

DOCUMENT TECHNIQUE DE LA FORMATION DE REACT NATIVE

Réalisé par : HAFDI AHMED

Encadré par : Mr SERRAJ Mohammed Amine

Table des matières

Table des Figures	6
Liste des Tableaux :	7
CHAPITRE 1 : SET UP POUR L'ENVIRONNEMENT WINDOWS	9
Configuration pour Windows :	9
CHAPITRE 2 : BONJOUR LE MONDE	11
Bonjour Tout le monde!	11
Chapitre 3 : Coiffant (stylesheet)	13
Introduction :	13
Syntaxe :	13
Remarques :	13
Exemples :	13
Chapitre 6 : Composants	15
Exemples :	15
Composant de base :	15
Composant avec état	15
Composant sans état (composant without state)	16
Chapitre 7 : Disposition	17
Flexbox :	17
Chapitre 8 : States (les états)	19
Syntaxe :	19
Exemples :	19
SetState :	19
Chapitre 10 : Exécuter une application sur l'appareil (version android) & Expo	21
Exécuter sur l'appareil :	21
Chapitre 11 : Images	23
Exemples :	23
Module d'image	23
• Image locale et Image externe :	23
Source d'image conditionnelle :	24
Screen :	24
.....	24
.....	24
Chapitre 12 : Instructions en ligne de commande	25
Exemples :	25
Vérifier la version	25

Exemples de sortie	25
Chapitre 13 : Le débogage	26
Syntaxe :	26
Le débogage avec Expo (Exponent) et VSCode :	26
• C'est quoi Exponent ?	26
Etapas de débogage par Expo & VScode :	26
Chapitre 14 : Le Routage	28
Exemples :	28
Stack Composant :	28
Top Tabs :	30
Bottom Tabs	31
Chapitre 15 : Liaison de l'API native	32
Introduction :	32
Exemples :	32
Liens sortants	32
Liens entrants	32
Chapitre 16 : ListView	34
FlatList	34
SectionList	35
Screen :	35
.....	35
Map Array	37
Chapitre 17 : Input Data	38
Exemples :	38
TextInput :	38
Chapitre 18 : Modal	39
Introduction :	39
Paramètres :	39
Chapitre 19 : Module Plateforme	41
Introduction :	41
Exemple :	41
Chapitre 20 : Notification push	42
Expo Push Notifications Tool :	42
Chapitre 21 : Polices Personnalisés	44
Exemples :	44
Chapitre 22 : RefreshControl	45

Chapitre 23 : Requêtes http	46
Syntaxe :	46
Exemple : Posting application avec json server.....	46
Chapitre 24 : Test d'unité	48
Introduction :.....	48
Exemples :.....	48
Chapitre 25 : Mobx	49
Introduction :.....	49
C'est quoi Mobx :.....	49
Les Principes de Mobx :.....	49
Exemple 1 : une application simple des taches avec state management useState ().....	50
Exemple 2 :une application simple des taches avec state management Mobx.....	52
Exemple 3 : Exemple 2 :une application simple des taches avec state management MST (Mobx State Tree)	53
.....	53
C'est quoi MST :.....	53

Table des Figures

Figure 1:react native logo	9
Figure 2: Hello World.....	11
Figure 3: Hello World Screen.....	12
Figure 4: basic component	15
Figure 5: component avec état.....	15
Figure 6: composant sans état	16
Figure 7: flexdirection : 'row'.....	17
Figure 8 : flexdirection row screen.....	18
Figure 9: counter add	19
Figure 10: counter add screen.....	20
Figure 11: adb devices	21
Figure 12: expo start server.....	22
Figure 13: expo run on android emulator	22
Figure 14: image locale et externe	23
Figure 15: image locale & externe screen	23
Figure 16: image conditionnelle	24
Figure 17: image conditionnelle screen	24
Figure 18: react native check version.....	25
Figure 19: expo logo	26
Figure 20: react native tools.....	26
Figure 21 : debug in exponent.....	27
Figure 22: debug.....	27
Figure 23 : Navigator	28
Figure 24 : MyStack	28
Figure 25: First Screen	29
Figure 26: Second Screen	29
Figure 27: Top Tabs	30
Figure 28: Tab Tops screen.....	30
Figure 29: Bottom tabs.....	31
Figure 30: lien sortant	32
Figure 32 : lien entrant	32
Figure 31: lien sortant screen.....	32
Figure 33: Linking.....	32
Figure 34: lien entrant	33
Figure 35:flatList	34
Figure 36: flatlist users	34
Figure 37:SectionList	35
Figure 38: SectionList Screen.....	35
Figure 39:map array view	37
Figure 40: TextInput	38
Figure 41: add user screen	38
Figure 42: Modal.....	39
Figure 43: Modal 1.....	40
Figure 44: modal 2.....	40
Figure 45: Plateforme.....	41
Figure 46: Platform Screen	41

Figure 47: Push Notification	42
Figure 48: expo push notification tool	42
Figure 49: notification screen.....	43
Figure 50: fonts.....	44
Figure 51: fonts exemples	44
Figure 52: refresh control.....	45
Figure 53:Refresh control screen	45
Figure 54: fetch1.....	46
Figure 55: fetch 2.....	46
Figure 56: Test for buttons component	48
Figure 57: buttonns.test.js.....	48
Figure 58: Test passed	48
Figure 59:Mobx + React Native	49
Figure 60: Mobx state architecture	49
Figure 61: Task with UseState	50
Figure 62: Tasks UseState screen	51
Figure 63: Tasks With Mobx	52
Figure 64: Mobx Tasks Screen	52
Figure 65: MST RN	53
Figure 66: Book Store	53
Figure 67: BookView 1.....	55
Figure 68: BookView 2.....	55

Liste des Tableaux :

Tableau 1: Modal parameters	39
-----------------------------------	----

CHAPITRE 1 : SET UP POUR L'ENVIRONNEMENT WINDOWS



Figure 1:react native logo

React Native vous permet de créer des applications mobiles en utilisant uniquement JavaScript. Il utilise la même conception que React, vous permettant de composer une interface utilisateur mobile riche à partir de composants déclaratifs. Avec React Native, vous ne créez pas une

«application Web mobile», une «application HTML5» ou une «application hybride».

Vous construisez une véritable application mobile qui ne se distingue pas d'une application créée avec Objective-C ou Java. React Native utilise les mêmes éléments fondamentaux que les applications iOS et Android classiques. Vous venez de mettre ces blocs de construction ensemble en utilisant JavaScript et React.

Il est open-source et maintenu par Facebook.

- Site Internet (<https://reactnative.dev/>)
- Documentation (<https://reactnative.dev/docs/getting-started>)
- GitHub Repository(<https://github.com/facebook/react-native>)

Configuration pour Windows :

Remarque: vous ne pouvez pas développer d'applications réactives pour iOS sur Windows, mais uniquement des applications Android réactives.

Outils / Environnement

- Windows 10
- outil de ligne de commande (par exemple, ligne de commande Powershell ou Windows)
- Chocolaté (étapes pour configurer via PowerShell via ce lien <https://blog.fbalashov.com/2016/07/react-native-android-apps-on-windows.html#setup-choco>)
- Le JDK (version 8)
- Studio Android Une machine Intel avec la technologie de virtualisation activée pour HAXM (facultatif, nécessaire uniquement si vous souhaitez utiliser un émulateur)

1) Configurez votre machine pour réagir au développement natif :

Démarrez la ligne de commande en tant qu'administrateur exécutez les commandes suivantes:

```
choco install nodejs.install  
choco install python2
```

Redémarrez la ligne de commande en tant qu'administrateur pour pouvoir exécuter npm

```
npm install -g react-native-cli
```

2) Définissez vos variables d'environnement :

Ouvrez la fenêtre Variables d'environnement en naviguant vers: [Clic droit] Menu "Démarrer" -> Système -> Paramètres système avancés -> Variables d'environnement Dans la section inférieure, recherchez la variable système "Path" et ajoutez l'emplacement d'installation de react-native à l'étape 1. Si vous n'avez pas ajouté de variable d'environnement ANDROID_HOME, vous devrez également le faire ici. Dans la fenêtre "Variables d'environnement", ajoutez une nouvelle variable système nommée "ANDROID_HOME" et la valeur correspondant au chemin d'accès à votre SDK Android. Redémarrez ensuite la ligne de commande en tant qu'administrateur pour pouvoir y exécuter des commandes réactives.

3) Créez votre projet

Créez votre projet En ligne de commande, accédez au dossier dans lequel vous souhaitez placer votre projet et exécutez la commande suivante:

```
react-native init ProjectName
```

4) Lancez votre projet

Démarrez un émulateur depuis Android Studio Accédez au répertoire racine de votre projet en ligne de commande et exécutez-le:

```
cd ProjectName
```

```
react-native run-android
```

CHAPITRE 2 : BONJOUR LE MONDE

On crée un projet avec la commande :

expo init Hello World

on lance le project avec la commande : (on lance la commande sur le path du projet)

expo start

on ouvre le fichier App.js

Après on écrit `<Text>Bonjour Tout le Monde </Text>` :

On rafraichit le fichier avec la commande « control + S »

Et voila Félicitations! Vous avez écrit avec succès votre premier Hello World!

Bonjour Tout le monde!

```
1  import { StatusBar } from 'expo-status-bar';
2  import React from 'react';
3  import { StyleSheet, Text, View } from 'react-native';
4
5  export default function App() {
6    return (
7      <View style={styles.container}>
8      |   <Text>Bonjour Tout le Monde </Text>
9      |   <StatusBar style="auto" />
10     </View>
11   );
12 }
13
14 const styles = StyleSheet.create({
15   container: {
16     flex: 1,
17     backgroundColor: 'white',
18     alignItems: 'center',
19     justifyContent: 'center',
20   },
21 });
22
```

Figure 2: Hello World

Vous devriez voir Hello World! écrit à l'écran!

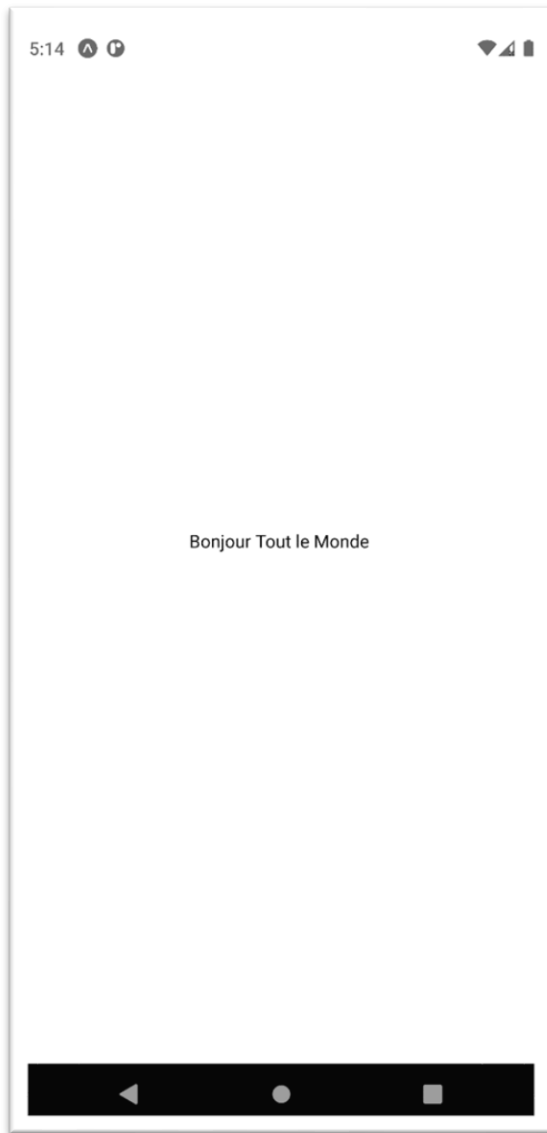


Figure 3: Hello World Screen

Chapitre 3 : Coiffant (stylesheet)

Introduction :

Les styles sont définis dans un objet JSON avec des noms d'attributs de style similaires, comme dans CSS. Un tel objet peut soit être mis en ligne dans le style prop d'un composant, soit être transmis à la fonction `StyleSheet.create(StyleObject)` et être stocké dans une variable pour un accès en ligne plus court en utilisant un nom de sélecteur similaire à une classe. en CSS.

Syntaxe :

- `<Component style={styleFormaStyleSheet }/>`
- `<Component style={styleObject }/>`
- `<Component style={style1,style2 }/>`

Remarques :

La plupart des styles React Native sont leurs formulaires CSS, mais dans un cas camel. Ainsi, la text-decoration devient `textDecoration` . Contrairement aux CSS, les styles ne sont pas hérités. Si vous souhaitez que les composants enfants héritent d'un certain style, vous devez le fournir explicitement à l'enfant. Cela signifie que vous ne pouvez pas définir une famille de polices pour une View entière. La seule exception à cette règle est le composant `Text` : les `Text` imbriqués héritent de leurs styles parents.

Exemples :

Il y a plusieurs manière pour définir un style :

- Style en ligne :
`<Text style={{color : 'red'}} >Red Text </Text>`

- Styling à l'aide d'une feuille de style :

```
Const styles = StyleSheet.create {{  
  red : {  
    color : 'red'  
  },  
}}
```

- Styling à l'aide de plusieurs styles :

```
const styles = StyleSheet.create({  
  red: { color: 'red' },  
  greenUnderline: { color: 'green',  
    textDecoration: 'underline' },
```

```
big: { fontSize: 30 } });
```

```
//code of the class
```

```
<Text style={{styles.red,styles.big } > Big red </Text>
```

```
<Text style={{styles.red,styles.greenUnderline } > Big red </Text>
```

Chapitre 6 : Composants

Exemples :

Composant de base :

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'
class Example extends Component {
  render () {
    return (
      <View>
        <Text> I'm a basic Component </Text>
      </View>
    )
  }
}
AppRegistry.registerComponent('Example', () => Example)
```

Figure 4: basic component

Composant avec état

Ces composants auront des états changeants.(composant with states)

On utilise un constructeur pour créer un état / state car on utilise ici une Classe comme un composant

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'
class Example extends Component {
  constructor (props){
    super(props)
    this.state={
      name:"state 1"
    }
  }
  render () {
    return (
      <View>
        <Text> I'm a basic Component </Text>
      </View>
    )
  }
}
AppRegistry.registerComponent('Example', () => Example)
```

Figure 5: component avec état

Composant sans état (composant without state)

(ce sont les fonctions , funtional componants are stateless , class componants are statefull)

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

function App(props) {
  return (
    <View style={styles.container}>
      <Text style={{color:'black'}} >composant sans état</Text>
    </View>
  );
}

export default App;
const styles = StyleSheet.create({
  container : {
    flex : 1,
    justifyContent:'center',
    alignItems:'center'
  },
});
```

Figure 6: composant sans état

Chapitre 7 : Disposition

Exemples :

Flexbox :

Flexbox est une mode de mise en page permettant la disposition des éléments sur une page de manière à ce que les éléments se comportent de manière prévisible lorsque la mise en page doit

Prendre en charge différentes tailles d'écran et différents périphériques d'affichage .par défaut ,

Flexbox représente les enfants (children) sous forme d'une colonne (column), Mais on peut le changer en utilisant « flexDirection : 'row » dans le styling .

flexDirection

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
function App(props) {
  return (
    <View style={styles.container}>
      <Text style={{color:'black'}} >flexDirection : row</Text>
      <View style={styles.bboxes_container}>
        <View style={styles.box} ></View>
        <View style={styles.box} ></View>
        <View style={styles.box} ></View>
      </View>
    </View>
  );
}
export default App;
const styles = StyleSheet.create({
  container : {
    flex : 1,justifyContent:'center',alignItems:'center',
  },
  bboxes_container:{
    flexDirection:'row'
  },
  box : {
    margin:10,backgroundColor:'green', height:50,width:50,
  },
});
```

Figure 7: flexdirection : 'row'

screen :

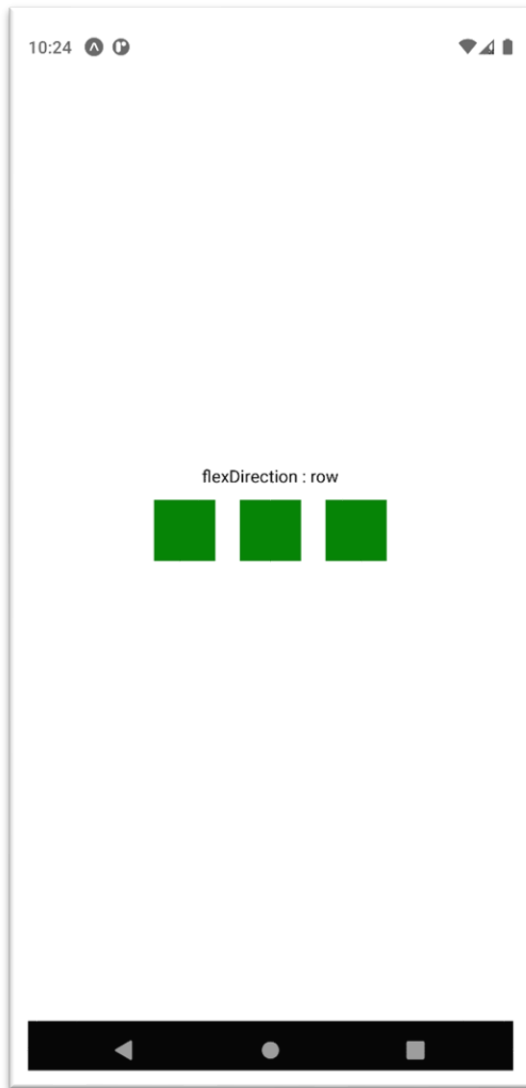


Figure 8 : flexdirection row screen

Il y a plusieurs possibilités pour changer vers une colonne inversée ou ligne inversée (flexDirection : 'row-reverse' or 'column-reverse')

Chaptire 8 : States (les états)

Syntaxe :

- Void **setState** (fonction | object nextState , [rappel de fonction])

Exemples :

Pour créer des applications qui contient par exemple des composants qui ont besoin d'être changé instantanément et être affiché sur l'écran , alors React Native offre les états ou states ,

Pour modifier la vue d'une application , on peut utiliser **setState** ,setState effectue une fusion superficielle entre l'état nouveau et précédent et déclenche un re-render du composant .

Pour bien expliquer setState et comment ça fonctionne je vais prendre l'exemple d'un compteur avec button

SetState :

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

function States(props) {
  const [counter, setCounter] = useState(1)

  return (
    <View style={styles.Counter_Container}>
      <Text>Counter Value : {counter}</Text>
      <Button title="add" onPress={() => setCounter(counter+1)} />
    </View>
  );
}

export default States;
const styles = StyleSheet.create({
  Counter_Container: {
    flex : 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});
```

Figure 9: counter add

Screen :

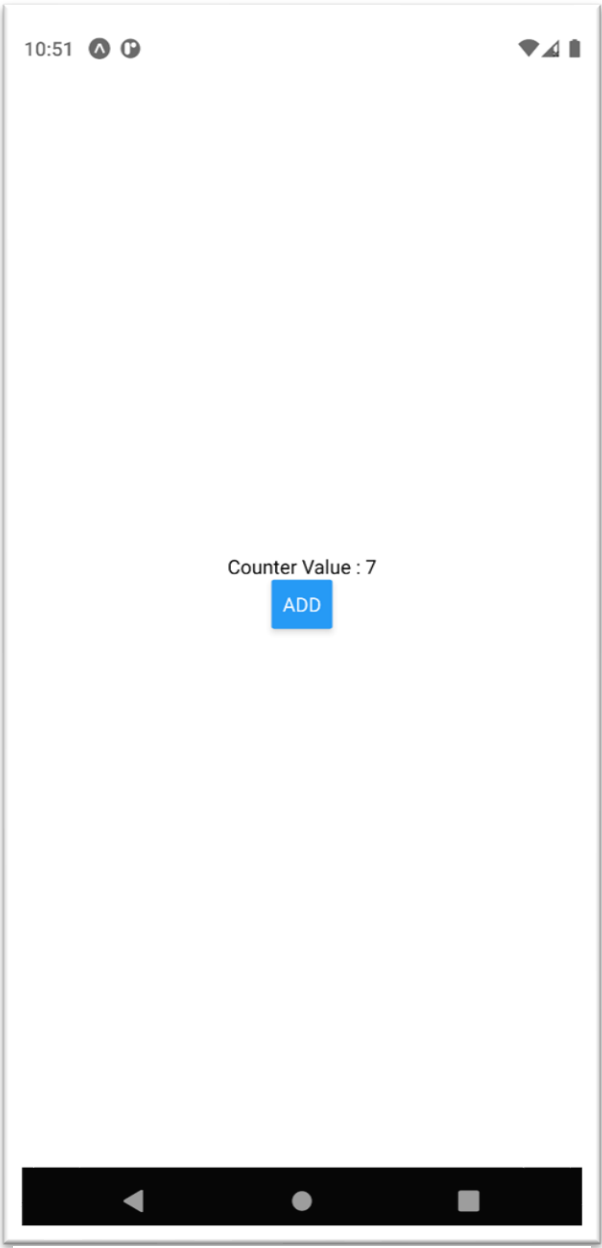
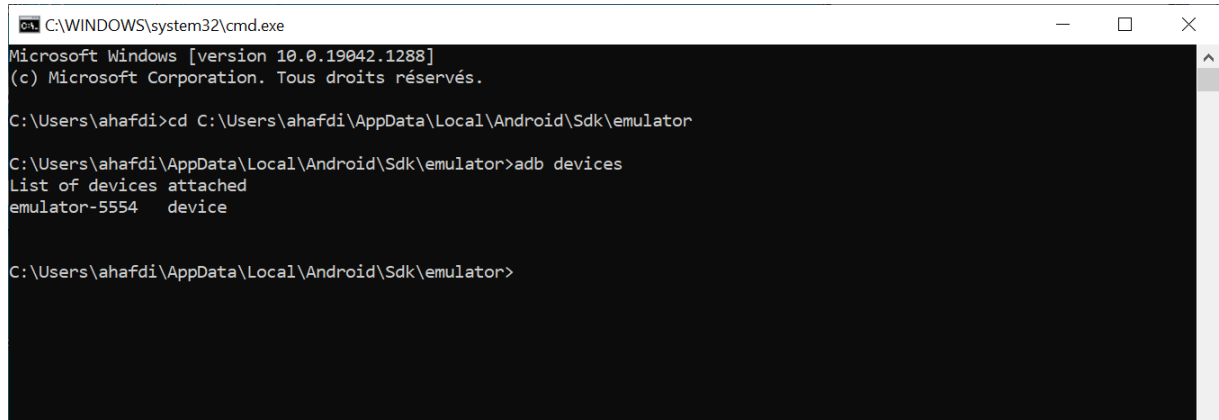


Figure 10: counter add screen

Chapitre 10 : Exécuter une application sur l'appareil (version android) & Expo

Exécuter sur l'appareil :

1. Adb devices (cette commande pour afficher tous les appareil /emulator connectés à l'ordinateur



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [version 10.0.19042.1288]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\ahafdi>cd C:\Users\ahafdi\AppData\Local\Android\Sdk\emulator

C:\Users\ahafdi\AppData\Local\Android\Sdk\emulator>adb devices
List of devices attached
emulator-5554    device

C:\Users\ahafdi\AppData\Local\Android\Sdk\emulator>
```

Figure 11: adb devices

2. emulator -avd Pixel_XL_API_30 (lancer l'emulator /simulateur)
3. react-native run-android (exécuter le projet sur le simulateur android)
4. pour Expo (j'utilise expo sur cette documentation)

- a. expo start (on lance cette commande sur la destination du projet pour lancer le serveur d'Expo)

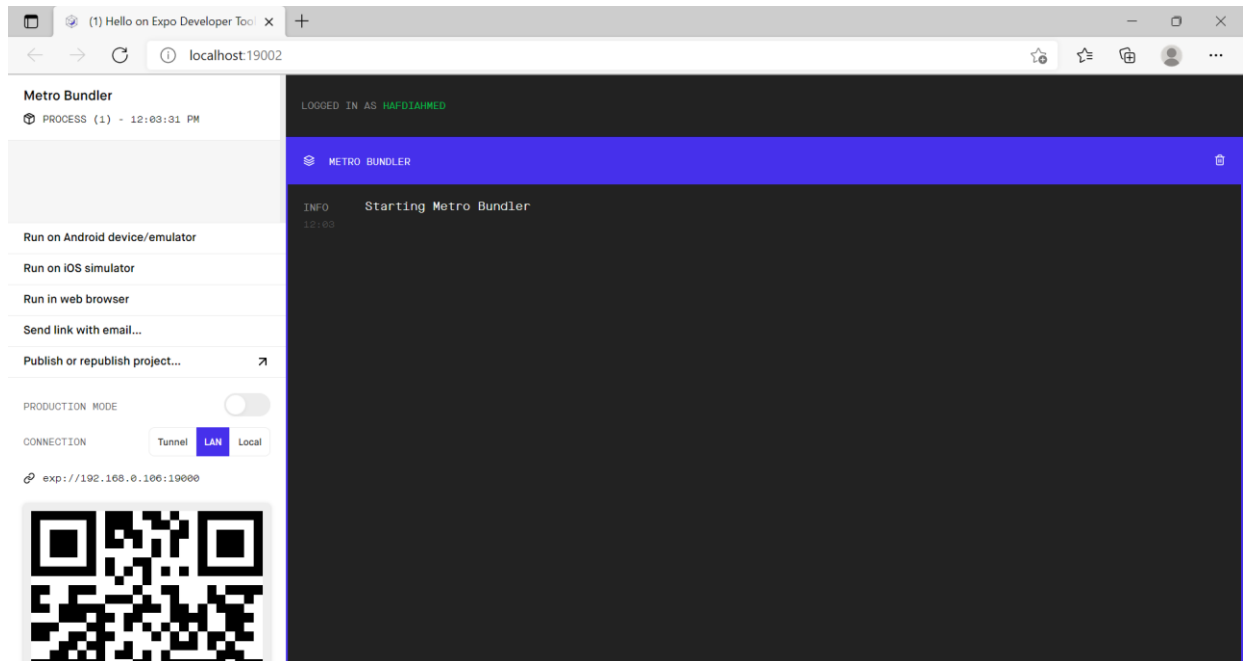


Figure 12: expo start server

- b. On clique sur 'run on Android device/emulator' :

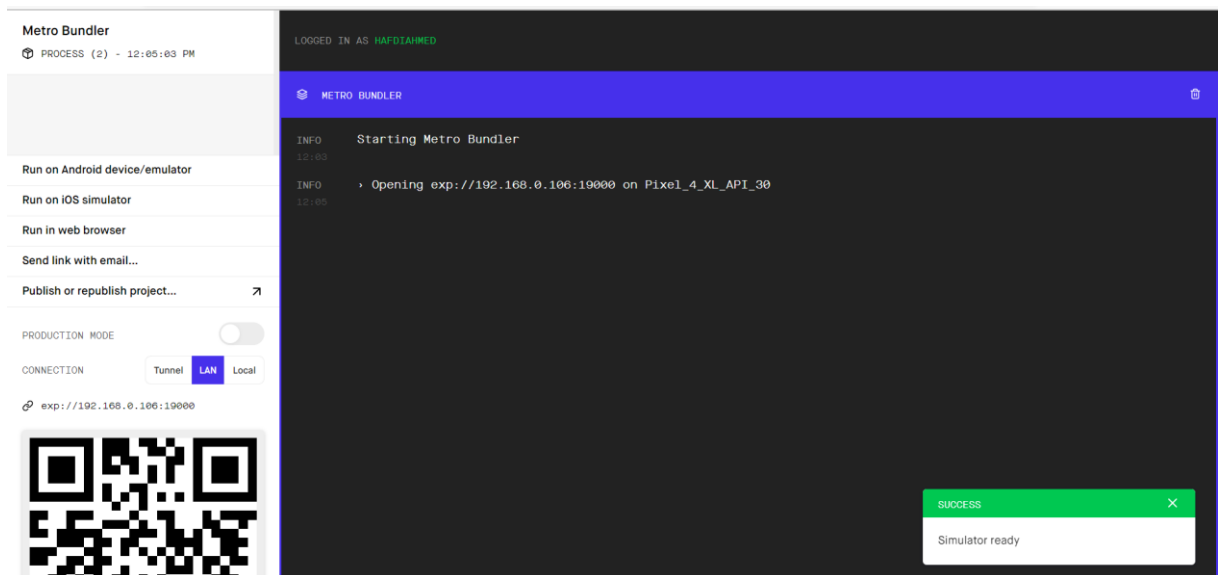


Figure 13: expo run on android emulator

Chapitre 11 : Images

Exemples :

Module d'image

On peut importer Image à partir du package « react-native »

- Image locale et Image externe :

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

function Images(props) {
  return (
    <View style={styles.Images_Container} >
      <Text>image locale</Text>
      <Image source={require("../Hello/assets/gear9_1.jpg")} style={styles.image_style}/>
      <Text>image externe</Text>
      <Image source={{uri : "https://yt3.ggpht.com/ytc/AKedOLQvLGRnf5cnRwTcsXTwRhyI_uwbJ4R_b7HYKm6ciA=s176-c-k-c0x00ffffff-no-rj"}}
        style={styles.image_style}/>
    </View>
  );
}

export default Images;
const styles = StyleSheet.create({
  Images_Container: {flex :1,justifyContent:'center',alignItems:'center'},
  image_style : { height:200,width:200},
});
```

Figure 14: image locale et externe



- Screen :

Figure 15: image locale & externe screen

Source d'image conditionnelle :

```
import React from 'react';
import { Image, StyleSheet, Text, View } from 'react-native';
function Images(props) {
  const path_1= require("../Hello/assets/favicon.png");
  const path_2=require("../Hello/assets/gear9_1.jpg");
  let image_path= image_path ? path_2 : path_1 ;
  return (
    <View style={styles.Images_Container} >
      <Text style={styles.text_style}>image conditionel</Text>
      <Image style={styles.image_style} source={image_path}/>
    </View>
  );
}
export default Images;
const styles = StyleSheet.create({
  Images_Container: {flex :1,justifyContent:'center',alignItems:'center',backgroundColor:"#212120"},
  image_style : { height:150,width:150},
  text_style :{color:"white" ,fontSize:20,},
});
```

Figure 16: image conditionnelle

Screen :

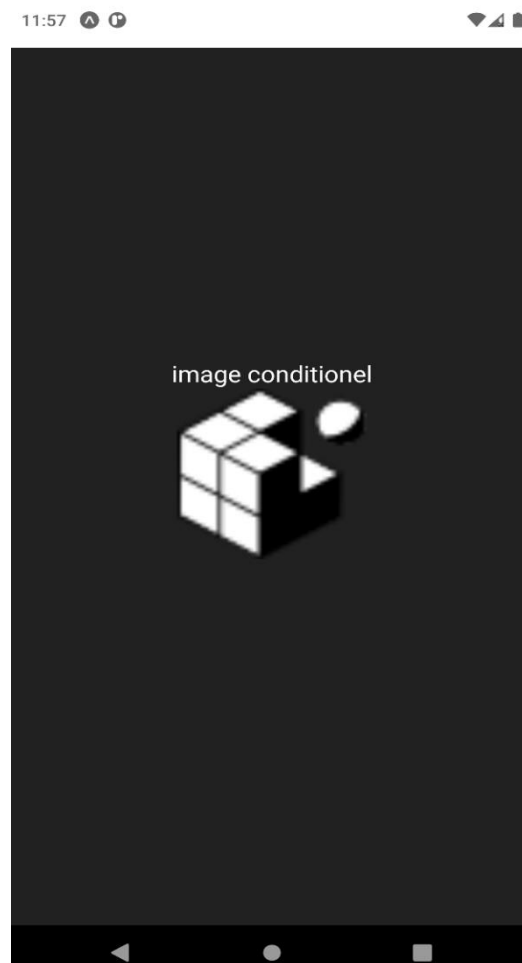


Figure 17: image conditionnelle screen

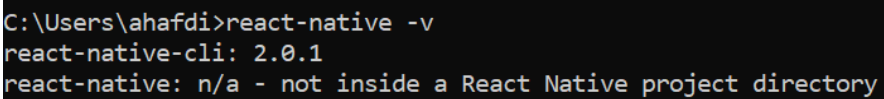
Chapitre 12 : Instructions en ligne de commande

Exemples :

Vérifier la version

- **\$ react-native -v**

Exemples de sortie



```
C:\Users\ahafdi>react-native -v
react-native-cli: 2.0.1
react-native: n/a - not inside a React Native project directory
```

Figure 18: react native check version

Pour Initialiser

- **\$ react-native init MyFirstProjectRN**
- **\$ expo init MyFirstProjectEXPO**

Pour courir pour Android

- **\$ cd MyFirstProjectRN**
- **\$ react-native run-android**

Chapitre 13 : Le débogage

Syntaxe :

- Débogueur

Le débogage avec Expo (Exponent) et VSCode :

- C'est quoi Exponent ?



Figure 19: expo logo

Exponent /Expo est un framework et une plateforme pour les applications de React . c'est un ensemble d'outils et services construits autour de React Native et des plate-formes natives qui aident à développer , créer , déployer et itérer rapidement sur ios ,android et Web à partir de la même base de code JavaScript/TypeScript .

Etapes de débogage par Expo & VScode :

1. Installation de l'extension « React Native Tools » sur VSCode

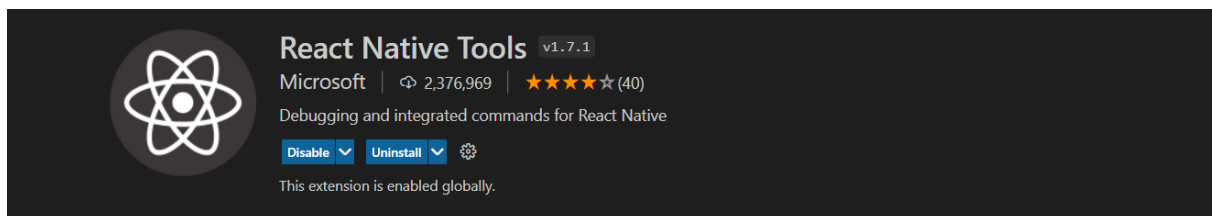


Figure 20: react native tools

2. Après l'ajout de la configuration « Exponent » sur le fichier launch.json dand le dossier. Vscod

```
"configurations": [  
  {  
    "name": "Debug in Exponent",  
    "request": "launch",  
    "type": "reactnative",  
    "cwd": "${workspaceFolder}",  
    "platform": "exponent",  
    "expoHostType": "local"  
  },  
]
```

Figure 21 : debug in exponent

3. On lance le debug de l'application :

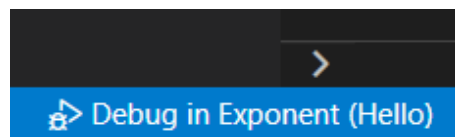


Figure 22: debug

Chapitre 14 : Le Routage

Introduction :

Le routage ou la navigation permet de naviguer entre différents écrans . le routage est très important pour les applications mobile car il fournit un contexte à l'utilisateur sur l'endroit où il se trouve , il se déplace entre les écrans et les fenêtres .

Exemples :

StepUp Préparation :

Installation des modules :

- `npm install @react-navigation/native @react-navigation/native-stack`
- `expo install react-native-screens react-native-safe-area-context`
(c'est mieux d'utiliser **yarn** cli)

Stack Composant :

Exemple de navigation entre deux screens :

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { useNavigation } from '@react-navigation/native';
import { Button, View, Text } from 'react-native';

const Stack = createStackNavigator()

function Stacks(props) {
  const navigation = useNavigation()
  const First_Screen = () => {
    return (
      <View>
        <Text>First Screen</Text>
        <Button title="go to 2 screen" onPress={()=>navigation.navigate("Second")} />
      </View>
    );
  };

  const Second_Screen = () => {
    return (
      <View>
        <Text>Second Screen</Text>
      </View>
    );
  };

  const MyStack = () => {
    return (
      <Stack.Navigator>
        <Stack.Screen name="First" component={First_Screen}/>
        <Stack.Screen name="Second" component={Second_Screen}/>
      </Stack.Navigator>
    );
  };

  return (
    <MyStack/>
  );
}

export default Stacks;
```

Figure 24 : MyStack

```
export default function App() {
  return (
    <NavigationContainer>
      <Stacks/>
    </NavigationContainer>
  );
}
```

Screens :

Figure 23 : Navigator

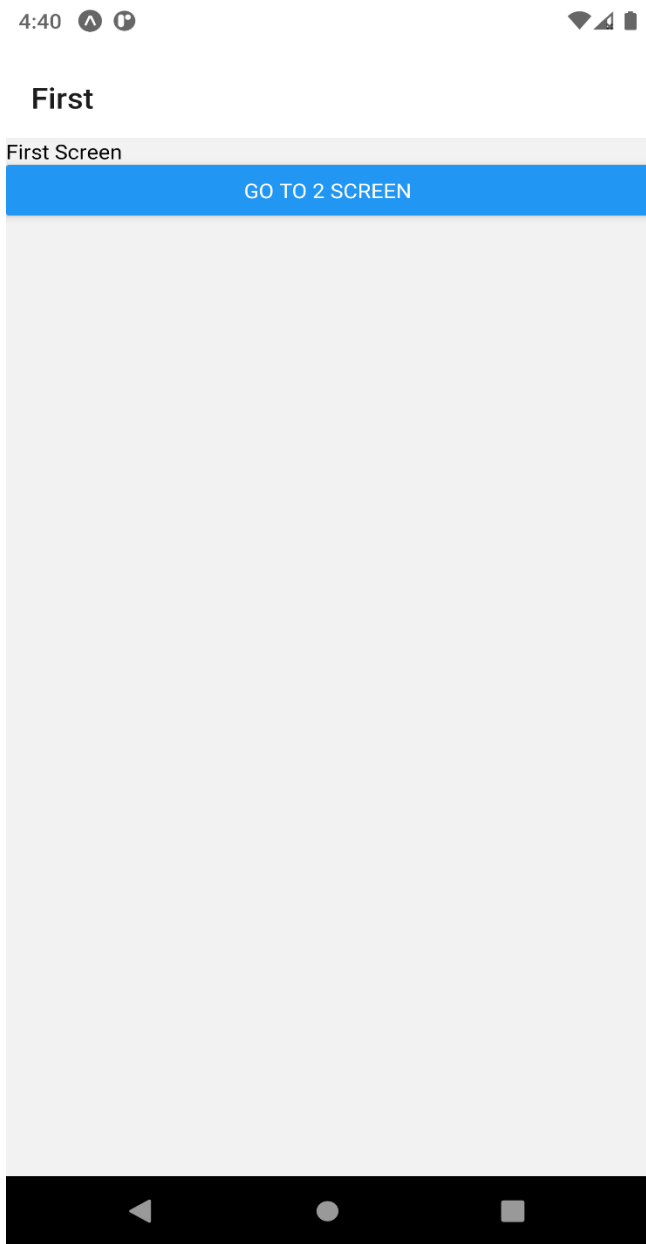


Figure 25: First Screen

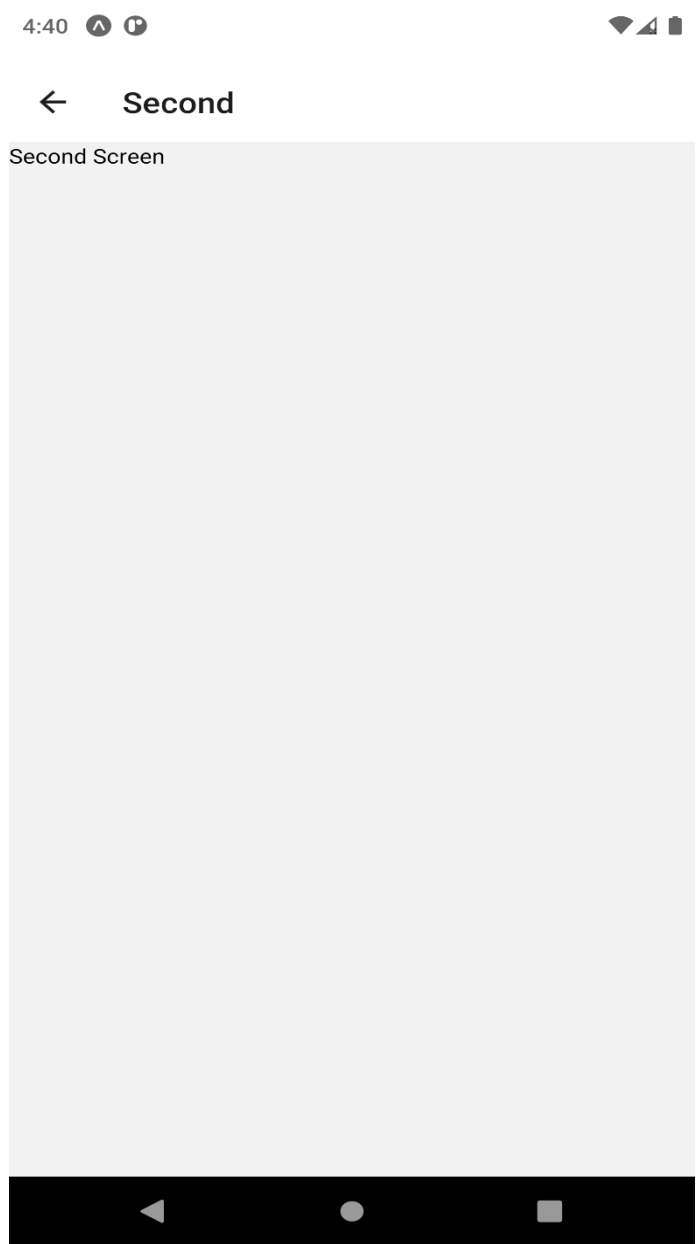


Figure 26: Second Screen

Top Tabs :

```
const Tab = createMaterialTopTabNavigator();
const First_Top={()=>{
  return (
    <View style={styles.Screen}>
      <Text>First Top</Text>
    </View>
  );
}
const Second_Top={()=>{
  return (
    <View style={styles.Screen}>
      <Text>Second Top</Text>
    </View>
  );
}
const MyTops={()=>{
  return(
    <NavigationContainer>
      <Tab.Navigator style={{ marginTop:40,}} >
        <Tab.Screen name="First Top" component={First_Top}/>
        <Tab.Screen name="Second Top" component={Second_Top}/>
      </Tab.Navigator>
    </NavigationContainer>
  );
}
function TabTop(props) {
  return (
    <MyTops/>
  );
}
export default TabTop;
const styles = StyleSheet.create({
  Screen: {
    flex : 1,alignItems:'center',justifyContent:'center',
  },
})
```

Figure 27: Top Tabs

Screen :

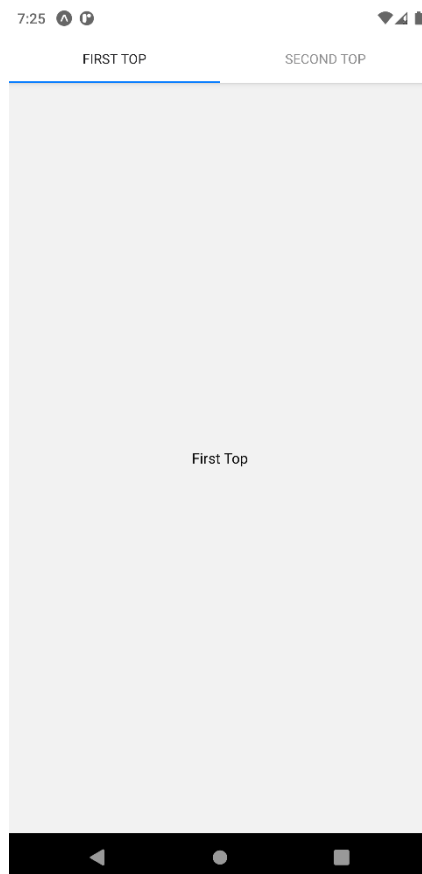


Figure 28: Tab Tops screen

Bottom Tabs

```
function BottomTab(props) {
  const Bottom=createBottomTabNavigator()
  const Tab1 =()=>{
    return (
      <View style={styles.screen}>
        <Text>Tab 1 </Text>
      </View>
    );
  }
  const Tab2 =()=>{
    return (
      <View style={styles.screen} >
        <Text>Tab 2 </Text>
      </View>
    );
  }
  return (
    <NavigationContainer>
      <Bottom.Navigator>
        <Bottom.Screen name="Tab 1" component={Tab1} />
        <Bottom.Screen name ="Tab 2" component={Tab2}/>
      </Bottom.Navigator>
    </NavigationContainer>
  );
}
export default BottomTab;
const styles = StyleSheet.create({
  screen : { flex :1 , justifyContent:'center',alignItems:"center"}
})
```

Figure 29: Bottom tabs

Screen :

7:43



Tab 1

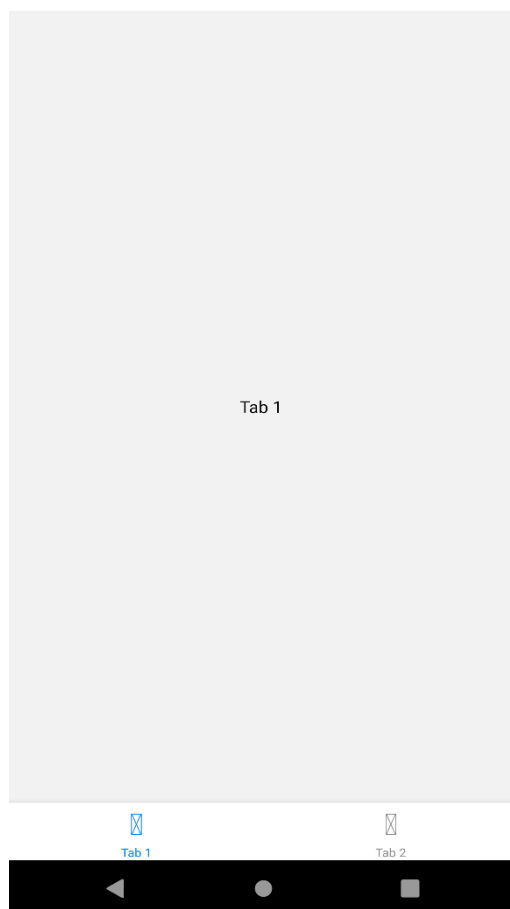


Figure 30: Bottom tabs1

Chapitre 15 : Liaison de l'API native

Introduction :

L'api de liaison permet d'envoyer et de recevoir des liens entre les applications . par exemple ouvrir l'application téléphone avec le numéro composé ou Google Maps .on peut également utiliser **Linking**

Pour répondre aux liens externes depuis d'autres applications .

Pour utiliser 'Linking ' on doit l'importer depuis react-native.

Exemples :

Liens sortants

```
<View style={styles.open_row}>
  <Text> call Gear9 </Text>
  <Button title="call" onPress={()=>Linking.openURL("tel:0666666666")} />
</View>
```

Figure 31: lien sortant

Liens entrants

```
<View style={styles.open_row}>
  <Text> open gear9 web site </Text>
  <Button title="open website" onPress={()=>Linking.openURL("https://gear9.ma/")} />
</View>
```

Figure 32 : lien entrant

Screens :

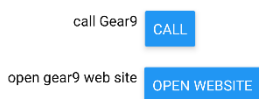


Figure 34: Linking

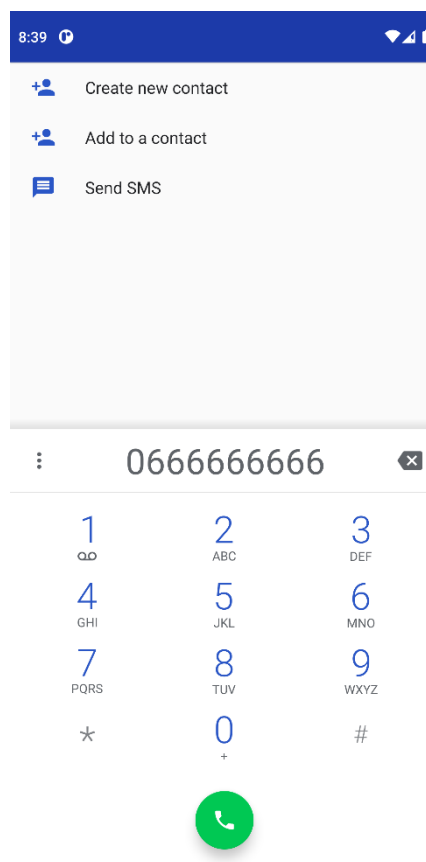


Figure 33: lien sortant screen

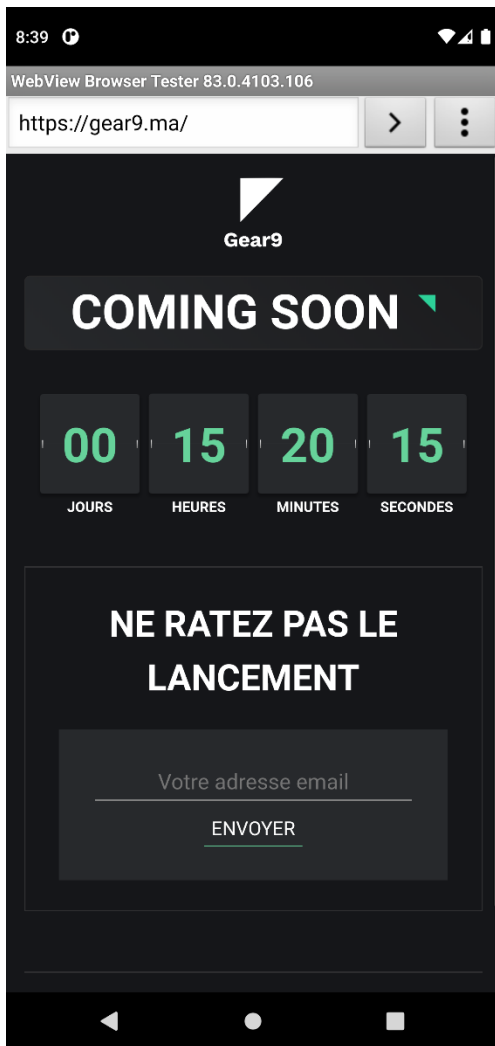


Figure 35: lien entrant

Chapitre 16 : ListView

Exemples :

ListView est un composant conçu pour un affichage efficace des listes de données changeantes défilant verticalement .il y a plusieurs de types pour l'utilisation de listView FlatList

```
import React from 'react';
import { FlatList, StyleSheet, Text, View } from 'react-native';
const users=[{id : 1,name : "user 1",job : "job 1"},{id : 2,name : "user 2",job : "job 2"},{id : 3,name : "user 3",job : "job 3"},{id : 4,name : "user 4",job : "job 4"}];
function Flatlist_View(props) {
  return (
    <View style={styles.container}>
      <FlatList
        data={users}
        renderItem={({item})=>
          <View style={styles.user}>
            <Text style={styles.Items}>id : {item.id}</Text>
            <Text style={styles.Items}>name: {item.name}</Text>
            <Text style={styles.Items}>job :{item.job}</Text>
          </View>
        }
      />
    </View>
  );
}
export default Flatlist_View;
```

Figure 36:flatList

Screen :

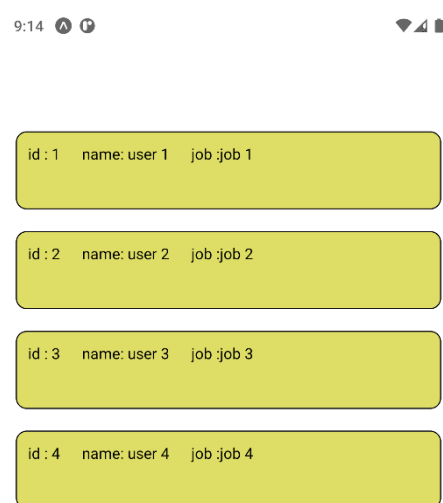


Figure 37: flatlist users



SectionList

```
import React from 'react';
import { SectionList, StyleSheet, Text, View } from 'react-native';
const users=[
  {name : "user 1",data: ["ahmed","samir","jhon"]},
  {name : "user 2",data: ["user 1","samir","jhon"]},
  {name : "user 3",data: ["user 1","samir","jhon"]},
  {name : "user 3",data: ["user 1","samir","jhon"]},]

function Sectionlist_View(props) {
  return (
    <View style={styles.container}>
      <Text style={{fontSize:20,marginHorizontal:60,}}> Users and their friends</Text>
      <SectionList
        sections={users}
        renderItem={({item})=><Text>{item}</Text>}
        renderSectionHeader={({section})=><Text style={styles.name_style}>{section.name}</Text>}
        keyExtractor={(item,index)=>index}
      />
    </View>
  );
}
```

Figure 38:SectionList

Screen :

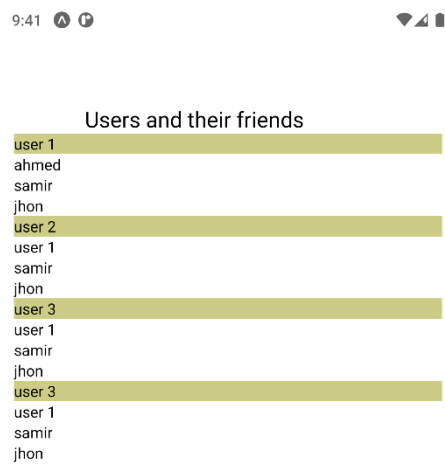


Figure 39: SectionList Screen

Map Array

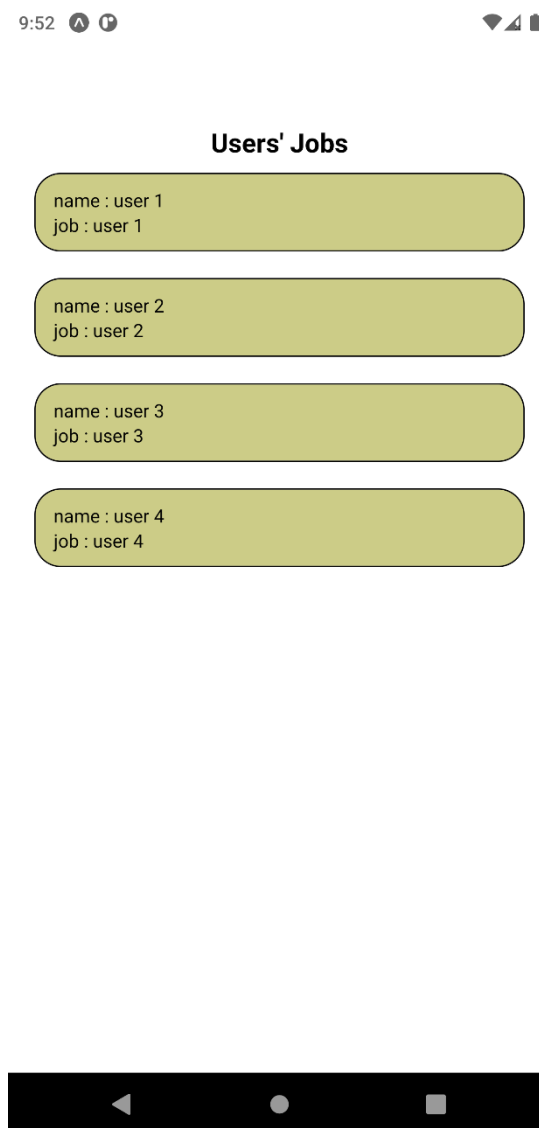
```
import React from 'react';
import { SectionList, StyleSheet, Text, View } from 'react-native';

function Map_Array_View(props) {
  const users=[{id : 1,name : "user 1",job : "job 1"},{id : 2,name : "user 2",job : "job 2"},{id : 3,name : "user 3",job : "job 3"},{id : 4,name : "user 4",job : "job 4"}];
  return (
    <View style={styles.container}>
      <Text style={styles.Title_Text}>Users' Jobs</Text>
      {
        users.map((user,index)=>{
          return (
            <View style={styles.user_view}>
              <Text > name : {user.name}</Text>
              <Text > job : {user.name}</Text>
            </View>
          );
        })
      }
    </View>
  );
}

export default Map_Array_View;
```

Figure 40:map array view

screen :



Chapitre 17 : Input Data

Exemples :

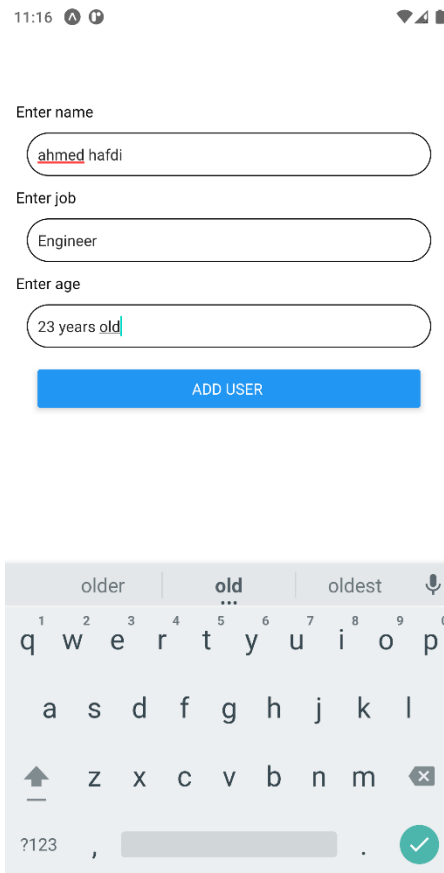
TextInput :

TextInput est un composant pour entrer les données via le clavier .

```
<View style={styles.container}>
  <Text>Enter name</Text>
  <TextInput style={styles.input}
    onChangeText={(text)=>{user.user_name=text}}
    placeholder="name"
  />
  <Text>Enter job</Text>
  <TextInput style={styles.input}
    placeholder="job"
    onChangeText={(text)=>{user.user_job=text}}
  />
  <Text>Enter age</Text>
  <TextInput style={styles.input}
    onChangeText={(text)=>{user.user_age=text}}
    placeholder="age"
  />
  <View style={styles.button}>
    <Button title="add user " onPress={()=>setUser({
      user_name:user.user_name, user_age: user.user_age,user_job:user.user_job
    })}/>
  />
</View>
```

Figure 41: TextInput

Screen :



11:16

Enter name

ahmed hafdi

Enter job

Engineer

Enter age

23 years old

ADD USER

older old oldest

q w e r t y u i o p

a s d f g h j k l

z x c v b n m

?123 , .

Figure 42: add user screen

Chapitre 18 : Modal

Introduction :

Modal est composant ou un moyen simple de présenter du contenu au-dessus d'une vue englobante.

Paramètres :

Tableau 1: Modal parameters

Fonction	Détails
animationType	('none', 'slide', 'fade') il contrôle l'animation modele.
Visible	Est un booléen qui contrôle la visibilité.
onShow	Il permet de passer une fonction qui sera une fois le modal affiché.
transparent	Bool pour la transparence.
onRequestClose (android)	Méthode pour le bouton retourner
onOrientationChange(ios)	Méthode lorsque l'orientation change
SupportedOrientations(IOS)	enum («portrait», «portrait à l'envers», «paysage», «paysage à gauche», «paysage à droite»)

```
import React, { useEffect, useState } from 'react';
import { Button, ImageBackground, Modal, SafeAreaView, SectionList, StyleSheet, Text, TextInput, View } from 'react-native';

function Modal_View_Comp(props) {
  const [open, setopen] = useState(false)
  return (
    <View style={styles.container}>
      <Modal visible={open}>
        <ImageBackground style={styles.modal} source={require("../Project_Hello/assets/gear9_1.jpg")} >
          <View>
            <Button title="close gear9 logo" onPress={()=>setopen(false)} />
          </View>
        </ImageBackground>
      </Modal>
      <Text>gear 9 logo : </Text>
      <Button title="see gear9 logo" onPress={()=>setopen(true)} />
    </View>
  );
}
```

Figure 43: Modal

Screens :

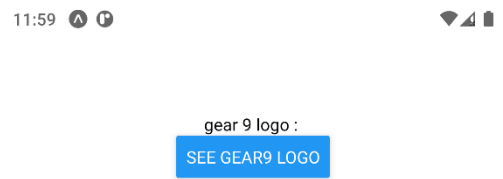


Figure 44: Modal 1

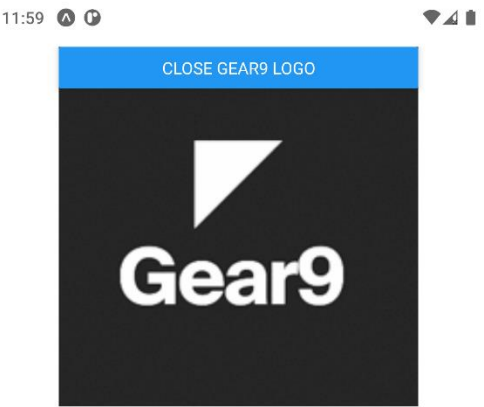


Figure 45: modal 2

Chapitre 19 : Module Plateforme

Introduction :

Module plateforme permet de détecter le type de système d'exploitation.

Exemple :

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

function Plateforme_android_ios_comp(props) {
  const android_image = "../../Project_Hello/assets/android.png"
  const ios_image = "../../Project_Hello/assets/ios.png"
  const Android = () => {
    return (
      <View style={styles.container}>
        <Text style={styles.text}> I am {Platform.OS}</Text>
        <Image source={require(android_image)} style={styles.image_style}/>
      </View>
    );
  }
  const Ios = () => {
    return (
      <View style={styles.container}>
        <Text style={styles.text}> I am {Platform.OS}</Text>
        <Image source={require(ios_image)} style={styles.image_style}/>
      </View>
    );
  }
  const Plateforme_Ios_or_android = () => {
    if (Platform.OS === 'android') {
      return <Android/>
    } else {
      return <Ios/>
    }
  }
  return (
    <Plateforme_Ios_or_android/>
  );
}
```

Figure 46: Plateforme

Scren :

2:04

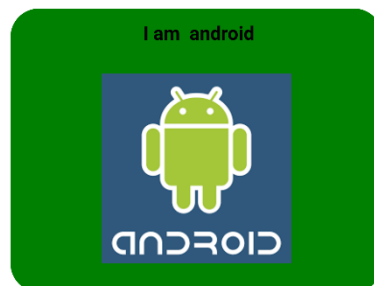


Figure 47: Platform Screen

Chapitre 20 : Notification push

On utilise Push Notification pour réagir avec une application native en utilisant le module react-native-push-notification pour cette documentation j'ai utilisé un autre module pour Expo

C'est expo-permissions et expo-notifications.

Exemple :

```
function PushNotif(props) {
  const notificationListener = useRef();
  const responseListener = useRef();
  useEffect(() => {
    registerForPushNotification().then(token=>console.log(token));
    return () => {
      Notifications.removeNotificationSubscription(notificationListener.current);
      Notifications.removeNotificationSubscription(responseListener.current);
    }
  }, []);
  async function registerForPushNotification(){
    const {status} = await Permissions.getAsync(Permissions.NOTIFICATIONS);
    if (status !== 'granted') {
      const { status } = await Permissions.askAsync(Permissions.NOTIFICATIONS);
      finalStatus = status;
    }
    if (status !== 'granted') {
      alert('Failed to get push token for push notification!');
      return;
    }
    const token = (await Notifications.getExpoPushTokenAsync()).data;
    return token
  }
  return (
    <View style={styles.container}>
      <Text style={styles.text}> Push Notification RN expo</Text>
    </View>
  );
}
```

Figure 48: Push Notification

Expo Push Notifications Tool :

To (Expo push token from your app)	From (Expo push token from your app)	From (Expo push token from your app)	From (Expo push token from your app)	From (Expo push token from your app)	From (Expo push token from your app)	From (Expo push token from your app)	From (Expo push token from your app)
ExponentPushToken[202PqaGpYP9_BOCAdsB8Ny]							
Access Token (if you have enabled push security)							
Message title							
notification gear9							
Message body							
Gear9 is here							
Data (JSON)							
TTL (seconds)							

Figure 49: expo push notification tool

Screen :

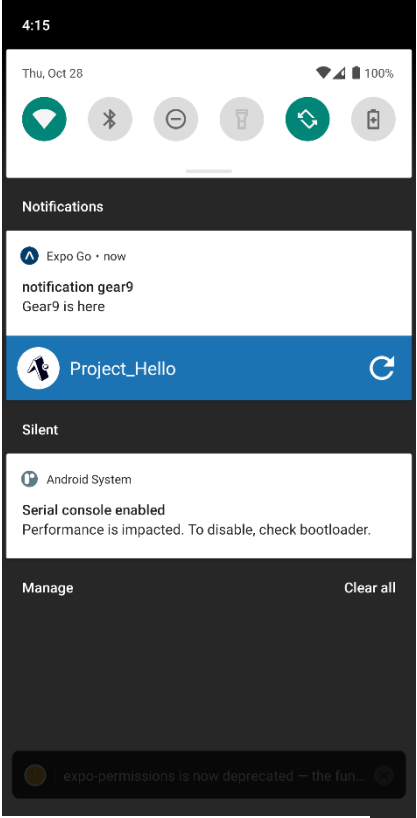


Figure 50: notification screen

Chapitre 21 : Polices Personnalisées

Exemples :

On Peut télécharger n'importe quelle type de police (par exemple à partir de google fonts)

```
import React from 'react';
import { View, Text } from 'react-native';
import AppLoading from 'expo-app-loading';
import { useFonts, Inter_900Black, } from '@expo-google-fonts/inter';

function AddPolice(props) {
  let [fontsLoaded] = useFonts({
    'Inter-black': require('./assets/fonts/InterGrover-Regular.ttf'),
    'Festive-black': require('./assets/fonts/Festive-Regular.ttf'),

  });

  if (!fontsLoaded) {
    return <AppLoading />;
  } else {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text style={{ fontFamily: 'Inter-black', fontSize: 40 }}>Irish Grover </Text>
        <Text style={{ fontFamily: 'Festive-black', fontSize: 40 }}>Festive </Text>
      </View>
    );
  }
}

export default AddPolice;
```

Figure 51: fonts

Screen :

5:03



Irish Grover
Festive



Figure 52: fonts exemples

Chapitre 22 : RefreshControl

```
const [refreshing, setrefreshing] =useState(false)
const wait =(timeout)=>{
  return new Promise(resolve=>setTimeout(resolve,timeout))
}
const OnRefresh =()=>{
  setrefreshing(true)
  wait(2400).then(()=>setrefreshing(false))
}
return (
  <View style={styles.container}>
    <ScrollView
      refreshControl={
        <RefreshControl
          refreshing={refreshing}
          onRefresh={OnRefresh}
        />
      }
    />
    <Text style={styles.Title_Text}>Users' Jobs</Text>
    <Text style={{alignSelf:"center"}}> pull down to refresh</Text>
    {
      users.map((user,index)=>{
        return (
          <View style={styles.user_view}>
            <Text > name : {user.name}</Text>
            <Text > job : {user.name}</Text>
          </View>
        );
      })
    }
  </ScrollView>
</View>
```

Figure 53: refresh control

Screens :

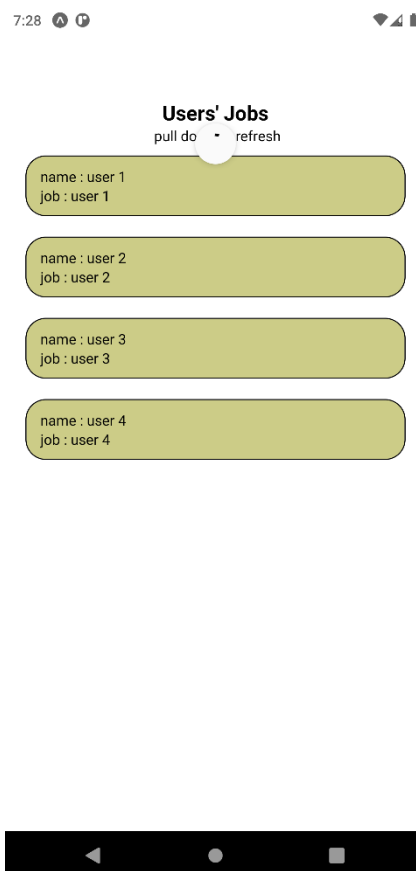


Figure 54:Refresh control screen

Chapitre 23 : Requêtes http

Syntaxe :

- `Fetch(url , options) [.alors(...)[.catch(...)]`

Exemple : Posting application avec json server

```
import { TouchableOpacity } from 'react-native-gesture-handler';
import { Ionicons } from '@expo/vector-icons';
function Requet(props) {
  const [data, setData] = useState([]);
  const [likes_num, setLikes]=useState(0);
  const Like =()=>{
    setLikes(likes_num+1)
  }
  const Dislike =()=>{
    if (likes_num===0)
    {
      setLikes(0)
    } else {
      setLikes(likes_num-1)
    }
  }

  const getPosts = async () => {
    try {
      const response =await fetch('http://192.168.0.106:3000/posts');
      const json = await response.json();
      setData(json);
    }catch(error) {
      console.log(error)
    }
  }

  useEffect(() => {
    getPosts();
  }, []);
  return (
    <ScrollView>
      <View style={styles.container}>
        <Text style={styles.Title_Text}>YouPost</Text>
        {
          data.map((post,index)=>{
            return (

```

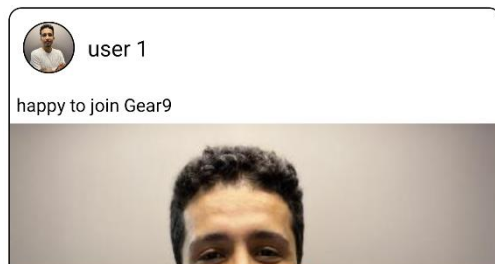
Screen :

Figure 55: fetch1

```
      <ScrollView>
        <View style={styles.container}>
          <Text style={styles.Title_Text}>YouPost</Text>
          {
            data.map((post,index)=>{
              return (
                <View style={styles.post_container}>
                  <View style={styles.avatar_row}>
                    <Image source={{uri : post.user_avatar}} style={styles.avatar}/>
                    <Text style={styles.user_text}>{post.user}</Text>
                  </View>
                  <Text style={styles.description_text}>{post.description}</Text>
                  <View>
                    <Image source={{uri : post.picture}} style={styles.image_post}/>
                  </View>
                  <View style={styles.like_dislike_row} >
                    <TouchableOpacity onPress={()=>setLikes(likes_num+1)}>
                      <Ionicons name="heart-outline" size={32} color="red" />
                    </TouchableOpacity>
                    <TouchableOpacity >
                      <Ionicons name="chatbubbles-outline" size={32} color="black" />
                    </TouchableOpacity>
                    <TouchableOpacity onPress={()=>Dislike()}>
                      <Ionicons name="heart-dislike-outline" size={32} color="black" />
                    </TouchableOpacity>
                  </View>
                </View>
              );
            }
          )
        </View>
      </ScrollView>
    );
  }
}
```

Figure 56: fetch 2

YouPost



Chapitre 24 : Test d'unité

Introduction :

Le test unitaire est une pratique de test de bas niveau où les plus petites unités ou composants du code sont testés.

Exemples :

```
import React, { useEffect, useState } from 'react';
import { Button, StyleSheet, Platform, Text, View, Image, ScrollView } from 'react-native';

function Buttons(props) {
  return (
    <View style={styles.container}>
      <Text style={styles.Title_Text}>TESTING JEST {props.text}</Text>
      <Button title="button for testing" />
    </View>
  );
}

export default Buttons;
const styles = StyleSheet.create({
  container: {marginTop:90,margin:10,marginBottom:10,},
  Title_Text: {fontSize:20,fontWeight:'bold',alignSelf:'center',margin : 20,},
});
```

Figure 57: Test for buttons component

```
import { exportAllDeclaration } from "@babel/types";
import React from "react";
import renderer from 'react-test-renderer';
import Buttons from "../components_testing/buttons";
describe ( '<Buttons/>', () =>{
  it('has 1 child ', ()=>{
    const tree=renderer.create(<Buttons text="test from jest"/>).toJSON();
    expect(tree.children.length).toBe(2);
  });
});
```

Figure 58: buttons.test.js

```
C:\Users\ahafdi\Desktop\react native\gear9_projects\Project_Hello>yarn test
yarn run v1.22.11
$ jest
PASS __tests__/components/buttons.test.js
  <Buttons/>
    ✓ has 1 child (487 ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 2.192 s, estimated 3 s
Ran all test suites.
Done in 3.23s.
```

Figure 59: Test passed

Chapitre 25 : Mobx

Introduction :

State



Figure 60: Mobx + React Native

Management ou la gestion de états fait partie de développement des applications avec JavaScript. En particulier , les applications de React et React Native . dans cette documentation on va apprendre à utiliser la bibliothèque Mobx pour la gestion des états , comprendre les concepts de base .

C'est quoi Mobx :

Mobx est une bibliothèque éprouvée qui simplifier la gestion des états .

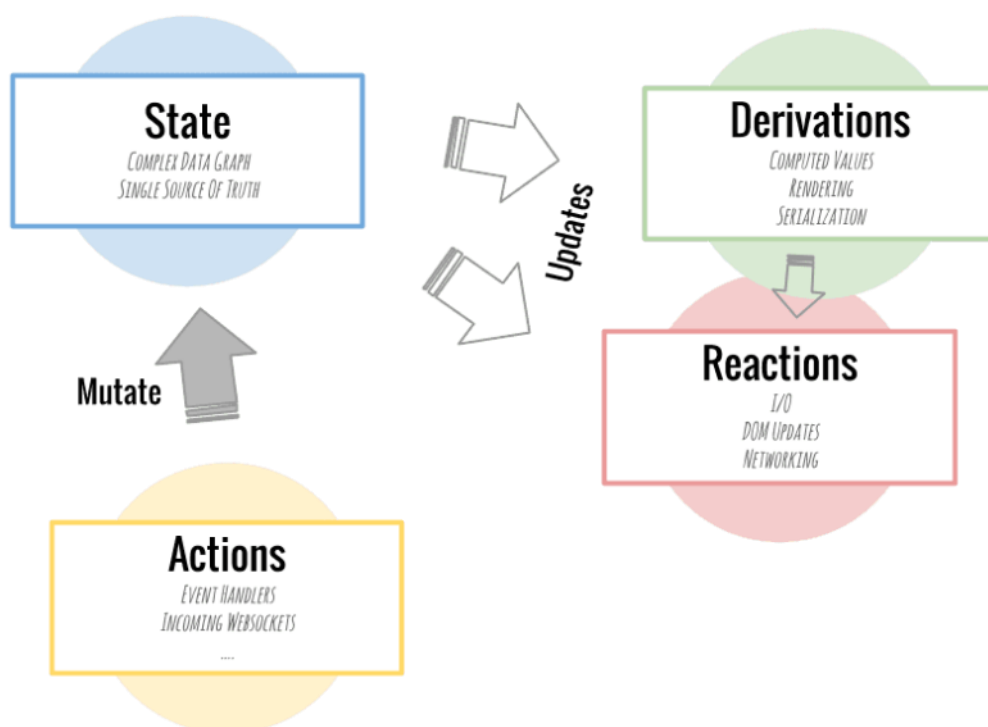


Figure 61: Mobx state architecture

Les Principes de Mobx :

- **STATE**

STATE correspond aux données que contient l'application , il s'agit aussi le contenu de sa mémoire . ceci s'applique également à les composants .

- **DERIVATIONS**

Dans Mobx ; tout ce qui peut être dérivé de l'état sans interactions est une dérivation .

- **ACTIONS**

Contrairement , aux dérivations les actions sont tous les composants et les fonctions qui mettent un changement sur les données.

Exemple 1 : une application simple des taches avec state management useState ()

```
function Task_SetState(props) {
  const [item,setItem]=useState('task 0');
  const [items,setItems]=useState(['task 1']);
  const RemoveTask =(index)=>{
    let CopyTasks=[...items];
    CopyTasks.splice(1,index);
    setItems(CopyTasks)
  }
  return (
    <ScrollView>
      <View style={styles.container}>
        <Text style={styles.Title_Text}> ToDo Tasks setstate</Text>
        <View style={styles.input_container}>
          <TextInput
            placeholder='write task'
            onChangeText={(text)=>setItem(text)}
          />
          <Button title="add task" onPress={()=>setItems([...items,item])}/>
        </View>
        {
          items.map((tasks,index)=>{
            return (
              <View style={styles.task_container} >
                <Text key={index}> {tasks} </Text>
                <TouchableOpacity onPress={()=>RemoveTask(index)}>
                  <Icons name="trash-outline" size={32} color="red" />
                </TouchableOpacity>
              </View>
            );
          })
        }
      </View>
    </ScrollView>
  )
}
```

Figure 62: Task with UseState

Screen :

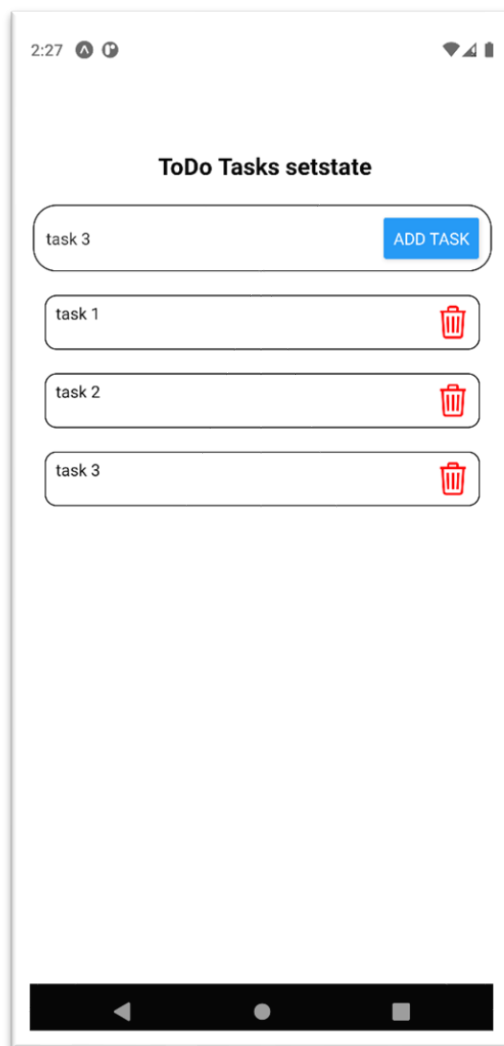


Figure 63: Tasks useState screen

Exemple 2 :une application simple des taches avec state management Mobx

```
export default Task_Mobx = observer(()=> {  
  return (  
    <ScrollView>  
      <View style={styles.container}>  
        <Text style={styles.Title_Text}> ToDo Tasks Mobx</Text>  
        <View style={styles.input_container}>  
          <TextInput  
            placeholder='write task'  
            onChangeText={(text)->{new_store.tache=text}}  
          />  
          <Button title="add task" onPress={()=>new_store.addTaches(new_store.tache)} />  
        </View>  
        {  
          new_store.taches.map((tasks,index)->{  
            return (  
              <View style={styles.task_container} >  
                <Text key={index}> {tasks} </Text>  
                <TouchableOpacity onPress={()=>new_store.removetache(index)}>  
                  <Ionicons name="trash-outline" size={32} color="red" />  
                </TouchableOpacity>  
              </View>  
            );  
          })  
        }  
      </View>  
    </ScrollView>  
  )  
})
```

Figure 64: Tasks With Mobx

Screen :

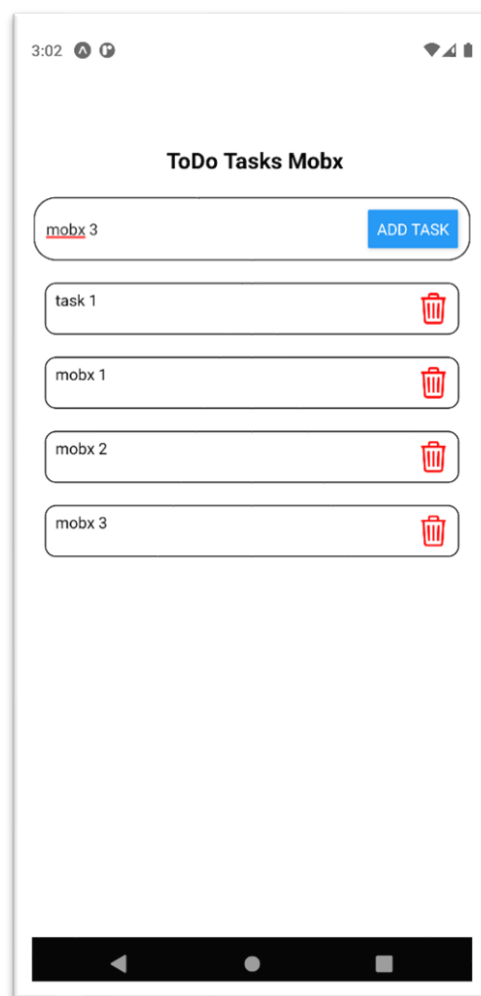


Figure 65: Mobx Tasks Screen

Exemple 3 : Exemple 2 :une application simple des taches avec state management MST (Mobx State Tree)

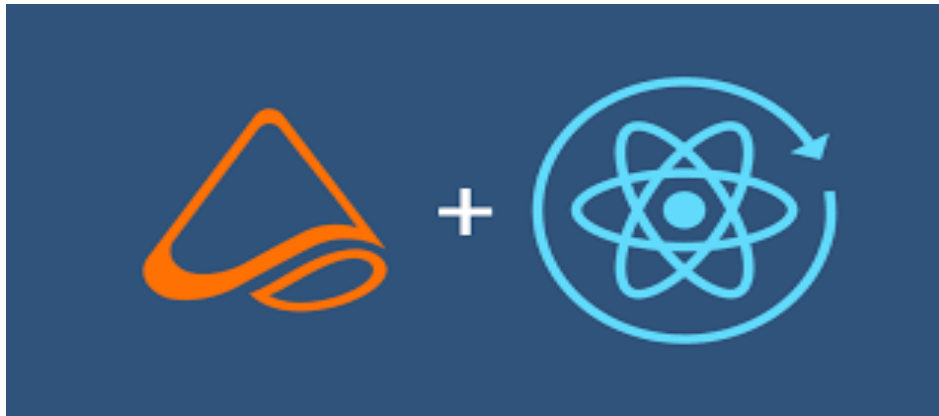


Figure 66: MST RN

C'est quoi MST :

Mobx State Tree est un bibliothèque de react et react native il permet de gérer les états mais mieux que Mobx car il permet de gérer les types de données et créer des modeles .

```
import {destroy, types} from "mobx-state-tree";
const Book = types.model('Book',{
  title : types.string,
  author : types.string,
})
const BookStore = types.model('Books',{
  books: types.array(Book)
})
.actions(self =>({
  addBook(book) {
    self.books.push(book)
  },
  removeBook(book) {
    destroy(book)
  }
}))
.create({
  books : [{title : "book 1",author : "author 1"},{title : "book 2",author : "author 2"}]
})
export default BookStore
```

Figure 67: Book Store

```

@Observer
export default class BookApp extends Component {
  state=initalestate;
  Value_Key (key,value){
    this.setState({
      [key]:value
    })
  };
  AjouterBook(){
    BookStore.addBook(this.state);
    this.setState(initalestate)
  };
  deleteBook(book) {
    BookStore.removeBook(book)
  }
  render() {
    return (
      <ScrollView>
        <View style={styles.container}>
          <Text style={styles.Title_Text}> Books APP</Text>
          <View style={styles.Book_Container}>
            <View style={styles.input_container}>
              <TextInput placeholder="Book Title"
                onChangeText={(titre)->this.Value_Key('title',titre)}
              />
            </View>
            <View style={styles.input_container}>
              <TextInput placeholder="Book Author"
                onChangeText={(ecriv)->this.Value_Key('author',ecriv)}
              />
            </View>
            <View style={styles.button_add}>
              <Button title ="Add Book"
                onPress={()=>this.AjouterBook()}
              />
            </View>
          </View>
        </ScrollView>
      )
    );
  }
}

```

Figure 68: BookView 1

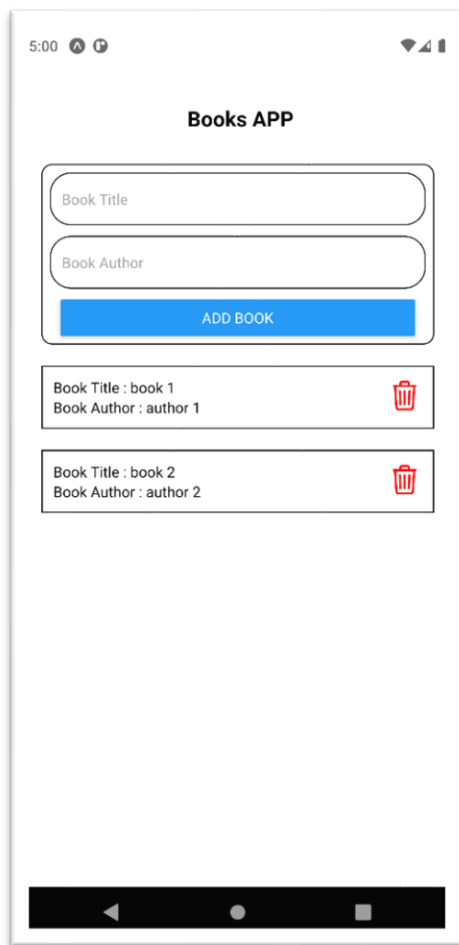
```

return (
  <ScrollView>
    <View style={styles.container}>
      <Text style={styles.Title_Text}> Books APP</Text>
      <View style={styles.Book_Container}>
        <View style={styles.input_container}>
          <TextInput placeholder="Book Title"
            onChangeText={(titre)->this.Value_Key('title',titre)}
          />
        </View>
        <View style={styles.input_container}>
          <TextInput placeholder="Book Author"
            onChangeText={(ecriv)->this.Value_Key('author',ecriv)}
          />
        </View>
        <View style={styles.button_add}>
          <Button title ="Add Book"
            onPress={()=>this.AjouterBook()}
          />
        </View>
      </View>
    </ScrollView>
    <View>
      {
        books.map((livre,index)->{
          return (
            <View style={styles.Book_Info}>
              <View style={styles.book_view}>
                <Text>Book Title : {livre.title}</Text>
                <Text>Book Author : {livre.author}</Text>
              </View>
              <TouchableOpacity style={styles.trash} onPress={()=>this.deleteBook(livre)}>
                <Ionicons name="trash-outline" size={32} color="red" />
              </TouchableOpacity>
            </View>
          );
        })
      }
    </View>
  )
);

```

Figure 69: BookView 2

Screen :



Chapitre 26: Mini Projet 1 (Firebase users app)

Firebase users app : est une application simple d'ajouter /créer , supprimer et mettre à jour des comptes des utilisateurs c'est application pour admin pour gérer des comptes tout ça en utilisant la base de données firebase.

Code source :

Database/firebase.js :

```
// Import the functions you need from the SDKs you need
import firebase from "firebase/app";
import "firebase/firestore";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyDz6remGPgHoo00gsZxx5QFjZzsTfI00oJI",
  authDomain: "crudbook-127e5.firebaseio.com",
  projectId: "crudbook-127e5",
  storageBucket: "crudbook-127e5.appspot.com",
  messagingSenderId: "85964288116",
  appId: "1:85964288116:web:b6bd7b6a7b5544b8326121",
  measurementId: "G-XXPMR15E06"
};
// Initialize Firebase
firebase.initializeApp(firebaseConfig);
firebase.firestore();

export default firebase ;
```

Figure 70: firebase

Add User Screen :

```
class AddUserScreen extends Component {
  constructor () {
    super();
    this.dbRef=firebase.firestore().collection('users');/*connection to DB */
    this.state={
      name:'',
      email :'',
      mobile : '',
      isLoading : false
      /*this.state takes a column of data from the db */
    };
  }

  inputValueUpdate =(val,prop)=>{
    const state = this.state;
    state[prop]=val;
    this.setState(state);
    /*affecting the value inputed to this.state(column of data) */
  }

  storeUser () {
    if(this.state.name===''){
      alert("name is empty fill it !")
    } else {
      this.setState(
        {
          isLoading:true,
        }
      );
      this.dbRef.add({
        name : this.state.name,
        email : this.state.email,
      })
    }
  }
}
```

Figure 71: adduserscreen1


```

        this.setState({
          name : '',
          email : '',
          mobile : '',
          isLoading : false,
        });
        this.props.navigation.navigate('UserScreen')
      }
    ).catch((err)=>{
      console.error("error found : ", err);
      this.setState({
        isLoading : false
      });
    })
  }
}

```

```

render() {
  if (this.state.isLoading){
    return (
      <View style={styles.preloader}>
        <ActivityIndicator
          size="large" color = "#9E9E9E"
        />
      </View>
    )
  }
}

```

Figure 72: adduserscreen2

```

return (
  <ScrollView style={styles.container}>
    <View style={styles.input_container}>
      <View style={styles.title_container}>
        <Icons name="people" size={23} color="blue" />
        <Text style={styles.title}> USERS APP</Text>
      </View>

      <View style={styles.inputgroup}>
        { /*name */}
        <Icons name ="person" size={23} style={{marginRight:10,}} color="green"/>

        <TextInput
          multiline={true}
          placeholder={'Name'}
          value={this.state.name}
          onChangeText={ (val)=>this.inputValueUpdate(val , 'name')}
        />

      </View>

      <View style={styles.inputgroup}>
        { /*email */}
        <Icons name ="mail" size={23} style={{marginRight:10,}} color="green"/>

        <TextInput
          placeholder={'Email'}
          value={this.state.email}
          onChangeText={ (val)=>this.inputValueUpdate(val , 'email')}
        />

      </View>

      <View style={styles.inputgroup}>
        <Icons name ="call" size={23} style={{marginRight:10,}} color="green" />
        { /*phone */}

```

Figure 73: adduserscreen3

```

<Ionicons name="mail" size={23} style={{marginRight:10,}} color="green"/>

<TextInput
  placeholder={'Email'}
  value={this.state.email}
  onChangeText={(val)=>this.inputValueUpdate(val , 'email')}
/>
</View>
<View style={styles.inputgroup}>
<Ionicons name="call" size={23} style={{marginRight:10,}} color="green" />
  { /*phone */ }

  <TextInput
    placeholder={'Mobile'}
    value={this.state.mobile}
    onChangeText={(val)=>this.inputValueUpdate(val , 'mobile')}
  />
</View>
</View>

<View style={{flexDirection:'row',justifyContent:'space-evenly'}}>

  <Button
    title="Add User"
    color ='green'
    onPress={()=>this.storeUser()}
  />
  <Button
    title="List Users"
    color="blue"
    onPress={()=>this.props.navigation.navigate("UserScreen")}
  />
</View>
</ScrollView>

```

Figure 74: adduserscreen4

List Users :

```

class UserScreen extends Component {
  constructor () {
    super();
    this.firestoreRef=firebase.firestore().collection('users');
    this.state = {
      isLoading : true,
      userArr : []
    }
  }
  componentDidMount () {
    this.unsubscribe=this.firestoreRef.onSnapshot(this.getCollection);
  }
  componentWillUnmount() {
    this.unsubscribe();
  }
  getCollection =(querySnapshot)=>{
    const userArr =[];
    querySnapshot.forEach((res)=>
    {
      const{name,email,mobile}=res.data();
      userArr.push({
        key:res.id,
        res,
        name,
        email,
        mobile,
      });
    }
  );
  this.setState({
    userArr,
    isLoading : false,
  })
}

```

Figure 75: list users 1

```

render() {
  if(this.state.isLoading){
    return(
      <View style={styles.preloader}>
        <ActivityIndicator size="large" color="#9E9E9E"/>
      </View>
    )
  }
  return (
    <ScrollView style={styles.container}>
      {
        this.state.userArr.map((item,i)=>
          [
            return (
              <ListItem key={i} bottomDivider
                onPress={()=>{this.props.navigation.navigate('UserDetailScreen',{
                  userKey : item.key
                })}}
              >
                <ListItem.Content>
                  <ListItem.Title>
                    {item.name}
                  </ListItem.Title>
                  <ListItem.Subtitle>
                    {item.email}
                  </ListItem.Subtitle>
                </ListItem.Content>
              </ListItem>
            );
          ]
        )
      }
    )
  )
}

```

Figure 76: list users 2

User Details Screen :

```

class UserDetailScreen extends Component {
  constructor () {
    super();
    this.state = {
      name: '',
      email: '',
      mobile: '',
      isLoading: true,
    }
  }
  inputValueUpdate = (val,prop)=>{
    const state=this.state;
    state[prop]=val;
    this.setState(state);
  }
  updateUser () {
    this.setState(
      {
        isLoading: true,
      }
    );
    const updateDBRef=firebase.firestore().collection('users').doc(this.state.key);
    updateDBRef.set({
      name : this.state.name,
      email : this.state.email,
      mobile : this.state.mobile,
    }).then((docRef)=>{
      this.setState({
        key: '',
        name: '',
        email: '',
        mobile: '',
        isLoading: false,
      });
    });
  }
}

```

Figure 77: user screen details 1

```

    }).catch((error)=>{
      console.error("Error : ",error);
      this.setState({
        isloading : false,
      });
    });
  });
}
deleteUser (){
  const dbRef = firebase.firestore().collection('users').doc(this.props.route.params.userKey);
  dbRef.delete().this((res)=>
  {
    console.log('Item removed from database')
    this.props.navigation.navigate('UserScreen');
  }
  )
}
openTwoButtonAlert=()=>{
  Alert.alert(
    'Delete User',
    'Are you sure?',
    [
      {text: 'Yes', onPress: () => this.deleteUser()},
      {text: 'No', onPress: () => console.log('No item was removed'), style: 'cancel'},
    ],
    {
      cancelable: true
    }
  );
}
}

```

Figure 78: user details screen 2

```

      <TextInput
        placeholder={"Email"}
        value={this.state.email}
        onChangeText={(val)=>this.inputValueUpdate(val, 'email')}
      />
    </View>
    <View style={styles.inputs}>
      <TextInput
        placeholder={"Mobile"}
        value={this.state.mobile}
        onChangeText={(val)=>this.inputValueUpdate(val, 'mobile')}
      />
    </View>
  </View>
  <View style={styles.buttons}>
    <Button
      title ="Update User"
      color="green"
      onPress ={()=>this.updateUser()}
    />
    <Button
      title="Delete User"
      color="red"
      onPress ={()=>this.openTwoButtonAlert()}
    />
  </View>
</ScrollView>

```

Figure 79: user screen details 3

Screens :

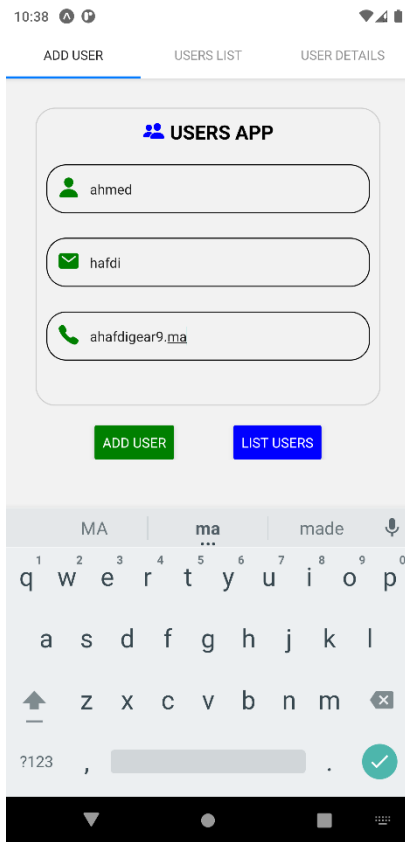


Figure 81: add user

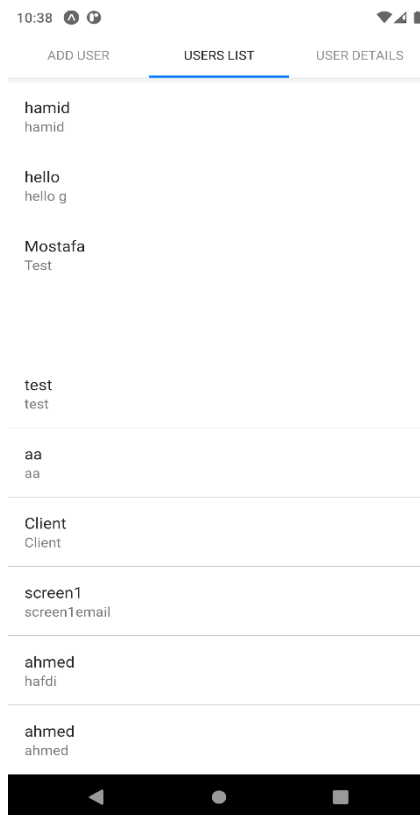


Figure 82: list users

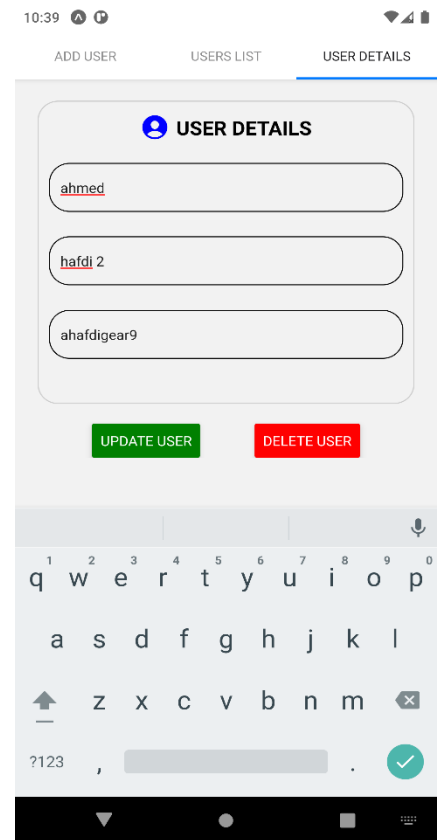


Figure 80: user details

Chapitre 27: Mini Projet 2 (Tasks app)

Voir Chapitre 25

