Développement build et développement natif en React Native

Développement natif :

Pour faire un développement natif avec React Native, on opte pour l'option d'utiliser React Native sans Framework. Il peut arriver que notre application React Native nécessite d'interagir avec des API de plateforme natives non fournies par React Native ou une bibliothèque existante,

Dans ce sens , on écrit un code d'intégration à l'aide d'un module Turbo Native. Il y a des étapes de base qu'on suit :

- Dans un premier temps, on défini une spécification JavaScript en utilisant l'un des langages d'annotation de type JavaScript les plus populaires : Flow ou TypeScript ;
- Ensuite, on configure le système de gestion des dépendances pour exécuter Codegen, qui convertit la spécification en interfaces en langage natif ;
- On écrit après notre code d'application en utilisant nos spécifications ;
- Enfin, on écrit notre code de plateforme natif en utilisant les interfaces générées pour écrire et connecter notre code natif à l'environnement d'exécution React Native ;

Avant toute chose, on implémente l'API du WebStorage (localStorage). Cette API est liée à un développeur React qui pourra écrire du code d'application sur notre projet.
Pour mieux suivre, je vais commencer par créer le projet suivant :

Step 1: npx @react-native-community/cli@latest init AwesomeProject

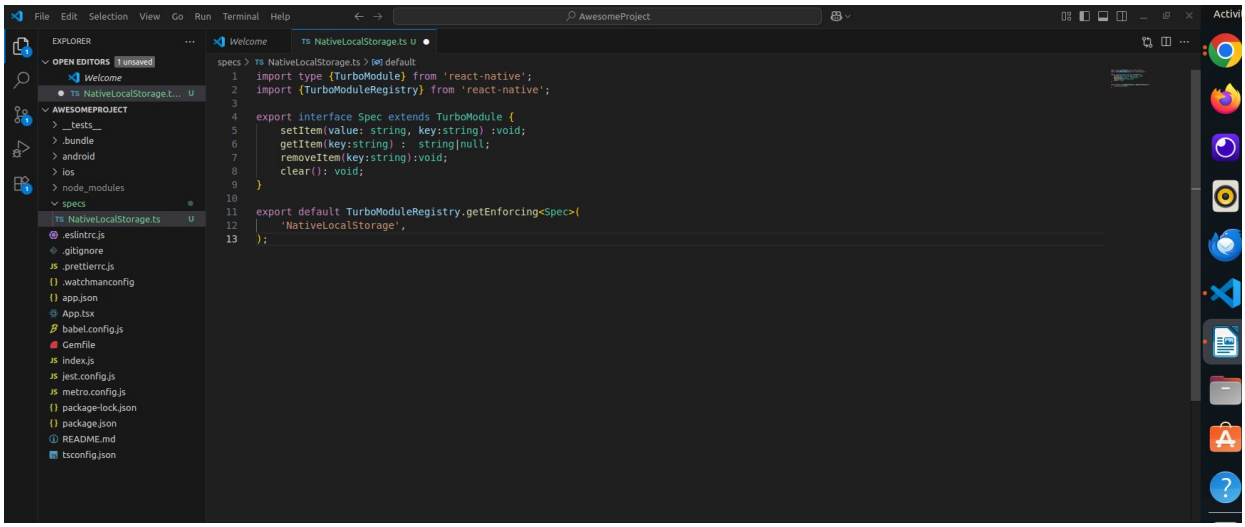
Cette commande permet de démarrer un nouveau projet, ici le nom du projet est **AwesomeProject**

Après avoir créer le projet , on va maintenant créer un fichier de spécifications. En effet, React Native fournit Codegen qui utilise une spécification écrite en TypeScript ou Flow et génère du code spécifique à la plateforme pour Android et iOS.

La spécification déclare les méthodes et les types de données qui seront échangés entre le code natif et l'environnement d'exécution JavaScript de React Native. Le module Turbo Native comprend à la fois notre spécification, le code natif qu'on écrit et les interfaces Codegen générées à partir de cette spécification.

Step 2:

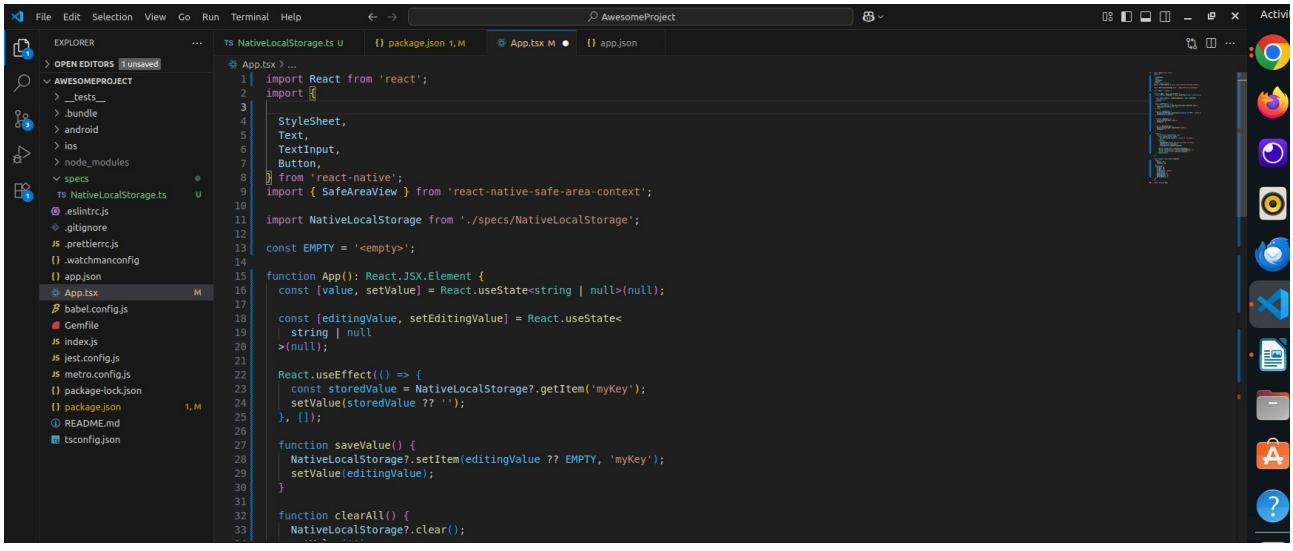
- Après avoir run le projet on l'ouvre : On crée un dossier specs à la racine du projet et à l'intérieur un fichier NativeLocalStorage.ts

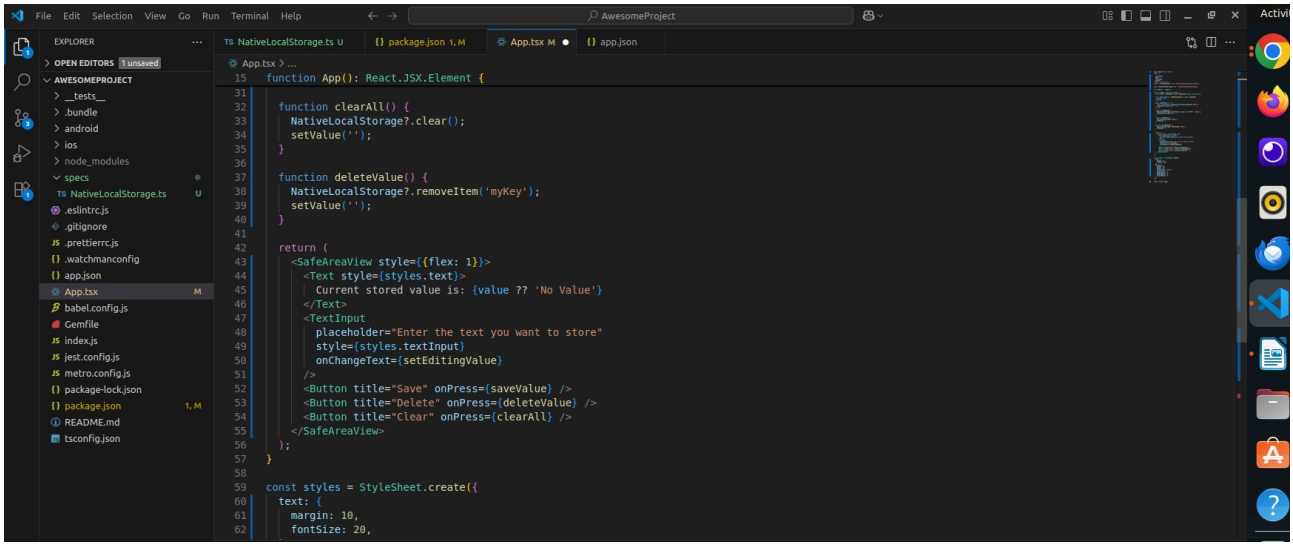


Une fois que cela est fait, nous allons préparer notre code natif à se connecter à notre code généré.

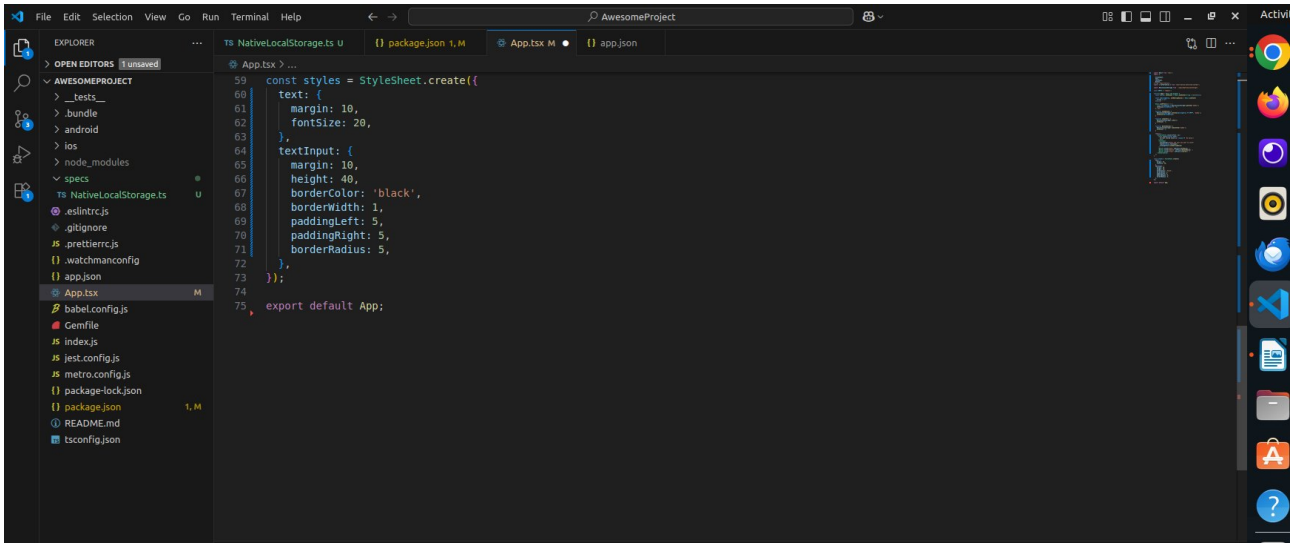
Step 3:

- Après cette étape, nous allons écrire le code de l'application à l'aide Turbo Native Module.





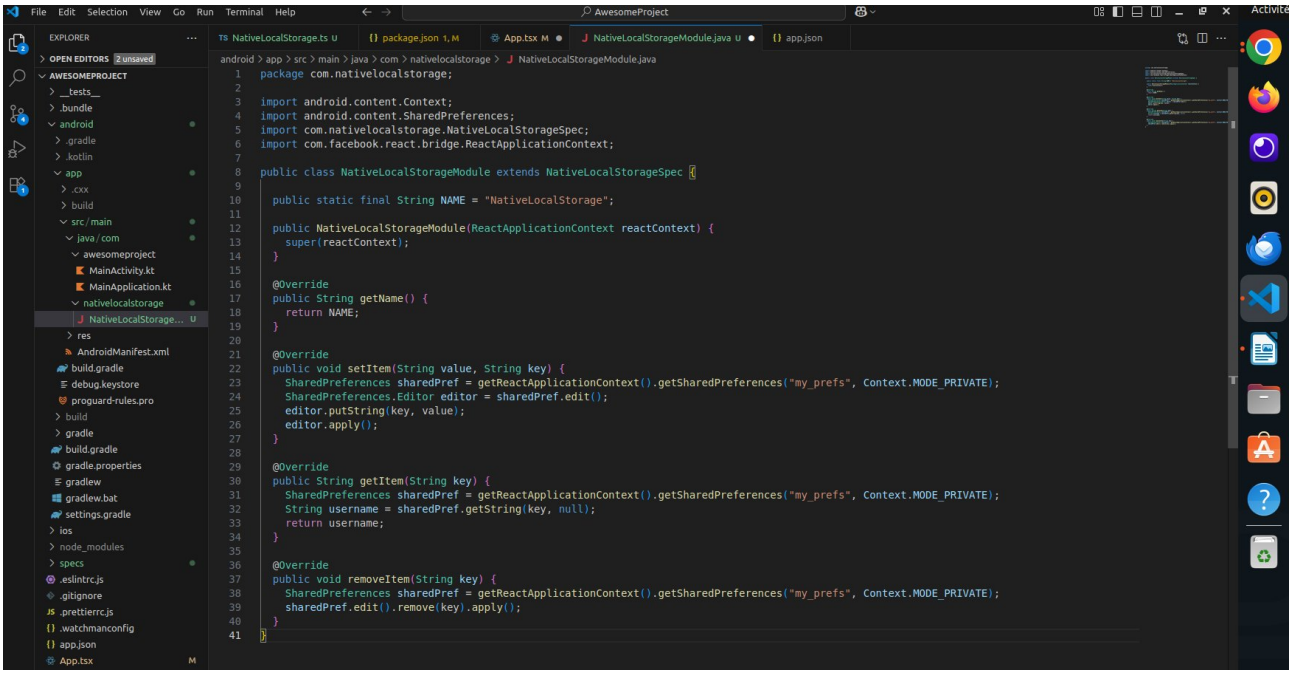
```
function App(): React.JSX.Element {

  function clearAll() {
    NativeLocalStorage?.clear();
    setValue('');
  }

  function deleteValue() {
    NativeLocalStorage?.removeItem('myKey');
    setValue('');
  }

  return (
    <SafeAreaView style={{flex: 1}}>
      <Text style={styles.text}>
        Current stored value is: {value ?? 'No Value'}
      </Text>
      <TextInput
        placeholder="Enter the text you want to store"
        style={styles.textInput}
        onChangeText={setEditingValue}
      />
      <Button title="Save" onPress={saveValue} />
      <Button title="Delete" onPress={deleteValue} />
      <Button title="Clear" onPress={clearAll} />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  text: {
    margin: 10,
    fontSize: 20,
```



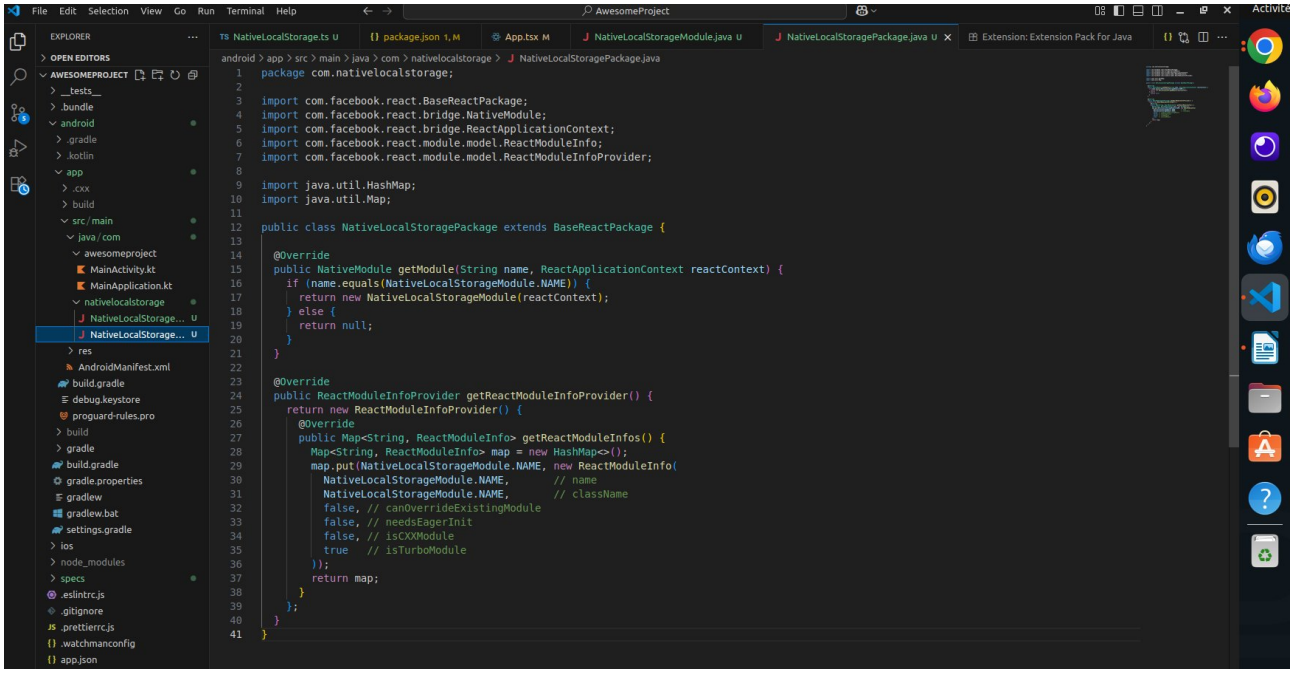
```
const styles = StyleSheet.create({
  text: {
    margin: 10,
    fontSize: 20,
  },
  textInput: {
    margin: 10,
    height: 40,
    borderColor: 'black',
    borderWidth: 1,
    paddingLeft: 5,
    paddingRight: 5,
    borderRadius: 5,
  },
});

export default App;
```

Step 4:

Une fois que les étapes précédentes sont respectés, nous allons commencer à écrire du code natif.



```
package com.nativelocalstorage;

import android.content.Context;
import android.content.SharedPreferences;
import com.nativelocalstorage.NativeLocalStorageSpec;
import com.facebook.react.bridge.ReactApplicationContext;

public class NativeLocalStorageModule extends NativeLocalStorageSpec {

  public static final String NAME = "NativeLocalStorage";

  public NativeLocalStorageModule(ReactApplicationContext reactContext) {
    super(reactContext);
  }

  @Override
  public String getName() {
    return NAME;
  }

  @Override
  public void setItem(String value, String key) {
    SharedPreferences sharedPref = getReactApplicationContext().getSharedPreferences("my_prefs", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPref.edit();
    editor.putString(key, value);
    editor.apply();
  }

  @Override
  public String getItem(String key) {
    SharedPreferences sharedPref = getReactApplicationContext().getSharedPreferences("my_prefs", Context.MODE_PRIVATE);
    String username = sharedPref.getString(key, null);
    return username;
  }

  @Override
  public void removeItem(String key) {
    SharedPreferences sharedPref = getReactApplicationContext().getSharedPreferences("my_prefs", Context.MODE_PRIVATE);
    sharedPref.edit().remove(key).apply();
  }
}
```



Développement build :

Le projet Perxonomiz tournait déjà en développement build, les différentes étapes sont les suivantes :

- npx create-expo-app nom-projet
- npx expo prebuild
- npm run android

Les différences clés notées :

- Expo fournit un environnement prêt-à-l'emploi.
- Pas besoin de toucher au code natif (Android/iOS)
- Builds personnalisés avec Expo EAS

- Avec le natif, on travaille directement avec React Native CLI
- Si besoin, possibilité d'implémenter directement du code natif (Kotlin ou Swift) pour des usages avancés
- on a /android dès le départ
- index.js lance l'application