© Donkey Geek

L’Ultime Formation

React Native

© Donkey Geek

Introduction

© Donkey Geek

Pourquoi apprendre React Native?

© Donkey Geek

Ce que vous allez apprendre

© Donkey Geek

Prérequis

© Donkey Geek

À qui s'adresse la formation ?

© Donkey Geek

C’est quoi React Native ?

React JS

Une bibliothèque JavaScript pour créer des interfaces utilisateurs (UI) à base de composants autonomes qui maintiennent leur propre état et qui seront affichés sur le **web** via le **ReactDOM.render()**

Une autre bibliothèque JavaScript qui se base sur la logique de **React** (Gestion de la mise à jour de l’interface utilisateur, la gestion des états, les composants - particuliers comme View, Text, Image, entre autres) pour créer des **applications natives Android et iOS**. Donc, on code une seule fois en **JavaScript** et on génère du **code natif** Android et iOS.

React Native sait compiler ses composants particuliers pour en faire des widgets en code natif qui fonctionnent sur ces deux plateformes.

Il permet également l’accès à certaines API natives qui nous permettent de faire plus de choses comme par exemple l’accès à la caméra .

© Donkey Geek

En résumé :

React Native nous permet de coder en JavaScript, avec la logique de React JS, et de compiler notre code en véritable code natif Android et iOS.

Les applications ainsi codées sous React Native sont de vraies applications Android et iOS et peuvent donc êtres chargées sur Apple Store pour les utilisateurs iOS et sur Google Play Store pour les utilisateurs Android

© Donkey Geek

Prérequis

- **Les bases de JavaScript**

- **Les bases de la librairie React JS**
  - Les bases de React JS *
  - Les HOOKS de React JS *

- **CSS FlexBox**
  - Maîtriser le CSS Flexbox *

* Vidéos disponibles dans les derniers chapitres de cette formation

© Donkey Geek

Différences entres les plateformes

Composants React Vs React-Native

```
import React, { useState } from "react";

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count +1)}>+ 1<button/>
    </div>
  );
}
```

```
import React, { useState } from "react";

function App() {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>{count}</Text>
      <Button onPress={() => setCount(count +1)} title="+1" />
    </View>
  );
}
```

Contrairement au Web, le langage HTML n’est pas accepté dans les applications Android et iOS !

Pour cela, React Native utilise des composants Spécifiques qui, une fois compilés en code natif, seront interprétés sur Android et iOS

Le code JavaScript qui contient toute votre logique n’est pas compilé, ce qui est une bonne chose en terme d’optimisation de votre application.

© Donkey Geek

Les Views seront compilées en code natif mais pas le JavaScript

```
import React, { useState } from "react";

function App() {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>{count}</Text>
     <Button onPress={() => setCount(count +1)} title="+1" />
    </View>
  );
}
```

© Donkey Geek

Comment va fonctionner mon code JavaScript?

Comment coder notre application ?

Visual Studio Code

Gratuit: https://code.visualstudio.com/

Quelques extensions facultatives

- Extension: ES7 React/Redux/GraphQL/React-Native snippets
- J’utilise le thème Primal avec la font Operator Mono

Comment tester & visualiser notre application ?

L’outil - EXPO CLI

Méthode de travail très simple, gratuite et permet de coder facilement de vraies applications React Native car elle nous offre tout ce dont on aura besoin pour mener à bien nos projets React Native.

Inconvénient: on est limités à l'écosystème Expo.

Cependant, si nécessaire, on peut toujours passer à la deuxième option React Native CLI via un "eject".

REACT NATIVE CLI

Méthode de travail un peu plus complexe, mais qui offre plus de contrôle et une plus grande marge de manœuvre..

Cette méthode nécessite également une configuration relativement complexe et l’utilisation d’un certain nombre d’outils comme Android Studio, Xcode et même l’utilisation de packages tiers notamment pour accéder et travailler sur les modules natifs comme la caméra, les maps etc.

© Donkey Geek

Comment fonctionne Expo?

```
import React, { useState } from "react";

function App() {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>{count}</Text>
     <Button onPress={() => setCount(count +1)} title="+1" />
    </View>
  );
}
```

Pendant le développement, on peut voir le rendu de notre application directement sur notre téléphone ou dans un simulateur / émulateur (version virtuelle)

© Donkey Geek

Environnement de développement

Documentations:

React Native: reactnative.dev

Expo: docs.expo.io/

Utilisateur macOS

Watchman via Homebrew

Expo CLI

(>=12 LTS)

Téléchargement: nodejs.org

Ne vous inquiétez pas, vous n’êtes pas sensés connaître Node Js!

Installation Homebrew (Après le lancement de l'installation, on vous demandera de cliquer sur RETURN)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

```
➜  ~ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
Password:
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew
==> The following existing directories will be made group writable:
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/share
/usr/local/share/doc
/usr/local/share/man
/usr/local/share/man/man1
==> The following existing directories will have their owner set to axxam:
/usr/local/bin
/usr/local/include
```

Mot de passe de votre macOS

mac OS

Installation de Watchman

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: zsh

  brew create [URL [--no-fetch]]
  brew edit [FORMULA...]

Further help:
  brew commands
  brew help [COMMAND]
  man brew
  https://docs.brew.sh
➜  ~ brew update
Already up-to-date.
➜  ~ brew install watchman
Updating Homebrew...
==> Downloading https://homebrew.bintray.com/bottles/openssl%401.1-1.1.1i.catalina.bottle.tar.gz
==> Downloading from https://d29vzk4ow07wi7.cloudfront.net/066b9f114617872e77fa3d4afee2337daabc2c181d75
################################################################## 100.0%
==> Downloading https://homebrew.bintray.com/bottles/pcre-8.44.catalina.bottle.tar.gz
==> Downloading from https://d29vzk4ow07wi7.cloudfront.net/f8ac266e04f984fa55091a43f0fdc39a40d57c2489d2
################################################################## 100.0%
==> Downloading https://homebrew.bintray.com/bottles/gdbm-1.18.1_1.catalina.bottle.tar.gz
==> Downloading from https://d29vzk4ow07wi7.cloudfront.net/f7b5ab7363961fa6defcb66b4ffdf5365264fcb97d35
```

Vérification: **watchman --version**

Installer Expo-CLI en global: sudo **npm install --global expo-cli**

Ignorez les erreurs qui s'affichent dès lors que l'installation s'effectue à la fin **expo-cli@XXXX**

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: zsh

npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for @expo/ngrok-bin-sunos-x64@2.2.8:
 wanted {"os":"sunos","arch":"x64"} (current: {"os":"darwin","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: @expo/ngrok-bin-freebsd-ia32@2.2.8 (node_modules/expo-c
li/node_modules/@expo/ngrok-bin/node_modules/@expo/ngrok-bin-freebsd-ia32):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for @expo/ngrok-bin-freebsd-ia32@2.2
.8: wanted {"os":"freebsd","arch":"ia32"} (current: {"os":"darwin","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: @expo/ngrok-bin-win32-ia32@2.2.8-beta.1 (node_modules/e
xpo-cli/node_modules/@expo/ngrok-bin/node_modules/@expo/ngrok-bin-win32-ia32):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for @expo/ngrok-bin-win32-ia32@2.2.8
-beta.1: wanted {"os":"win32","arch":"ia32"} (current: {"os":"darwin","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: @expo/ngrok-bin-freebsd-x64@2.2.8 (node_modules/expo-cl
i/node_modules/@expo/ngrok-bin/node_modules/@expo/ngrok-bin-freebsd-x64):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for @expo/ngrok-bin-freebsd-x64@2.2.
8: wanted {"os":"freebsd","arch":"x64"} (current: {"os":"darwin","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: @expo/ngrok-bin-win32-x64@2.2.8-beta.1 (node_modules/ex
po-cli/node_modules/@expo/ngrok-bin/node_modules/@expo/ngrok-bin-win32-x64):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for @expo/ngrok-bin-win32-x64@2.2.8-
beta.1: wanted {"os":"win32","arch":"x64"} (current: {"os":"darwin","arch":"x64"})
npm WARN @pmmmwh/react-refresh-webpack-plugin@0.3.3 requires a peer of react-refresh@^0.8.2 but none is
 installed. You must install peer dependencies yourself.

+ expo-cli@4.0.17
added 1898 packages from 825 contributors in 102.907s
→  ~
```

Ne vous inquiétez pas pour les erreurs et les warnings

Vérification: **expo-cli --version**

Initialiser notre premier projet React Native

> cd Desktop

> **expo init mon-projet**

- Choisir le template « blank » pour démarrer avec un minimum de dépendances
+ un composant racine vide

- Une fois le projet installé, rendez-vous sur ce dernier et lancer le script **npm start**

Attention: utilisateurs macOS: vous devez donner l'autorisation d'accès à Watchmen

- Une fois **npm start** lancé, on aura accès à notre console en local dans un nouvel onglet

© Donkey Geek

La Console - ExpoDevTool

© Donkey Geek

Visualiser notre application

Pour accéder et travailler sur votre projet, n’arrêtez pas Expo que nous avons lancé via « npm start »

Expo Client

Google Play

iOS Apple Store

© Donkey Geek

Client Expo (L’application Android Play Store)

L’application va se charger sur votre smartphone

© Donkey Geek

Bouygues Telecom
46 %
12:43
New update available, downloading...

Bouygues Telecom
46 %
12:44
Open up App.js to start working on your app!

Utiliser un simulateur (Émulateur)

On peut utiliser notre smartphone pour tester notre application

Inconvénient: On peut tester que la version de notre téléphone

On peut utiliser un simulateur qui permet de tester notre application sur:

Plusieurs plateformes (iOS - Android)

Plusieurs versions

Plusieurs dimensions d’écran ..etc

© Donkey Geek

Quel simulateur (Émulateur) ?

iOS Simulator (Xcode)

Android Studio Emulator

© Donkey Geek

Installation de Android Studio

Installation possible sur macOS - Windows - Linux

© Donkey Geek

Installation de iOS Simulator

Installation possible uniquement sur macOS

© Donkey Geek

Exercice

+ 1

© Donkey Geek

Composants View & Text

View est le composant le plus fondamental pour créer une interface utilisateur. C’est un conteneur qui prend en charge la mise en page avec flexbox, le style via JavaScript, certaines manipulations tactiles et les contrôles d’accessibilité.

C’est aussi un élément parent qui peut envelopper d’autres View. Il peut donc contenir d’autres composants de tous types.

Note: Pas de texte direct dans un View !

Style

Pour styliser notre View et lui donner l'apparence qu'on souhaite, on dispose d'un certain nombre de styles via les objets « JavaScript » dont on utilisera les propriété pour jouer le rôle de propriétés CSS

La liste de ces propriétés est évidement accessible au niveau de la documentation React Native

https://reactnative.dev/docs/components-and-apis#basic-components

https://reactnative.dev/docs/view-style-props

JSX

Comme dans la librairie React.JS, React-Native utilise également le JSX. C’est une syntaxe qui nous permet de coder les éléments et des composants directement dans le JavaScript.

Puisque le JSX c’est une forme de JavaScript, on peut donc y utiliser des datas dynamiques comme les variables, invoquer des fonctions, effectuer des conditions, des boucles, des map etc.

Le JSX fonctionne dans React-Native car nous avons importé la librairie « React »

Props (Propriétés)

Le mot « props » est un abrégé du mot « propriétés »

Il nous permet de customiser nos composants React et ReactNative en passant des datas entres les composants.

Exercice:

Nous allons passer des noms via des props pour les afficher dans un autre composant.

© Donkey Geek

Props children

Les props children sont des props définis entre les balises des composants

<Person> Data **</Person>**

Exercice:

Nous allons passer l’espèce du personnage comme props children pour l’afficher (Human Vs Alien).

Note

Les props permettent de customiser nos composants et de ce fait, certains sont obligatoires comme dans le cas des images qu'on verra un peu plus tard. D'autres, sont des informations supplémentaires qu'on applique en fonctions de nos besoins

Exemples:

Pour afficher une image, on aura besoin d'un props « source »

Pour appliquer un style, on aura besoin d'un props « style »

Note 2

On peut coder en JavaScript directement dans le JSX. Pour cela, on ouvre **{ }**

Cela nous permet d'afficher la valeur d'une variable, invoquer des fonctions, effectuer des boucles, mettre en place des conditions, ou simplement passer un array javaScript à un props.

```
["pizza", "fromage"]

<Person age={20}  food={ ["pizza", "fromage"]} />

{ "pizza", "fromage" }

<Person age={20}  food={ { "pizza", "fromage" } } />
```

© Donkey Geek

State - Etat d’un composant

Avec seulement les props et les composants de React Native « View, Text, Images, Buttons ..etc », on peut coder déjà énormément de choses.

Mais pour créer quelque chose d’interactif ou de dynamique on aura besoin d’un ou plusieurs états dans notre composant »

A qui sert le « State » d’un composant React Native?

© Donkey Geek

Les States & les événements

L'état donne de la mémoire à vos composants!

Nous allons pouvoir créer ces états grâce à la librairie React qui nous offre un moyen très simple pour générer autant d'états qu'on veut dans un composant. Pour cela, nous avons le hook « useState ».

Un State (Un état local) est donc le moyen qui permet le stockage des données personnelles d'un composant. Il est utile pour gérer les données qui changent au fil du temps ou qui proviennent de l'interaction de l'utilisateur via des événements.

Exemple: « onPress » sur un composant <Button />.

Composant Button

Le composant <Button /> est un des composants de base offert par la librairie React Native. On peut lui appliquer un certain nombre de props:

« title » pour afficher le texte du bouton.

« color » pour lui appliquer une couleur de text sur iOS et pour le background color dans le cas d'Android.

« accessibilityLabel » pour lui attribuer un texte pour les fonctionnalités d'accessibilité sur les supports destinés aux personnes non-voyantes

ou simplement pour accéder au JavaScript pour activer un événement comme « onPress ».

© Donkey Geek

Hook useState

Le Hook **useState** est une fonction offerte par la librairie React pour nous permet de générer une variable d'état locale à notre composant

Grace au hook useState, on peut créer plusieurs états dans un composant

Exercice 1:

Créer un état via le hook useState pour contenir une chaîne de caractères et l'afficher dans un <Text>. Via un props « onPress », nous allons modifier notre State et afficher la nouvelle valeur

Exercice 2:

On passe un objet dans useState( )

Composant TextInput

Le composant TextInput est un autre composants de base de la librairie React Native.

Il est également dotés d’un certains nombre de props comme « auto-correction, multiline, maxlength, auto-capitalization, placeholder .. »

On peut lui appliquer des événements bien spécifiques comme « onChangeText, onSubmitEditing, onFocus, onBlur, onScroll, onEndEditing .. »

© Donkey Geek

Afficher le contenu d’un objet

Passer un objet comme valeur initiale au Hook useState et l’afficher dans un <View>

Pour cela, on aura besoin de la méthode MAP de JavaScript qui permet de créer un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément de notre objet state

© Donkey Geek

Composant ScrollView

ScrollView est un composant React-Native qui nous permet d'effectuer un scroll afin d'accéder à la totalité du composant verticalement.

React-Native nous offre également une autre alternative via le composant **<FlatList>** Qu'on verra dans une autre vidéo

Moins de code et une meilleure optimisation car on charge seulement les éléments visibles …

© Donkey Geek

Exercice Pratique

Nous avons vu

Quelques composants React-Native

Comment appliquer du styles

Le state et sa gestion et l'affichage des données

Les événements

Nous allons utiliser ces connaissances pour réaliser un petit projet ensemble

© Donkey Geek

Shopping List

Indiquer l’article à acheter au niveau d’un TextInput

Valider le nom de l’article et l’enregistrer dans un Array

Afficher l’ensemble des éléments enregistrés

© Donkey Geek

Modules & Props

Les props nous permettent de customiser nos composants React et ReactNative en passant des datas entres les composants.

Exercice:

Nous allons utiliser les props dans notre application Shopping-List pour communiquer et passer les informations entre différents composants

Créer un nouveau composant <Products> pour y afficher l’ensemble des produits via un props

© Donkey Geek

Fonction dans Props

Exercice:

Créer un nouveau composant <AddProduct> pour y afficher le <TextInput> et <Button>

Gérer les deux événements « onChangeText » et « onPress » pour invoquer leurs fonctions respectives et mettre à jour les deux states « product » et « myProducts »

Touchable… & Pressable

Les Touchable et les Pressable sont des composant React-Native qu’on peut appliquer sur d’autres composants en l’enveloppant.

Ils permettent, entre autres, de détecter les différentes étapes d’événement « onPress » effectuées sur ses composants enfants

Grace à ces composants, on peut transformer d’autres composants comme <View> ou <Text> en composants « cliquables » comme les boutons.

Alert

Le Alert en React-Native nous permet de générer une boîte de dialogue d’alerte avec, entre autre, un titre et un message qu’on peut customiser.

Il nous permet également l’affichage d’une liste de boutons sur lesquels on peut appliquer des événements « onPress » ..

© Donkey Geek

Le hook useEffect

useEffect est un hook React. C’est une fonction qui nous permet de gérer les différentes phases « cycles de vie » d’un composant React.

Montage du composant

Mise à jour du composant (Modification)

Démontage du composant

© Donkey Geek

Composant Modal

Modal est un autre composant React-Native. Il nous permet de créer un pop-up pour afficher du contenu superposé.

Exercice

Créer un modal pour informer l'utilisateur qu'il ne peut pas enregistrer un produit qui contient moins de deux caractères

Gérer le state du Modal via useState

Le bouton permettant la fermeture du Modal

Utiliser quelques props du composant Modal

© Donkey Geek

Exercice 1

Afficher les personnages indiqués dans l’objet « obj » comme indiqué dans le visuel ci-contre. Veuillez consulter le fichier App.js se trouvant dans le dossier « Ressources », vous y trouverez le code pour démarrer cet exercice.

Tous les personnages doivent se trouver dans une liste centrée verticalement.

© Donkey Geek

Exercice 2

Afficher la même liste en la collant vers le bas de l’écran

© Donkey Geek

Exercice 3

Afficher la même liste en la collant vers le haut de l’écran

© Donkey Geek

Exercice 4

Etirer la liste verticalement pour occuper tout l’écran

Exercice 5

Je reviens à la configuration précédente (Liste collée vers le haut) et j'agrandie la police du Text à 40px et je corrige le problème d'affichage avec un ScrollView afin d'accéder à tous les personnages

© Donkey Geek

Exercice 6

Afficher un « Spinner » quand on rafraîchit la liste

Indice: La doc est ton amie ;-)

© Donkey Geek

Exercice 7

Gérer le RefreshControl via un state

Gérer l'événement « onRefresh » pour afficher une Alerte

Afficher un « console.warn( ) » pour confirmer le rafraîchissement

Indice: La doc est ton amie ;-)

© Donkey Geek

Exercice 8

Changer la couleur du Spinner

Indice: La doc est ton amie ;-)

Exercice 9

Retirer le composant <ScrollView> et le remplacer par le composant <FlatList />

Exercice 10

Retirer la propriété « id » du array « obj » et maintenir l’affichage dans le composant <FlatList />

© Donkey Geek

Exercice 11

Afficher les éléments de la liste horizontalement sans toucher au CSS FlexBox

Utiliser le props « refreshControl » pour afficher le spinner et à chaque rechargement inverser le sens de la liste horizontale

Stan - Francine - Heyley - Steve - Roger - Klaus

Klaus - Roger - Steve - Heyley - Francine - Stan

© Donkey Geek

Exercice 12 Composant <SectionList />

SectionList est un autre composant React-Native qui nous permet d'effectuer le rendu de listes sectionnées

Exercice:

Afficher les personnages du Array « obj »

```
[
  { role: "Père", data: [ "Stan", 45 ] },
  { role: "Mère", data: [ "Francine", 45 ] }
]
```

© Donkey Geek

Exercice 13

Permettre l’affichage de texte sur plusieurs lignes dans un composant <TextInput>

Autoriser 9 caractères maximum

Cacher le mot de passer

Désactiver le <TextInput>

© Donkey Geek

Module Keyboard

Le module Keyboard (Clavier) nous permet d'écouter les événements natifs et d'y réagir, ainsi que d'apporter des modifications au clavier: Exemple: Retirer le clavier.

Nous allons appliquer ce module « Keyboad » à notre application en association avec « TouchableWithoutFeedback » afin de retirer le clavier en cliquant à l'extérieur du TextInput

© Donkey Geek

Module Keyboard

Méthodes

- addListerner( )
- removeListerner( )
- dismiss( )

Hook useEffect( )

- Phase de montage
- Phase de démontage (Cleanup - Nettoyage)

Exercice 14

Afficher le <TextInput> dans un <Modal> en cliquant sur un <Button>

Le composant <Button> doit s'afficher dans le composant App.js et permettra de gérer l'affichage du <Modal> qui se trouve dans le composant <AddProduct>

Exercice 15

Retirer le <Modal> après validation du <TextInput>

Exercice 16

Retirer le <Modal> sans valider le <TextInput>

© Donkey Geek

Exercice 17

Aligner les deux boutons horizontalement (Flexbox)

© Donkey Geek

Composant Image

Image est un autre composant React-Native qui, comme son nom l’indique, nous permet d’afficher des images dans nos applications.

Ce composant nous permet d’afficher différents types d'images, y compris les images réseau, les ressources statiques, les images locales temporaires et les images du disque local, les images via une adresse URL en remote, etc.

Les formats actuellement pris en charge sont: png, jpg, jpeg, psd, gif, bmp, webp (iOS encore plus)

© Donkey Geek

Composant ImageBackground

ImageBackground est un autre composant React-Native qui nous permet d'afficher une image d'arrière-plan

Pour cela, nous devons envelopper les éléments enfants d'un composant <ImageBackground> afin de leur appliquer une image d'arrière-plan

© Donkey Geek

Custom Components

Créer notre custom component <ButtonComponent> pour éviter de redéfinir un composant <Button> dans plusieurs composants

Cela nous permet d'utiliser un seul composant partout dans notre application et ainsi réduire la quantité de code.

Nous allons également utiliser les props pour customiser notre <ButtonComponent> en fonction de nos besoins.

<Text> et <View>

Nous avons déjà vu les composants <Text> & <View> mais j’aimerais revenir dessus une nouvelle fois afin de vous rappeler quelques points importants à garder en mémoire.

<View> applique systématiquement flexDirection: « column », ce qui n’est pas le cas de <Text>

<View> peut contenir autant de composants enfants <Text>.

<Text> aussi peut contenir des composants enfants <View> mais cette pratique est vivement déconseillée.

On ne peut pas styliser un composant enfant <Text> via un composant parent <View>. En revanche on peut le faire via un autre composant parent <Text>

© Donkey Geek

Créer un Header

Maintenant que nous avons vu les Custom Components, on va en créer un autre afin d'appliquer un Header à notre application pour y afficher le logo

Là aussi les composants <View> et <Text> et StyleSheet permettent de réaliser ce nouveau Custom Component.

© Donkey Geek

Constantes de couleurs

Maintenant que avons appliqué plusieurs styles à différents composants, on réalise qu'on commence à répéter certaines couleurs dans différents composants.

Imaginez qu'on décide de changer le thème de notre application et changer toutes les couleurs. On sera alors obligé d'aller chercher chaque couleur individuellement pour la remplacer.

Pour éviter cette situation, on va opter pour les constantes dans séquelles on aura toutes nos couleurs. Ainsi, il suffit de mettre à jour ces dernières et la mise à jour s'appliquera partout dans notre application.

© Donkey Geek

Composants réutilisables

Nous avons utilisé un composant <TextInput> directement dans le composant <AddProduct>. Ceci est parfaitement correct.

Cependant, si on aura besoin d'un autre composant <TextInput>, ailleurs dans notre application, nous allons devoir en créer un tout nouveau composant qui sera stylisé séparément du précédent. On va donc devoir dupliquer notre code inutilement.

Pour éviter cela, on va créer un composant « Input réutilisable » qu'on pourra appliquer partout dans notre application.

© Donkey Geek

Validation sans numbers

Nous avons utilisé un Custom Composant <Input>

Nous allons, à présent, interdire les valeurs numériques

Nous allons, à présent, interdire les valeurs numériques

A B c d

1 2 3 4 5

© Donkey Geek

Customiser les Fonts

iOS et Android utilisent leur propre ensemble de polices bien définies. Cependant, si vous souhaitez personnaliser (customiser) votre application avec votre font préférée vous pouvez parfaitement le faire.

C'est une pratique vivement recommandée car elle permet d'offrir une expérience utilisateur beaucoup plus cohérente sur différentes plateformes.

C'est ce que nous allons découvrir dans cette vidéo.

Customiser les Fonts

Expo offre un moyen nous permettant de customiser nos fonts

Pour cela, vous devez vous assurez que vous avez EXPO-FONT pré-installé. Sinon, vous pouvez l'installer de 2 façons:

Option 1 : **npm install --save expo-font**

Option 2 : **expo install expo-font** (Option recommandée)

Et on l'importe dans notre projet

import * as Font from 'expo-font'

Expo App Loading

Expo App Loading est un composant qui s’occupera de charger les données asynchrones avant l’affichage de notre application. Ainsi, les composant qui auront besoin des fonts ne seront pas affichés tant que les datas relatives aux polices ne sont pas récupérées.

Le chargement des autres composants vas s’effectuer seulement une fois, les données reçues et cela va être géré par « Expo App Loading »

Ensuite, on installe EXPO-APP-LOADING **expo install expo-app-loading**

Et on l’importe dans notre projet **import AppLoading from 'expo-app-loading';**

© Donkey Geek

Package Expo Google Fonts

@Expo-Google-Fonts est un package Expo qui nous permet d'obtenir facilement autant de polices de Google Fonts

Ces package ainsi obtenus sont en open-source et compatibles iOS et Android

Installation: **expo install @expo-google-fonts/NOM**

Importation: **import {useFonts, PACKAGES} from '@expo-google-fonts/NOM'**

© Donkey Geek

Les Icons

Vector Icons

@expo/vector-icons est un package Expo qui nous permet d'obtenir facilement des icons pour les utiliser dans nos applications React Native

Cette librairie fait partie du package expo. Elle est donc installée par défaut si vous avez initialisé votre projet via **expo init —**.

Une fois importée elle nous permet d'accéder à plusieurs ressources et librairies proposant des icons comme FontAwesome, Glyphicons, Ionicons, AntDesign etc

© Donkey Geek

Gestion Globale des Styles

Nous avons vu le moyen qui permet de styliser un composant via StyleSheet. Cependant, on constate ici une certaine redondance dans notre code.

Pour remédier à cette problématique, on peut opter pour les styles globales qu'on pourra utiliser partout dans notre application.

Le Debugging

Le debugging (débogage) est une étape inévitable par laquelle passent tous les développeurs. Les ***erreurs*** et autres ***dysfonctionnements*** (incohérence de logique ou d'affichage UI ..) sont des choses inévitables quand on code un programme informatique.

Note:

En tant que développeur, vous devez être débrouillards! Vous devez avoir un tempérament clame et éviter tout affolement en cas de bug ou autres problèmes que vous pouvez rencontrer dans votre code. Vous avez plusieurs moyens et techniques vous permettant de repérer et de corriger toutes les erreurs qui existent et c'est ce que nous allons voir dans ce chapitre.

© Donkey Geek

Comment Déboguer ?

Prendre le temps de bien lire les messages d'erreurs

Das 99% des cas, le message d'erreur explique clairement l'origine du bug et parfois on y trouve la solution déjà proposée dans le message d'erreur.

Réviser son code

Parfois un simple caractère mal rédigé peut causer un sérieux dysfonctionnement

Google est ton ami (pour le code)!

Si le message d'erreur n'est pas clair ou qu'ils ne vous guide pas assez, il suffit simplement de le copier et de le coller dans le moteur de recherche (Google), d'autres ont certainement eu ce genre de message avant vous et la réponse leur a peut-être apportée.

Quel outil pour Déboguer ?

Console.log( )

Dans la plupart dans cas et notamment sur les petites application, un debugging rapide via un simple console.log() suffit pour repérer et corriger un bug.

Chrome Debugger

Dans le cas de grandes applications complexes, le debugging via console.log() peut ne pas permettre repérer et corriger un bug. Dans ces cas-là, un outil de debugging plus performant comme « Chrome Debuggeur » peut s’avérer indispensable car offre plus de fonctionnalités pour traquer les bugs d’une manière beaucoup plus ciblée notamment via les Breakpoints.

React Developer Tools

Vous pouvez utiliser la version autonome (standalone) de React Developer Tools pour déboguer la hiérarchie des composants React.

Installation globale du package react-devtools:

```
npm install -g react-devtools
```

Installation en tant que dépendance de projet

```
npm install --save-dev react-devtools
```

> Ajouter "react-devtools" à la section scripts de votre package.json

> **npm run react-devtools** pour ouvrir les DevTools.

React Native Debugger

React Native Debugger est un outil autonome (standalone) qui permet d'accéder aux éléments de notre application (Interface Utilisateur), leurs States, leurs Props, leurs Styles…

- Remote Debugger
- React Inspector (react-devtools-core vu précédemment)
- Redux DevTools (redux-devtools-extension)
- Apollo Client DevTools

© Donkey Geek

Navigation Web

https://donkeygeek.com/index.html → SERVEUR → Index.html

https://donkeygeek.com/login → REACT ROUTER → <Login />

App React Native

https://donkeygeek.com/login

Dans une application React Native on n’a pas d’URL. On se déplace d’un écran un autre en interne. Clicks sur les boutons, les menus ‘les tabs’ pour naviguer entre le stack d’écrans.

React Navigation

React Navigation est une librairie qui va nous permettre de gérer facilement les Routes et la Navigation dans nos applications React Native

- Une facilité d'utilisation avec des « Navigateurs intégrés »

- Adaptabilité des composants conçus pour iOS et Android

- Entièrement personnalisable!

- Une plateforme extensible à chaque couche. On peut coder nos propres navigateurs

© Donkey Geek

React Navigation (versions)

Nous allons commencer avec la version 4, actuellement utilisée massivement dans beaucoup de projets React-Native, avant de passer à la version 5 qui apporte beaucoup de modifications et surtout de simplifications.

React Navigation

- Initialiser une nouvelle application « Pixels »
- Créer 3 écrans dans un dossier /screens
- Lancer l’application sur les deux simulateurs
- Mettre en place nos custom fonts avec AppLoading

© Donkey Geek

Stack Navigation

Installer la librairie « React Navigation » pour gérer la navigation des différents écrans.

© Donkey Geek

Stack Navigation

Comment naviguer entre les différents écrans?

© Donkey Geek

Stack Navigation

Différence entre ***navigate( )*** et ***push( )***

© Donkey Geek

Stack Navigation

Scénario 1

© Donkey Geek

Stack Navigation

Scénario 2

© Donkey Geek

Stack Navigation

© Donkey Geek

Stack Navigation

La méthode **pop( )** est une alternative à **goBack( )** mais elle existe que sur le Stack Navigator

Stack Navigation

Passer de la data entre les différents composants de React pour le web

Home (Parent)

```
const DATA = {
    name: ‘Maria’,
    age: 30
}

< Title name=‘DATA.name’ />

< Infos age=‘DATA.age’ />
```

Title (Enfant)

```
<h1>{props.name}</h1>
```

Infos (Enfant)

```
<p>J’ai {props.age} ans</p>
```

© Donkey Geek

Stack Navigation

Passer de la data entre les différents écrans de React Native

Ici, nous avons des écrans indépendants les uns des autres et ne sont donc pas des enfants des autres écrans pour pouvoir recevoir des informations via les props!

© Donkey Geek

Stack Navigation

On peut gérer la configuration du Header de différentes façons

Option #1: Une configuration pour chaque Route

Option #2: Une configuration par défaut pour tous les Routes

Option #3: Une configuration des Headers dans les composants

© Donkey Geek

Stack Navigation

Hiérarchie des styles entre les trois techniques

Propriété « navigationOptions » pour chaque Route

↓

Propriété « navigationOptions » directement sur le composant

↓

Propriété « defaultNavigationOptions » par défaut à tous les écrans

© Donkey Geek

Stack Navigation

Comment passer des valeurs dynamiques dans le Header ?

Changer la couleur du background en fonction de la couleur préférée de l’utilisateur

Changer le titre pour afficher le nom de l’utilisateur

Stack Navigation

La façon la plus courante d'interagir avec un en-tête « Header » consiste à appuyer sur un bouton à gauche ou à droite du titre.

Le bouton est évidemment un composant React-Native mais aussi du JSX. On aura donc besoin des deux librairies dans les fichiers où le <Button /> sera appliqué.

Ajouter un bouton sur le côté droit de l'en-tête

React Navigation Header Buttons

React-Navigation-Header-Buttons est un package qui nous aide à afficher des boutons dans la barre de navigation mais aussi à gérer le style pour que nous n'ayons pas à le faire.

Ce package essaie d'imiter l'apparence des boutons natifs de la barre de navigation pour nous permettre d'obtenir une interface simple et flexible avec laquelle les utilisateurs pourront interagir.

Ce package prend en charge iOS et Android. Il est codé avec Flow (version TS de Facebook) mais livré en TypeScript.

PS: Si vous êtes intéressés, vous pouvez consulter la formation TypeScript pour Tous !

© Donkey Geek

Exercice

Sur l’écran d’accueil, afficher cette une icône de ***Material Icons***

En cliquant sur l’icône, affichez ***alert(‘Afficher le menu latéral’)***

© Donkey Geek

Drawer Navigation

Le **Drawer Navigation** est une autre dépendance du package React Navigation et qui nous permet d'afficher un menu latéral dans notre application React Native.

- React Navigation version <4: pas de dépendance à installer

- React Navigation version 4: Il faut installer la dépendance

« react-navigation-drawer »

© Donkey Geek

App.js
Drawer Navigator
(Main Navigator)
createDrawerNavigator(
{
Stack Navigator
Composant Faq
}
)
Stack Navigator (Menu)
<Faq />

© Donkey Geek

Exercice

- Afficher le Drawer tout en affichant l'icône du menu
- Changer le texte des onglets du Menu Latéral (Drawer)

  Home → Accueil

  Faq → FAQ

- Augmenter la largeur du Drawer à 110 px
- Augmenter la taille de la font à 19 px
- Définir la couleur du texte en blanc si onglet désactivé
- Changer la couleur du texte de l'onglet une fois activé

© Donkey Geek

Tab Navigation

Tab Navigation (La navigation par onglets) est probablement le moyen de navigation le plus courant dans les applications mobiles modernes.

- Onglets en bas de l'écran
- Onglets en haut de l'écran sous le Header
- Onglets à la place du Header

React Navigation v: 4: Package « react-navigation-tabs »
React Navigation v: <4: Pas de package à installer

© Donkey Geek

createAppContainer(MainNavigator)
Accueil
StackNav
Home
Portfolio
Photo
FAQ
FaqStackNav
Faq

© Donkey Geek

createBottomTabNavigator(StackNav + Selected)
createAppContainer(MainNavigator)
Home
FAQ
StackNav
Home
Portfolio
Photo
+
Selected
FaqStackNav
Faq

© Donkey Geek

Tab Navigation

- Changer la couleur du texte de l’onglet activé

- Changer la couleur du texte de l’onglet désactivé

- Afficher des icônes dans les onglets de Tab Navigation

- Appliquer les mêmes couleurs aux icônes que celles définies pour les onglets (Activé - Désactivé)

- Changer le texte des onglets en français

© Donkey Geek

Tab Navigation

- Changer l’apparence des onglets sur Android pour leur donner un style plus adapté aux applications Android.

Pour cela, on aura besoin de deux packages « react-navigation-material-bottom-tabs » et « react-native-paper »

© Donkey Geek

Tab Navigation

- Créer un StackNavigator pour contenir le composant « Selected » et « Photo »

- Afficher un Header au niveau du l'écran « Selected » avec un background « darkGrey » pour Android et « lightbrown » pour iOS

© Donkey Geek

Exercice

- Travailler l'affichage de l'écran Portfolio. Pour cela, nous devons utiliser les informations passées de l'écran « Home » vers « Portfolio »

- Afficher un « header customisé » pour chaque membre en fonction de sa couleur préférée

- Afficher la photo du membre dans un cercle

- Afficher la biographie du membre

- Afficher ses photos cliquables et passer leurs informations vers l'écran « Photo »

© Donkey Geek

Exercice

- Travailler l’affichage de l’écran Photo. Pour cela, nous devons utiliser les informations passées de l’écran « Portfolio » vers « Photo »

- Afficher les informations relatives à la photo sélectionnée

- Afficher le titre de la Photo dans le Header

© Donkey Geek

React Navigation V4 vers V5

Avant de passer à la version 5 de React Navigation, nous devons terminer notre application. Pour cela, on aura besoin de Redux.

Redux & React Native

Pour utiliser la librairie « Redux » avec « React Native » on aura besoin de deux packages

© Donkey Geek

Redux & React Native - Sélection

- Installer les deux packages « Redux » & « React-Redux »
- Initialiser notre Store et connecter Redux à notre application
- Récupérer la data des utilisateurs dans le Redux Store et gérer l’affichage en cas contenu vide
- Définir une action de sélection d’images via une fonction de création d’action
- Afficher le pouce vers le haut dans le header pour sélectionner les photos d’un utilisateur
- Dispatcher l’action précédemment créée + utilisation setParams( )
- Gérer l’action dispatchée au niveau du Reducer
- Afficher les photos des utilisateurs enregistrés dans l’écran Selected
- Modifier l’icône pour les utilisateurs sélectionnés + le Alert quand on efface les images

Redux & React Native - Settings

- Créer une nouvelle icône dans le Header de l’écran « Home » pour afficher un Modal
- Afficher un composant Switch pour la gestion des Settings « Réglages »
- Gérer la validation des Switch et mettre les States dans un objet
- Créer une action setCategorySettings pour dispatcher l’objet généré via les Switch
- Dispatcher l’objet généré via les Switch

© Donkey Geek

Switch

Switch est un autre composant "React Native" qui retourne un booléen.

Il s'agit d'un composant contrôlé qui nécessite un prop « ***onValueChange »*** qui met à jour le prop ***« value »*** afin que le composant reflète les actions de l'utilisateur.

© Donkey Geek

React Navigation 5

React Navigation 5

- - Installation du package React Navigation 5
- - Installation des dépendances de React Navigation 5
- - Créer 2 écrans Home + Portfolio (dans /screens)

© Donkey Geek

React Navigation 5

- Naviguer entre nos deux écrans via un StackNavigator

Fichier de configuration

Du StackNavigator

setParams

Configuration

DYNAMIQUE

du StackNavigator

Format Composants

© Donkey Geek

Stack Navigator

- Changer le titre du header de l’écran Home
- Définir le route « Portfolio » comme route initial
- Retirer le Header uniquement sur l’écran Portfolio
- Retirer le Header sur tous les écrans
- Changer le titre de l’écran Home dynamiquement en cliquant sur le bouton

© Donkey Geek

Stack Navigator - Exercice

- Changer la couleur de background en ‘slateblue’ pour le header de « Portfolio ».
- Changer le couleur du texte du header de « Portfolio » en ‘white’.
- Changer le style de la police en gras & en majuscules.
- Appliquer les 3 points ci-dessus à l’ensemble des headers du stackNavigator.

© Donkey Geek

Stack Navigator - Exercice 2

- - Sur Android, dévoiler l’écran en dessous en effectuant un balayage avec le doigt. Cette fonctionnalité existe par défaut sur iOS.
- - Effectuer un balayage de gauche à droite pour dévoiler l’écran caché
- - Effectuer un balayage de droite à gauche pour dévoiler l’écran caché (transition bas vers haut)
- - Effectuer un balayage du bas vers le haut pour dévoiler l’écran caché (transition bas vers haut)

Stack Navigator - Exercice 3

- - Effectuer une animation d’ouverture avec un effet de d’amortissement
- - Effectuer une animation de fermeture avec un effet d’assouplissement « easing »
- - Sur android, obtenir le même effet de slide que celui sur iOS (Horizontalement)
- - Sur iOS, obtenir le même effet de slide que celui sur android (Verticalement)

Stack Navigator - Exercice 4

- Passer l’objet dataObj de l’écran Home vers le route Portfolio et afficher le « name » et « age » de la personne dans un composant <Text>.
- Afficher le nom de la personne « name » dans le header de l’écran Portfolio
- Utiliser la couleur passée dans l’objet comme backgroundColor du header de l’écran Portfolio

```
let dataObj = {
    name: "Selma",
    age: 28,
    color: 'royalblue'
}
```

© Donkey Geek

Stack Navigator - Exercice 5

- Créer un Text cliquable via le composant TouchableOpacity dans le header de l'écran Portfolio pour incrémenter un une valeur « count » dans un state existant dans le composant Portfolio

- Dupliquer le même TouchableOpacity pour décrémenter le même « count »

© Donkey Geek

Drawer Navigation

- Installation du package @react-navigation/drawer
- Créer un Tab Navigator avec 2 onglets pour naviguer entre l'écran Home et Portfolio
- Afficher une icône dans le header des deux écrans pour ouvrir le Drawer

© Donkey Geek

Drawer Navigation - Exercice

- Définir « Portfolio » comme le route Initial
- Afficher le Drawer du côté droit de l’écran
- Afficher le Drawer d’une façon permanente
- Afficher le Drawer sous l’écran
- Pousser l’écran lors de l’affichage du Drawer
- Ouvrir le drawer via un balayage du milieu de l’écran
- Appliquer une couleur rouge pour recouvrir l’écran lors de l’ouverture du Drawer
- Changer la couleur du Drawer + largeur à 100%

© Donkey Geek

Drawer Navigation - Exercice 2

- Changer le texte de l’onglet de Profil en « Ma Page »
- Afficher une icône à coté de Home
- Préciser la couleur de l’onglet activé en « white »
- Préciser la couleur de l’onglet désactivé : « yellow »
- Préciser une marge entre les onglets de 30px

© Donkey Geek

Drawer Navigation - Exercice 3

- Retravailler le Drawer pour ressembler à celui de Twitter

© Donkey Geek

Tab Navigation

- Installer le package @react-navigation/bottom-tabs
- Créer deux onglets en bas de l'écran pour naviguer vers l'Accueil et Settings

© Donkey Geek

Tab Navigation - Exercice

- Définir une icône pour chaque onglet et changer la couleur de l'icône quand l'onglet est sélectionné
- Agrandir la taille de l'icône de l'onglet sélectionne
- Changer la couleur de l'icône de l'onglet sélectionne
- Eviter la répétition pour l'application de couleur de l'icône tout en l'appliquant également au texte de l'onglet
- Changer la couleur du background de l'onglet activé en « #ccc »
- Agrandir la taille du texte de l'onglet
- Afficher uniquement les icônes (Retirer la couleur de background)
- Afficher une notification 8 sur l'onglet « Accueil »

© Donkey Geek

App. e-commerce avec React Navigation V5

- Initialiser l’application « Success Coder » via Expo (Vente de formation Dev)
- Importer la data de teste
- Installer Redux et React-Redux
- Intégrer le Redux Store dans notre application avec un reducer pour les cours
- Créer et importer un écran dans App.js pour y afficher les cours

© Donkey Geek

Redux

© Donkey Geek

App. e-commerce avec React Navigation V5

- Afficher les cours disponibles. Si aucun cours n’est disponible, on affiche un message
- Afficher les icônes « plus de détails » et « ajouter au panier »
- Créer deux nouveaux écrans « CourseInfos + Cart »
- Installer React Navigation V5 et ses dépendances
- Créer un stackNavigator pour les écrans « Landing + CourseInfos »
- Afficher le nom du cours dans le header de CourseInfos
- Styliser le header du stackNavigator
- Afficher une icône sur le côté droit pour naviguer vers Cart

© Donkey Geek

App. e-commerce avec React Navigation V5

- Travailler l’affichage du composant CourseInfos
- Ajouter un cours dans le panier
- Retirer du catalogue le cours ajouté dans le panier
- Afficher les cours enregistrés dans le panier
- Retirer les cours du panier
- Afficher dans le catalogue le cours effacé du panier
- Effectuer l’achat des articles déposés dans le panier + vider le panier
- Créer un Drawer Navigator avec trois onglets « Catalogue - Panier - Achats »

© Donkey Geek

App. e-commerce avec React Navigation V5

- Afficher une icône sur tous les StackNavigators pour ouvrir le Drawer
- Afficher l’historique des formations achetées
- Créer deux nouveaux écrans utilisateur pour modifier et créer des cours
- Afficher les deux écrans dans un stackNavigator avant de les intégrer dans le Drawer
- Afficher les cours dans l’écran userCourses
- Supprimer nos cours (supprimer dans le catalogue + panier)
- Afficher le formulaire pour modifier ou créer un nouveau cours
- Modifier un cours et créer un cours

© Donkey Geek

App. e-commerce avec React Navigation V5

Validation du formulaire de création ou de modification de cours

- ✓ - Afficher un clavier numérique pour le prix
- ✓ - Permettre d’avoir un champ multi-lignes (Ex: 4 lignes) pour les informations
- ✓ - Gérer tous les inputs via un seul handler (Créer un custom Input)
- ✓ - Désactiver le bouton de validation tant que tous les TextInput ne sont pas renseignés

AsyncStorage (stockage asynchrone)

AsyncStorage est une API de stockage simple sous forme de clé-valeur.

- Stockage **non crypté** *(ne pas l'utiliser pour stocker une data sensible comme un mot de passe ..)*
- Il est asynchrone, global pour l'application, et persistant *(La data ne s'efface pas quand on relance l'application ou même le téléphone).*
- C'est un stockage « Offline » généralement utilisé comme alternative au « localStorage » des navigateur web (dans le cas des applications Web).
- Sur iOS, AsyncStorage est soutenu par un code natif qui stocke les valeurs dans des fichiers internes à l'appareil. Sur Android, AsyncStorage utilise soit RocksDB ou SQLite en fonction de ce qui est disponible sur l'appareil.

© Donkey Geek

Inscription
AsyncStorage
Vérifier / Récupérer
Vers l'écran Login
Vers Espace membre
4:27
Donkey Geek
@DonkeyGeek
24 Abonnements 48 Abonnés
Profil
Listes
Sujets
Signets
Moments
Réglages
Déconnexion

Exercice: AsyncStorage

- Retravailler notre React Navigation pour y inclure une page de Login (Connexion)
- Créer l’écran Login avec un petit formulaire
- Installer AsyncStorage via expo pour stocker les données utilisateur
- Enregistrer le nom de l’utilisateur (string) dans AsyncStorage et accéder à l’espace de membre
- Interroger AsyncStorage pour récupérer le nom de l’utilisateur pour l’afficher dans le Drawer
- Effacer la clé enregistrée dans AsyncStorage et rediriger l’utilisateur vers l’écran Login
- Passer un objet dans AsyncStorage et l’afficher dans le Drawer
- Effacer l’objet enregistré dans AsyncStorage + Effacer tous les clés

© Donkey Geek

Authentification & requêtes HTTP (Firebase)

ActivityIndicator

Activity Indicator est un composant React Native qui permet d'afficher facilement un spinner animé.

Il peut, entre autres, servir à indiquer à l'utilisateur que le système est en attente d'une réponse coté base de donnée ou simplement le chargement d'un contenu.

Dans notre exercice, il va servir à indiquer à l'utilisateur que son inscription est en cours de traitement et qu'il doit patienter un petit instant.

© Donkey Geek

Base de données (Firebase)

© Donkey Geek

iPhone 12 Pro Max – 14.4
Inscription
Connexion
Email
Mot de passe
Auth
Informations
Nom
Prénom
Photo
URL
Galerie
BDD
ID user
Objet
Donkey Geek
Carte
9:45
Map
San Francisco
Maps
vers Home
Géolocalisation
Enregistrement SQLite
BDD Locale (Offline)

© Donkey Geek

Responsive & UI Adaptative

Nous avons vu comment styliser nos applications avec Flexbox mais, vous avez certainement constaté que, dans certaines situations, cela pourrait ne pas suffire.

De nos jours, nous avons une large gamme d'appareil, avec des dimensions d'écrans très variées, et auxquels devront s'adapter nos applications. Non seulement pour le bon fonctionnement de ces dernières mais aussi pour offrir la meilleure expérience utilisateur.

Donc une «Interface Utilisateur» bien adaptée.

Tester sur un petit écran

Pour commencer, nous allons tester notre application Pixels sur un nouveau simulateur ayant un écran de petite taille.

Cela va nous permettre de voir rendu visuel et, si nécéssaire, essayer de trouver des solutions pour adapter notre Interface Utilisateur aux dimensions de l'appareil.

Exercices

- Modifier la hauteur de l'image pour qu'elle occupe la moitié de la hauteur de l'écran sur Android & iOS.
- Afficher deux images avec les informations sur iOS et une seule sur Android.
- Afficher deux images avec les informations sur les écrans dont la hauteur est >600px et une seule sur les autres. (Utiliser un Opérateur Conditionnel - Ternary Expression)
- Appliquer un style spécifique au nom de la personne pour les écrans >600px à la manière des break points via les « media queries » en CSS. Cibler les écran dont la hauteur >600px.
- Afficher un message d'erreur si la hauteur de l'écran est < 600px.
- Réduire la taille de la font de biographie sur les appareils avec un écran < 600px de hauteur.

API DIMENSIONS

Grâce à l'API Dimensions, nous allons pouvoir obtenir, entre autres, les informations relatives aux dimensions de l'appareil.

Cela va nous permettre d'ajuster nos contenus en fonction de la taille de notre écran.

Note : Les valeurs sont obtenues lors du chargement ! Si votre application autorise les deux modes (Portrait et Paysage), cela peut causer des problèmes d'affichages.

On verra cette situation plus tard.

Exercices (suite)

- Corriger l’affichage des images sur l’écran Portfolio pour les écrans <600px de hauteur.
- Corriger le padding gris sur les petits écrans <600px de hauteur.
- Permettre l’utilisation de l’application en mode paysage seulement
- Permettre l’utilisation de l’application en mode paysage et portrait
- Corriger le problème d’affichage suite au changement de mode (Landscape/Portrait)
- Corriger le problème d’affichage suite au changement de mode sans définir un listener!

  Pour cela, on aura besoin de ***useWindowDimensions*** *qui permet de* mettre automatiquement à jour les valeurs de largeur et de hauteur lorsque la taille de l'écran change.
- Corriger le problème causé par l’encoche du téléphone (Simulateur iOS)

© Donkey Geek

Chargement

Nouvelle orientation

Exercices (suite)

- Modifier l’affichage sur les écrans < 400px (Android Mode Paysage)

© Donkey Geek

Quiz N°3

Quelques petites révisions autour des thématiques abordées dans le chapitre « Responsive & UI adaptative » mais aussi pour tester vos capacités à chercher les informations d’une manière autonome.

Notifications

Notifications Locales:

Sont déclenchées par l’application en local sur l’appareil de l’utilisateur.

Elles ne sont pas déclenchées pour d’autres utilisateurs (Sans passer par un serveur).

Elles peuvent se déclencher même si l’application est fermée.

Exemple: Une application **Todo-List** qui vous rappel que vous avez une tâche à accomplir.

Notifications Push

Sont envoyées (déclenchées) de quelque part (serveur, autre utilisateur..) et reçues par l’application.

Une fois les Notifications Push reçues sur l’app, celle-ci pourra les déclencher via les notifications locales.

Exemple: Serveur envoie une notification de vente, email, like, YouTube vidéos, un follower…

© Donkey Geek

Notifications Locales

- Créer une nouvelle application via Expo.
- Générer une simple notification local en cliquant sur un bouton.
  > On aura besoin d'installer le package *Expo Notifications*
- Mettre en place un listener pour détecter la notification quand l'app est lancée
- Afficher le titre, body et data de la notification dans le JSX
- Nettoyage des listeners

Explicitement accorder la permissions sur iOS !

© Donkey Geek

Notifications Push

(Événement)

Publier une nouvelle vidéo sur Facebook

© Donkey Geek

Notifications Push

Identifier les Applications via un Token (id / jeton unique) pour afficher les notification Push

Serveur de notifications (Google & Apple)

Notifications Push

- Avant d’attaquer les notifications Push, tester les notifications Locales sur un vrai téléphone
- Les Notifications Push ne fonctionnent pas sur les simulateurs. Mettre en place une condition pour vérifier si on utilise un simulateur pour afficher une alerte.
- Demander la permission pour les Notifications Push. Si elle est accordée, afficher le jeton (Token) dans le JSX

Avec Expo on code de vraies applications natives !

- Configuration de notre application
- Code JavaScript
- Code Expo

Déploiement App Expo

Déploiement Sans Expo

Développement

Chargement dans l'app Expo

Simulateur / Vrai appareil

Même si votre application utilise Expo en arrière-plan, les utilisateurs n'auront pas besoin d'installer Expo !

Les utilisateurs vont télécharger une vraie application Native.

Cette légère couche Expo n'a pas un grand impact en terme de performance. Cependant, vous pouvez coder sans Expo si ce dernier ne vous convient pas.

Pour les besoins de cette formation et afin de que tout le monde puisse suivre sans difficultés, nous avons opté pour EXPO CLI.

EXPO CLI

Managed Workflow | Flux de travail géré

Possibilité de passer vers le Bare Workflow | Flux de travail non-géré via « eject »

Flux de travail non-géré

(Bare Workflow)

App Non-Expo. Pas de code supplémentaire

Quelques package Expo disponibles
Installation relativement complexe comparée à Expo

Tous les modules natifs accessibles

Code & déploiement via Android Studio ou Xcode

React Native CLI

Flux de travail non-géré

© Donkey Geek

AVEC EXPO

Coder facilement des App iOS/Android

Coder des App Natives !

Facilité d'intégration de modules Natifs

Léger impact en terme de performance
à cause du code en plus (Couche Expo)

Limitations aux modules natifs pré-définis

SANS EXPO

Pas de code supplémentaire

Pas de limitation en terme de modules
(Librairies et modules tiers)

Configuration Manuelle requise !!!

React Native CLI (MAC)

Pour changer votre environnement de développement et passer à React Native CLI, vous devez effectuer quelques configurations en fonction de votre système.

Je code sur un MAC, je vais vais donc devoir suivre attentivement les instructions relative à la configuration **masOS**.

Avec React Native CLI, nous allons devoir installer Android Studio (App Android) ou Xcode (App iOS) pour pouvoir coder ces applications en local car nous n’avons plus Expo.

Créer un nouveau projet

Présentation du contenu du projet initialisé via React Native CLI

Nous pouvons lancer notre application sur un simulateur ou sur un vrai appareil via USB

Lancement de l'app sur un simulateur iOS et modifier le titre ou la couleur

Comme sur Expo, les erreurs seront affichées sur l'appareil + le terminal

Exercices via React Native CLI

Installer React Navigation V6 dans notre application React Native CLI

Installer Le Stack Navigator V6 dans notre application React Native CLI

Créer deux écrans « Home » et « Profile »

Naviguer entres les écrans

- Mettre les Paths des routes dans ./constants/navigationPaths.js
- Naviguer de l’écran d’accueil vers le profil

Passer une data de l’écran d’accueil vers l’écran profil et l’afficher dans le titre

Changer les titres dans le header en français: « Accueil » et « Mon Profil »

Changer la couleur des headers et la couleur de la police + seulement Profile

Cacher tous les headers

© Donkey Geek

Exercices via React Native CLI

Installer le Tab Navigation

Passer le Tab Navigation dans le Stack.Screen du Stack Navigation

Afficher deux onglets en bas de l’écran pour naviguer entre l’accueil et le profil

Exercices via React Native CLI

Enlever le Header du Stack

Enlever le Header du TabNav

Cacher les Labels des Tabs

Donner une couleur de background « dodgerblue » à la TabBar

Afficher les Icônes sur les Tab

En cliquant sur une icône on change sa couleur

Sélectionner une image dans la galerie et l'afficher dans l'écran Profil
Pour cela, utiliser le package React Native Image Picker (sans Expo)

Prendre une photo avec la caméra de l'appareil pour l'afficher

© Donkey Geek

EXPO CLI

REACT NATIVE CLI

Vous vous sentez à l'étroit avec Expo mais en même temps la configuration de certains packages RN-CLI sont compliqués ?

Vous pensez à effectuer le « eject » mais vous avez peur de passer à coté d'un package Expo qui pourrait vous intéresser ?

Expo est toujours là pour vous aider avec le Bare Workflow !

Bare Workflow

Avant d'aborder « eject » qui permet de passer d'une application Expo CLI vers React-Native CLI, nous allons voir le « bare Workflow ».

Le « Bare Workflow de Expo » est une autre façon « intermédiaire entre Expo CLI et React Native CLI ». En d'autres termes, avec le bare workflow on aura une application comme celle de React Native CLI mais configurée de manière à pouvoir prendre en charge plusieurs packages Expo.

Pourquoi ?

- Utilisation de la configuration « Brute » sans le wrapper de Expo tout en profitant de ce dernier
- Pouvoir utiliser Android Studio et Xcode comme dans RN-CLI
- Avoir moins de configurations à faire, contrairement à RN-CLI (Profiter des packages natifs Expo)

Expo nous offres énormément d'outils ainsi qu'une grande facilité dans la réalisation de nos applications. Cependant, avec Expo, vous pourriez vous confronter à l'incapacité de gérer certains modules natifs inexistants via Expo. Vous pouvez alors effectuer un eject pour passer de la version Expo CLI et RN CLI. Opération est irréversible !!!

© Donkey Geek

Création du build - ApplicationsExpo

Etape 1

Projet Expo

Projet Expo

bare Workflow

Etape 2

Compte utilisateur

Gratuit

Effectuer le build sur le serveur Expo pour les différentes plateformes et appareils

On peut effectuer un Build pour tester localement sur les simulateurs

Etape 3

Installer EAS CLI

sudo npm install -g eas-cli

Google Play

Apple Store

Configurations

Avant de publier votre application, certaines configurations peuvent êtres requises dans votre projet.

Nom de l’application et son identifiant

Vous devez définir le nom de votre application, son identifiant dans les stores et ses versions.

Variables d’environnement

Si vous utilisez de la data sensibles dans votre applications, pensez à mettre en place les variables d’environnements pour sécuriser ces informations.

Personnalisation

Pensez à personnaliser l’apparence de votre application en précisant son logo qui s’affichera sur les stores et une fois l’application installée sur l’appareil de l’utilisateur ainsi que l’écran de démarrage.

Permissions:

Si vous utilisez certaines fonctionnalités natives de l’appareil comme la caméra ou encore la géolocalisation, vous devez alors vous assurez d’avoir correctement configuré les permissions. Cela dépend à la fois des packages que vous utilisez mais aussi des systèmes Android et iOS. Surtout, si vous n’utilisez pas avec Expo.

BRAVO !!!!

Ici, se termine cette version de la formation **React Native Pour Tous - L’ultime formation**.

D’autres chapitres pourraient arriver très prochainement. Pour cela, je vous recommande de me suivre sur les réseaux sociaux ci-dessous pour rester informés.

https://twitter.com/donkeygeek

Merci pour votre confiance et excellente formation à tous.

Cordialement,

Donkey Geek :-)