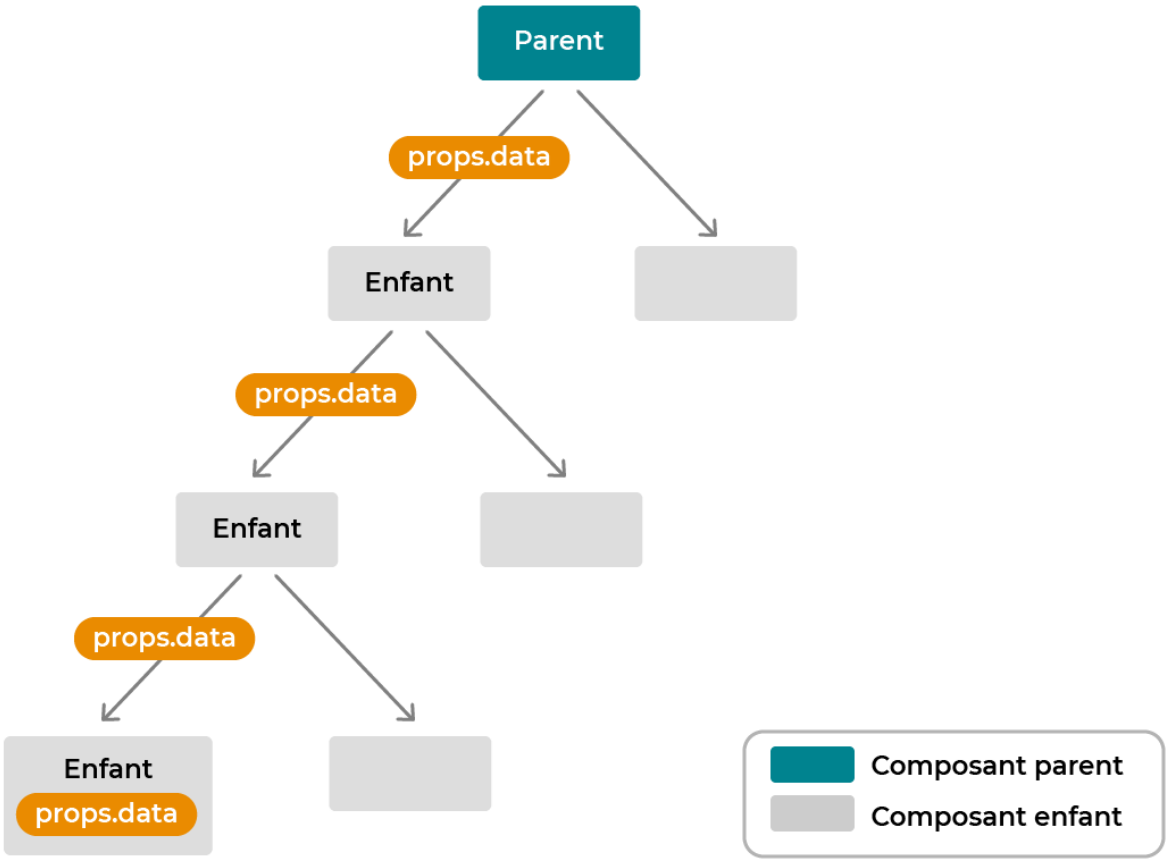
```
import React, { Component, useContext, useState } from "react";
import { View, Text, StyleSheet } from "react-native";
import Button from "../../UI/Button/Button";
import InputWithError from "../../UI/InputWithError/InputWithError";
import { FontAwesome5 } from "@expo/vector-icons";
import { UserContext } from "../../../Contexts/UserContext";

const SignUpForm = () => {
  const userContext = useContext(UserContext);

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

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

  const [passwordInput, setPasswordInput] = useState("");
  const [passwordError, setPasswordError] = useState("");

  const [confirmPasswordInput, setConfirmPasswordInput] = useState("");
  const [confirmPasswordError, setConfirmPasswordError] = useState("");

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

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

  function handlePassword(t) {
    setPasswordInput(t);
    setPasswordError("");
  }

  function handleConfirmPassword(t) {
    setConfirmPasswordInput(t);
    setConfirmPasswordError("");
  }
```

```
function signup() {
    if (
      emailInput.includes("@") &&
      usernameInput.length >= 3 &&
      usernameInput.length <= 12 &&
      passwordInput.length >= 6 &&
      confirmPasswordInput === passwordInput
    ) {
      userContext.setUser({ email: emailInput, username: usernameInput });
    } else {
      setEmailError(!emailInput.includes("@") ? "Email invalide!" : "");

      setUsernameError(
        usernameInput.length < 3
          ? "Username trop court!"
          : usernameInput.length > 12
          ? "Username trop long!"
          : ""
      );
      setPasswordError(
        passwordInput.length < 6 ? "Mot de passe trop court!" : ""
      );
      setConfirmPasswordError(
        confirmPasswordInput !== passwordInput
          ? "Les mot de passe ne sont pas identiques"
          : ""
      );
    }
  }

  return (
    <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='email-address'
      />
```

```
      <InputWithError
        holder='Mot de passe'
        valeur={passwordInput}
        action={handlePassword}
        errorMessage={passwordError}
        type='default'
        isPassword
      />

      <InputWithError
        holder='Confirmez votre mot de passe'
        valeur={confirmPasswordInput}
        action={handleConfirmPassword}
        errorMessage={confirmPasswordError}
        type='default'
        isPassword
      />
      <Button action={signup}>
        <FontAwesome5 name='sign-in-alt' size={20} color='whitesmoke' />
        <Text
          style={{
            color: "whitesmoke",
            marginHorizontal: 5,
            fontSize: 20,
          }}
        >
          Valider
        </Text>
      </Button>
    </View>
  );
};

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

export default SignUpForm;
```

5/ Le contexte:

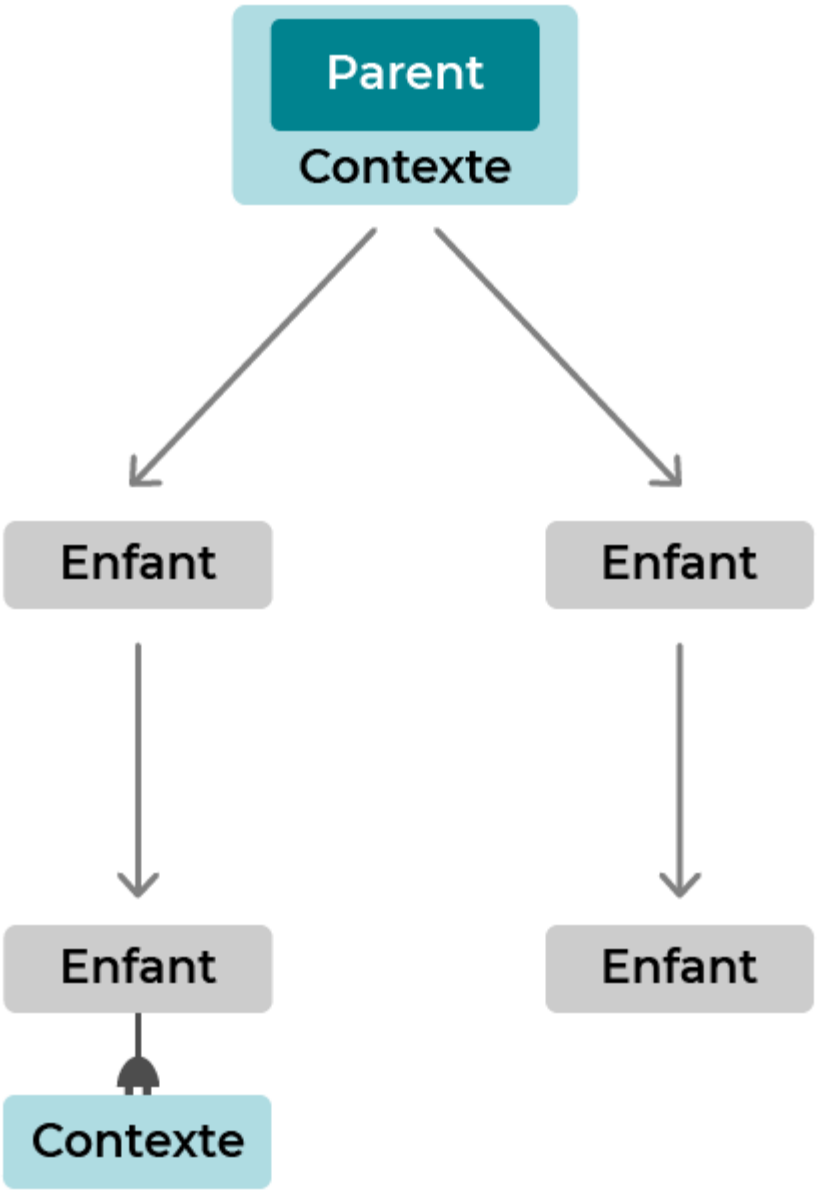
Il arrive que plusieurs composants aient besoin d'accéder à des variables ou fonctions déclarer dans un parent commun:



Afin d'éviter un cascade de propriétés, et de passer des propriétés à des composants qui ne l'utilisent pas. Nous pouvons utiliser le **contexte**.

Le contexte nous permet de récupérer simplement nos datas sans avoir à tout passer manuellement.

Tous les composants enfants pourront alors se connecter au Provider et ainsi accéder aux propriétés, de manière globale. On dit que les composants enfants sont les **consumers**.



5.1/Création du contexte:

Nous allons commencer par créer notre Provider/Fournisseur de contexte dédié à la variable d'états user.

Pour cela, nous allons créer un dossier dédié au contextes: **./contexts**

Puis le fichier: **./contexts/UserContext.js**

```
import { createContext } from "react";

export const UserContext = createContext();
```

On commence par importer la fonction **createContext()** depuis react.

Puis on initialise et exporte notre contexte dans une constante **UserContext**.

5.2/ Le Provider:

On peut maintenant l'utiliser dans **App.js**, après l'avoir importé:

```
import { useState } from "react";
import { StyleSheet, Text, View, ScrollView } from "react-native";
import Auth from "./components/page/Auth/Auth";
import Profil from "./components/page/Profil/Profil";
import { UserContext } from "./contexts/UserContext";

export default function App() {
  const [user, setUser] = useState(null);
  return (
    <UserContext.Provider value={{ user, setUser }}>
      <View style={styles.container}>{user ? <Profil /> : <Auth />}</View>
    </UserContext.Provider>
  );
}

const styles = StyleSheet.create({
  container: {
    paddingVertical: 15,
    backgroundColor: "rgba(0, 0, 0, 0.9)",
  },
});
```

On englobe toute notre application avec le **Provider**, afin de rendre accessible la variable **user** et la fonction **setUser()**, à tous les enfants de App.js.

5.3/ Le Consumer:

Nous pouvons maintenant utiliser le **hook** **useContext()**, qui permet de se “brancher” depuis un composant enfant qui a été englobé par un Provider, et donc d’accéder aux données partagées.

Dans : **./components/container/LoginForm/LoginForm.jsx** :

```
const userContext = useContext(UserContext);
```

Puis dans la fonction **login():**

```
if (emailInput.includes("@") &&
    usernameInput.length >= 3 &&
    usernameInput.length < 12 &&
    passwordInput.length >= 6 &&
    confirmPasswordInput === passwordInput
    ) {
      userContext.setUser({
        email: emailInput,
        username: usernameInput,
      });
    }
```

Maintenant, quand l’utilisateur s’inscrit et que les entrées passent les tests, nous utilisons la fonction **setUser()** accessible depuis le contexte, pour affecter a **user** un objet contenant l’email et le username.

Ce qui a pour conséquence d’afficher le composant **Profil**, car dans l’**opérateur ternaire** de **App.js**, **la variable d’état user** n’est plus **null.**

5.4/ Exercice: Informations de l'utilisateur:

Dans **./components/page/Profil/Profil.jsx**, utiliser le contexte pour:

- Afficher l'**email** de l'utilisateur.
- Afficher le **username** de l'utilisateur.
- Afficher la **description** de l'utilisateur si l'objet **user** en a une; Sinon afficher un texte par défaut (Utiliser un opérateur ternaire.).

5.5/ Solution: Informations de l'utilisateur:

```
import { useContext } from "react";
import { View, Text, StyleSheet, ScrollView } from "react-native";
import { UserContext } from "../../../contexts/UserContext";

export default function Profil(){
  const { user, setUser } = useContext(UserContext);
  return (
    <ScrollView contentContainerStyle={styles.container}>
      <View style={styles.infosContainer}>
        <View style={styles.infoContainer}>
          <Text style={styles.infoLabel}>Email:</Text>
          <Text style={styles.info}>{user.email}</Text>
        </View>
        <View style={styles.infoContainer}>
          <Text style={styles.infoLabel}>Username:</Text>
          <Text style={styles.info}>{user.username}</Text>
        </View>
        <View style={styles.infoContainer}>
          <Text style={styles.infoLabel}>Description:</Text>
          <Text style={styles.info}>
            {user.description
              ? user.description
              : "Veuillez entrer un description..."}
          </Text>
        </View>
      </View>
    </ScrollView>
  );
};
```

```
const styles = StyleSheet.create({
  container: {
    justifyContent: "center",
    alignItems: "center",
    height: "100%",
  },
  infosContainer: {
    borderBottomColor: "#2C3E50",
    borderBottomWidth: 2,
    borderTopColor: "#2C3E50",
    borderTopWidth: 2,
    padding: 20,
    backgroundColor: "rgb(220,220,220)",
  },
  infoContainer: {
    borderBottomColor: "gray",
    borderBottomWidth: 2,
    padding: 10,
  },
  infoLabel: { color: "#2C3E50", fontSize: 18, fontWeight: "bold" },
  info: { color: "black", fontSize: 20 },
});
```