React Native

Version : 1.0.0

Introduction

Dans ce cours vous allez apprendre React Native, vous allez apprendre à construire des applications mobiles.

Objectifs pédagogiques

- Apprendre à connecter React native
- Construire une application complète à partir de zéro

Prérequis

ReactJS

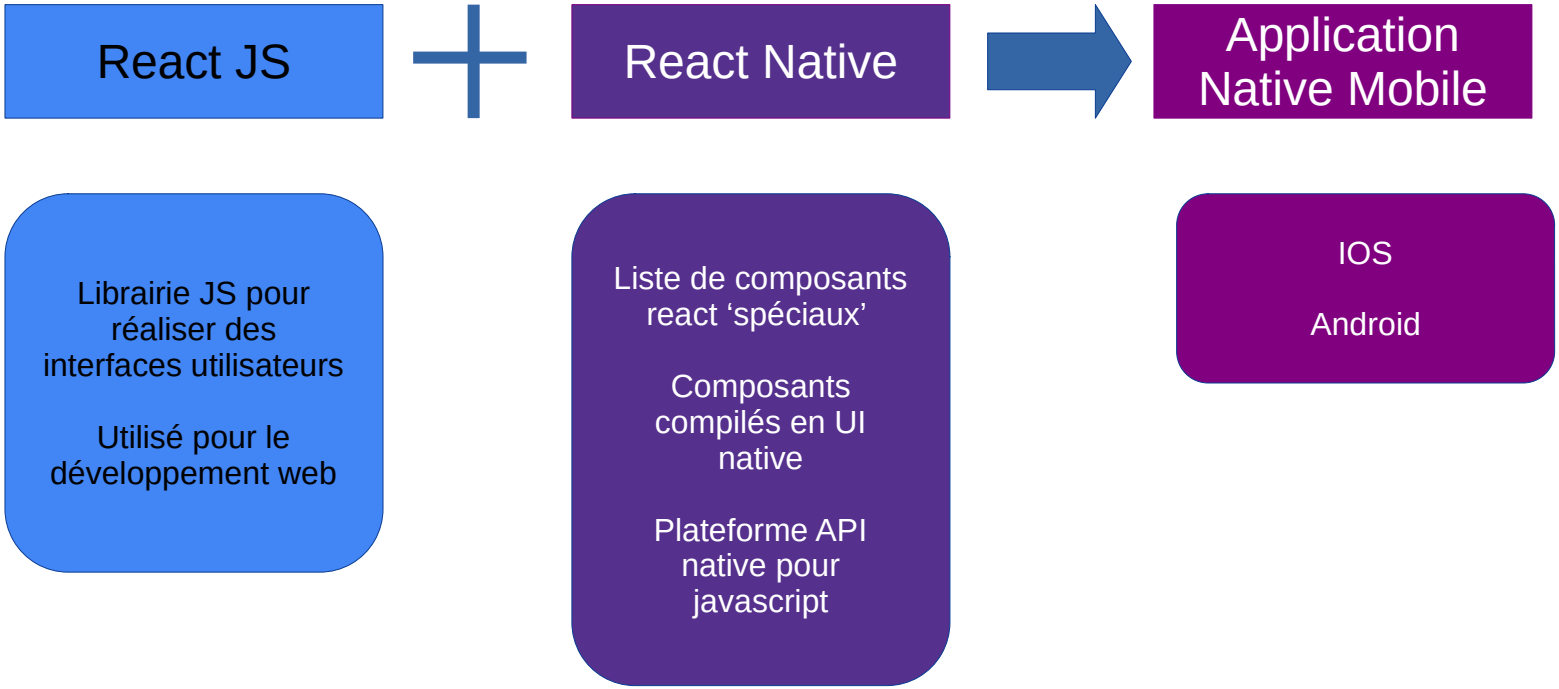
Outils nécessaires

Postman

Plan de la semaine

Jour 1 :
Intro React Native (installation, composants, styles, routing)
Jour 2 :
communications API
Jour 3 :
Les composants natifs
Jour 4 :
Projet + contrôle de connaissance
Jour 5 :
Projet + soutenance de projet

Qu’est-ce que React Native ?



Comment ça fonctionne ?

```
const App = props =>{
    return(
        <View>
            <Text> Bonjour -_- </Text>
        </View>
    )
}
```

Un code React Native.

Beaucoup de similitude avec React JS

Composant ‘view’ Compilé en application native (partie JSX)

Exemple de composants compilés

Navigateur Web (react)		<input>	Etc...
Composant Natif (android)	Android.View	EditText	Etc...
Composant Natif (IOS)	UIView	UITextField	Etc...
React Native	<View>	<TextInput>	Etc...

Et la logique ?

Les composants de React Native (UI Element) sont compilés en vues Natives.

La logique Javascript, n’est pas compilé. Elle est hébergé dans l’application par React Native.

Pour Résumé : Eléments User Interface = compilé
Logique JS = non compilé

Créer notre application React Native

Allez sur le site : https://reactnative.dev/ qui est la documentation officiel de React Native.

Dans la page Guide, vous allez avoir un onglet ‘Environnement Setup’

Dans cet onglet, vous avez le choix d’installez Expo CLI ou ou le React Native CLI

Pour le cours, nous utiliserons la solution Expo.

Expo ou React Native CLI ?

EXPO

Service d'une entreprise tier (et gratuit)

Outil données afin de facilité la création ou le développement. Utilisation des fonctionnalités natives + facile.

Vous pouvez quitter l'environnement n'importe quand.

React Native CLI

Développé par l'équipe de React Native et la communauté.

Départ de 0 (+ de mise en place)

Moins de fonctionnalités pratique (ex : l'appareil photo demande + de travail)

Intégration de code native + facile (Java, Swift, KOTLIN)

Créer notre application React Native

Utilisez la commande :

NPM :

```
npx create-expo-app NomProjet
```

YARN :

```
yarn create expo-app NomProjet
```

Puis entrez dans votre projet et lancé la commande `npx expo start`
Ou **npm start**

Créer notre application React Native

Une fois lancé, installez l’application mobile ‘Expo Go’

Créez vous un compte dessus, Puis Scannez le QR code
Que vous avez reçu sur votre terminal.

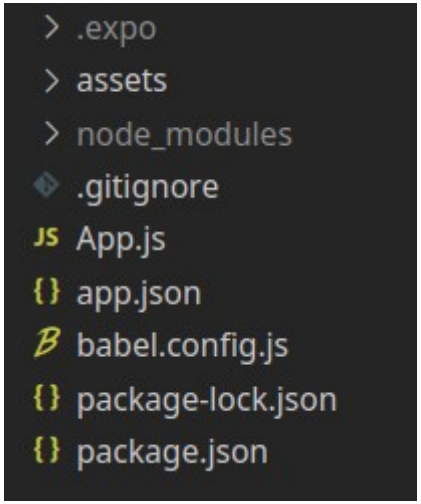
/!\ Faites bien attention a ce que votre téléphone et votre ordinateur soit sur le même réseau /!\

Analyse de l’application

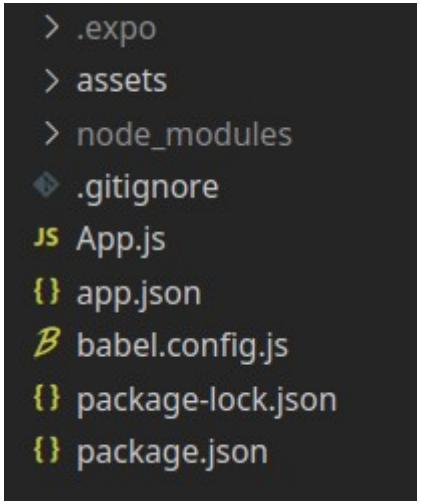
.expo => contiens les informations des appareils utilisés dans ce projet (propre à expo).

Assets => mettre nos images

Babel.config.js => converti la syntaxe React en code pouvant être exectué dans un environnement JS qui prend de base, pas en chagre ces fonctionnalités.



Analyse de l’application

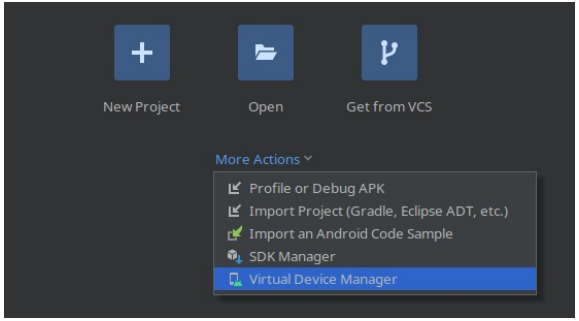


App.json => permet la configuration de paramètres de notre application. (ex : fichier utilisé par expo pour la prévisualisation).

App.js => notre code de départ.

Environnement de développement Local

Si vous souhaitez émulé un téléphone android sur votre ordinateur, vous pouvez utilisé Android Studio



Une fois installé, vous pouvez créer un nouvel appareil (attention a bien prendre un appareil avec google play store).

Environnement de développement Local

Une fois votre émulateur lancé, si votre application est encore lancé, cliquer sur A afin de lancer l'application sur votre émulateur android.

Pour utilisez un téléphone sous IOS, il faut utiliser Xcode disponible uniquement sur MacOS.

Stylisé le code React Native

Deux façons :

-dans le composant:

```
<View style={{borderColor: 'red', borderWidth: 3}}>
```

- En important StyleSheet de react native :

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

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Stylisé le code React Native

Attention ce n’est pas du CSS !

```
style={{ border: '1px solid red'}}
```

Certaines propriétés ont été reprises pour être le plus raccord avec CSS, mais pas toutes

```
<View style={{borderColor: 'red', borderWidth: 3}}>
```

Mises en pages et flexbox

Si nous retirons tous le css mis en place, vous remarquerez que le texte se situe tout en haut de votre téléphone.

```
import { Button, StyleSheet, Text, TextInput, View }

export default function App() {
  return (
    <View>
      <View>
        <TextInput placeholder='votre mission'/>
        <Button title='ajouter'/>
      </View>
      <Text>listes d'objectifs:</Text>
    </View>
  );
}

const styles = StyleSheet.create({

});
```

Mises en pages et flexbox

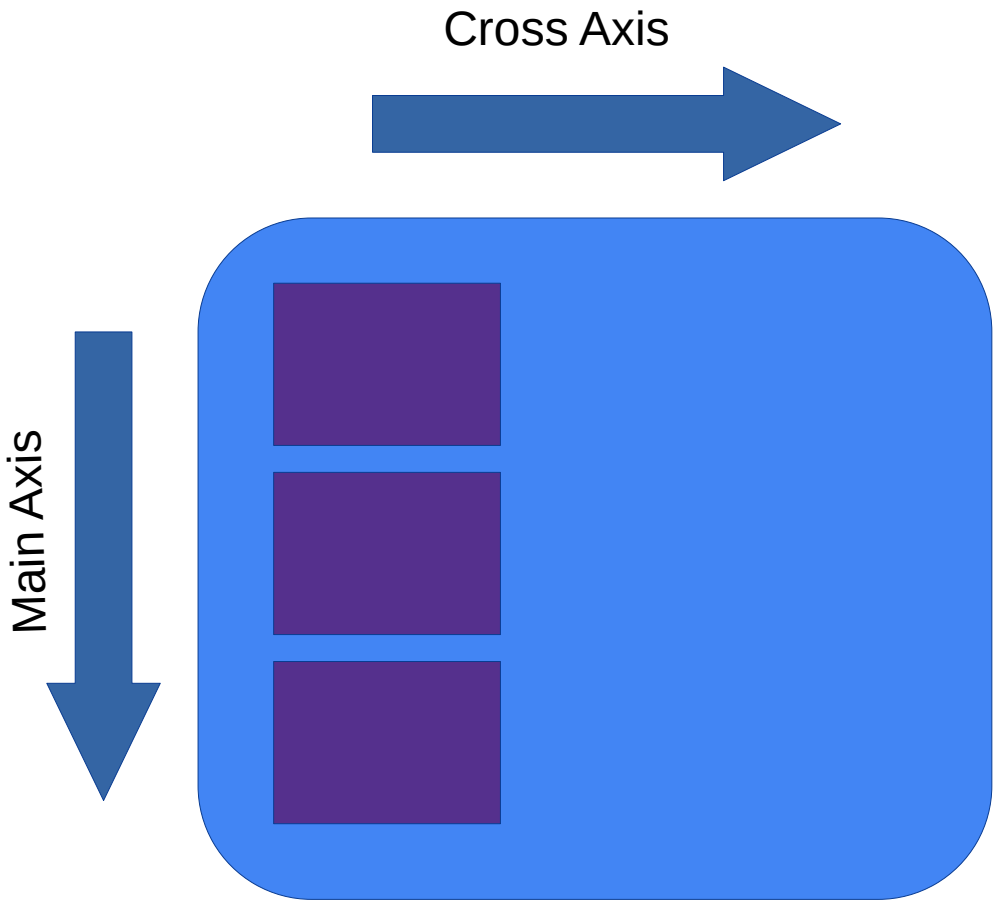
La mise en page de React Native est essentiellement basé sur les flexbox

Les flexboxs utilisés sont très similaire à celle de CSS

Pour + d’info sur les flexbox en react native :http://reactnative.dev/docs/flexbox

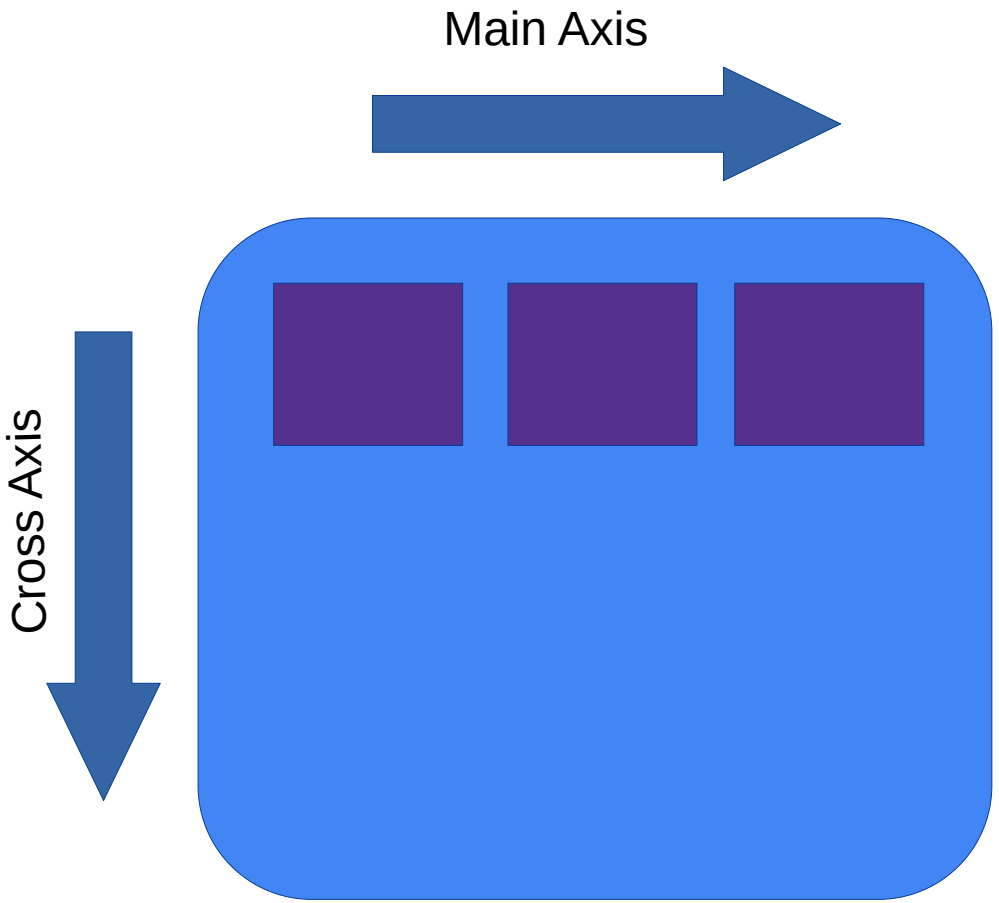
Mises en pages et flexbox

```
container:{
  flex: 1,
  flexDirection: 'column',
  justifyContent: 'flex-start',
  alignItems: 'flex-start'
}
```



Mises en pages et flexbox

```
container:{
  flex: 1,
  flexDirection: 'row',
  justifyContent: 'flex-start',
  alignItems: 'flex-start'
}
```



Mises en pages et flexbox

Testez avec un exemple ci-après :

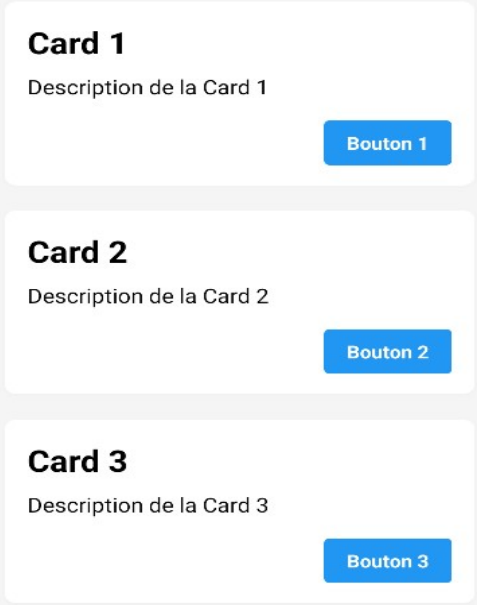
```
export default function App() {
  return (
    <View style={styles.appContainer}>
      <View style={styles.inputContainer}>
        <TextInput placeholder='votre mission'
        style={styles.textInput}/>
        <Button title='ajouter'/>
      </View>
      <Text>listes d'objectifs:</Text>
    </View>
  );
}
```

```
const styles = StyleSheet.create({
  appContainer:{
    padding: 50
  },
  inputContainer:{
    flexDirection: 'row',
    justifyContent: 'space-between'
  },
  textInput:{
    borderWidth:1,
    borderColor: '#cccccc',
    width: '70%'
  }
});
```

Exercice de style

Réaliser un composant React Native ressemblant à ceci :

Attention, ici on n’attend pas une repoduction à l’identique, uniquement un résultat ressemblant à ceux-ci.



Gestion des évènements :

La gestion d'évènement est la même qu'en JS classique. Cependant, certains évènement ont changé et d'autres apparaissent.

```
onChangeText={votreFonction}/>
```

Par exemple : la fonction **onClick** n'existe plus, elle a été remplacé par **onPress** car techniquement, on ne clique pas sur un bouton du téléphone, on appui dessus.

```
onPress={votreFonction}
```

UseState

React Native utilise également des useState

UseState erst un hook permettant d’ajouter le state local React à des fonctions composants

```
import { useState } from 'react';
```

```
const[etat, setEtat]=useState()
```

Affichage d’une liste

- Utilisez map() afin de parcourir une liste.

- Utilisez les accolades dans votre rendu.

- Utilisez la propriété key afin d’éviter les erreurs React : ‘each child in a list should have a unique key prop’

```
<Text>listes d'objectifs:</Text>
{objectifs.map((objectif)=>
<Text key={objectif}>{objectif}
</Text>)}
```

Tricks feuille de style

Certaines propriétés de styles ne sont utilisables uniquement sur IOS ou Android

```
<Text style={styles.objectif} key={objectif}>{objectif}
</Text>
```

```
objectif:{
  padding:8,
  margin: 8,
  backgroundColor: 'orange',
  color: 'white',
  borderRadius: 6
}
```

IOS :

Android :

Tricks feuille de style

La propriété borderRadius n’est pas pris en compte sur IOS.
Deux Choix

On règle pas le problème : IOS c’est le mal 

On règle le problème : Créer une View qui va pouvoir contourner le problème. Car l’élément sous-jacent sur lequel View est compilé est un élément qui prend en compte les bordures.

```
<View style={styles.objectif}
 key={objectif}>
    <Text >
      {objectif}
    </Text>
</View>
```

Comment le savoir ? Documentations React Native :-)

Tricks feuille de style

Si vous faites cela, une autre erreur apparaît

Le texte n’est plus blanc sur aucun des deux. Pour rappel, ce que nous faisons n’est pas du CSS mais s’y rapproche.

Ici le style n’est pas en cascade. Donc mettre le style dans un composant parent, se sera pas hérité par ses enfants.

Composant ScrollView

ScrollView permet d’effectuer un défilement continue (exemple avec un map())

Il est utilisé après un View car la View permet de définir l’espace dans lequel le défilement va s’effectuer

```
<View>
  <ScrollView>
  {objectifs.map((objectif)=>
  vos_Infos
  )}
</ScrollView>
</View>
```

FlatList

Problème ScrollView : Même si les éléments ne sont pas à l'écran, l'élément est quand même rendu (dans la face non visible). Si vous avez 10000 articles c'est compliqué.

```
<ScrollView>
```

FlatList est un Scroll qui n'affichera que les éléments réellement visibles et les éléments hors écran seront chargés que lors du défilement

```
<FlatList>
```

FlatList

Cependant FlatList ne prend pas en compte map() car nous n'afficherons plus nos données manuellement.

Cette tâche sera effectué par FlatList grâce à deux propriétés :
- **data** : qui stockera les données à listé
- **renderItem**: fonction qui explique comment les données seront rendu

RenderItem a un paramètre qui possèdent des métadonnées comme
Index : connaître l'id des différents objets que l'on rend
Item : Récupère les données contenu dans '**data**'

FlatList

Exemple :

```
<FlatList data={objectifs} renderItem={(objectif) =>{
  objectif.index
  return(
    <View style={styles.objectif}>
      <Text >
        {objectif.item}
      </Text>
    </View>
)}}/>
```

Séparation de composants et props

La séparation des composants se fait comme en ReactJS à savoir :
- création du composant
- exporter le composant
- importer le composant dans un composant parent

Il en va de même pour les props

Dans App.js :

```
import Objectif from './components/objectif'
```

```
<Objectif objectif={objectif.item}/>
```

Dans Objectif.js :

```
export default function Objectif(props){
    return(
    <View style={styles.objectif}>
        <Text >
            {props.objectif}
        </Text>
    </View>)
}
```

Pressable et Modal

Deux composant JSX disponible en react Native :
- Pressable : A associer avec l'event onPress
- Modal : Deux propriétés : visible et animationType

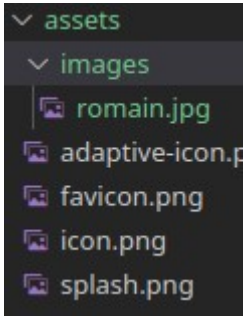
```
<Pressable onPress={                                    }>
    <View style={styles.objectif}>
        <Text >
            {props.objectif}
        </Text>
    </View>
</Pressable>
```

```
<Modal visible={props.visibilite} animationType='fade'>
    <View style={styles.inputContainer}>
    ...
    </View>
</Modal>
```

Image

- Ajouter une image dans le dossier ‘assets’

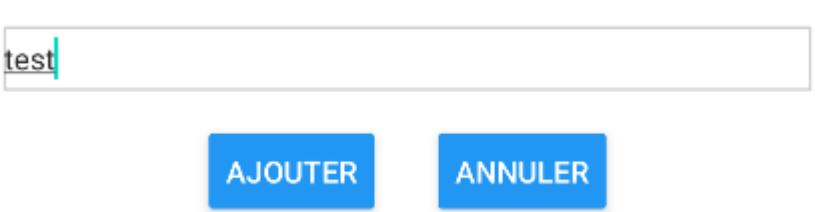
- utiliser la balise Image et un require afin d’afficher l’image

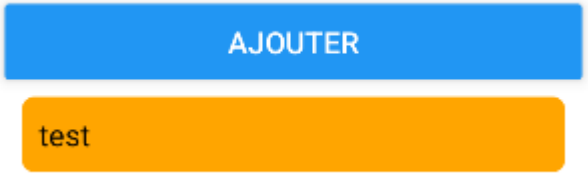



```
<Image style={styles.image} source={require('../assets/images/romain.jpg')}/>
```

Exercice modal et liste

- Créer un composant qui listera des tâches à effectuer, avec un bouton qui enverra sur un modal qui demandera la dans un **TextInput** la tâche à effectuer, et qui lorsque l'on cliquera sur un bouton, renverra vers la liste des tâches. Si vous cliquer sur la tâche, elle se supprime.





Dimension entre les différents téléphones et rotation

Pour connaître les dimensions du téléphones :
- Import ‘Dimensions’ from ‘react native’

- Utilisez Dimensions.get(‘’) puis selectionner si vous voulez ‘screen’ ou ‘window’ (screen c’est tout l’écran, windows c’est sans les bordures).

- Récupérer les dimensions avec Dimensions.get(‘window’).width (ou autres)

Tips : Pour ajuster l’orientation, allez dans app.json et changer la valeur de ‘orientation’ par default.

La navigation en RN

Pour mettre ne place la navigation, nous allons installer le package react navigation :

https://reactnavigation.org/

Pour installer le package :

```
npm install @react-navigation/native
```

La navigation en RN

Si nous utilisons expo, il faut installer d’autres dépendances :

```
npx expo install react-native-screens react-native-safe-area-context
```

Si vous n’utilisez pas expo, il y a + de dépendances à installer, lisez la documentation officiel.

La navigation en RN

Le composant **<NavigationContainer>** entoure tous les composants de votre app qui doivent bénéficier de la navigation :

```
<NavigationContainer>
  <Text>...</Text>
</NavigationContainer>
```

La navigation en RN

Après cela, il va falloir mettre en place un ‘navigateur’, il en existe plusieurs comme :
- Stack
- Native Stack
- Drawer
- Material Top Tabs

La différence entre ces différents navigateurs sera dans leur comportement (ex : Draw fera un tirage latéral).
Nous utiliserons pour l’exemple native stack.

```
npm install @react-navigation/native-stack
```

La navigation en RN

Après avoir installer la dépendance, nous allons importer ‘createNativeStackNavigator’, puis créer une variable (en dehors de notre fonction) :

```
import { createNativeStackNavigator } from '@react-navigation/native-stack'
```

```
const Stack = createNativeStackNavigator();
// créer un objet avec deux propriétés (Navigator et Screen)
```

La navigation en RN

A quoi servent ces propriétés :

Navigator va vous permettre de naviger entre les différents Screen disponibles :

Tips : la premiere page sera le Screen le plus haut placer dans votre Navigator

```
<NavigationContainer>
  <Stack.Navigator>
    <Stack.Screen name="page1" component={Page1}/>
    <Stack.Screen name="page2" component={Page2}/>
  </Stack.Navigator>
</NavigationContainer>
```

La navigation en RN

C’est bien beau mais comment naviguer entre nos deux composants du coups ? Le composant que vous mettez dans screen hérite d’un props appelé **navigation** que vous pouvez récupérer qui vous permet de vous diriger vers d’autres pages :

```
export default function Page1({ navigation }) {
```

```
const navigationPage2 = () =>{
    navigation.navigate('page2')
}
```

mettre nom de votre screen

La navigation en RN

Tips : si vous voulez utiliser la navigation dans un composant d'un composant d'un composant… d'un composant Screen, au lieu de passer le props, vous pouvez utilisez le **useNavigation**

```
import { useNavigation } from '@react-navigation/native';
```

La navigation en RN

Comment envoyer des informations entre composants ?

Quand on utilise le props **navigation**, on peut envoyer un second paramètre qui sera un objet

```
const navigationPage2 = () =>{
    navigation.navigate('page2', {
      info: "information a envoyer"
    })
}
```

La navigation en RN

Ensuite nous utilisons le props route qui est également envoyer en props pour les composants de Screen. Route est un objets et peux ressortir pas mal d'information. Notamment params :

```
export default function Page2({ route }) {
```

```
<Text>
  Bienvenue sur la page2,
  {route.params.info}
</Text>
```

Ici avec params, on peut ensuite selectionner l'information que l'on a envoyer en objet via le navigation de la page précédente

TP

Gestion des livres