React Native

NovaldiDemoApp Procédure d'utilisation

11 Août 2021

Table des matières

1 Language et Framework

1.1 TypeScript & React Native

Tout le projet se base sur la framework mobile React Native qui permet d'utiliser React (site web) et de créer du code native pour iOS et Android simultanément.
Contrairement à React qui manipule le DOM directement avec des balises HTML, React Native consiste en une superposition de composants. Il existe des composants de base (<Text>,<Image>,etc), mais aussi une multitude de composants OpenSource disponible sur Internet.

React Native consiste donc à créer des composants indépendants et réutilisables qui s'affichent les uns par dessus les autres.
On peut ensuite appliquer une feuille de style semblable au CSS pour chaque composant. Le lien OpenClassRoom suivant renvoie sur une bonne notice explicative de comment marche les composants React Native (https://openclassrooms.com/fr/courses/4902061-developpez-une-application-mobile-react-native/4915611-faites-vos-premiers-pas-avec-les-components-react).

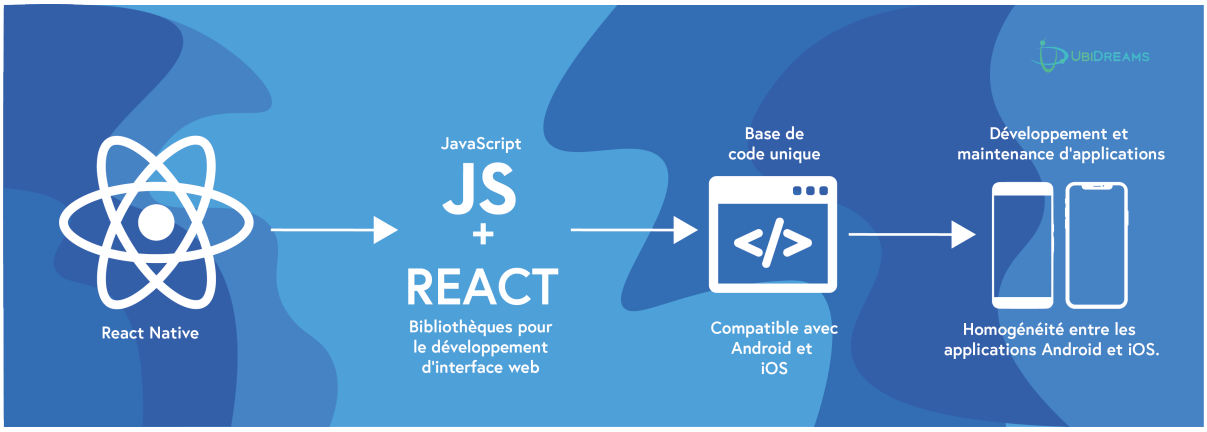

FIGURE 1 – Schéma React Native

1.2 Gestionnaire de paquet - Yarn

Yarn est le gestionnaire de paquet utilsé. Il permet une plus grande rapidité que NPM nottamment sur l'installation de nouveaux modules. Ainsi, toute la gestion des modules ou encore le lancement du serveur et des simulateurs iOS/android se feront avec les commandes **yarn**.

On retrouve pour lancer le serveur et vider le cache : **yarn start**.
Pour lancer le projet et le serveur sur iOS/android : **yarn ios** ou **yarn android**.
Pour ouvrir le contenu des fichiers iOS sur XCode : **yarn xcode**.
Pour installer tous les modules nécessaires si le projet vient d'être importé : **yarn install**.

2 Architecture du projet

Le projet possède jusqu'à présent 4 fonctionnalités : **'Agenda'**,**'Actualités'**,**'Propreté/Déchets'** et **'Autour de Moi'**.
L'écran d'accueil est géré par les composants **'Home'** qui regroupe l'accès à ces 4 fonctionnalités et enfin, l'écran de chargemenet est géré par les composants **'Splash'**.

Le projet est organisé ainsi :

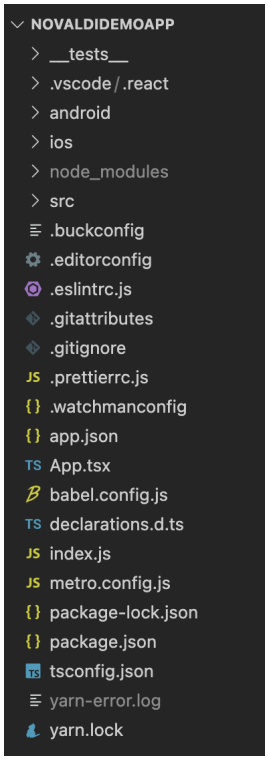

FIGURE 2 – Arborescence du Projet

2.1 Android & iOS

Les dossiers **android** et **iOS** contiennent tous les fichiers de configuration propres à leur système d'exploitation. On y retrouve par exemple les demandes d'accessibilité au micro, à la localisation, ...
La lecture et l'écriture dans les fichiers android peut se faire directement depuis l'environnement. En revanche, pour manipuler les fichiers iOS, il faut passer par XCode (plateforme d'Apple).
La commande depuis le terminal **yarn xcode** ouvre directement l'espace de travail.

2.2 node_modules

Il s'agit de tous les modules installés contenant des aides pour le développement, des composants Open-Source importés (react-native-maps par exemple). Il est important après avoir récupéré le travail d'un ordinateur distant d'utiliser la commande **yarn install** pour installer tous les modules nécéssaires.

2.3 src

Ce dossier contient l'ensemble des différents composants créés. Voir prochaine section.

2.4 Autres

Tous les fichiers restants concernent la configuration du projet.
On retrouve les fichiers git pour travailler depuis le repository local de Novaldi.

Sur le projet a aussi été installé deux outils, **eslint** et **prettier** qui permettent d'analyser le code écrit et de proposer des simplifications ou des corrections en fonction de certaines règles.

Babel est un compilateur de code JS configurable depuis babel.config.js.

Le packager/bundler **Metro** sert à convertir à l'éxécution tout le code JS/JSX en un seul fichier lisible par l'ordinateur.

Enfin, le package.json répertorie tous les modules installés dans le **node_modules**.

3 Structure du code source

3.1 Dossier config

Ce dossier regroupe les configurations d'utilisation, c'est l'interface pour changer les liens, les langues ou les horaires de l'application.

Le dossier locales comporte un index qui détecte la langue de l'appareil utilisé (pour le développement, mis par défaut sur fr) et qui renvoie sur quel fichier utilisé pour le language. En effet, chaque zone texte du projet ne comporte pas directement du texte. On fait appel à I18n.t('objet') qui pointe sur notre objet créé dans fr.js par exemple. Et c'est ici que l'on définie le texte à mettre.

Les fichiers config.js et Constants.js regroupent tous les liens utilisés pour lire les flux.

3.2 Dossier shared

Comme React Native permet l'utilisation de composants réutilisable, il est possible de créer des composants assez général pour être utiliser dans plusieurs parties du projet. On le place alors dans ce dossier partagé/components.

Il y est aussi stocké les **'assets'**, c'est-à-dire toutes les icônes ou images utilisées dans tout le projet.
On retrouve aussi le dossier theme qui comporte nos variables de taille ou de famille de fonts et aussi les couleurs utilisées.

Le dossier **'permissions'** enfin, regroupe les fichiers de vérification des permissions de localisation ou de l'appareil photo par exemple.

3.3 Dossier features

Il est d'usage pour chaque fenêtre ou fonctionnalité différente d'avoir un dossier nommé pour. On y retrouve alors un premier composant XXXScreen.tsx qui correspond à la fenêtre affichée sur le téléphone. On peut y ajouter alors une série de composants classés dans un dossier **components**. Si on souhaite afficher des jeux de données, un Store sera alors aussi ajouté qui permet stocker toutes nos données et de les rendre accessibles à tout endroit de l'application en les injectant.

3.3.1 Chargement - Splash

L'écran de chargement est pour le moment très sommaire. Il affiche juste pendant 3 secondes NovaldiDemoApp puis redirige sur le menu d'accueil.

Il sera possible d'y ajouter une animation ou une personnalisation dédiée à la ville.

3.3.2 Accueil - Home

Le dossier home est structuré de la sorte :

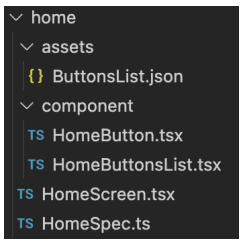

FIGURE 3 – Structure home

L'écran d'accueil comporte d'abord un Slider affichant les 3 dernières actualités.
Ensuite, il y a l'ensemble des boutons permettant de naviguer sur les différentes fonctionnalités.

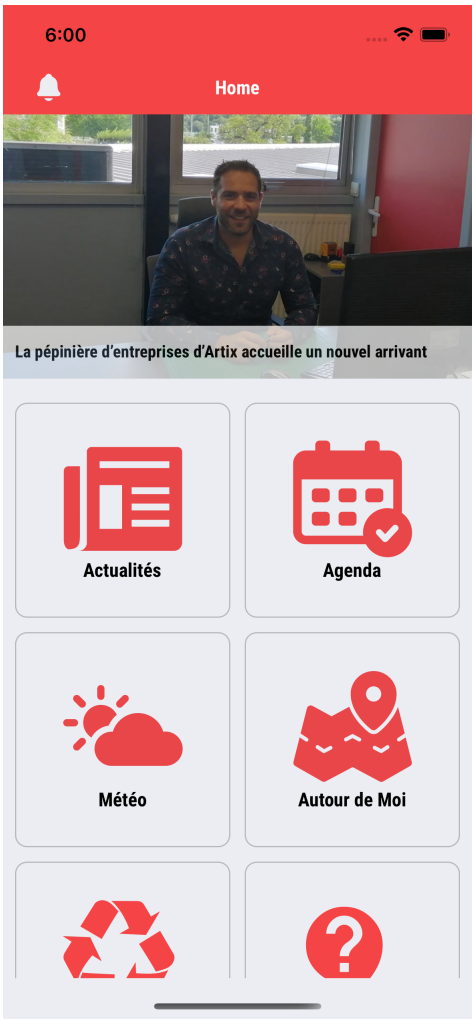

FIGURE 4 – Structure home

L'espace des boutons est prévu pour être très configurable. Il est divisé en 31 cellules en largeur. La liste des boutons est alors renseigné dans le **ButtonsList.json** avec leur nombre de cellules de longueur et largeur, leur icône ou image de fond, etc.

3.3.3 Actualités - News

Les actualités ont été prises sur le flux RSS de la CCLO. Une première fenêtre **NewsListScreen.tsx** montre la liste de toutes les actualités, et chaque actualité présée dirige vers **NewsDetailScreen.tsx**. Le contenu du flux RSS étant un bloc HTML, la lecture se fait grâce au composant <RenderHtml> qu'il faudra remplacer si le contenu des articles d'un autre flux n'est pas fourni en balise HTML.

Le composant **NewsSlideShow.tsx** utilise le Slider créé dans le dossier Shared pour l'affichage sur le menu d'Accueil des dernières actualités.

3.3.4 Agenda

Les évènements sont pris depuis le flux JSON de Saint-Jean-De-Luz sur Tourinsoft. Il est possible de les filtrer d'abord par la date (par défault ne sont affichés que les évènements des 7 prochains jours) puis par catégorie en cliquant sur le filtre en haut à droite. Les catégories correspondent à un type créé en particulier (Cateogory.ts) qui prend le nom de la catégorie et si celle-ci est demandée pour le filtre.

L'écran de l'agenda est divisé en deux parties recto-verso, la liste des évènements comme pour les actualités et la map qui représente la localisation de chaque évènement.

FIGURE 5 – Agenda Liste

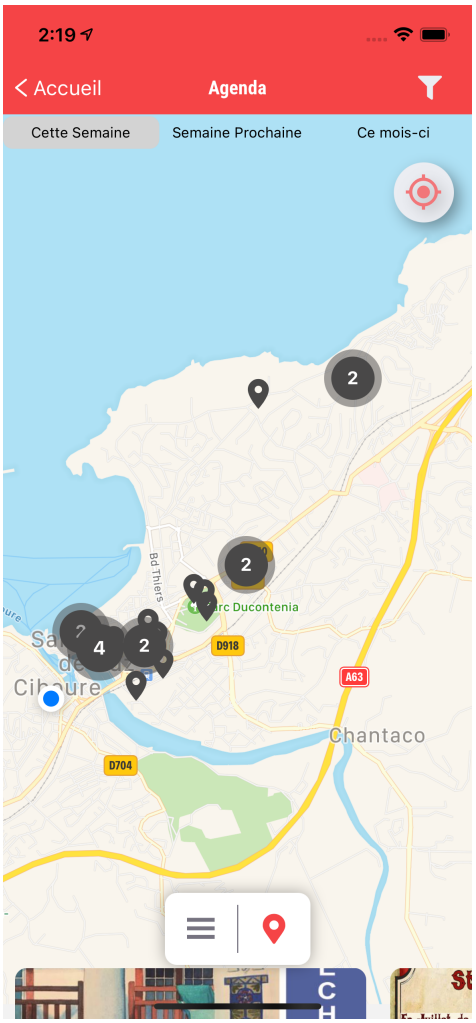

FIGURE 6 – Agenda Map

Pour la carte, il y a un rappel de la liste d'évènements en bas de l'écran qui est disponible en scrollant vers le haut. Les markers sont alors directement reliés à cette liste : en cliquant sur un marker, cela amène directement à l'évènement lié et vice-versa.
Cela passe par l'usage de composants 'Interactable' et 'Animated' qui permettent de gérer les animations sur l'écran.

Aussi, react-native-map-clustering est utilisé pour utiliser le composant <MapView> de react-native-maps avec en propriété supplémentaire, la possibilité ou non de créer des clusters. Si beaucoup d'évènements sont rapprochés dans une ville, cela permet d'éviter l'amas de markers.

3.3.5 Propreté/Déchets - Recycling

Pour cette section, le flux pris est celui des Points de Tri de Sitcom40. Il s'agit d'une carte pointant tous les centres de tris (RecyclingCenter.ts). Il y a aussi un filtre s'activant ou non en fonction de ce que l'on veut pouvoir jetter.

Il faudra sûrement modifier la structure RecyclingCenter si il s'agit pour un autre flux de déchetterie avec des horaires d'ouverture / fermeture.

4 Personnalisation

Le projet a été construit de telle sorte qu'il soit rapidement et facilement personnalisable pour ce qu'il s'agit des couleurs, des logos/icônes, des fonts, etc.

4.1 Changement des fonts

Les tailles d'écritures sont prédéfinies dans le fichier /src/shared/theme/sizes.ts. Il est possible de rajouter une variable pour une nouvelle taille ou de juste modifier la taille des variables déjà existantes.

Pour la police, elle est pour le moment la même que sur les maquettes fournies : RobotoCondensed. Si besoin est de la changer, il faut alors télécharger la nouvelle police (en regular, bold, italic, ...), puis l'ajouter aux polices lisibles par iOS et Android (voir paragraphe suivant), et enfin, renommer les variables du fichier /src/shared/theme/fonts.ts avec celles voulues.

Pour intégrer de nouvelles polices à iOS, il faut lancer le projet depuis XCode (yarn xcode depuis le terminal ou à la main). Il faut ensuite s'assurer de placer les fonts dans le dossier 'fonts' puis dans NovaldiDemoApp/Info.plist, à la rubrique 'Fonts provided by application', il faut ajouter un à un chacun des fichiers de fonts téléchargés.

4.2 Modification d'un composant

Il est possible de modifier les composants déjà créés. S'il s'agit d'en modifier l'aperçu, il suffit de se déplacer en bas de chaque fichier .tsx et d'y modifier le style CSS correspondant.
Cependant, les composants étant généralement imbriqués, certaines modification de taille ou d'emplacement peuvent entraîner des modifications pour les composants 'enfants' (i.e imbriqués).

S'il s'agit de modifier le composant en lui même (ajouter un texte, enlever une photo, etc), il faut alors modifier le render/return du composant.

React Native intègre un certain nombre de composants qui permettent un fonctionnement de base. On peut trouver tous ces composants au lien suivant :
https://reactnative.dev/docs/components-and-apis
D'autres composants sont aussi disponibles dans React Native Elements :
https://reactnativeelements.com/docs
Enfin, si besoin est de créer un nouveau composant pour quelque chose de plus perfectionner, il en existe déjà une multitude en OpenSource sur internet. Il faut toutefois faire attention à si le module est sur la bonne version et est bien fonctionnel/maintenu.