

Cours De React Native

SANKARA Sarata & TOURE Chabane

28 mars 2025

Table des matières

0.1	Qu'est-ce que React native ?	2
0.2	Création d'un Projet React Native	2
0.3	Rôle des Fichiers et Dossiers du Projet	2
0.4	Démarrage et Exécution du Projet	2
1	Composants	2
1.1	Qu'est ce qu'un composant	2
1.2	Les types composants	4
2	Les routes,Prop et state	5
2.1	Prop	5
2.2	state	5
2.3	Différences entre props et state	5
3	Description du projet	5
3.1	Architecture Du Projet	5
3.2	L'implémentation de projet et explication	5
3.2.1	Provider	5
3.2.2	Navigation	6
4	Élaboration Des Pages	9
4.1	La Page Index	9
4.2	Les Pages : Register, Login, addCours, editCours, ListCours	9
5	Le déploiement	22
5.1	Installer EAS CLI à l'échelle mondiale	22
5.2	Vérifier l'installation EAS	23
5.3	réuez et connectez-vous à votre compte EXPO	23
5.4	création d'une version de développement	23
5.5	Exécution avec l'application Expo Go	25
6	Conclusion	26

Introduction à React native

0.1 Qu'est-ce que React native ?

React native est un framework puissant pour créer des applications mobiles multi-plateformes avec JavaScript et React. Il a été développé par facebook

0.2 Création d'un Projet React Native

— :Pour créer un projet React Native, utilisez les commandes suivantes.

— `npx create-expo-app@latest`

0.3 Rôle des Fichiers et Dossiers du Projet

Voici une description des principaux fichiers et dossiers générés :

- **le dossier App** : est le coeur de notre application il contient le fichier `index.tsx` qui est le point d'entrée de l'application. Tout les fichiers qui sont dans ce dossier sont des navigations.
- **Le dossier assets** : Contient les images et fichiers statiques.
- **node modules** : Contient les dépendances du projet.
- **Le dossier components** : Contient les composants du projet.
- **Le dossier constantes** : Contient les constantes du projet.
- **app.json** : Configuration de l'application. • **package.json** : Liste des dépendances et configurations.
- **Le dossier script** : contient les scripts de notre projet
- **babel.config.js** : Configuration de Babel pour transpiler le code.
- **package-lock.json** : Assure la cohérence des versions des dépendances.

0.4 Démarrage et Exécution du Projet

Commandes pour démarrer le projet :

`npm run start` : Démarre le projet ReactNative .

`npm run android` : Démarre le projet sur Android .

`npm run web` : Démarre le projet sur le web

1 Composants

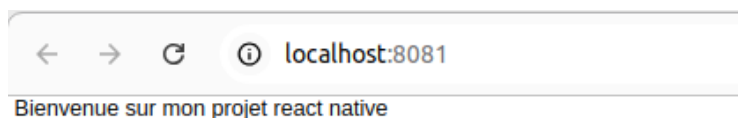
1.1 Qu'est ce qu'un composant

Un composant en React Native est une unité de code représentant une partie de l'interface utilisateur. Quelques composants natifs de react native :

- ✚ **Le composant Text** : sert à afficher des chaines de texte et gère les événements tactiles. Fourni par React Native, il est analogue à la balise `p` en HTML.

```
1 import React from 'react';
2 import { Text, View } from 'react-native';
3 export default function Home() {
4   return (
5     <View>
6       <Text> Bienvenue sur mon projet react native</Text>
7     </View>
8   )
9 }
```

8 `npm install @apollo/server graphql`



1. **Le composant StyleSheet** : Module utilisé pour créer et gérer les styles.
Avantages :
Organisation : Les styles sont centralisés.
Validation : Les propriétés des styles sont vérifiées.
Réutilisabilité : Les styles peuvent être appliqués à plusieurs composants.
Clarté : Les styles sont séparés de la logique. Les noms des propriétés suivent le format camelCase (ex. backgroundColor)
1. **Le composant Image** sert à afficher différents types d'images. L'attribut source prend uri, qui contient le chemin de l'image, et on peut avoir l'attribut style pour personnaliser l'affichage de l'image.

```
1 import React from 'react';
2 import { Text, View, Image, StyleSheet } from 'react-native';
3 export default function Home() {
4   return (
5     <View style={styles.container}>
6       <Text>Bienvenue sur mon projet React Native</Text>
7       <Image style={styles.image} source={{ uri: 'https://reactnative.dev/img/tiny_logo.png' }} />
8     </View>
9   );
10 }
11
12 const styles = StyleSheet.create({
13   container: {
14     flex: 1,
15     alignItems: 'center',
16     justifyContent: 'center',
17   },
18   image: {
19     width: 50,
20     height: 50,
21   },
22 });
```

Bienvenue sur mon projet React Native



1. **Le composant Link** : est utilisé pour naviguer entre les écrans

```
1 import { Link } from 'expo-router';
2 import React from 'react';
3 import { Text, View, Image, StyleSheet } from 'react-native';
4 export default function Home() {
5   return (
6     <View style={styles.container}>
7       <Text>Bienvenue sur mon projet React Native</Text>
8       <Image style={styles.image} source={{ uri: 'https://reactnative.dev/img/tiny_logo.png' }} />
9       <Link href="/propos">
10         <Text>a propos</Text>
11       </Link>
12     </View>
13   );
14 }
15
16 const styles = StyleSheet.create({
17   container: {
18     flex: 1,
19     alignItems: 'center',
20     justifyContent: 'center',
21   },
22   image: {
23     width: 50,
24     height: 50,
25   },
26 });
```

Bienvenue sur mon projet React Native



a propos de nous

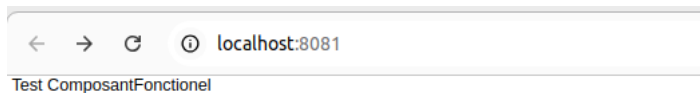
1. [Le composant View](#) : Conteneur prenant en charge la mise en page, le style et les contrôles d'accessibilité. Analogue à la balise div en HTML.

1.2 Les types composants

Nous avons deux types de composants les composants fonctionnels et les composants de classe

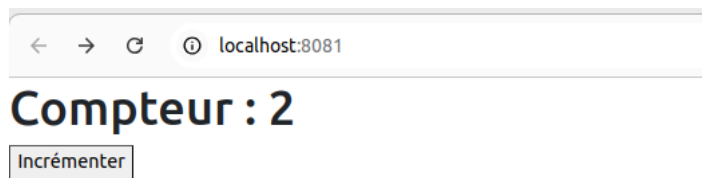
1. [Les composants fonctionnels](#) : ce sont des fonctions qui retournent du JavaScript XML (JSX) ou TypeScript XML (TSX). Exemple

```
1 import React from 'react';
2 import { StyleSheet, Text, View } from 'react-native';
3 export default function ComposantFonctionnel() {
4   return (
5     <View style={styles.container}>
6       <Text> Test ComposantFonctionnel </Text>
7     </View>
8   );
9 }
```



1. [Les composants de classe](#) : ce sont des classes qui héritent de la classe Component. Exemple

```
1 import React, { Component } from 'react';
2
3 class Compteur extends Component {
4   constructor(props) {
5     super(props);
6     this.state = { count: 0 };
7
8     incrementer = () => {
9       this.setState({ count: this.state.count + 1 }); // Modification du state
10    };
11
12    render() {
13      return (
14        <div>
15          <h1>Compteur : {this.state.count}</h1>
16          <button onClick={this.incrementer}>Incr menter </button>
17        </div>
18      );
19    }
20  }
21
22  export default Compteur;
```



Les composants fonctionnels sont plus utilisés que les composants de classe

2 Les routes, Prop et state

En react native un fichier qui se trouve dans le dossier app est une route

Prop et state

2.1 Prop

Les props : Transmettent des informations à un composant

2.2 state

Le state : Variable interne qui existe à l'intérieur d'un composant

2.3 Différences entre props et state

Props :

Données immuables.

Transmises d'un composant à un autre

State

Données mutables.

Utilisées uniquement à l'intérieur du composant

Nous allons les manipuler dans le projet

3 Description du projet

Il s'agit d'un projet de **learning** qui consiste à fournir une plateforme qui propose des cours, des quiz et dont on peut s'inscrire pour suivre des cours et passer des quiz pour les cours.

3.1 Architecture Du Projet

- **app** :

– **index.tsx** : c'est le point d'entrée de notre application.

- **composant** :

– **screens.tsx** : c'est un fichier qui définit les composants (fonction qui appelle les composants pour former des écrans) pour nos écrans

– **auth** :

– **register.tsx** : c'est un fichier qui permet de créer un compte

– **login.tsx** : Cet fichier permet de se connecter / s'authentifier .

– **AuthProvider.tsx** : Cet fichier définit un provider

– **cours** :

– **tabCours.tsx** : C'est un fichier qui permet de lister les cours.

– **addCours.tsx** : C'est un fichier qui permet l'ajout des cours

– **editCours.tsx** : C'est un fichier qui permet la modification des cours

3.2 L'implémentation de projet et explication

3.2.1 Provider

Définir un provider pour gérer l'authentification

Provider permet de transmettre les données du contexte aux composants enfants. Dans notre cas, il va permettre de rendre les

informations de l'utilisateur connecté dans tous les composants
createContext permet de créer un contexte que les composants peuvent fournir ou lire :
const SomeContext = createContext(defaultValue)

```
1 import React, { createContext, useState, useEffect, ReactNode } from 'react';
2 const initialValues = {
3   email: "",
4   token: "",
5   authorities: "",
6 };
7 export const AuthContext = createContext();
8
9 export const AuthProvider = ({ children }: {children : ReactNode}) => {
10
11   const [user, setUser] = useState(initialValues);
12   return (
13     <AuthContext.Provider value={{ user, setUser }}>
14       {children}
15     </AuthContext.Provider>
16   );
17 };
```

Un **provider** ne transmet les données du context aux composants qu'il englobe (composant enfant) et c'est **children** qui fait qu'il doit prendre des composants enfants . Dans notre cas on a definit le provider dans le fichier **AuthProvider.tsx**

3.2.2 Navigation

* Le fichier **index.tsx** :

C'est dans cet fichier on déclare nos écrans grâce au **navigation** (c'est une bibliothèque qui permet de configurer les écrans d'une application) . Pour cet faire , il faut d'abord installer les dépendances et voila les commandes qui le permettent :

npm install @react-navigation/native @react-navigation/native-stack
npx expo install react-native-screens react-native-safe-area-context

Code du fichier index :

```
1
2
3 import { StyleSheet } from 'react-native';
4 import { AuthProvider } from '@context/AuthProvider';
5 import { createNativeStackNavigator } from '@react-navigation/native-stack';
6 //import { CreationCoursScreen, HomeScreen, listeCours, loginScreen, registerScreen, UpdateCours } from '@components/navigation';
7 import { LoginScreen, RegisterScren, HomeScreen, CoursScreen, CreationCoursScreen, UpdateCoursScreen } from '@components/screens';
8
9 const Stack = createNativeStackNavigator();
10
11 export default function Home() {
12   return (
13     <AuthProvider>
14       <Stack.Navigator>
15         <Stack.Screen name="index" component={HomeScreen}></Stack.Screen>
16         <Stack.Screen name="login" component={LoginScreen}></Stack.Screen>
17         <Stack.Screen name="register" component={RegisterScren}></Stack.Screen>
18         <Stack.Screen name="cours" component={CoursScreen}></Stack.Screen>
19         <Stack.Screen name="addCours" component={CreationCoursScreen}></Stack.Screen>
20         <Stack.Screen name="editCours" component={UpdateCoursScreen}></Stack.Screen>
21
22       </Stack.Navigator>
23     </AuthProvider>
24
25   )
26 }
27
28 }
```

Explication :

createNativeStackNavigator : Cette fonction permet de créer un stack navigator, un système de navigation qui empile les écrans les uns sur les autres.

Stack.Screen : permet de créer un écran , qui prend des props tels que

name : permet de scpécifier le nom de l'écran,

component indique le composant qui va afficher si l'on évoque l'écran.

`<Stack.Navigator>` : Ce composant englobe tous les écrans de l'application.

```
<Stack.Screen name="index" component={HomeScreen}></Stack.Screen>
```

Définit un écran nommé "index", qui affiche le composant HomeScreen.

```
<Stack.Screen name="login" component={LoginScreen}></Stack.Screen>
```

Définit l'écran "login", qui affiche le composant LoginScreen.

```
<Stack.Screen name="register" component={RegisterScreen}></Stack.Screen>
```

Définit l'écran "register", qui affiche le composant RegisterScreen

```
<Stack.Screen name="cours" component={CoursScreen}></Stack.Screen>
```

Définit l'écran "cours", qui affiche le composant CoursScreen.

```
<Stack.Screen name="addCours" component={CreationCoursScreen}></Stack.Screen>
```

Définit l'écran "addCours", qui affiche le composant CreationCoursScreen.

```
<Stack.Screen name="editCours" component={UpdateCoursScreen}></Stack.Screen>
```

Définit l'écran "editCours", qui affiche le composant UpdateCoursScreen.

* Le fichier **screens.tsx** :

C'est dans cet fichier qu'on définit des fonctions (composants) qui permettent d'appeler les composants pour les regroupés par écrans. Et ces fonctions prennent en paramètre la méthode **navigation** (qui permet de naviguer entre les écrans) fournit par les dépendances installer ci dessus.

Par exemple l'écran **index**, cet écran a pour **component** le composant **HomeScreen** , cet composant a été défini dans le fichier **screens.tsx** qui contient deux boutons, le bouton se connecter et le bouton Créer Un Compte .Si on clique sur se connecter l'écran login sera appelé et si c'est sur Créer Un Compte l'écran register est appelé :

Code :

```
1
2 // Definition du composant HomeScreen
3 export function HomeScreen({ navigation }) {
4
5     return (
6         <View >
7             // Cet bouton permet d'aller sur la page de login grace a navigation
8             <Button
9                 title="Se Connecter"
10                 onPress={() => navigation.navigate('login')}
11             />
12             <Text>ou</Text>
13             // Cet bouton permet d'aller sur la page de register
14             <Button
15                 title="Créer Un Compte"
16                 onPress={() => navigation.navigate('register')}
17             />
18         </View>
19     );
20 }
21
```

navigation.navigate('login') permet de naviguer vers l'écran login et navigation.navigate('register') permet de naviguer vers l'écran register

L'écran **login**, cet écran a pour **composant** le composant **LoginScreen** , cet composant a été défini dans le fichier **screens.tsx** qui conteint le formulaire de login definit dans le fichier login.tsx :

```
1
2 // Definition du composant LoginScreen
3 import FormulaireLogin from './auth/login';
4
5 export function LoginScreen({navigation}) {
6   return (
7     <View>
8       <FormulaireLogin navigation={navigation}> </FormulaireLogin>
9     </View>
10   );
11 }
```

L'écran **register**, cet écran a pour **composant** le composant **RegisterScreen** , cet composant a été défini dans le fichier **screens.tsx** qui conteint le formulaire de register definit dans le fichier register.tsx :

Voici exemple du composant **RegisterScreen** :

```
1
2 // Definition du composant RegisterScreen
3 import FormulaireRegister from './auth/register';
4
5 export function RegisterScreen({navigation}){
6   return(
7     <View>
8       <Text>
9         Inscire
10      </Text>
11      <br />
12      <FormulaireRegister navigation={navigation}></FormulaireRegister>
13    </View>
14  )
15 }
```

L'écran **cours**, cet écran a pour **composant** le composant **ListeCours** , cet composant a été défini dans le fichier **screens.tsx** qui conteint le composant ListeCours (definit dans le fichier tabCours.tsx) qui affiche une liste de cours :

```
1
2 // Definition du composant CoursScreen
3 import ListeCours from './cours/tabCour';
4
5 export function CoursScreen({navigation}){
6   return (
7     <View>
8       <ListeCours navigation={navigation}></ListeCours>
9     </View>
10   );
11 }
```

L'écran **addCours**, cet écran a pour **composant** le composant **Formulaire** , cet composant a été défini dans le fichier **screens.tsx** qui conteint le formulaire de addCours definit dans le fichier addCours.tsx :

```
1
2 // Definition du composant UpdateCoursScreen
3
4 import Formulaire from './cours/addCours';
5
6 export function CreationCoursScreen({ navigation }) {
7   return (
8     <View>
9
10     <Formulaire navigation={navigation} />
11
12     </View>
13   );
14 }
```

L'écran **editCours**, cet écran a pour **composant** le composant **UpdateCoursScreen** , cet composant a été défini dans le fichier **screens.tsx** qui conteint le formulaire de editCours definit dans le fichier editCours.tsx :


```

1 // Definition du composant UpdateCoursScreen
2 import ModifierCours from '../cours/editCours';
3
4
5 export function UpdateCoursScreen(props) {
6   const {navigation, route} =props
7   return (
8     <View>
9       <ModifierCours navigation={navigation} route={route}></ModifierCours>
10    </View>
11  );
12 }

```

La plus part des fonctions definient dans le fichier screens.tsx appellent les composants et on voit que ces composants prennent un props navigation (c'est un objet qui permet de naviguer entre les pages) et un props route (qui permet de recuperer les données depuis de l'url).

Exemple :

```

<FormulaireRegister navigation=navigation></FormulaireRegister>
<ModifierCours navigation=navigation route=route></ModifierCours>

```

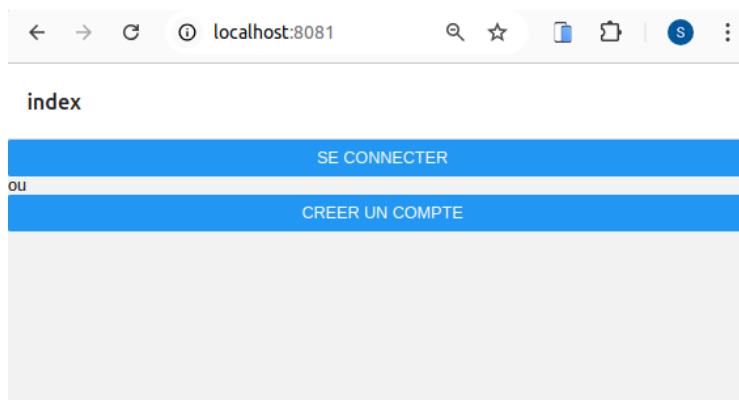
4 Élaboration Des Pages

On distingue plusieurs pages qui sont :

4.1 La Page Index

Il s'agit de la page d'accueil qui affiche les boutons **Creer Un Compte** et **Se Connecter** (C'est l'ecran index qui a pour component HomeScreen definit dans le fichier screens.tsx . Donc le contenu de cette page s'agit du composant HomeScreen).

L'image de la page Index :



4.2 Les Pages : Register, Login, addCours, editCours, ListCours

**** La Page Register**

Cette page permet de créer un compte qui servira a s'inscrire aux cours de notre plateforme.Et le composant qui permet de le faire se trouve dans le fichier **register.tsx** , dans notre cas on l'a nommer **FormulaireRegister** .

Les Besoins pour ces page sont :

Un Formulaire : il s'agit d'un formulaire qui permet d'enregistrer les données qui seront envoyées a l'API pour créer un compte user.

Pour faire cet formulaire les composants qu'on vas utilisés sont :

* Le composant **Formik** qui englobe le formulaire. Il est utilisé pour gérer l'état du formulaire ,ses propriétés sont :

initialValues :

Type : object

Décrit les valeurs initiales du formulaire.

Sans **initialValues**, le formulaire n'a aucune idée des champs qu'il doit suivre.

validationSchema :

Type : object (généralement défini avec Yup)

Sert à valider les données du formulaire.

onSubmit :

Type : fonction

Fonction qui s'exécute lorsque le formulaire est soumis (appelée via handleSubmit)

La fonction **resetForm** de Formik : réinitialise le formulaire à ses valeurs initiales.

* Le composant **TextInput** est l'un des éléments fondamentaux de React Native pour la saisie de texte. Il permet aux utilisateurs d'entrer des données et offre plusieurs options pour personnaliser l'expérience utilisateur. Mais avec cet composant il faut appliquer un style pour que la bordure des champs soit visible.

Ses attributs sont :

style : Applique des styles CSS au champ de saisie

placeholder : Affiche un texte d'exemple avant la saisie.

value : Contrôle la valeur actuelle du champ.

onChangeText : Fonction appelée à chaque modification de texte.

Exemple :

onChangeText=handleChange('prenom') **handleChange('prenom')** est une fonction qui met automatiquement à jour la valeur prenom dans l'état du formulaire

keyboardType : Définit le type de clavier affiché.

onBlur : Fonction exécutée lorsque le champ perd le focus.

Exemple

onBlur=handleBlur('prenom') Se déclenche quand l'utilisateur quitte le champ (perte de focus). Avec Formik, **handleBlur('prenom')** est souvent utilisé pour la validation.

secureTextEntry : Active le mode mot de passe (masque le texte).

* **Yup** est une bibliothèque de validation d'objets pour JavaScript qui vous permet de définir des règles de validation pour des données, généralement dans le cadre de la gestion de formulaires. IL offre une variété de méthodes pour valider différents types de données généralement dans le cadre de la gestion de formulaires.

Code Du Formulaire Register :

```
1 // Importation Des Composants
2 import React from 'react';
3 import {Formik, Field, Form, ErrorMessage} from 'formik';
4 import * as Yup from 'yup';
5 import "bootstrap/dist/css/bootstrap.css";
6
7 // Validation Des Champs avec Yup
8 const validationSchema = Yup.object().shape({
9   nom: Yup.string()
10     .min(3, "trop petit")
11     .max(255, "trop long!")
12     .required("Ce champ est obligatoire"),
13   prenom: Yup.string()
14     .min(3, "trop petit")
15     .max(255, "trop long!")
16     .required("Ce champ est obligatoire"),
17   sexe: Yup.string()
18     .min(2, "trop petit")
19
20
```

```

21 .max(10, "trop long!")
22 .required("Ce champ est obligatoire"),
23 role: Yup.string()
24 .min(2, "trop petit")
25 .max(10, "trop long!")
26 .required("Ce champ est obligatoire"),
27
28 email: Yup.string()
29 .email("email invalide")
30 .required("l'email est obligatoire"),
31 password: Yup.string()
32 .required('Mot de passe est obligatoire')
33 .min(8, "Mot de passe doit tre plus grand que 8 caract res")
34 .max(50, "Mot de passe doit tre plus petit que 50 caract res"),
35 });
36
37 // Formulaire De Register
38 const FormulaireRegister = ({ navigation }) => {
39   // Initialisation Des Valeurs (Champs De Saisie)
40   const initialValues = {
41     nom: "",
42     prenom: "",
43     email: "",
44     sexe: "",
45     role: "user",
46     password: "",
47   };
48   return (
49     <Formik
50       initialValues={initialValues}
51       validationSchema={validationSchema}
52       onSubmit={handleSubmit}
53     >
54       ({ handleChange, handleBlur, resetForm, handleSubmit, values, errors }) => (
55         <View style={styles.container}>
56           <Text>Nom</Text>
57           <TextInput
58             style={styles.input}
59             placeholder="Entrez votre nom"
60             onChangeText={handleChange('nom')}
61             onBlur={handleBlur('nom')}
62             value={values.nom}
63           />
64           {errors.nom && <Text style={styles.error}>{errors.nom}</Text>}
65
66           <Text>Pr nom </Text>
67           <TextInput
68             style={styles.input}
69             placeholder="Entrez votre pr nom"
70             onChangeText={handleChange('prenom')}
71             onBlur={handleBlur('prenom')}
72             value={values.prenom}
73           />
74           {errors.prenom && <Text style={styles.error}>{errors.prenom}</Text>}
75
76           <Text>Email</Text>
77           <TextInput
78             style={styles.input}
79             placeholder="Entrez votre email"
80             onChangeText={handleChange('email')}
81             onBlur={handleBlur('email')}
82             value={values.email}
83             keyboardType="email-address"
84             autoCapitalize="none"
85           />
86           {errors.email && <Text style={styles.error}>{errors.email}</Text>}
87
88           <Text>Sexe</Text>
89           <TextInput
90             style={styles.input}
91             placeholder="Homme ou Femme"
92             onChangeText={handleChange('sexe')}
93             onBlur={handleBlur('sexe')}
94             value={values.sexe}
95           />
96           {errors.sexe && <Text style={styles.error}>{errors.sexe}</Text>}
97

```

```

98         <Text>Mot de passe</Text>
99         <TextInput
100             style={styles.input}
101             onChangeText={handleChange('password')}
102             onBlur={handleBlur('password')}
103             value={values.password}
104             secureTextEntry
105         />
106         {errors.password && <Text style={styles.error}>{errors.password}</Text>}
107         <Button title="Annuler" onPress={resetForm} />
108         <Button title="S'inscrire" onPress={handleSubmit} />
109     </View>
110   )}
111 </Formik>
112 );
113 };
114 export default FormulaireRegister;
115
116
117
118 const styles = StyleSheet.create({
119   container: { padding: 20 },
120   input: {
121     height: 40,
122     borderColor: '#ccc',
123     borderWidth: 1,
124     marginBottom: 10,
125     paddingHorizontal: 10,
126     borderRadius: 5,
127   },
128   error: { color: 'red', marginBottom: 10 },
129 });

```

Envoie Des Données Du Formulaire A l'API :

Il y a une méthode qu'on a nommer handleSubmit , qui prend les values (données saisies du formulaire) et contacte l'API en faisant un **fetch** .

* L'Endpoint contacter est : **http://localhost:8080/api/users** et la méthode est POST.

Code De handleSubmit :

```

1 // definition de le methode handleSubmit
2 const handleSubmit = async (values) => {
3
4     try {
5         // contacte a l'api
6         const response = await fetch('http://localhost:8080/api/users', {
7             method: 'POST',
8             headers: {
9                 'Content-Type': 'application/json',
10             },
11             body: JSON.stringify(values),
12         });
13
14
15         // Verification d'erreur
16         if (!response.ok) {
17             throw new Error('Une erreur est survenue lors de l\'enregistrement des donn es.');

```

Donc , dans ce cas si le compte a été créer on le redirige sur la page login (cette redirection se fait avec navigation) pour s'authentifier.

Voila l'image de la page register :

The screenshot shows a web browser at localhost:80... displaying a registration page. The page has a back arrow and the title 'register'. Below the title is a section titled 's'inscrire'. It contains five input fields: 'Nom' (placeholder: 'Entrez votre nom'), 'Prénom' (placeholder: 'Entrez votre prénom'), 'Email' (containing 'chabane'), 'Sexe' (placeholder: 'Homme ou Femme'), and 'Mot de passe' (masked with dots). Red error messages are present: 'Ce champ est obligatoire' for Nom, Prénom, and Sexe, and 'email invalide' for the Email field. At the bottom are two blue buttons: 'ANNULER' and 'S'INSCRIRE'.

** La Page Login :

Cette page permet de s'authentifier pour accéder aux cours de notre plateforme. Et le composant qui permet de le faire se trouve dans le fichier **login.tsx** , dans notre cas on l'a nommé **FormulaireLogin** .

Les Besoins pour cette page sont :

Un Formulaire : il s'agit d'un formulaire qui permet d'enregistrer les données qui seront envoyées à l'API pour se connecter. Et les composants qui permettent de créer un formulaire ont déjà été expliqués ci dessus.

Code Du Formulaire Login :

```

1 import React from "react";
2 import { Form,Field,ErrorMessage, Formik } from "formik";
3 import * as Yup from 'yup';
4 import { Button, Text, TextInput, View } from "react-native";
5
6 const FormulaireLogin = ({navigation}) => {
7   const initialVal={
8     email:"",
9     password:""
10  };
11   return (
12     <Formik
13       initialValues={initialVal}
14       validationSchema={validationSchema}
15       onSubmit={handleSubmit}
16     >

```

```

17   ({ handleChange, handleBlur, handleSubmit, values, errors }) => (
18     <View style={styles.container}>
19       <Text>Email</Text>
20       <TextInput
21         style={styles.input}
22         placeholder="Entrez votre email"
23         onChangeText={handleChange('email')}
24         onBlur={handleBlur('email')}
25         value={values.email}
26         keyboardType="email-address"
27         autoCapitalize="none"
28       />
29       {errors.email && <Text style={styles.error}>{errors.email}</Text>}
30
31       <Text>Mot de passe</Text>
32       <TextInput
33         style={styles.input}
34         placeholder="Entrez votre mot de passe"
35         onChangeText={handleChange('password')}
36         onBlur={handleBlur('password')}
37         value={values.password}
38         secureTextEntry
39       />
40       {errors.password && <Text style={styles.error}>{errors.password}</Text>}
41
42       <Button title="Soumettre" onPress={handleSubmit} />
43     </View>
44   )}
45 </Formik>
46 );
47 };
48
49 export default FormulaireLogin;
50
51
52 const styles = StyleSheet.create({
53   container: { padding: 20 },
54   label: { fontSize: 16, fontWeight: 'bold', marginBottom: 5 },
55   input: {
56     height: 40,
57     borderColor: '#ccc',
58     borderWidth: 1,
59     marginBottom: 10,
60     paddingHorizontal: 10,
61     borderRadius: 5,
62   },
63   error: { color: 'red', marginBottom: 10 },
64 });

```

Envoi Des Données Du Formulaire A l'API :

Il y a une méthode qu'on a nommer handleSubmit , qui prend les values (données saisies du formulaire) et contacte l'API en faisant un **fetch** .

Dans cette methode nous allons stoker les informations de l'utilisateur qui est connecté pour avoir le token pour acceder les entPoints protégés.

Pour ce faire nous allons utilisées useContext pour mettre a jour le context du provider qui a pour etat user.En plus utilisé le useEffect pour atendre que l'etat se mette à jour .

* **L'Endpoint** contacter est : **http://localhost:8080/api/login** et la **méthode** est POST.

Code De handleSubmit :

```

1  const { user , setUser } = React.useContext(AuthContext);
2
3  const handleSubmit = async (values) => {
4    try {
5      const response = await fetch('http://localhost:8080/api/login', {
6        method: 'POST',
7        headers: {
8          'Content-Type': 'application/json',
9        },

```

```

10     body: JSON.stringify(values),
11   });
12
13   if (!response.ok) {
14     throw new Error('Une erreur est survenue lors de l\'enregistrement des donn es. ');
15     //alert("mot de passe ou email incorrect");
16   }
17
18   const data = await response.json();
19   console.log('Donn es enregistr es:', data);
20   setUser(data)
21   console.log(" l1111111111:",user)
22   navigation.navigate('cours');
23
24   //navigation.navigate('cours');
25
26 }
27
28 catch (error) {
29   console.error('Erreur:', error);
30   // Optionally handle errors appropriately, e.g., show a notification
31 }
32
33 useEffect (() => {
34   console.log(user);
35 }, [user]);

```

Voici l'image de la page login :

Si l'utilisateur se connecte il doit être rediriger sur la liste des cours.

** La Page ListCours :

Cette page permet d'afficher la liste des cours. Pour ce faire nous allons utilisés le composant **FlatList**.

Il a trois propriétés importantes :

data : La source des données (le tableau des éléments à afficher).

keyExtractor : Fournit une clé unique pour chaque élément.

renderItem : Détermine comment chaque élément doit être affiché.

Avant d'afficher les cours nous allons contacter l'api pour les récupérer. Pour se faire nous allons utilisé la méthode **fetch()** pour contacter l'api , le **useState** pour déclarer l'état qui va contenir les cours récupérés et **useEffect** pour le chargement des donnés.

Dans la liste des cours nous allons avoir trois boutons **ajouter un cour**, **modifier un cours** et **supprimer un cours**. La methode SupprimerCours sera appeler si on clique sur supprimer un cours va être dans le composant composantCours.

```

1  import React, { useEffect, useState } from "react";
2  import { Button, FlatList, ScrollView, StyleSheet, Text, View } from "react-native";
3  import { AuthContext } from "@context/AuthProvider";
4
5  interface Cours {
6      id: number;
7      titre: string;
8      description : string;
9      prix: number;
10     credit: number;
11 };
12
13 // Composant pour afficher La liste des cours
14 const ListeCours= ({navigation}) =>{
15     // Definition des etats
16     const [dataCours, setDataCours] = useState([]); //etat pour stocker les cours
17     const {user} = React.useContext(AuthContext); // etat pour stocker les donnees du context
18
19     // Contacte l'api pour recuperer les cours
20     useEffect(()=>{
21         fetch('http://localhost:8080/api/cours',{
22             method:'GET',
23         })
24         .then((reponse)=>reponse.json()) // convertit la reponse en json
25         .then((data)=>{
26             console.log(data);
27             setDataCours(data);
28         })
29         .catch((error)=>{
30             console.log(error);
31         });
32     });
33     },[])
34
35 //Definition de la fonction : SupprimerCours
36 const SupprimerCours = async (id:number) => {
37     try{
38         const reponse=fetch('http://localhost:8080/api/cours/${id}','{
39             method:'DELETE',
40             headers:{
41                 'Content-Type':'application/json',
42                 'Authorization':'Bearer ${user.token}'
43             }
44         } );
45         // Mise A jour de la liste de cours :
46         setDataCours((dataCours) => dataCours.filter((cour:Cours) => cour.id !==id));
47     }
48     catch (error){
49         console.log(error);
50     }
51 }
52
53
54 // Rendu du composant ListeCours
55 return (
56     <ScrollView>
57     <View style={styles.container}>
58         <Button
59             title="Ajouter Un Cours"
60             onPress={()=>{navigation.navigate('addCours')}}
61         ></Button>
62         <Text style={styles.cell}> Listes Des Cours </Text>
63
64         <View style={styles.row}>
65             <Text style={styles.cell}>Titre</Text>
66             <Text style={styles.cell}>Description</Text>
67             <Text style={styles.cell}>Prix</Text>
68             <Text style={styles.cell}>Credit</Text>
69             <Text style={styles.cell}>Action</Text>
70         </View>
71
72         // Utilisation de FlatList
73         <FlatList
74             data={dataCours}
75             keyExtractor={(item:Cours)=> item.id.toString()}
76             renderItem={({item}) => (
77                 <View style={styles.row}>

```



```

78     <Text style={styles.cell}>{item.titre} </Text>
79     <Text style={styles.cell}>{item.description} </Text>
80     <Text style={styles.cell}>{item.prix} </Text>
81     <Text style={styles.cell}>{item.credit} </Text>
82     <Text style={styles.cell}>
83       // Definition des boutons :
84       <Button title="Supprimer" onPress={()=>{SupprimerCours(item.id)}}></Button>
85       //button Modifier : lance l'ecran editCours avec navigation
86       // en prenant l'objet item (cour)
87       <Button title="Modifier" onPress={()=>{
88         navigation.navigate("editCours",item)
89       }}></Button>
90     </Text>
91   </View>
92   })
93 ></FlatList>
94
95 </View>
96 </ScrollView>
97 );
98
99 }
100
101 // Definition du style
102 const styles =StyleSheet.create({
103   container: {
104     flex:1,
105     padding:20,
106     width:'100%',
107     borderCollapse:true,
108   },
109   row :{
110     flexDirection:'row',
111     marginBottom:10,
112   },
113   listContainer: {
114     flexGrow: 1, // Assure que le FlatList peut scroller
115   },
116   cell :{
117     borderWidth:1,
118     borderColor:'#ccc',
119     padding:10,
120     flex:1,
121     textAlign:'center',
122   },
123 },
124 });
125
126 export default ListeCours;

```

L'image de l'écran cours :

AJOUTER UN COURS				
Listes Des Cours				
Titre	Description	Prix	Credit	Action
lolololi	aaa	7	1	<div>SUPPRIMER</div> <div>MODIFIER</div>
ertyulo	ertyulo	222	222	<div>SUPPRIMER</div> <div>MODIFIER</div>
aaaaaaaaaaaaa	aaaaaaaaaaaaa	5	5	<div>SUPPRIMER</div> <div>MODIFIER</div>
uyilthj	gugljjk	56	78	<div>SUPPRIMER</div> <div>MODIFIER</div>

**** La Page addCours :**

Cette page permet d'ajouter un cours. Et le composant qui permet de le faire se trouve dans le fichier **addCours.tsx** , dans notre cas on l'a nommer **Formulaire** .

Les Besoins pour cette page sont :

Un Formulaire : il s'agit d'un formulaire qui permet d'enregistrer les données qui seront envoyées a l'API pour creer un cours. Et les composants qui permettent de créer cet formulaire ont déjà été expliqués ci dessus.

Code Du Formulaire création de cours :

```
1 import React from "react";
2 import { Formik } from 'formik';
3 import * as Yup from 'yup';
4
5 import { Button, StyleSheet, Text, TextInput, View } from 'react-native';
6
7 const initiaVal= {
8   titre:"",
9   description:"",
10  prix:"",
11  credit:"",
12 };
13
14 // Validation
15 const validationSchema = Yup.object().shape({
16   titre: Yup.string()
17     .min(2, "trop petit")
18     .max(255, "trop long!")
19     .required("Ce champ est obligatoire"),
20   description: Yup.string()
21     .min(2, "trop petit")
22     .max(255, "trop long!")
23     .required("Ce champ est obligatoire"),
24   prix: Yup.number()
25     .min(0, "le prix doit etre positif")
26     .required("le prix est obligatoire"),
27   credit: Yup.number()
28     .min(0, "le credit doit etre positif")
29     .required("le credit est obligatoire"),
30 });
31
32 // Definition du formulaire pour creer un cours
33 const Formulaire = ( {navigation}) => {
34
35
36   return(
37     <View>
38       //Utilisation de Formik
39       <Formik
40         initialValues={initiaVal}
41         validationSchema={validationSchema}
42         onSubmit={handleSubmit}
43       >
44         ({ handleChange, handleBlur, handleSubmit, values, errors, resetForm })=>(
45           <View style={styles.container}>
46             <Text>Titre</Text>
47             <TextInput
48               style={styles.input}
49               placeholder="Entrez votre titre"
50               onChangeText={handleChange('titre')}
51               onBlur={handleBlur('titre')}
52               value={values.titre}
53               // keyboardType="titre-address"
54               autoCapitalize="none"
55             />
56             {errors.titre && <Text style={styles.error}>{errors.titre}</Text>}
57             <Text>Description</Text>
58             <TextInput
59               style={styles.input}
60               placeholder="Entrez votre description"
61               onChangeText={handleChange('description')}
62               onBlur={handleBlur('description')}
```

```

63         value={values.description}
64         // keyboardType="description-address"
65         autoCapitalize="none"
66     />
67     {errors.description && <Text style={styles.error}>{errors.description}</Text>}
68     <Text>Prix</Text>
69     <TextInput
70         style={styles.input}
71         placeholder="Entrez votre prix"
72         onChangeText={handleChange('prix')}
73         onBlur={handleBlur('prix')}
74         value={values.prix}
75         // keyboardType="prix-address"
76         autoCapitalize="none"
77     />
78     {errors.prix && <Text style={styles.error}>{errors.prix}</Text>}
79     <Text>Cr dit </Text>
80     <TextInput
81         style={styles.input}
82         placeholder="Entrez votre credit"
83         onChangeText={handleChange('credit')}
84         onBlur={handleBlur('credit')}
85         value={values.credit}
86         // keyboardType="credit-address"
87         autoCapitalize="none"
88     />
89     {errors.credit && <Text style={styles.error}>{errors.credit}</Text>}
90     <Button title="Ajouter" onPress={handleSubmit} />
91     <Button title="Annuler" onPress={resetForm} />
92   </View>)}
93 </Formik>
94 </View>
95 )
96 };
97
98 export default Formulaire
99 const styles = StyleSheet.create({
100   container: { padding: 20 },
101   input: {
102     height: 40,
103     borderColor: '#ccc',
104     borderWidth: 1,
105     marginBottom: 10,
106     paddingHorizontal: 10,
107     borderRadius: 5,
108   },
109   error: { color: 'red', marginBottom: 10 },
110 });

```

Envoi Des Données Du Formulaire A l'API :

Il y a une méthode qu'on a nommer handleSubmit , qui prend les values (données saisies du formulaire) et contacte l'API en faisant un **fetch**. On utilise le provider pour recuperer le token :

```

1  const { user } = React.useContext(AuthContext);
2
3  const handleSubmit = async (values) => {
4    try {
5      const response = await fetch('http://localhost:8080/api/cours', {
6        method: 'POST',
7        headers: {
8          'Content-Type': 'application/json',
9          'Authorization': 'Bearer ${user.token}', // Utilisez le token de l'utilisateur
10        },
11        body: JSON.stringify(values),
12      });
13
14      if (!response.ok) {
15        throw new Error('Une erreur est survenue lors de l\'enregistrement des donn es.');

```

```

23   }
24   };

```

Si le cours a été créé avec succès on le ramène sur la page cours qui affiche la liste des cours avec navigation .

L'image de la page addCours :

** La Page editCours :

Cette page permet de modifier un cours. Et le composant qui permet de le faire se trouve dans le fichier **editCours.tsx**, dans notre cas on l'a nommé **ModifierCours**.

Les Besoins pour cette page sont :

Un Formulaire : il s'agit d'un formulaire qui permet d'enregistrer les données qui seront envoyées à l'API pour modifier un cours. Et les composants qui permettent de créer cet formulaire ont déjà été expliqués ci dessus.

Code Du Formulaire de modification de cours :

```

1  import React from "react"
2  import { Formik, Field, Form, ErrorMessage } from 'formik';
3  import * as Yup from 'yup';
4  import { AuthContext } from "@context/AuthProvider";
5  import { Button, StyleSheet, Text, TextInput, View } from 'react-native';
6
7  // Definition de validation :
8  const validationSchema = Yup.object().shape({
9    titre: Yup.string()
10     .min(2, "trop petit")
11     .max(255, "trop long!")
12     .required("Ce champ est obligatoire"),
13    description: Yup.string()
14     .min(2, "trop petit")
15     .max(255, "trop long!")
16     .required("Ce champ est obligatoire"),
17    prix: Yup.number()
18     .min(0, "le prix doit etre positif")
19     .required("le prix est obligatoire"),
20    credit: Yup.number()
21     .min(0, "le credit doit etre positif")
22     .required("le credit est obligatoire"),
23  });
24
25  // Definition du formulaire
26  const ModifierCours = ({navigation,route}) =>{
27    const { user } = React.useContext(AuthContext);
28    const initailVal= route.params;

```

```

29 console.log(initailVal);
30 return(
31   <View>
32     // Utilisation du Formik
33     <Formik
34       initialValues={initailVal}
35       validationSchema={validationSchema}
36       onSubmit={handleSubmit}
37     >
38       ({ handleChange, handleBlur, handleSubmit, values, errors, resetForm }) => (
39         <View style={styles.container}>
40           <Text>Titre</Text>
41           <TextInput
42             style={styles.input}
43             placeholder="Entrez votre titre"
44             onChangeText={handleChange('titre')}
45             onBlur={handleBlur('titre')}
46             value={values.titre}
47             // keyboardType="titre-address"
48             autoCapitalize="none"
49           />
50           {errors.titre && <Text style={styles.error}>{errors.titre}</Text>}
51           <Text>Description</Text>
52           <TextInput
53             style={styles.input}
54             placeholder="Entrez votre description"
55             onChangeText={handleChange('description')}
56             onBlur={handleBlur('description')}
57             value={values.description}
58             // keyboardType="description-address"
59             autoCapitalize="none"
60           />
61           {errors.description && <Text style={styles.error}>{errors.description}</Text>}
62           <Text>Prix</Text>
63           <TextInput
64             style={styles.input}
65             placeholder="Entrez votre prix"
66             onChangeText={handleChange('prix')}
67             onBlur={handleBlur('prix')}
68             value={values.prix}
69             // keyboardType="prix-address"
70             autoCapitalize="none"
71           />
72           {errors.prix && <Text style={styles.error}>{errors.prix}</Text>}
73           <Text>Cr dit</Text>
74           <TextInput
75             style={styles.input}
76             placeholder="Entrez votre credit"
77             onChangeText={handleChange('credit')}
78             onBlur={handleBlur('credit')}
79             value={values.credit}
80             // keyboardType="credit-address"
81             autoCapitalize="none"
82           />
83           {errors.credit && <Text style={styles.error}>{errors.credit}</Text>}
84           <Button title="Modifier" onPress={handleSubmit} />
85           <Button title="Annuler" onPress={resetForm} />
86         </View>
87       </Formik>
88     </View>
89   )
90 }
91
92 export default ModifierCours;
93
94 const styles = StyleSheet.create({
95   container: { padding: 20 },
96   input: {
97     height: 40,
98     borderColor: 'red',
99     borderWidth: 1,
100    marginBottom: 10,
101    paddingHorizontal: 10,
102    borderRadius: 5,
103  },
104   error: { color: 'red', marginBottom: 10 },
105 });

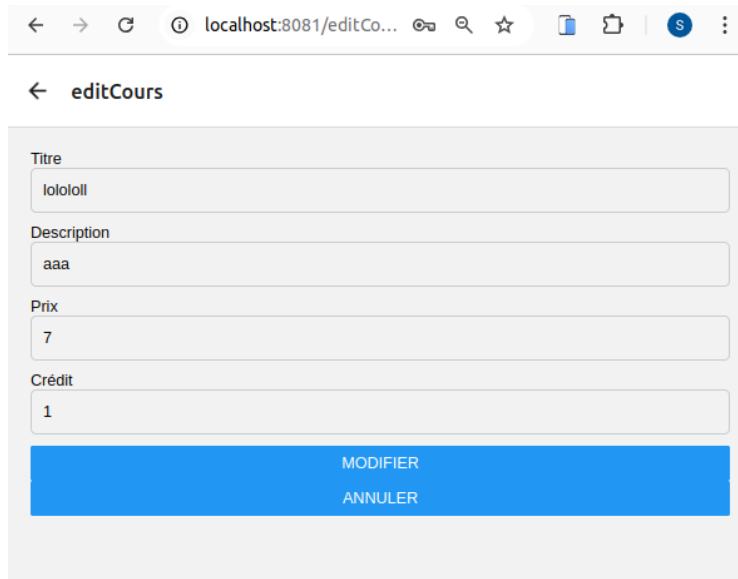
```

Envoie Des Données Du Formulaire A l'API :

Il y a une méthode qu'on a nommer `handleSubmit`, qui prend les values (données saisies du formulaire) et contacte l'API en faisant un **fetch**. On utilise le provider pour recuperer le token :

```
1 // Recuperation des donnees du context grace au provider
2 const { user } = React.useContext(AuthContext);
3 const handleSubmit = async (values) => {
4   try {
5     const response = await fetch('http://localhost:8080/api/cours/${values.id}', {
6       method: 'PUT', // Utilisez PUT pour mettre jour
7       headers: {
8         'Content-Type': 'application/json',
9         'Authorization': 'Bearer ${user.token}', // Ajouter le token d'authentification
10      },
11      body: JSON.stringify(values), // Convertit les valeurs en JSON
12    });
13
14    if (!response.ok) {
15      throw new Error('Erreur lors de la mise jour');
16    }
17
18    const data = await response.json();
19    console.log('Cours mis jour:', data);
20    navigation.navigate('cours');
21  } catch (error) {
22    console.error('Erreur:', error);
23  }
24 }
25
26
27
28
```

L'image de la page editCours :



The screenshot shows a web browser window with the address bar displaying 'localhost:8081/editCo...'. The page title is 'editCours'. The form contains four input fields: 'Titre' with the value 'lolololl', 'Description' with the value 'aaa', 'Prix' with the value '7', and 'Crédit' with the value '1'. At the bottom of the form, there are two blue buttons: 'MODIFIER' and 'ANNULER'.

5 Le déploiement

5.1 Installer EAS CLI à l'échelle mondiale

EAS CLI (Expo Application Services Command Line Interface) est un outil en ligne de commande qui permet de gérer le build, les mises à jour et le déploiement des applications Expo

Utilisez la commande suivante pour installer EAS CLI à l'échelle globale sur votre système :

```
npm install -g eas-cli
```

```

bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ npm install -g eas-cli
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and test
ed way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated lodash.get@4.4.2: This package is deprecated. Use the optional chaining (?.) operator instead.
npm warn deprecated sudo-prompts@1.1.1: Package no longer supported. Contact Support at https://www.npmjs.com/support for more info.
npm warn deprecated @xalton/mldom@0.7.13: this version is no longer supported, please update to at least 0.8.*
npm warn deprecated glob@6.0.4: Glob versions prior to v9 are no longer supported
npm warn deprecated rimraf@2.5.4: Rimraf versions prior to v4 are no longer supported
npm warn deprecated goclif/screeng@3.0.8: Package no longer supported. Contact Support at https://www.npmjs.com/support for more info.
added 443 packages in 38s
53 packages are looking for funding
run 'npm fund' for details

```

5.2 Vérifier l'installation EAS

Une fois installé, vérifiez la version EAS pour confirmer l'installation :

eas --version

```

bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ eas --version
eas-cli/15.0.15 linux-x64 node-v22.9.0
bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ eas login

```

5.3 réez et connectez-vous à votre compte EXPO

Pour utiliser EAS (Expo Application Services) pour créer et déployer votre application, vous avez besoin d'un compte Expo. Après avoir créé le compte taper la commande **login eas** pour se connecter

```

eas-cli/15.0.15 linux-x64 node-v22.9.0
bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ eas login
(node:246741) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Log in to EAS with email or username (exit and run eas login --help to see other login options)
  Email or username _ beveri
  Password _ *****
Logged in
bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$

```

5.4 création d'une version de développement

Pour générer une version de développement pour la plateforme Android, accédez à votre projet et exécutez la commande suivante :

eas build --profile development --platform android

```

Logged in
bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ eas build --platform android
eas build --profile development --platform android
(node:247064) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Generated eas.json. Learn more

You want to build a development client build for platforms: Android
However, we detected that you don't have expo-dev-client installed for your project.
  Do you want EAS CLI to install expo-dev-client? f
  Do you want EAS CLI to install expo-dev-client for you? ... yes

Running expo install expo-dev-client

[expo-cli] > Installing 1 SDK 52.0.0 compatible native module using npm
[expo-cli] > npm install
[expo-cli] added 8 packages, changed 5 packages, and audited 1202 packages in 12s
[expo-cli] 113 packages are looking for funding
[expo-cli] run 'npm fund' for details
[expo-cli] 6 vulnerabilities (4 low, 2 moderate)
[expo-cli] To address all issues, run:
[expo-cli] npm audit fix
[expo-cli] Run 'npm audit' for details.

```

```

bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ eas login
(node:246741) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Log in to EAS with email or username (exit and run eas login --help to see other login options)
  Email or username _ beveri
  Password _ *****
Logged in
bever@beveri-HP-ProDesk-600-G3-MT:~/s5/les_Ue_Libres/react_natives/gestion_cours$ eas build --platform android
eas build --profile development --platform android
(node:247064) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Generated eas.json. Learn more

You want to build a development client build for platforms: Android
However, we detected that you don't have expo-dev-client installed for your project.
  Do you want EAS CLI to install expo-dev-client? f
  Do you want EAS CLI to install expo-dev-client for you? ... yes

Running expo install expo-dev-client

[expo-cli] > Installing 1 SDK 52.0.0 compatible native module using npm
[expo-cli] > npm install
[expo-cli] added 8 packages, changed 5 packages, and audited 1202 packages in 12s
[expo-cli] 113 packages are looking for funding
[expo-cli] run 'npm fund' for details
[expo-cli] 6 vulnerabilities (4 low, 2 moderate)
[expo-cli]

```

Comme il s'agit de votre première version, EAS vous demandera d'installer expo-dev-client .

Tapez yes et appuyez sur Entrée pour procéder à l'installation.

Ce package est essentiel pour exécuter efficacement les builds de développement
Maintenant, EAS vous demandera de créer automatiquement un projet EAS avec un nom par défaut comme indiqué dans l'image

ci-dessous.

Vous pouvez accepter le nom par défaut ou saisir un nom de projet personnalisé.
Une fois terminé, choisissez « Oui » pour continuer.

Cela configurera votre projet pour les futures versions EAS de manière transparente.

```
EAS project not configured.
✓ Would you like to automatically create an EAS project for @doctorsdesks/novahealthpatientapp? ... yes
✓ Created @doctorsdesks/novahealthpatientapp on EAS
✓ Linked local project to EAS project 944afb74-1749-46ab-be23-0e8d1d88a166
Resolved "development" environment for the build. Learn more
No environment variables with visibility "Plain text" and "Sensitive" found for the "development" environment on EAS.

Android application id Learn more
? What would you like your Android application id to be? > com.doctorsdesks.novahealthpatientapp
```

Pendant le processus de construction, l'invite vous demandera de créer un ID d'application Android avec une valeur par défaut comme indiqué dans l'image ci-dessus.

Vous pouvez accepter l'ID par défaut ou en saisir un personnalisé.
Appuyez sur « Entrée » pour continuer.

Cet identifiant est indispensable pour identifier votre application sur le Google Play Store.

Maintenant, à nouveau pendant le processus de construction, EAS vous demandera de générer un nouveau Keystore Android comme indiqué dans l'image ci-dessous.

Tapez « oui » et appuyez sur Entrée pour permettre à EAS d'en créer un pour vous.

Ce Keystore est requis pour signer votre application avant de la publier sur le Google Play Store.

```
EAS project not configured.
✓ Would you like to automatically create an EAS project for @beveri/gestion_cours? ... yes
✓ Created @beveri/gestion_cours on EAS
✓ Linked local project to EAS project 8b6ef4fc-4795-476d-bc0b-f07193619237
Resolved "development" environment for the build. Learn more
No environment variables with visibility "Plain text" and "Sensitive" found for the "development" environment on EAS.

Android application id Learn more
? What would you like your Android application id to be? ... com.beveri.gestion_cours
We create keystore are configured for this project, versionCode will be initialized based on the value from the local project.
✓ Initialized versionCode with 1.
✓ Using remote Android credentials (Expo server)
Generate a new Android Keystore? ... yes
✓ Created keystore
Compressing project files and uploading to EAS Build. Learn more
✓ Uploaded to EAS
✓ Computed project fingerprint

Build details: https://expo.dev/accounts/beveri/projects/gestion_cours/builds/041a8ccd-e804-4278-ba99-5f7b4d2ebc47

Waiting for build to complete. You can press Ctrl+C to exit.
Build queued...

Start builds sooner in the priority queue.
Sign up for EAS Production or Enterprise at https://expo.dev/accounts/beveri/settings/billing
Waiting in Free tier queue
```

Félicitations ! Votre build est en cours, attendez qu'il soit terminé.
Une fois terminé, vous recevrez un lien de téléchargement pour votre fichier APK comme indiqué ci-dessous dans l'image, prêt à être testé !

```
Build queued...

Start builds sooner in the priority queue.
Sign up for EAS Production or Enterprise at https://expo.dev/accounts/beveri/settings/billing
Waiting in Free tier queue
=====
✓ Build finished

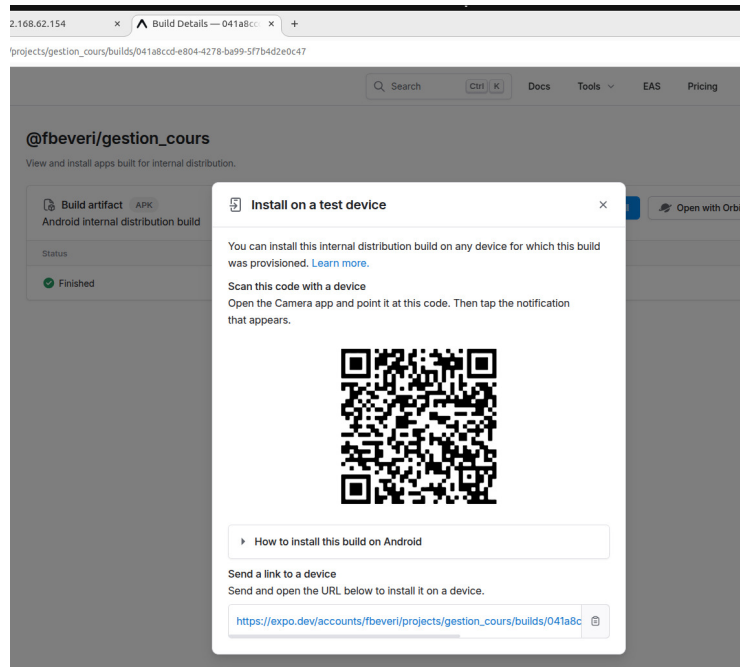
[QR Code] https://expo.dev/accounts/beveri/settings/billing (ctrl+click)

Open this link on your Android devices (or scan the QR code) to install the app:
https://expo.dev/accounts/beveri/projects/gestion_cours/builds/041a8ccd-e804-4278-ba99-5f7b4d2ebc47
? Install and run the Android build on an emulator? ... (Y/n)
```

Tapez « Non » et appuyez sur Entrée pour ignorer cette étape pour le moment. Nous aborderons ce sujet en détail plus tard dans le blog.

Votre construction se poursuivra comme prévu !

Bravo ! Vous avez créé avec succès une version de développement pour votre projet. Vous pouvez maintenant installer l'APK sur votre appareil Android et tester vos modifications en temps réel. si vous cliquez sur le lien sa vous amène sur le site expo.



5.5 Exécution avec l'application Expo Go

Accédez au Play Store et installez l'application Expo Go sur votre appareil mobile. Installez la version de développement — Transférez le fichier APK (généré à la dernière étape) sur votre appareil Android et installez-le.

Ouvrez votre application — Une fois installée, lancez l'application sur votre appareil mobile.

Redémarrez votre projet — Maintenant, redémarrez votre serveur de développement en exécutant : **npm start**



Vous pouvez désormais lancer votre application en effectuant l'une des opérations suivantes :

Scannez le code QR — Utilisez l'appareil photo de votre téléphone pour scanner le code QR fourni dans le terminal.
Saisissez manuellement le lien — Ouvrez l'application Expo Go et saisissez le lien affiché sous le code QR (par exemple, exp ://192.168.29.116 :8081).
Utilisez la version de développement — Ouvrez l'application de développement que vous avez installée et saisissez le même lien.

Votre application fonctionne désormais sur votre appareil Android et vous pouvez tester les modifications en temps réel !

6 Conclusion

Avec ces bases, vous pouvez commencer à créer des interfaces utilisateur simples et efficaces dans React Native