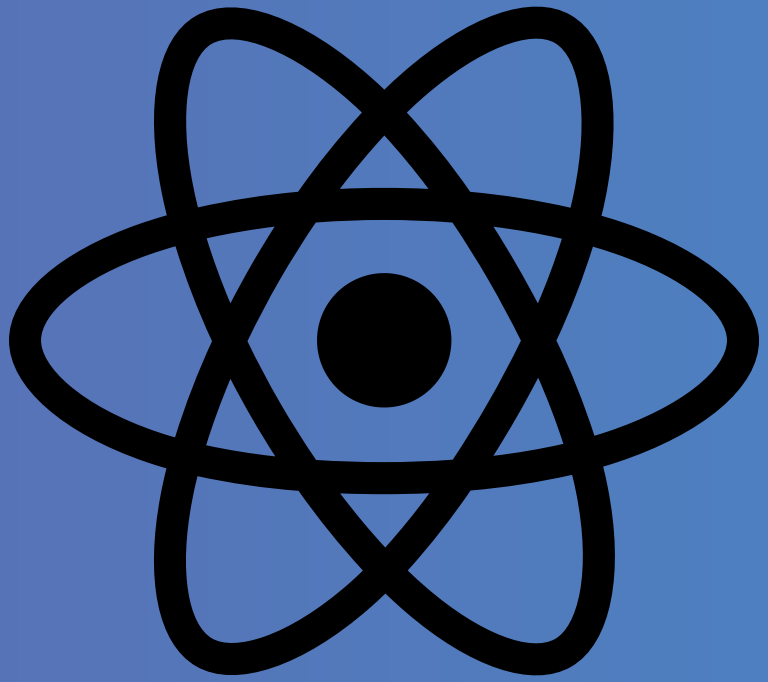
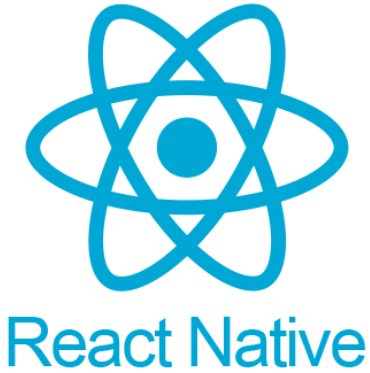
UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

React Native



Sommaire



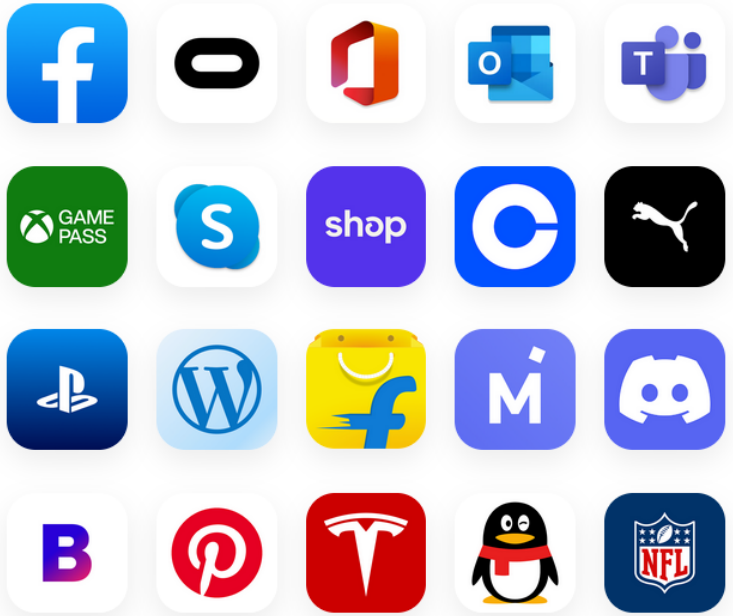
- Qu'est-ce que le Framework React Native ?
- Rappels React
- Composant Native et Core
- Environnement de développement React Native
- Core Components
- Navigation
- Persistance

Qu'est-ce que le Framework React Native ?

- React Native est un framework d'applications mobiles open sources créé par Facebook en 2015.
- Il permet de développer des applications pour Android, IOS mais aussi UWP (Univeral Windows Platform)
- Il reprend les principes de fonctionnement de React (Composant, props, state ...)

Qui utilise React Native ?

- React Native est utilisé par des milliers d'application que vous utilisez certainement tous les jours.



Rappel React

React Native fonctionne sur React, une bibliothèque open source populaire pour construire des interfaces utilisateur avec JavaScript. Pour tirer le meilleur parti de React Native, il est utile de comprendre React lui-même. Principe de base de React à connaître :

Components

JSX

props

state

Historique du développement mobile

Avant l'apparition de framework de développement d'application mobile comme React Native, Xamarin ou NativeScript le developpement mobile présentais des inconvénients, car le développement d'une application mobile multiplateforme nécessite un développement spécifique pour chaque plateforme (IOS, Android), ce qui présente plusieurs inconvénients :

- Temps de développement plus long
- Limitations de la portabilité du code
- Difficultés à obtenir le méme résultat sur chaque plateforme

Une solution temporaire pour pallier a cela fut de développer des applications web retranscris en version mobile mais celle-ci présente l'inconvénients de ne pas pouvoir accéder aux différentes fonctionnalités de la plateforme qu'elle utilise.

Composants Core et Native

React Native permet de développer des applications pour les système d'esploitation Android et IOS.

En React Native, il existe deux types de composants : les composants natifs (ou "native components") et les composants de base (ou "core components").

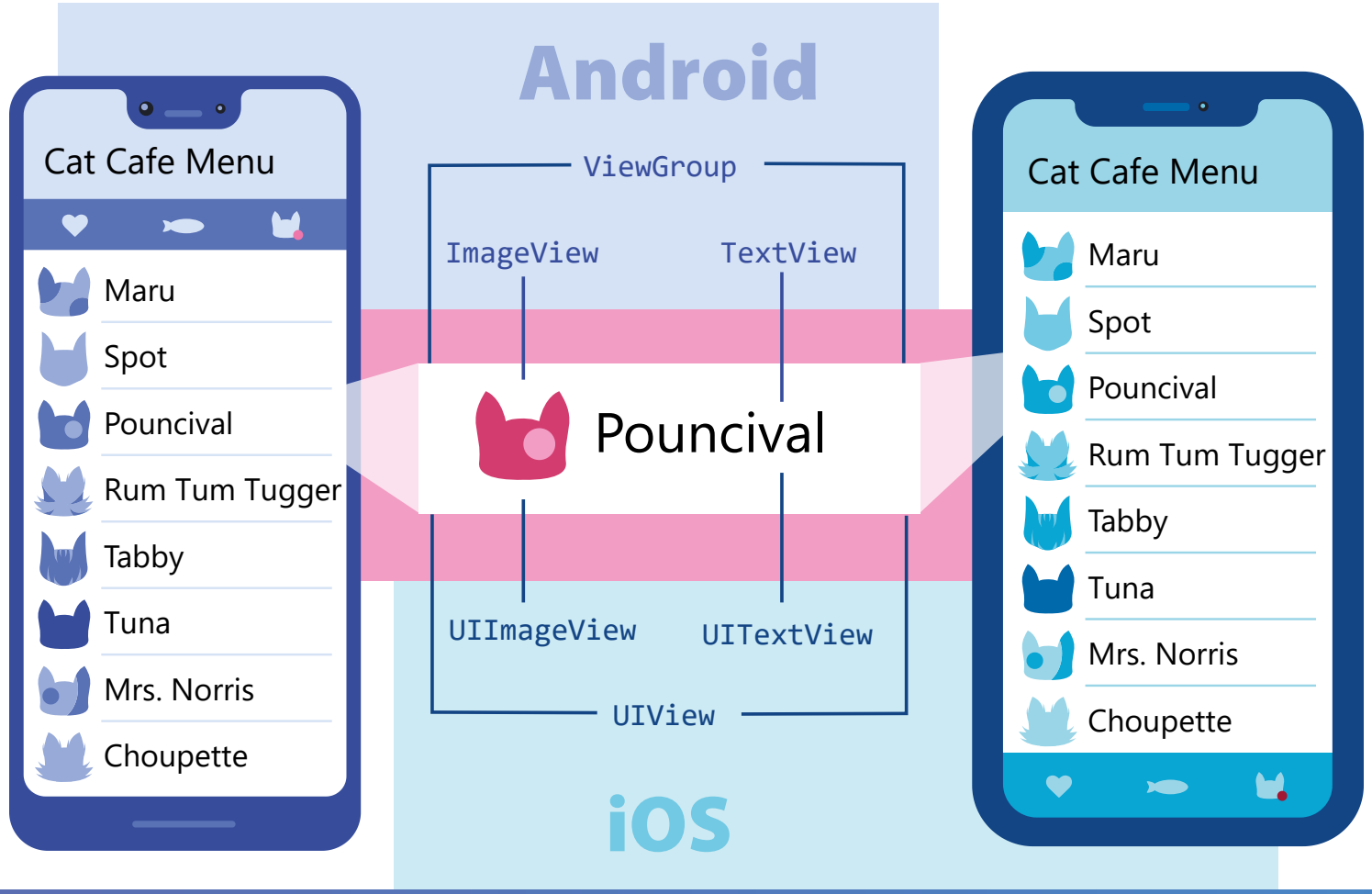
Composants Native

Les composants natifs (ou "native components") sont des composants directement liés aux éléments de l'interface utilisateur du système d'exploitation. Ecrit le plus souvent en Kotlin ou JAVA pour la plateforme Android et en Swift ou Objective-C pour la plateforme IOS. Ils sont généralement créés et rendus à l'aide des API natives du système d'exploitation, ce qui leur permet d'être plus performants et plus cohérents avec le style de l'interface utilisateur du système.

Composants Core

Les composants de base (ou "core components") sont des composants fournis par React Native qui peuvent être utilisés pour construire des interfaces utilisateur personnalisées. Ils ne sont pas directement liés aux éléments de l'interface utilisateur du système d'exploitation, mais ils peuvent être utilisés pour construire des interfaces utilisateur personnalisées. Parmi les composants de base, on trouve des éléments tels que View, Text, Image, ScrollView, etc. Ces composants sont écrits en JavaScript et sont donc moins performants que les composants natifs, mais ils offrent une grande flexibilité et sont souvent plus faciles à utiliser.

Composants Native



Queleques Composants React native et leur équivalent Android / IOS / web

React Native UI component	Android View	IOS View	Web Analog	Description
<View>	<ViewGroup>	<UIView>	non scrollable	Un conteneur qui prend en charge la mise en page avec flexbox, le style, certaines manipulations tactiles et les contrôles d'accessibilité
<Text>	<TextView>	<UITextView>	<p>	Affiche, stylise et imbrique des chaînes de texte et gère même les événements tactiles
<Image>	<ImageView>	<UIImageView>	<img>	Affiche différents types d'images
<ScrollView>	<ScrollView>	<UIScrollView>		Un conteneur de défilement générique pouvant contenir plusieurs composants et vues
<TextInput>	<EditText>	<UITextField>	<input type="text">	Permet à l'utilisateur de saisir du texte

Environnement de développement React Native

Afin de développer une application avec React Native nous avons plusieurs possibilités :

Expo Go :

Expo est un ensemble d'outils et de services construits autour de React Native et, bien qu'il possède de nombreuses fonctionnalités , la fonctionnalité la plus pertinente est qu'il peut vous permettre d'écrire une application React Native en quelques minutes. Vous n'aurez besoin que d'une version récente de Node.js et d'un téléphone ou d'un émulateur. Si vous souhaitez essayer React Native directement dans votre navigateur Web avant d'installer des outils, vous pouvez essayer Snack .
https://snack.expo.dev/.

Environnement de développement React Native

React Native CLI :

Afin d'utiliser React Native CLI dans un environnement windows nous alons avoir besoin de Node JS, JDK, Android Studio :

- Node JS, pour verifier son installation :

```
node -v // affiche la version de node
node --version // affiche la version de node
```

- JDK (Java SE Development Kit (JDK)) version 11 recommandé, il est recommandé de l'installé avec chocolatey.

```
choco install -y nodejs-lts microsoft-openjdk11
```

Chocolatey

- Commande pour installer chocolatey (dans une invite de commande en mode administrateur)

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]
::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072;
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

- Verifier la précense de chocolatey (dans une nouvelle invite de commande)

```
choco
```

Environnement de développement React Native

Android Studio : https://developer.android.com/studio

Une fois Android Studio installé nous devons télecharger, installer le SDK Android, et configuré notre premier Android Device Manager.

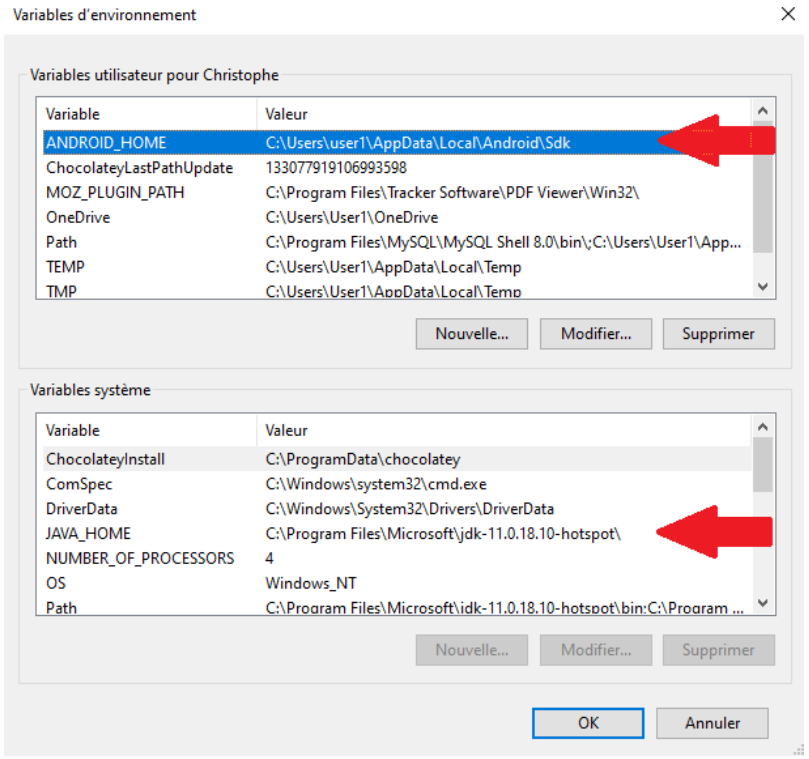
- SDK => File / Settings / Appearance & Behavior / System Settings / Android SDK
- Android Device Manager => Tools / Device Manager

Environnement de développement React Native

Variables d'environnement windows

Nous devons vérifier la présence de variables d'environnement :

- ANDROID_HOME
- JAVA_HOME



Notre premiére application React Native

Pour initialiser ce placer dans le répertoire de travail est à l'aide d'une invite de commande saisir :

```
npx react-native@latest init AwesomeProject
```

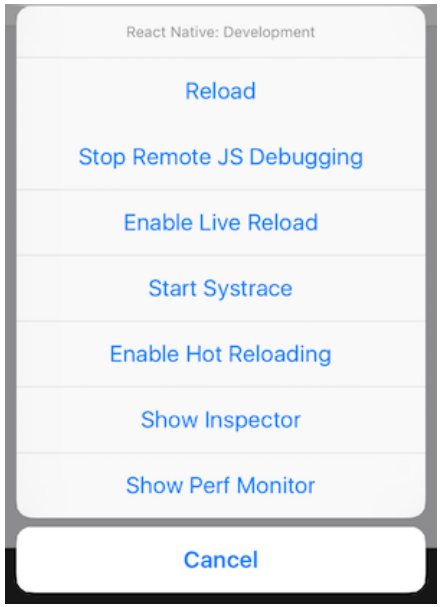
Ceci aura pour effet d'initialiser un projet React Native qui porte le nom AwesomeProjet (que vous pouvez personnaliser)
Et pour demarrer l'application

```
cd AwesomeProject // se placer dans le répertoire de notre application
npx react-native run-android
```

Le projet s'installera et ce lancera automatiquement sur le Android Device configuré ou sur votre telephone android connecté a l'ordinateur

Débogueur React Native

Si l'application est lancé sur un émulateur android vous avez la possibilité d'acceder à un débogueur en mode développement via le raccourci `CTRL + m`



Notre premier Composant React-Native

A la racine de notre projet ou dans un dossier nous pouvons à présent crée notre premier composant React-native, dans lesquelle nous alons utilisé des composant de base fourni par React-Native.

- View
- Text
- Image
- TextInput
- ScrollView

Chacun de ces composant peut prendre en compte des props qui pourront agir sur son affichage et/ou comportement.

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

const FirstComponent = () => {
  return (
    <View>
        <Text>FirstComponent</Text>
            <Image
                width={50}
                height={40}
                source={{
                    uri: 'https://reactnative.dev/img/tiny_logo.png',
                }}
            />
    </View>
  )
}

export default FirstComponent
```

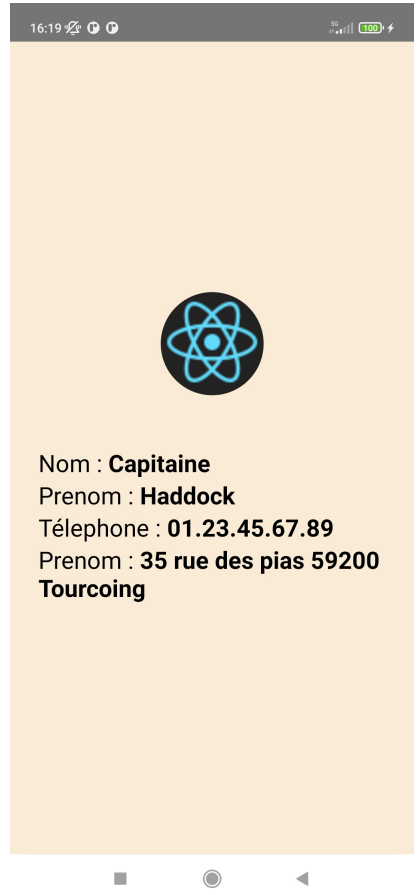
Ajouter du Style

Afin de styliser notre application nous pouvons utiliser StyleSheet qui est une abstraction similaire à CSS StyleSheets que nous passons dans le props style de nos composants React Native.

```
import { StyleSheet, View, Text, Image } from 'react-native'
import React from 'react'
const FirstComponent = () => {
    return (
        <View style={styles.container}>
            <Text style={styles.monTexte}>Mon premier Composant</Text>
            <Image
                style={styles.monImage}
                source={{
                    uri: 'https://reactnative.dev/img/tiny_logo.png',
                }}
            />
        </View>
    )
}
const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        backgroundColor: 'black',
    },
    monTexte: {
        fontSize: 30,
        color : 'white',
    },
    monImage: {
        width: 100,
        height: 100,
        margin: 30,
    }
})
export default FirstComponent
```

Exercice 1

Réalisé une application Android qui permet d'afficher les infos d'un contact.



Autre Composants

React Native nous fournis d'autre composants afin d'ameliorer notre interface en voici une liste non exhaustive :

Interface utilisateur :

- Button
- Switch

Liste :

- FlatList
- SectionList

Autres :

- ActivityIndicator
- Modal
- StatusBar
- Animated

Gestion des événements en React Native

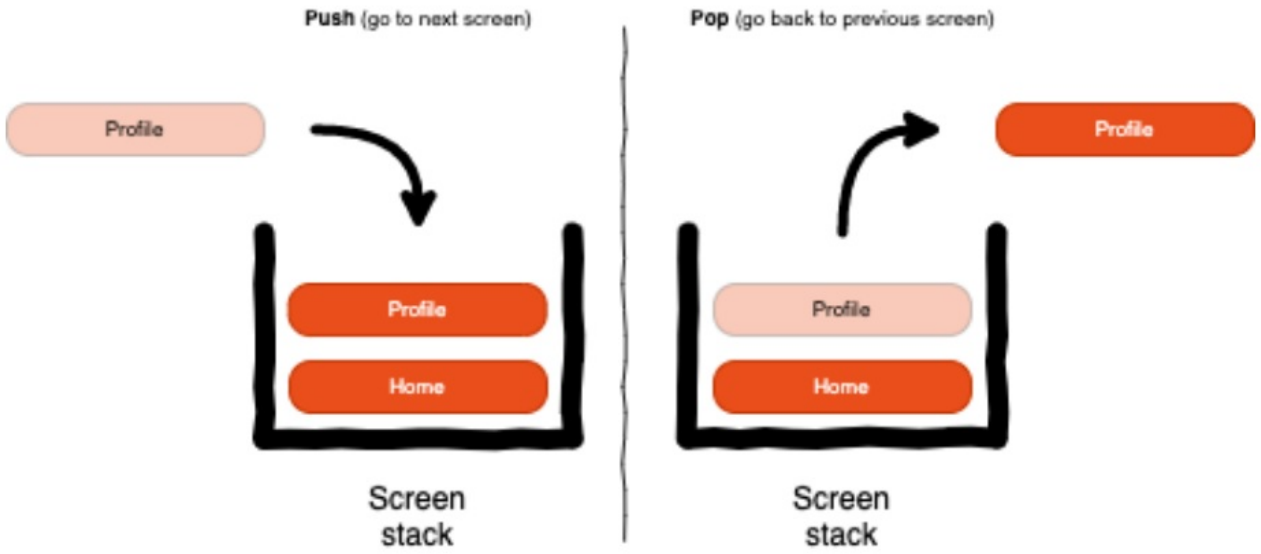
La gestion des événements avec react Native est similaire à celle de React.

Nous avons la possibilités d'ajouter des propriétes a certains de nos composants qui prenne en compte des evenements comme `onPress` `onTouchStart` `onTouchEnd` `onScroll` .

Et pour les composants qui ne prenne pas en compte ces types d'evenements dans leur props (exemple : <Views>) il est possible de les englobés dans un composant (wrapper) qui prendra en compte ces evenements. Wrapper disponible : `TouchableHighlight` `TouchableOpacity` `TouchableWithoutFeedback` .

La navigation avec React Native

En mobile, la navigation repose sur le modéle d'empilement d'écrans (Stack-Based-Navigation). Ces différents écrans (screens) sont empilés (stack)



La navigation avec React Native

Dans un premier temps nous devons ajouter des packages à notre projet :

```
npm install @react-navigation/native
npm install react-native-screens react-native-safe-area-context
```

Ensuite nous englobons (wrapping) nos différentes écrans (screens) dans un composant `NavigationContainer`

```
import { NavigationContainer } from '@react-navigation/native';
<NavigationContainer>{/* Votre navigateur et vos differents screens */}</NavigationContainer>
```

Pour fonctionner notre `NavigationContainer` va avoir besoin de la fonction `createNativeStackNavigator` qui va nous fournir 2 propriétés pour définir notre navigation qui sont : `Navigator` et `Screen`.
A partir de la nous pouvons définir autant de `Screen` qu nous le souhaitons.

```
    // In App.js in a new project

import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Pour naviguer entre les écrans, nous allons utiliser le props navigation transmis a chacun de nos écrans (screens) défini dans notre `Navigator`

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

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

function DetailsScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

La persistance avec React Native

Async Storage

AsyncStorage est une fonctionnalité de stockage de données dans React Native qui permet de stocker des données sous forme de clé-valeur dans le dispositif de stockage local de l'application.
Le stockage asynchrone signifie que le stockage et la récupération des données ne sont pas bloquants et ne ralentissent pas l'interface utilisateur de l'application. Les données stockées peuvent être récupérées ultérieurement à tout moment pendant l'exécution de l'application.
AsyncStorage est très pratique pour stocker des informations simples telles que des préférences utilisateur, des jetons d'authentification, des informations de connexion, des paramètres de l'application, etc.

On ajoute le Async Storage à notre application :

```
npm install @react-native-async-storage/async-storage
```

On ajoute l'import à notre composant :

```
npm install @react-native-async-storage/async-storage
```

Stockage de type String :

```
const storeData = async (value) => {
  try {
    await AsyncStorage.setItem('@storage_Key', value)
  } catch (e) {
    // saving error
  }
}
```

Pour le stockage d'un objet il suffit simplement de le "stringifiées" :

```
const storeData = async (value) => {
  try {
    const jsonValue = JSON.stringify(value)
    await AsyncStorage.setItem('@storage_Key', jsonValue)
  } catch (e) {
    // saving error
  }
}
```

Et pour la récuperation

Récupération de type String :

```
const getData = async () => {
  try {
    const value = await AsyncStorage.getItem('@storage_Key')
    if(value !== null) {
      // value previously stored
    }
  } catch(e) {
    // error reading value
  }
}
```

Récupération de type Objet :

```
const getData = async () => {
  try {
    const jsonValue = await AsyncStorage.getItem('@storage_Key')
    return jsonValue != null ? JSON.parse(jsonValue) : null;
  } catch(e) {
    // error reading value
  }
}
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

FIN