React Native

Si besoin de mettre à jour les dépendances ainsi qu’Expo voici la ligne de commande :

* npx expo install expo-constants@~16.0.1 expo-font@~12.0.4 expo-status-bar@~1.12.1 expo-web-browser@~13.0.3 react-native@0.74.1

React Native c’est quoi ? C’est une autre bibliothèque Js qui se base sur la logique de react pour créer des applications natives Android et IOS. Donc on code une seule fois en Js et on génère du code natif Android et IOS. Il 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.

Il y aura quelques différences par rapport à React JS sur les nom des « balises », pas de panique nous allons voir cela ensemble, React native va si on peut dire adapter ses balises selon les plateformes utilisés je vous montre un petit exemple rapide d’un composant React et React Natif.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

React

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

React Natif

Cela reste bien du Jsx, vous remarquez que les « balises » sont en majuscules et le onclick s’est transformé en onPress, pourquoi ?

Et bien sur un mobile on ne click pas on appui donc ce sera un onpress tout simplement, les « balises » sont appelés composant et sont donc en Majuscule, voilà, pas de panique encore une fois nous allons les voir ensemble.

Quelques précisions rapides, 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 Js qui contient toute votre logique n’est pas compilé, ce qui est une bonne chose en termes d’optimisation de votre application.

Nous allons également utilisés certains outils dont Expo, je vous propose de voir à quoi cela ressemble avec ce lien <https://snack.expo.dev/>.

Je vous propose quelques extensions Vs Code vous en avez déjà certaines mais je vous conseille vivement l’extension : ES7 React/Redux/GraphQl/React-Native Snippets.

**Comment tester et visualiser votre application ??**

Vous avez l’outil Expo CLI, on va utiliser celui-ci car c’est très simple et gratuit qui permet de code facilement de vraies applications React native. Néanmoins il y a un petit inconvénient à Expo malgré sa facilité il vous limite quand même à l’écosystème Expo. De ce fait si par hasard vous souhaiter accéder à une fonctionnalité native et que celle-ci n’est pas offerte au niveau d’expo vous aurez des petits bug. Cependant aucun souci à vous faire si jamais vous vous retrouvez dans cette situation vous pouvez effectuer un eject et sortir de expo et passer au niveau de React native CLI. Petite précision si jamais vous optez pour cette méthode sachez que sous window vous devrez si vous désirez coder un appli Ios obligatoirement passer par expo cli, il ne prend que le support android, en revanche sous mac aucun soucie donc sous window je conseille vivement Expo CLI. React native CLI nécessite une certaine configuration relativement complexe au départ et surtout un certain nombre d’outils on fera donc sur Expo pour plus de facilité.

**Comment fonctionne Expo ??**

C’est plutôt simple vous avez Vs code dans lequel vous coder votre appli, pour pouvoir visualiser votre appli vous aurez donc besoin de Expo, ce dernier on va le télécharger sous forme d’application pour pouvoir visualiser le rendu directement sur votre téléphone ou bien sur un simulateur. C’est-à-dire que vous aurez un émulateur en ligne, une version virtuelle dans lequel vous allez tester et visualiser directement sur votre écran d’ordinateur.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Rendez-vous sur la doc de React Native, <https://reactnative.dev/>. Sur docs et introduction, vous aurez un exemple de composant React native, vous reconnaissez certainement certaines choses, les imports, les imports des composants du packages react native View et Text afin de pouvoir les utiliser dans votre application. Rendez-vous sur le site d’Expo <https://expo.dev/>. Et profiter en pour regarder un peu la doc et vous familiariser avec, les étapes sont extrêmement claires et simples. Je vous encourage à aller sur DOCS et Create a universal Android, Ios and web app puis descendez et cliquer sur open in snack pour visualiser le rendu et agir dessus en direct, vous avez le choix android, ios ou web app. Avant vos grands début. Pour visualiser sur votre téléphone il vous faudra télécharger Expo go et scanner le Qr code au lancement de votre appli. Sinon vous pouvez taper la commande

**npx create-expo-app --template blank** et il sera initialisé en javascript et pas en typescript qui est le modèle par défaut.

Allez !! On se lance,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Après la première ligne vous pouvez ajouter ou pas -t pour avoir le template, ensuite vous allez devoir choisir le template en question, vous pouvez choir le Blank et patienter. Il vous demandera sur quel plateforme lancer le projet, l’émulateur ou votre téléphone fera très bien le travail. Je vais garder le même nom pour notre application.

Vous choisirez **Blank** pour cette fois ci, et on patiente le temp de l’install ensuite vous ouvrez votre projet dans Vs Code. Vérifiez le package.Json et que le script soi bien présent, vous pouvez ensuite lancez la commande **npx expo start** (ou bien même **npm start** qui fonctionnera également). Vous avez face à vous un Qr Code et quelques commande vous pouvez choisir celle qui vous convient le mieux, soit installez expo go et scanner le Qr code sinon vous taper le a et cela lancera un émulateur android. Vous patienter et votre projet sera ouvert, et voilà votre première appli mobile est initialisée.

Je sais déjà que vous avez les bonnes versions de node.js pour continuer ce cours, si vous voulez vérifier il suffit de taper **node –version**.

Pour ceux qui désirent un émulateur vous pouvez télécharger Android studio <https://docs.expo.dev/workflow/android-studio-emulator/> vous cliquer sur installer et accepter les conditions et termes et lancer le téléchargement, cela peut être un peu long. Ensuite vous l’ouvrez et cliquer sur standard, ensuite vous risquez d’avoir des components à installer là encore ça peut être un peu long alors soyez patient.

Ensuite more actions -> Android SDK -> SDK tools vous avez les packages vous allez ajouter Google Play Services -> apply et vous installez, finish -> Ok

Encore more actions -> Sdk manager -> Sdk platforms et vous pouvez ajouter la version android 11.0®, et la version 14.

Encore more actions -> virtual device manager si vous souhaiter créer un nouvel appareil et bien vous cliquez sur + et phone je vous conseille si vous ajoutez un nouvel appareil de choisir ceux qui ont le logo play store ce sera bcp plus simple pour vous, donc vous en avez choisi un (ou pas) click sur next et vous allez devoir choisir l’image system de l’appareil vous sélectionnez celle qui vous convient (en général pie ou la premiere) download et next -> finish, vous pourrez reclicker sur pie et faire quelques modifications si besoin.

Ensuite vous pourrez lancez votre emulateur en clickant sur play

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Et voilà votre émulateur fonctionne sur votre bureau, c’est fini. Je deconseille fortement de lancer deux émulateurs en même temps votre ordi risquerai de ne pas du tout apprécier.

Maintenant comment mettre votre appli dessus et bien on y va, vous allez sur votre terminal vs code, npm start ou npx expo start et cela va se lancer, votre metro bundler se lance vous avez le choix pour lancer votre appli sur un serveur web ou sur votre émulateur android.

Sachez que les changements se font automatiquement après sauvegarde de ces changements plus besoin d’actualiser et oui !!

OK abordons désormais les principes de React native, tout d’abord sachez que vous allez me voir importer la bibliothèque comme ceci : import React from ‘react’ mais cela n’est plus nécessaire c’est plus par habitude que je le fait, depuis la version 17 on a plus besoin de le faire.

Pour afficher sur le serveur web vous aurez besoin de ceci :

* npx expo install expo-constants expo-web-browser react-dom react-native-web
* npx expo install @expo/metro-runtime

Abordons les dossier créer à l’initialisation de l’appli :

* expo et expo shared ou programm data : ce sont les dossiers qui vont contenir les fichiers de configuration d’expo **on y touche pas**
* assets : ici vous aurez les images de votre application
* node modules : là ou toutes les dépendances sont enregistrés
* gitignore : gère le versionning via Git
* App.js : c’est le point d’entrée de votre application cela ne change pas de React
* app.json : il contient les infos relatives à vos projets
* babel.config.js : il vous permet de compiler votre code et de vous générer votre bundle
* package.lock.json : contient des infos sensible sur votre projet encore une fois **on y touche jamais**
* package.json : c’est ici que vous allez pouvoir accéder à vos dépendances

**Les Composants View et 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 Js, certaines manipulations tactiles et les contrôles d’accessibilité. C’est aussi un élément parent qui peut envelopper d’autre View, il peut contenir d’autre composants de tous types, en revanche **pas de texte direct dans un View**. Ils sont natifs à la librairie React native, voici la liste : <https://reactnative.dev/docs/components-and-apis>

On va commencer à coder vous pouvez déjà enregistrer le raccourci **rnf** pour créer le composant de type function.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le style est moche on va voir cela de suite, sur la doc vous cliquer sur le composant en question et vous avez un menu à droite vous chercher style et vous cliquez dessus vous aurez l’exemple (encore et toujours la doc), vous aurez toutes les props sur la droite encore une fois. Rappelez-vous que l’on ne peut pas ajouter une View dessous l’autre vous devez avoir une seule View parent. Allez mettons du style en inline style, **prenez en compte que le composant View est par défaut en display : flex et flex-direction : column**.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

On va ajouter du texte et les englober dans une autre View. A vous de jouer !!

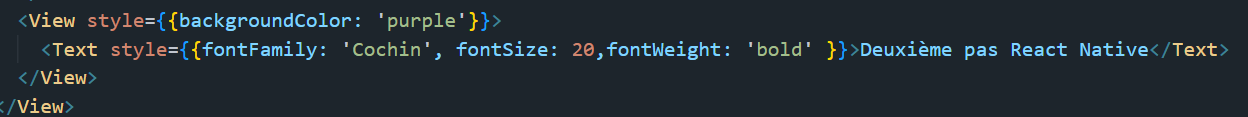
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

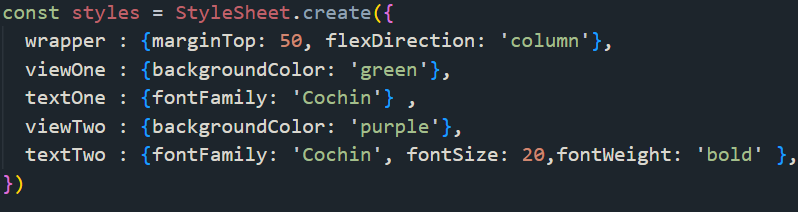
Je vais même copier cette View et la rajouter a cote de la précédente,

Une image contenant texte, capture d’écran, Police

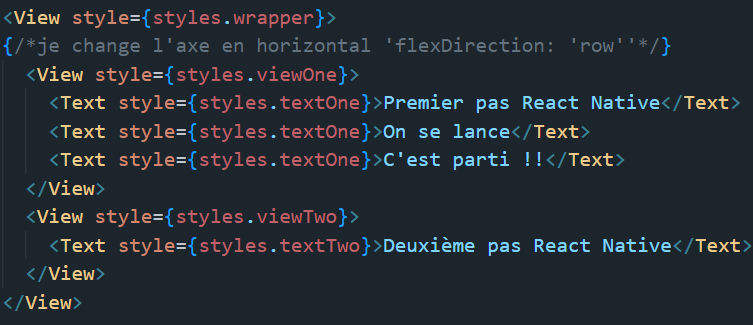
Description générée automatiquement

Voilà, là on voit bien la différence. 

Là on a fait du inline style et on peut voir que l’on répète le style et que cela surcharge le visuel, avec peu de composant on va donc voir une alternative que j’affectionne beaucoup. Vous allez importer StyleSheet, et on va l’utiliser de suite, vous allez créer une constante styles et appliquer la methode create() pour pouvoir générer votre objet et pouvoir définir les propriétés dont vous avez besoin.



Et on applique,



Et voilà c’est plus propre. Vous avez donc appris à externaliser votre style bravo !!

Ok on va avancer un peu maintenant que le style n’a plus de secret pour vous, vous pouvez soit commenter ce que l’on vient de faire soi l’effacer.

Vous allez créer une constant name et lui attribuer une valeur, et l’afficher. Parfait maintenant on va faire cela avec une fonction,



A vous de l’afficher !!

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Bravo à vous si vous avez réussi, vous avez invoqué la fonction.

**Les props en React Native**

Les props permettent de customiser vos composants React et React Native en passant des datas entre les composants.

**Exercice 1**

Correction exercice 1 :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

App.js

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Bien joué vous n’avez pas tout oublié, faisons-le avec le destructuring du coup.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Parfait, rajoutons une props age à cela, on va faire une condition pour le composant qui n’aurai pas d’âge précisé par exemple.

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, affichage, Police

Description générée automatiquement

Voilà tout fonctionne parfaitement.

**Le children en React Native**

Rappel : les props children sont des props définis entre les balises des composants.

**Exercice 2**

On va modifier légèrement notre code, et mettre nos children donc refermer nos composants, maintenant que c’est fait nous allons pouvoir passer des props children supplémentaires. Vous avez sans doute remarqué que ce sont des personnages du dessin animé American Dad, donc Roger est un alien, et Stan puis Steve sont des humains nous allons donc faire cela.

A vous de jouer !!! Voici la correction

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Et donc vous les passez au composant Person.js et les affichez :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Parfait pour ces petites révisions, les props et les childrens. Abordons le sujet des props on a bien créer des props nous-mêmes mais il y a également des props obligatoires. Prenons le cas d’une image que l’on n’a pas encore vu, car elle fait partie des composants spéciaux de React native, et bien vous aurez besoin d’un props source, c’est grâce a se props la que vous allez pouvoir accéder à l’url de l’image par exemple. Il faut que vous gardiez en tete que vous pouvez créer vos propre props avec le nom que vous désirez mais qu’il y a des props dont le nom ne peut pas être changé c’est comme ça.

**Le State en React Native**

Rien de nouveau par rapport à React c’est exactement la même chose, cela permet de créer quelque chose d’interactif ou de dynamique, vous aurez besoin d’un ou plusieurs états dans vos composants.

Un state c’est donc un moyen qui permet le stockage de 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 évènements.

Voyons le composant Button avec le state, il accepte un certain nombre de props :

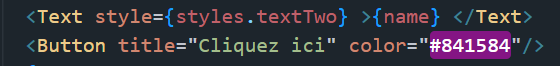
* title : pour afficher le texte du bouton
* color : pour lui appliquer une couleur de text sur ios et le background color pour android
* accessibilityLabel : pour lui attribuer un texte pour les fonctionnalités sur les supports destinés aux personnes en situation de handicap
* ou simplement pour accéder au Js pour activer en évènement comme onPress par exemple

Vous pouvez aller sur la docs et cliquer sur components et sélectionner le Button vous aurez des exemples.

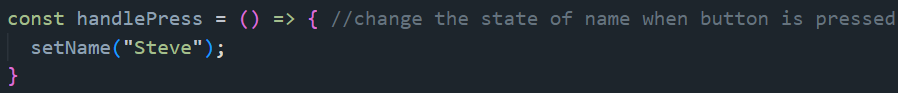
**Exercice 3**

Vous pouvez créer un composant Button, sur app.js et nous allons voir cela.





Voilà pour le début, tout est créer maintenant on a dit au click on va invoquer une fonction qui va modifier ce state. On va définir l’évènement onPress et appliquer cette méthode, je l’appelerai handlePress,



Et je l’applique au onPress, donc ensuite cela change la chaîne de caractère.

Rien ne vous interdit d’invoquer cette fonction directement après le onPress comme vu en react ou avec l’argument. Encore une fois on peut directement invoquer le setName cela fonctionnera de la même manière.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

**Exercice 3.2**

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

On aurai pu définir l’objet directement dans les parenthèses mais j’ai refait une const pour l’exemple, donc on passe l’objet en argument du useState, et avec l’annotation pointé j’affiche ce dont j’ai besoin.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Maintenant on s’occupe de l’évènement que le bouton et on aura terminé.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

On invoque la méthode dans la fonction onPress et voilà, vous pouvez également faire comme ceci

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

On en a fini pour le State.

**TextInput**

Le composant TextInput est un autre composant de la librairie React Native, il est lui aussi doté d’un certain nombre de props comme :

* auto-correction
* multiline
* maxlength
* auto-capitalization
* placeholder etc..

On peut également lui appliquer des évènements bien spécifiques : n’hesitez pas à regarder la doc.

* onChangeText
* onSubmitEditing
* onFocus
* onBlur
* onScroll etc..

Par exemple lors de la validation d’un form, vous pourriez demander au users de donner leur nom et là vous aurez besoin d’un champ input pour pouvoir capturer ces données.

Vous l’importez comme toujours et le mettez dans une view, il va falloir le styliser pour qu’il s’affiche bien évidemment.

On va lui donner un peu de style ainsi qu’au container, et créer un state,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Voilà pour le style, passons au state



Et on l’applique en props au value, essayer seul !!



Pfiouuu c’était dur eheh. Maintenant vous allez faire en sorte que quand vous changez cette info (la value), vous allez appliquer le changement au niveau du state afin de mettre à jour le texte qui lui-même va mettre à jour le value. Essayer seul c’est écrit dans le doc il suffit de trouver la bonne props de TextInput, réfléchissez un peu à l’action que vous voulez effectuer.



Et voilà pas plus difficile que cela, on a passé le text en argument, à chaque changement la fonction va s’enclencher et passer donc le text a la value grâce au setText. OK super maintenant vous allez faire en sorte de valider un text au niveau de ce champ (TextInput) et de l’afficher car écrire pour écrire ne sert pas à grand-chose.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le style je vous laisse choisir, on passe désormais le name et setName ainsi que la value c’est plus parlant je trouve.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Vous pouvez aller tester cela fonctionne très bien. Vous pouvez faire pareil pour l’âge allez-y.

Il suffit de tout copier sauf la view parent et c’est fait. Nous allons ajouter un petit truc sympa qui va nous permettre d’avoir directement les nombres sur le clavier du tel, c’est une props qui est dans la doc.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Une image contenant capture d’écran, gadget, Appareil électronique, Appareil mobile

Description générée automatiquement

Et voilà cela ce fait automatiquement au click sur le champ age. Bravo on en a fini avec cela.

**Affichage du contenu d’un objet avec méthode map**

Rappel : la méthode map() crée un nouveau tableau avec les résultats de l’appel d’une fonction fournie sur chaque éléments du tableau.

Comme toujours vous allez créer un state et également un array qui contiendra 6 objet avec id, name et age. A vous de jouer.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Ça s’est fait, maintenant que l’objet est créé vous allez le passer en valeur initial de votre state. Vous allez désormais faire en sorte d’afficher les membres de votre famille dans votre vue. Allez-y seul encore une fois, la pratique y’a rien de mieux. N’oubliez pas qu’en react pour la méthode map vous devez préciser la key.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ça fonctionne parfaitement mais on va ajouter du style pour un meilleur rendu visuel,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Et bien là vous devriez avoir un petit soucie, et oui votre dernier personnage ne se voit plus, comment remédier à cela ?? En sachant que c’est la config par défaut de React Native.

**Le ScrollView**

Vous allez donc utiliser un nouveau composant qui est le ScrollView, ce composant est plutôt simple, il vous permet d’effectuer un scroll afin d’accéder à la totalité de vos composants verticalement, il existe une alternative qui sera FlatList que nous verrons ensuite.

On n’oublie pas de l’importer évidemment, et une fois cela fait il suffit d’envelopper les différents éléments que vous retournez donc ici, juste avant vos accolades et la fermante juste après.



Une image contenant Police, capture d’écran, Graphique, graphisme

Description générée automatiquement

Donc de ce fait, à l’intérieur de cette view, vous aurez un scroll qui va vous permettre d’avoir une barre sur le côté droit afin de pouvoir visualiser l’ensemble de votre composant. Voyons donc son alternative la FlatList.

**La FlatList**

La FlatList est donc une alternative au ScrollView mais elle a une petite particularité, c’est qu’elle est bien plus performante que ScrollView, elle apporte moin de code et une bien meilleure optimisation, je vous explique cela. Dans le cas de votre appli il n’y a que 6 éléments (votre objet) donc cela ne fait rien mais si vous aviez 100 ou plus d’éléments, alors il vous chargerai tous les éléments en même temps et la cela deviendrai très lourd, il chargerai que ceux qui sont visible sur l’écran donc dans votre cas les 5 premiers et ainsi de suite, cool hein !!

Je vais commenter une grosse partie du code précédent pour que vous puissiez utiliser le flatList.



Et dans votre view il y aura cela a la place du code précédent,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Sauvegarder et allez voir le résultat. Il est possible qu’il vous mette un warning précisant que l’id est de type string si c’est le cas dans votre objet vous mettez l’id en apostrophe et cela enlèvera le warning. Et voilà vous avez vu la FlatList, vous avez pu vous rendre compte de l’optimisation du code et en même temp de votre application.

Nous avons déjà vu pas mal de composant, le View, Text, le Button, le TextInput vous savez également comment styliser vos composants, les states et sa gestion des données ainsi que les évènements, vous allez utiliser ses connaissances avec un mini projet.

On l’appellera shopping list ou comme bon vous semble, on va faire en sorte d’avoir un input qui indique l’article à acheter au niveau d’un TextInput, de valider le nom de cet article et l’enregistrer dans un array et bien sûr d’afficher l’ensemble des éléments du tableau.

**Exercice 4**

Voilà vous avez initialisé votre application et nettoyer votre espace de travail, vous allez pouvoir commencer cette application. Vous allez pouvoir créer un nasted components View et un Text à l’intérieur qui dira bonjour afin d’avoir un premier rendu visuel. Vous allez mettre en props Css sur container :

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Pour la seconde view, vous remplacerez le Text par un TextInput, et on stylise cela (à votre convenance le style je vous laisse gérer à votre goût).

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Vous ajouterez également un Button à cette seconde View,

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Voilà qui est beaucoup mieux. Pour le moment votre TextInput n’a que du style mais ajoutez lui d’autre props comme le placeholder, ensuite vous allez passer à l’évènement onChangeText, mais tout d’abord il faut penser au state, alors c’est parti, à vous de jouer.



Et donc l’évènement,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Et l’affichage bien sûr,

Une image contenant texte, Police, capture d’écran

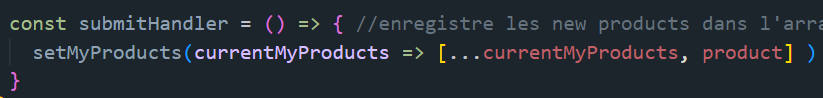
Description générée automatiquement

Maintenant vous avez donc la possibilité d’interagir avec votre state et modifier son état initial, il reste à prendre cette valeur là et l’enregistrer dans un autre state, en fait vous aurez besoin d’un autre state dans lequel vous ne passerez pas une chaine de caractère mais un array. Et c’est dans cet array la que vous allez enregistrer les nouveaux produits. Donc on fait un state comme toujours, mais qui contient un array,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

On va créer un évènement sur le bouton qui va vous permettre d’invoquer une fonction, onPress car on est sur mobile et on crée la fonction.



Quand vous lancer le submitHandler, l’objectif c’est de prendre l’info validée au niveau du input pour l’injecter dans l’array (myProducts), sauf que vous souhaitez garder les infos existantes parceque peut-être être vous avez déjà enregistré des infos et vous allez devoir les récupérer. Pour cela vous avez fait une callBack function et en tant qu’argument de celle-ci vous avez récupérer tous les éventuels produits existant dans l’array et ensuite vous allez rajouter le produit que vous venez de valider. Tout fonctionne bien, maintenant vous allez mettre en place le fait que votre dernier produit reste dans l’input et on va faire en sorte de l’effacer ce n’est pas joli, et c’est très simple. Vous prenez votre setter et vous mettez une chaine de caractère vide et voilà le tour est joué.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Ca ça va donc permettre de vider l’info de l’input, parceque c’est product(la value) du textInput. Maintenant on va donc récupérer l’ensemble des éléments du array (state MyProducts) pour les afficher sous le bouton. Vous allez donc préparer une autre view, afin de les séparer.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Et vous allez donc faire la méthode map sur votre array myProducts,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

On recupere donc les products grâce à leur index que l’on precise en key, et évidemment on affiche les products dans le Text, et j’ai ajouté du style dont une propriété que vous ne connaissez pas car elle n’existe pas en Css classique.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Super vous pouvez ajouter des produis à votre liste et ils s’affichent on est bien parti, continuons sur cette lancée. En revanche si vous ajoutez plusieurs produits vous risquez de ne pas pouvoir tous les voir ce qui serai dommage nous avons vu un composant de la librairie React native plus tôt dans le cours alors a vous de remédier à ce problème seul.

Super c’était bien le ScrollView, vous avez donc entouré votre View qui affiche vos items et vous avez bien un Scroll, petit détail mais au scroll vous avez vos items qui viennent toucher votre input et c’est moche vous allez donc rajouter un marginBottom a votre container de l’input et cela sera bcp plus agréable.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Super c’est beaucoup plus agréable maintenant, vous allez passer à la prochaine étape. Encore une fois là vous n’avez pas beaucoup d’élément mais si vous en aviez une centaine disons, vous vous retrouverez avec un soucie d’optimisation vous allez donc modifier votre appli avec un autre composant que nous avons déjà vu rappelez-vous.

C‘est bien le composant FlatList, alors corrigeons cela ensemble, on importe le composant évidemment, puis on le met en place en oubliant pas ses props obligatoire. On va préparer le key déjà, puis ajouter le product.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici vous avez générer un objet à la volée que vous injecter dans l’array currentMyProducts et se retrouver en haut de la liste donc en première position.



Et ensuite on va placer la FlatList,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

A partir de cela vous pouvez effacer le ScrollView ou le commenter, et allez tester de suite le résultat, parfait tout fonctionne très bien super. Bravo à vous.

Petit Rappel de ce que nous avons fait :

* Indiquer l’article à acheter au niveau du TextInput, on l’injecte ensuite dans un objet afin de lui attribuer la propriété **key** parceque vous en aurez besoin au niveau du FlatList.
* Valider le nom de l’article et l’enregistrer dans un array
* Afficher l’ensemble des éléments enregistrés

**Exercice 5**

Je vais vous montrer la base car il y a quelque modification à faire sur App.js et donc sur Products à ajouter. J’ai mis ce style ici et donc je l’ai enlevé de App.js, ainsi que le Text.



Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Pour le reste rien de spécial vous connaissez déjà, sauf que dans ce composant là, vous n’avez pas accès à ces informations et c’est là que les props vont vous servir. Vous importez le composant Product au niveau de la FlatList évidemment.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Vous allez donc devoir véhiculer le name de l’item via un props on l’appellera name pour faire simple.



Maintenant que vous avez passé ce props au niveau du composant product, on va donc pouvoir y accéder sur le composant Products en tant qu’argument et on l’affiche

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Parfait tout fonctionne très bien, super on va passer à la création d’un nouveau composant pour l’input et le bouton. Vous allez devoir faire en sorte d’importer la fonction inputHandler ainsi que le state dans votre nouveau composant. Et vous allez devoir faire communiquer la fonction submitHandler avec votre nouveau composant également.

Alors créer le nouveau composant que j’appellerai AddProduct vu que c’est son rôle, je fais toujours simple et compréhensif au premier regard. Vous allez donc copier ce bout de code dedans et l’effacer de App.js

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Et à la place vous importerez votre composant,

Une image contenant texte, Police, capture d’écran, conception

Description générée automatiquement

Ensuite pensez à importer les styles nécessaires, vous aurez également besoin du state product donc vous le mettez dans le new component et vous l’enlever de App.js, ensuite il vous faudra également la fonction inputHandler donc on fait la même opération,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**Petit récap :** Le TextInput invoque la fonction inputHandler qui elle va mettre à jour product via le setter et ensuite on affiche le product dans le value.

Ok il reste le bouton, au click on invoque également une fonction on aurai pu aussi la prendre, sauf que vous allez en avoir besoin avec le deuxième state qui va vous permettre justement d’afficher ses infos au niveau de FlatList. Donc vous allez maintenir le state sur App.js ainsi que ladite fonction. Vous allez donc devoir faire passer cette fonction via un props pour y accéder dans votre composant. Je vais appeler le props du même nom que la fonction personnellement.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Et voilà grâce au destructuring on peut y accéder facilement,



Vous y avez désormais accès sur le onPress. Une fois celle-ci invoqué elle aussi va invoquer le setProduct pour lui passer le produit en question. Maintenant il faut véhiculer cette info vers l’élément parent évidemment, comment faire ??

C’est très très simple, sachant que vous avez une autre fonction(submitHandler) qui vous permet de capturer un argument, et bien on va s’en servir. Je vais l’appeler product parceque c’est ce nom là que j’ai utiliser pour alimenter la propriété name,

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Ensuite il reste plus qu’à l’ajouter en argument de la fonction dans le button,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

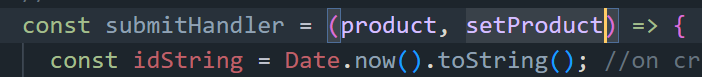
Maintenant que vous passez cette info à la fonction submitHandler, vous allez pouvoir récupérer dans App.js et l’appliquer à l’objet



Que vous injectiez éventuellement existant au niveau du state myProducts. Vous pouvez enregistrez et tester votre application.

Ok maintenant on a un petit soucie si vous avez remarqué ?? Et bien le setter (setProduct), du state dans le composant enfant on ne peut donc pas l’invoquer dans le composant parent (App.js), de manière à pouvoir vider la const product. Donc ce que vous allez faire et bien c’est l’inverse de ce que vous avez fait précédemment, on va passer du composant enfant vers le parent cette fois-ci.

Rendez-vous sur AddProduct, vous avez passez un argument mais on peut forcément en rajoutez un autre, allez l’ajouter sur App.js(le setter en argument),



Là vous allez l’ajouter en second argument sur AddProduct également,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Super, voilà tout fonctionne parfaitement ce n’était pas très compliqué, et vous avez résolu le problème. Mais on pourra gagner en optimisation vous allez modifier légèrement le code afin de faire cela d’une autre manière, et oui on modifie encore. Vous allez créer une fonction handleClick dans le AddProduct.js qui va gérer cela,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

On rappelle désormais celle-ci dans le button,

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Ensuite sur App.js, vous devez mettre à jour le submitHandler,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Le setter étant directement géré dans le composant enfant il n’y a plus besoin de la laisser ici. Vous pouvez retester tout cela, c’est cool tout fonctionne bien.

**Rappel :** Vous avez fait trois composants, un parent (App.js) dans lequel vous avez importé deux composants enfants, Product vous permet d’ afficher tous les produits que vous avez au niveau de votre state, et AddProduct qui vous permet d’afficher le TextInput et le Button donc là vous avez une appli qui fonctionne parfaitement bien, alors poussons un peu plus loin et découvrons de nouveaux composants.

Maintenant on va faire en sorte de pouvoir effacer un produit de votre liste en cliquant dessus, comment faire cela ??

En utilisant les Touchable et Pressable, sur la doc ils expliquent bien cela, ils permettent entre autres, de détecter les différentes étapes d’évènements ‘onPress’ effectués sur ses composants enfants, grâce à ces composants, vous pouvez transformer d’autres composants comme View ou Text par exemple en composants cliquables comme des boutons.

Découvrons le TouchableOpacity, pour commencer vous l’importer et c’est un composant recat-native je précise, on va l’essayer sur Product.js, et vous envelopper la View.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Vous pouvez mettre un console log pour voir son fonctionnement si besoin,

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Le console log peut s’écrire avec le raccourci clg, c’est cadeau 😊. Ce composant peut être styliser indépendamment de ses enfants, et il contient un bon nombre de props que vous voyez sur la droite dans la doc, on va essayer le activeOpacity,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

J’aime bien celui-ci il a un effet assez intéressant et vous l’avez déjà vu j’en suis certain. Regardons maintenant le TouchableHighlight qui est intéressant également, donc même opération que pour le précédent, nous allons juste lui ajouter une props de plus que le précédent, qui sera underlayColor. Si vous souhaiter avoir un effet sympa, garder le en tête également, sachez juste que si vous ne mettez pas la props underlayColor il y aura une couleur noir d’office et c’est moche. Ok passons au Pressable, et je vous encourage vivement à aller voir sur la doc ses props car il est très intéressant. Testons le ensemble, toujours de la même manière, vous pouvez garder la props onPress, je vais lui mettre un style avec une condition,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

N’hésitez pas à en tester plusieurs, j’en ai une que j’affectionne à vous de chercher celle qui peut vous plaire. Ok vous avez donc vu de nouveau composant, vous allez désormais passez à la suppression d’un item, on va garder le Pressable. Vous allez pouvoir enlever le console.log et vous allez plutôt invoquer une fonction que vous allez créer sur le App.js. Pour supprimer un item bien spécifique vous allez devoir vous baser sur son key,

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Maintenant que vous avez créer la fonction, il faudra la passer en tant que props au composant Product sur le App.js toujours,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

J’ai juste réorganisé afin que ce soit plus clair et lisible, ensuite sur le Product.js on l’applique via le destructuring, et j’ai créé la props idString car vous allez avoir besoin d’accéder au key dans Product.js, en l’ajoutant au destructuring également. Et donc j’applique cela sur le onPress, et voilà.

Une image contenant texte, Police, Graphique, capture d’écran

Description générée automatiquement

Allez tester votre application, tout fonctionne, bravo. On a encore un petit soucie si vous cliquer sur le bouton avec un produit vide que se passe-t-il ?? Et oui il génère un produit vide et ce n’est pas super, on va partir sur un petit scénario et forcer l’utilisateur à donner une chaine de caractère minimum sinon il aura une petite alerte. Alors il va falloir vérifier lors de la validation du bouton il devra y avoir une chaine de caractère minimum. Donc vous allez devoir mettre une condition dans le submiHandler,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Allez tester cela sur votre appli, super ça marche, maintenant au lieu d’afficher un alert classique je vous invite à découvrir le alert de react-native, il vous permet de générer une boite de dialogue d’alerte avec, entre autres, un titre, et un message customisable, une liste de boutons sur lesquels vous pouvez appliquer des évènements onPress par exemple. Il va falloir penser à l’importer bien sûr, donc sur App.js car c’est là que se trouve la condition avec le alert.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Toutes les options sont notées sur la doc, on part sur l’idée de l’android donc vous êtes limité à 3 boutons, ios aucune limite, je vais vous montrer une autre options sympa,

Une image contenant Police, texte, capture d’écran, Graphique

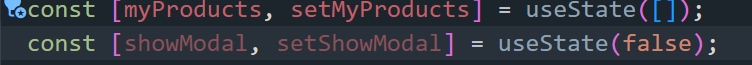
Description générée automatiquement

Cela permet de cliquer n’importe ou pour enlever l’alerte. OK découvrons ensemble le modal, donc il faut pour cela l’importer de react-native dans le App.js, (dans la doc il est sur others), vous allez le mettre juste avant le composant AddProduct, et vous mettez dans le modal un Text.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Vous verrez que le modal s’affiche automatiquement, il faudra ajouter la props visible pour gérer cela, et voilà il a disparu, on peut jouer sur ce visible et ce booléen on peut le travailler au niveau d’un state alors allez-y.



Maintenant vous pouvez modifier le visible en mettant le state,

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Si vous enregistrez vous n’aurez pas votre modal et pour pouvoir l’afficher vous allez devoir gérer le setter en true, cela se passe dans la condition else du submitHandler bien sùr là ou vous aviez l’alert.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Sur votre appli il ne s’affiche pas mais vous essayez de passer une chaine de caractère vide et bien le modal s’affiche, super mais désormais on va devoir faire en sorte de pouvoir fermer ce modal et gérer son style et son contenu. Occupons-nous de sa fermeture il y a un props qui gère cela, onRequestClose qui va permettre d’enclencher un évènement.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Ok on va pouvoir styliser un peu le modal, vous pouvez enlevez le Text et remplacer par une View sur laquelle vous appliquerez un style,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Parfait vous avez gérer le container, dans ce dernier vous allez ajouter une view que vous allez également styliser,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Voilà pour le début, vous allez ajouter une nouvelle view pour le header de la modal, et on va encore styliser cela, ainsi qu’un Text a l’intérieur,

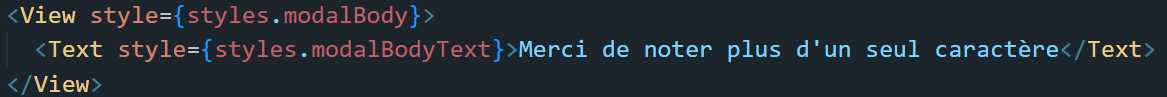
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Voici pour le header, passons au body, vous pouvez copier collez et modifier la classe,



Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Et ensuite bien évidemment le footer qui sera cliquable,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Voici pour la mise en place de la modale, passez au Pressable à présent, et ajouter lui la props onPress, son rôle est donc de prendre notre modal(setShowModal) et de pouvoir le remettre en false pour la refermer. Pour cela vous avez le setter, donc à vous de jouer.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Ok parfait votre modal s’ouvre et se ferme mais un peu brusquement, vous allez pouvoir ajouter une petite animation afin de rendre cela plus fluide. Pour cela vous allez retourner au niveau du modal, et ajouter la props animationType qui se trouve elle aussi dans la doc, elle a trois option :

* Fade
* Slide
* None

Je vais mettre slide personnellement,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Super bravo votre application est terminée, maintenant je vous propose certaine extensions pour visualiser les couleurs dans votre code css, color highLight ou colorInfo.

Voyons à présent comment insérer des images en react native, par exemple un petit logo dans notre modal, Image est un composant react-native qui permet d’afficher différents types d’image, les formats pris en charge sont nombreux, png, jpg, jpeg, gif, webp et bien plus encore. Je vais créer un dossier image dans le dossier assets et récupérer une icone d’une croix sur iconeFinder. Rendez-vous ensuite dans le App.js, car c’est ici que se trouve le modal, et juste avant le texte qui indique plus d’un caractère vous allez placer l’image.

Pour afficher l’image vous aurez besoin du props source, ainsi que la méthode require et son url,

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Ensuite vous pouvez allez tester votre appli, super cela s’affiche bien mais il va falloir modifier un peu pour que cela ne pousse pas le texte vers le bas, il suffit de mettre la height de modalContent a 300 et c’est parfait.

Désormais nous allons voir le composant ImageBackground vous avez compris son utilité, vous vous rendez sur App.js, vous allez remplacer la View principale par le composant ImageBackground,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

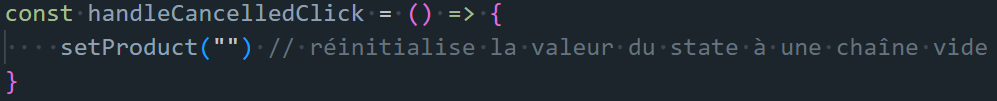
Ainsi que sa fermeture bien évidemment, maintenant vous devez lui appliquer le props source si elle vient d’internet il vous faudra mettre son url sinon c’est la même opération que précédemment,

**Exercice 6**

**Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement**

Voici le second bouton ainsi que sa méthode,



OK vous avez la base pour ensuite créer votre component et comprendre son utilité, vous pourriez créer tout un tas de bouton un peu partout dans votre appli et répéter le code puis les styliser, il suffit donc de créer un component button et de le rappeler la où vous en avez besoin ce serez beaucoup plus simple et clair, allons-y.

Vous pouviez taper rnfes, pour avoir la structure ainsi que le style déjà prêt et oui encore un petit tips,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Voici le style de base,

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Voilà votre composant de base créer, grâce au props vous allez pouvoir accéder au texte que vous souhaiter afficher pour votre bouton. Voilà avec l’aspect apparent de votre bouton, vous allez pouvoir l’importer dans le AddProduct.js. Vous avez donc deux boutons dans le App.js et on va commencer par le bouton Valider, qui est dans la View btnBlue et bien vous l’effacer, et à la place vous afficher le composant button que vous venez de créer.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Vous allez le répéter un seconde fois car vous avez deux boutons, dans le second pensez a mettre le onPress en props, et faites de même pour le premier bouton. Ensuite mettez le texte de votre bouton,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Maintenant que vous avez généré vos deux boutons, il va falloir pouvoir cliquer dessus, car pour le moment dans ButtonComponent vous n’avez qu’une vue avec du texte. Vous avez vu comment faire précédemment, comment rendre une vue cliquable ???

Pressable et oui !! Donc allez-y, et on n’oublie pas les props du pressable car vous devez pouvoir cliquer je le rappelle. Et donc on donne la props onPressHandler au onPress,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ok vous avez donc vos boutons qui fonctionnent mais qui n’on pas de style donc vous allez passer au style maintenant, ok est ce que ça marche ?? Et bien non car vous venez d’ajouter une props qui elle n’est pas encore exploité au niveau de votre ButtonComponent, vous allez donc en quelque sorte faire un merge de votre second style qui va venir écraser le premier, vous allez avoir besoin de transformer le style en objet et de du spread operator. A vous de jouer !!

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour le style de vos boutons je vous laisse décider, voici pour un custom component styliser, bien joué,mais ce n’est pas encore fini, on va s’occuper du children tant qu’à faire. On va modifier légèrement le code,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Et donc maintenant évidemment on doit modifier légèrement les boutons,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Voilà désormais ces derniers vont être accessibles via le props children. Un léger détail me chagrine et vous ???

On ne voit pas que le click ou onPress s’effectue c’est un peu bizarre, vous allez donc remédier à cela et remplaçant le Pressable par un TouchableOpacity, suite a cela vous pouvez ajoutez le props que l’on a vu plus tôt le activeOpacity,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Voilà qui est beaucoup mieux. OK maintenant que vous avez vu les custom components, vous allez avancer sur cette application.

**Exercice 7**

Voici le mien j’ai fait simple,

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

En revanche dans le App.js il va falloir modifier votre code car la balise parent ne peut pas être ImageBackground sinon le header ne serai pas a sa place. Je vais juste modifier le style de ce composant et modifier le code par la suite.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Avec juste un flex :1, en propriété, et ensuite vous allez pouvoir importer votre Header,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Ensuite tout ce qui suit ira dans une autre View, qui aura la classe container du début,

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Voilà qui est mieux en revanche le paddingTop n’est plus nécessaire, vous pouvez donc l’enlever si tout s’est bien passé. Si besoin de quelques ajustement n’hésitez pas vous savez gérer le Css seul. Bravo là on a vraiment bien avancé, profitons-en pour voir les constantes de couleurs. On commence à répéter certaines couleurs dans différents composants, imaginez que vous décidiez de changer le thème de votre appli, vous serez donc obligé d’aller changer individuellement chaque couleur et pour éviter cette situation et bien opter pour les constantes dans lesquelles vous aurez toutes vos couleurs, et la vous n’auriez besoin de mettre à jour que celle-ci. Sachez que bien évidemment les constantes en sont pas réservé qu’aux couleurs, elles pourraient gérer les fonts, pour le positionnement, l’espacement etc etc.. Vous allez créer un nouveau dossier a la racine du projet appelé constants, je vais pour l’exemple commencer sur AddProduct, dans ce dossier je vais créer un fichier nommé colors.js, et vous allez définir un objet Javascript dans lequel vous aurez les différentes couleurs.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Et voilà pour mon fichier colors.js, et maintenant on peut l’utiliser partout dans notre application grâce à l’import,





Et le style est donné par les constantes définis précédemment. Super tout est bon. Abordons les fonts désormais, il y a un package Expo google fonts, qui permet d’obtenir facilement autant de police de google fonts que possible, ces packages sont open-source et compatibles Ios et Android. Voici la ligne à taper pour l’installer, il est fort probable que vous ayez besoin de AppLoading a vous de chercher comment l’installer : (AppLoading etatn deprecie vous pouvez regardez spalshScreen, mais AppLoading reste utilisable bien sur.

* expo install @expo-google-fonts/nom de la font, ainsi que la ligne d’import,
* import {useFonts, PACKAGES} from ‘@expo-google-fonts/nom de la font

vous pouvez vous rendre sur la doc d’expo, et taper font, sur le droite vous avez useFonts et comment l’utiliser.Voici la piste des fonts avec leur packages : <https://github.com/expo/google-fonts/blob/master/GALLERY.md#readme>

Donc vous ouvrez un second terminal, après avoir récupérer le packages et on install donc pour ma part j’ai choisi celle-ci, **expo install @expo-google-fonts/shantell-sans**

On peut vérifier dans le package.json et elle est bien apparu



Vous vous rendez sur App.js, et on importe, c’est la ligne qui est plus haut dans le cours. Pour connaitre les packages qui se trouvent dans la font, il faut regarder dans la doc. Pour moi c’est celui-ci



Je vais mettre une petite condition pour le chargement ou si cela ne charge pas au cas où,

Une image contenant texte, Police, capture d’écran, écriture manuscrite

Description générée automatiquement

Ensuite on crée le state de la font,

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Ensuite vous prenez votre font, et vous vous rendez sur Header.js, et vous l’appliquer a votre header dans le style,

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Vous sauvegardez et allez vérifier le résultat, (pour ceux qui ont des problèmes de fichiers SDK si vous avez le message bien sur voici l’endroit où trouver ce qu’il vous faut <https://expo.dev/go?sdkVersion=50&platform=android&device=false>)

Voici pour les fonts, ce n’est pas bien compliqué.

Voyons les icones maintenant, vous allez utiliser le package @expo/vector-icons qui est un package d’expo qui vous permet d’obtenir facilement des icones pour les utiliser dans vos appli react-native, cette librairie fait partie du package expo donc elle est installé par défaut si vous avez initialisé votre projet via expo init. Une fois importée elle vous permezt d’accéder à plusieurs ressources et librairies proposant des icons comme Font-awesome, Glyphicons, Ionicons, AntDesign etc..

Vous pouvez vous rendre sur la doc d’expo et taper icons dans la searchBar, vous aurez la partie vector-Icons, allez sur la partie librairie direct et choisissez le filter, vous verrez il vous ajoutera certaines parties qui sont cachées.

Vous allez ajouter un icones pour préciser à l’utilisateur qu’il peut effacer son produit en cliquant dessus car pour le moment ce n’est pas très clair, je vais choisir une petite croix personnellement, donc je vais taper remove et je prendrais celui de fontAwesome.

Vous pouvez copiez la ligne de la doc expo pour l’import, on le mettra dans quel composant ?

Celui qui contient le produit, donc Product,



Et donc on affiche ce composant dans votre view, en précisant ce que l’on veut comme props, rappelez-vous la doc, le name, la size et éventuellement la color.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Maintenant vous allez faire en sorte que cela s’affiche sur la gauche du texte, a vous de jouer. Voilà pour moi,

Une image contenant texte, capture d’écran, Monde, espace

Description générée automatiquement

Et bien là on peut se dire que l’appli est terminée définitivement, alors bravo a vous tous vous avez votre première appli react-native qui fonctionne parfaitement.