```
import React, {useContext, useEffect, useState} from 'react';
import {
    Clipboard,
    Dimensions,
    Image,
    Linking,
    Platform,
    StyleSheet,
    Text,
    TouchableOpacity,
    TouchableWithoutFeedback,
    View,
    NativeModules
} from 'react-native';
import {ScrollView, TextInput} from 'react-native-gesture-
handler';
import {SafeAreaView} from 'react-native-safe-area-context';
import firebase from '@react-native-firebase/app';
import {FirebaseAuthTypes} from '@react-native-firebase/auth';
import {
    DeviceInterface,
    loginAttemptInterface,
    PolicyInterface,
    useSmartdiagnosticsInterface,
    loginAuxCredentials,
} from '../interfaces/Interfaces';
import AsyncStorage from '@react-native-async-storage/async-
storage';
import {ActivityIndicator} from 'react-native-paper';
import {useUpdateData} from '../hooks/useUpdateData';
import {DeviceContext} from
'../context/DeviceContext/DeviceContext';
import {PolicyContext} from
'../context/PolicyContext/PolicyContext';
import {useSmartdiagnosticsSDK} from
'../hooks/useSmartdiagnosticsSDK';
import {PermissionContext} from "../context/Permissions";
import SplashScreen from 'react-native-splash-screen';
import {DiagnosticContext} from
"../context/DiagnosticContext/DiagnosticContext";
import {useDevice} from '../hooks/useDevice';
import {usePolicy} from '../hooks/usePolicy';
import {DiagnosticResultSDK} from
"../interfaces/diagnosticResultSdk";
import Config from 'react-native-config';
import LegalsModal from "../components/LegalsModal";
import CustomAlert from "../components/CustomAlert";
import {isEnable, transformDiagnostics} from "../utils/utils";
```

```
const {Credentials} = NativeModules;

let StrConstants;
let errConstants =  require('./../assets/errorConstants');;
let imageSource = '';
let currentFlavor = Config.FLAVOR;
console.log('currentFlavor: ', currentFlavor)
if (currentFlavor === 'flavor1') {
    imageSource = require('./../assets/flavor1/VELOXX_V2.png');
    StrConstants = require('../constants/flavor1/strings');
} else if (currentFlavor === 'flavor2') {
    imageSource =
require('./../assets/flavor2/segurocell_logo.png');
    StrConstants = require('../constants/flavor2/strings');
} else {
    imageSource = require('./../assets/logo_fs_sinfondo.png');
    StrConstants = require('../constants/strings');
}

export const LoginScreen = ({navigation}) => {

    const {setDevice} = useContext(DeviceContext);
    const {storedPolicy, setPolicy} =
useContext(PolicyContext);
    const [isLoading, setIsLoading] = useState(true);

    const [numPoliza, setNumPoliza] = useState('');
    const [codSms, setCodSms] = useState('');
    const [codImei, setCodImei] = useState('');
    const [codPoliza, setCodPoliza] = useState('');
    const [codPolizaInput, setCodPolizaInput] = useState(0);
    const [textoMostrar, setTextoMostrar] = useState(0);
//Toggle txt & buttons
    const [waitingCode, setWaitingCode] = useState(false);
    const [confirm, setConfirm] = useState(null); // If null,
no SMS has been sent
    const [verificationId, setVerificationId] = useState(null);
    const {updateTokenAndLastLogin, updateLegals,
getSmsAuthTimeout, getLastAppVersion} = useUpdateData();
    const {askPermission, permissionGranted} =
useContext(PermissionContext);
    const {setDiagnostic} = useContext(DiagnosticContext);
    const [polizaLoaded, setPolizaLoaded] = useState(false);
    const [timeRemaining, setTimeRemaining] = useState(30);
    const [isReintento, setIsReintento] = useState(0);

    const {getStoredDevice, getDeviceByDeviceId,
suscribeDevice, checkDeviceExists, getDeviceByDeviceIdPromise}
= useDevice(null);
    const {
        getStoredPolicy, existsActivePolicyByPhone,
getPolicyActiveByPhone, transformPolicyArrayToObject,
        suscribePolicy, updatePolicyStatusByKey,
getPolicyByRef, updatePolicyStatusByKeyAndStatus,
deleteDeviceIdByKey, getPolicyActiveByPhoneAndNumber
    } = usePolicy(null);

    // Set an initializing state whilst Firebase connects
    const [initializing, setInitializing] = useState(true);
    const [user, setUser] = useState();

    const {getLastDiagnosticsReturn,
initSmartDiagnosticsReturn, doDiagnosticsReturn} =
useSmartdiagnosticsSDK();

    const [showAlertPerm, setShowAlertPerm] = useState(false);
    const showAlertPermissions = () => {
        setShowAlertPerm(true)
        setIsLoading(false);
    };

    const [showAlertError, setShowAlertError] =
useState(false);
    const [showAlertUpdateApp, setShowAlertUpdateApp] =
useState(false);
    const [msgError, setMsgError] = useState("Se ha producido
un error");
    const alertError = (e) => {
        setIsLoading(false);
        let msg = "Se ha producido un error"
        try {
            msg = e.toString()
        } catch (e) {
            console.log(e)
        }
        setMsgError(msg)
        setShowAlertError(true)
    };
    const alertUpdateApp = (e) => {
        setIsLoading(false);
        let msg = "Hay disponible una nueva versión de la
aplicación. Descargue la última versión para continuar"
        try {
            msg = e.toString()
        } catch (e) {
            console.log(e)
        }
        setMsgError(msg)
        setShowAlertUpdateApp(true)
    };
    const [showAlertDebug, setShowAlertDebug] =
useState(false);
    const [msgDebug, setMsgDebug] = useState("");
    const alertDebug = (msg) => {
        setMsgDebug(msg)
        setShowAlertDebug(true)
    };

    const [showAlertFb, setShowAlertFb] = useState(false);
    const showAlert = () => {
        setShowAlertFb(true)

    }

    const [showAlertDiagnosticDone, setShowAlertDiagnosticDone]
= useState(false);

    const [showAlertPolizaAsignada, setShowAlertPolizaAsignada]
= useState(false);

    const openStoreLink = () => {
        // TODO: Reemplazar 'app_store_link' y
'play_store_link' con los enlaces de la tienda de aplicaciones
        const appStoreLink = Config.APP_STORE_LINK;
        const playStoreLink = Config.PLAY_STORE_LINK;

        if (Platform.OS === 'ios') {
            Linking.openURL(appStoreLink);
        } else {
            Linking.openURL(playStoreLink);
        }
    };

    async function getCredentials() {
        return new Promise(async (resolve, reject) => {
            try {
                const [email, password] = await
Credentials.getCredentials();
                let credentials: loginAuxCredentials = {
                    email: email,
                    pass: password
                }
                resolve(credentials)
            } catch (error) {
                console.error('Error getting credentials:',
error);
                reject(error)
            }
        })
    }

    const _onPressButtonPoliza = () => {
        loginAttemptLogic().then(response => {
            console.log('loginAttemptLogic response:',
response);
            if (response) {
                showAlert();
            } else {
                alertError('Ya existe una operación de
verificación de número en curso. Por favor, espere al menos 5
minutos antes de volver a intentarlo.');
            }
        })
    };

    // Función para enviar un código de verificación al número
de teléfono del usuario
    const sendVerificationCode = async (phoneNumber: string):
Promise<{ auto: boolean, verificationId?: string, credential?:
FirebaseAuthTypes.AuthCredential }> => {
        return new Promise<{ auto: boolean, verificationId?:
string, credential?:
FirebaseAuthTypes.AuthCredential }>((resolve, reject) => {
            const phoneAuthListener =
firebase.auth().verifyPhoneNumber(
                phoneNumber,
                30,
                true
            );
            setWaitingCode(true);
            phoneAuthListener.on('state_changed',
(phoneAuthSnapshot) => {
                console.log('Snapshot state: ',
phoneAuthSnapshot.state);
                if (phoneAuthSnapshot.state ===
firebase.auth.PhoneAuthState.CODE_SENT) {
                    //alertDebug("El código de verificación se
ha enviado al número " + phoneNumber)
                    getSmsAuthTimeout().then(time => {
                        console.log('setTimeRemaining: ', time)
                        setTimeRemaining(time)
                        startTimer()
                        console.log('CODE_SENT');
                        console.log("Código de verificación
enviado exitosamente")
                        console.log("verificationId: ",
phoneAuthSnapshot.verificationId)
                        // Código de verificación enviado
exitosamente
                        const verificationId =
phoneAuthSnapshot.verificationId;
                        setVerificationId(verificationId);
                        resolve({
                            auto: false,
                            verificationId: verificationId
                        });
                    }).catch((error) => {
                        setTimeRemaining(30)
                        startTimer()
                        console.log('CODE_SENT');
                        console.log("Código de verificación
enviado exitosamente")
                        console.log("verificationId: ",
phoneAuthSnapshot.verificationId)
                        // Código de verificación enviado
exitosamente
                        const verificationId =
phoneAuthSnapshot.verificationId;
                        setVerificationId(verificationId);
                        resolve({
                            auto: false,
                            verificationId: verificationId
                        });
                    });
                } else if (phoneAuthSnapshot.state ===
firebase.auth.PhoneAuthState.ERROR) {
                    console.log('ERROR');

//alertDebug(phoneAuthSnapshot.error.message)
                    // Ocurrió un error al enviar el código de
verificación
                    setWaitingCode(false);
                    const error = phoneAuthSnapshot.error;
                    reject(error);
                } else if (phoneAuthSnapshot.state ===
firebase.auth.PhoneAuthState.AUTO_VERIFY_TIMEOUT) {
                    console.log('AUTO_VERIFY_TIMEOUT');
                    //alertDebug("[AUTO_VERIFY_TIMEOUT]No se
ha podido realizar la verificación automatica")
                    console.log(phoneAuthSnapshot);
                    setWaitingCode(false);
                    reject('No se ha podido realizar la
verificación');
```

```
                } else if (phoneAuthSnapshot.state ===
firebase.auth.PhoneAuthState.AUTO_VERIFIED) {
                    console.log('AUTO_VERIFIED');
                    //alertDebug("[AUTO_VERIFIED]Código de
verificación enviado")
                    const verificationId =
phoneAuthSnapshot.verificationId;
                    setVerificationId(verificationId);
                    // Código de verificación enviado
exitosamente
                    const credential =
firebase.auth.PhoneAuthProvider.credential(
                        verificationId,
                        phoneAuthSnapshot.code,
                    );
                    console.log("Código de verificación
enviado exitosamente")
                    console.log(credential);
                    setCodSms(phoneAuthSnapshot.code)
                    resolve({
                        auto: true,
                        verificationId: verificationId,
                        credential: credential
                    });
                } else {
                    //alertDebug(phoneAuthSnapshot.state + ' -
No se ha podido realizar la verificación')
                    console.log(phoneAuthSnapshot);
                    reject('No se ha podido realizar la
verificación');
                }
            }, (error) => {
                console.log('Ocurrió un error al enviar el
código de verificación');
                // Ocurrió un error al enviar el código de
verificación
                //alertDebug(error.message)
                reject(error);
            }/*, (phoneAuthSnapshot) => {
                console.log(phoneAuthSnapshot);
                // Código de verificación enviado exitosamente
                const verificationId =
phoneAuthSnapshot.verificationId;
                resolve(verificationId);
              }*/);
        });
    };

    /**
```

```
     * Firebase SignIn With PhoneNumber
     * @param numPoliza
     */
    async function signinWithPhoneNumber(numPoliza) {
        try {
            let authVar = firebase.auth()
            //authVar.onAuthStateChanged(onAuthStateChanged);
            console.log('-Número: ', numPoliza)
            setWaitingCode(true);

            sendVerificationCode(numPoliza)
                .then((response) => {
                    if (response.auto) {
                        console.log('')
                        console.log('')
                        console.log('Response auto')
                        console.log(response)
                        console.log('')
                        console.log('')
                        console.log('')

setVerificationId(response.verificationId)
                        setWaitingCode(false);
                        loginWith(response.credential)

                    } else {
                        console.log('')
                        console.log('')
                        console.log('Response no auto')
                        console.log(response)
                        console.log('')
                        console.log('')
                        console.log('')
                        // Mostrar una pantalla para que el
usuario ingrese el código de verificación

setVerificationId(response.verificationId)
                        setTextoMostrar(1);
                        setWaitingCode(false);
                    }
                })
                .catch((error) => {
                    // Error al enviar el código de
verificación
                    console.log('Error sending verification
code:', error);
                    console.error(error);
                    if
(error.toString().includes('[auth/invalid-phone-number]')) {
```

```
alertError(StrConstants.invalid_phone_number);
                    } else if
(error.toString().includes('[auth/too-many-requests]')) {

alertError(StrConstants.too_many_request);
                    } else {

alertError(errConstants.login_sms_code_error1);//TODO: Ir a
login alternativo
                    }
                    console.error('153');
                });

        } catch (e) {
            console.error(e);
            if (e.toString().includes('[auth/invalid-phone-
number]')) {
                alertError(StrConstants.invalid_phone_number);
            } else if (e.toString().includes('[auth/too-many-
requests]')) {
                alertError(StrConstants.too_many_request);
            } else {

alertError(errConstants.login_sms_code_error1_1);//TODO: Ir a
login alternativo
            }
            console.error('154');
        }
    }

    const _onPressButtonSms = () => {
        try {
            confirmCode()/*.then(_r => {
                setWaitingCode(false);
            });*/
        } catch (e) {
            console.error('164');
            alertError(StrConstants.login_sms_code_error);
            console.error(e);
        }
    };

    async function confirmCode() {
        try {
            const credential =
firebase.auth.PhoneAuthProvider.credential(
                verificationId,
                codSms,
            );

            loginWith(credential)

            /* await verifyCode(verificationId, codSms)
                 .then((userCredential) => {
                     // Usuario ha iniciado sesión
exitosamente
                     const user = userCredential.user;
                     console.log('User logged in:', user);
                 })
                 .catch((error) => {
                     // Error al verificar el código de
verificación
                     console.log('Error verifying code:',
error);

alertError(StrConstants.login_sms_code_error);
                 });*/
        } catch (error) {
            console.error(error);
            // Teléfono incorrecto
            console.error('178');
            alertError(StrConstants.login_sms_code_error);
        }
    }

    const loginWith = (credential) => {
        console.log('')
        console.log('loginWith')
        console.log(credential)
        console.log('')
        firebase.auth()
            .signInWithCredential(credential)
            .then((result) => {
                storeData('@smsIdentification', true).then(_r
=> {
                    console.log('smsIdentification actualizado
en local');
                    setWaitingCode(false);
                    // Manejar resultado del inicio de sesión
exitoso
                    setIsLoading(false);
                    setWaitingCode(false);
                    setTextoMostrar(2);
                }).catch((error) => {
                    console.log('loginWith error')
                    setWaitingCode(false);
                    // Manejar error en el inicio de sesión
alertError(StrConstants.login_sms_code_error);
                });

            })
            .catch((error) => {
                console.log('loginWith error')
                setWaitingCode(false);
                // Manejar error en el inicio de sesión

alertError(errConstants.login_sms_code_error2);//TODO: Ir a
login alternativo
            });
    };

    const onVerificationCompleted = (num) => {
        console.log("onVerificationCompleted num:", num)
        if (numPoliza != '' || num != '') {
            setTimeout(function() {
                setIsLoading(false);
                setWaitingCode(false);
                setTextoMostrar(2);
            }, 1500);
        } else {
            //comprobarDatosLocales()
            clearAll()
        }
    };

    const loginAttemptFunction = (currentTimestamp) => {
        let loginAttempt: loginAttemptInterface = {
            phone: numPoliza,
            timestamp: currentTimestamp
        }
        loginAttempt.phone = numPoliza;
        loginAttempt.timestamp = currentTimestamp;
        console.log('loginAttempt: ', loginAttempt);
        storeData('@loginAttempt_' + numPoliza,
loginAttempt).then(_r => {
            console.log('loginAttempt actualizado');
        });
    };

    const loginAttemptLogic = async () => {
        return new Promise(async (resolve, reject) => {
            const currentDate = new Date();
            const currentTimestamp = currentDate.getTime();
            console.log('storageKey: ', '@loginAttempt_' +
numPoliza)
```

```
            retrieveData('@loginAttempt_' +
numPoliza).then(resp => {
                console.log(resp)
                let loginAttemptStored: loginAttemptInterface
=
                    resp != null ? JSON.parse(resp) : null;
                console.log('loginAttemptStored: ',
loginAttemptStored);
                if (resp != null) {
                    let timestamp =
loginAttemptStored?.timestamp
                    console.log('loginAttemptStored.phone ==
numPoliza: ', loginAttemptStored.phone == numPoliza);
                    if (loginAttemptStored.phone == numPoliza)
{
                        // Verifica si han pasado 5 minutos
(300,000 milisegundos)
                        const diff = currentTimestamp -
timestamp;
                        console.log('diff: ', diff);
                        console.log('diff >= 300000: ',
diff >= 300000);
                        if (diff >= 300000) {
                            console.log('Han pasado 5
minutos');

loginAttemptFunction(currentTimestamp)
                            resolve(true);
                        } else {
                            console.log('Aún no han pasado 5
minutos');
                            resolve(false);
                        }
                    } else {
                        loginAttemptFunction(currentTimestamp)
                        resolve(true);
                    }
                } else {
                    loginAttemptFunction(currentTimestamp)
                    resolve(true);
                }
            }).catch(error => {
                console.error(error);
                resolve(false);
            });

            const keys = await AsyncStorage.getAllKeys();
            // Iterar sobre todas las claves y verificar si
contienen la cadena "loginAttempt"
```

```
            for (const key of keys) {
                if (key.includes('loginAttempt')) {
                    retrieveData('@loginAttempt_' +
numPoliza).then(resp => {
                        if (resp != null) {
                            let loginAttemptStored:
loginAttemptInterface =
                                resp != null ?
JSON.parse(resp) : null;
                            // Verifica si han pasado 5
minutos (300,000 milisegundos)
                            let timestamp =
loginAttemptStored?.timestamp
                            const diff = currentTimestamp -
timestamp;
                            if (diff >= 300000) {
                                console.log('Han pasado 5
minutos');

                                AsyncStorage.removeItem(key)
                            }
                        }
                    }).catch(error => {
                        console.error(error);
                    });
                }
            }
        });
    };

    const _onPressButtonImei = () => {
        getPolice();
    };

    const _onPressButtonCodPoliza = () => {
        console.log('' +
            'Haciendo login para:\n' +
            'Numero telefono: ', numPoliza)
        console.log('Numero poliza: ', codPoliza)
        //Get credentials
        getCredentials().then((credentials) => {
            let cred: LoginAuxCredentials = credentials
            signInWithGenericAccount(cred.email,
cred.pass).then((result) => {
                if (result){
                    //Guardar datos en dispositivo
                    storeData('@auxLoginPhone',
numPoliza).then(()=>{
                        storeData('@auxLoginPolicy',
codPoliza).then(()=>{
```

```
                            storeData('@smsIdentification',
false).then(()=>{
                                storeData('@isAuxLogin',
true).then(()=>{

getPolicyActiveByPhoneAndNumber(numPoliza,
codPoliza).then(result => {
                                        console.log(result)
                                        let policies: any =
result
                                        if (policies.length <=
0){

alertError(StrConstants.no_policy_active_text_info);
                                        }else{

policies.forEach(childSnapshot => {
                                                let childData:
PolicyInterface = transformPolicyArrayToObject(childSnapshot);
                                                if
(childData.number == codPoliza && childData.phone ==
numPoliza){

setTextoMostrar(2);
                                                }else{

alertError(errConstants.policy_phone_mismatch_error);
                                                }
                                            })
                                        }
                                    })
                                })
                            })
                        })
                    })
                }else{

alertError(errConstants.generic_account_login_error);
                }
            })
        }).catch(err =>{
            //ERROR
            alertError(errConstants.GET_CREDENTIALS_ERROR);
        })

    };

    async function signInWithGenericAccount(email, password) {
        return new Promise(async (resolve, reject) => {
            try {
                try {
                    await
firebase.auth().signInWithEmailAndPassword(email,
password).then(() => {
                        console.log('Inicio de sesión con
cuenta genérica exitoso');
                        resolve(true)
                    })
                } catch (error) {
                    console.error('Error al iniciar sesión con
cuenta genérica:', error);
                    resolve(false)
                }
            } catch (e) {
                resolve(false)
            }
        })
    }

    const mostrarAlertaPolizaAsignada = () => {
        setShowAlertPolizaAsignada(true)
    };

    const getPolice = async () => {
        //Buscando poliza para el telefono indicado
        console.log('Buscando poliza para el telefono
indicado')

        console.log('---------------numPoliza: ',numPoliza)
        if (numPoliza != "") {
            existsActivePolicyByPhone(numPoliza).then(existe
=> {
                if (existe == 0) {

getPolicyActiveByPhone(numPoliza).then(snapshot => {
                        let policies: any = snapshot
                        policies.forEach(childSnapshot => {
                            // childData will be the actual
contents of the child
                            let childData: PolicyInterface =
transformPolicyArrayToObject(childSnapshot);
                            //Input para initSmartDiagnostics
                            let input = {} as
useSmartdiagnosticsInterface;
                            input.policyId = childData.number
                            input.phone = numPoliza
                            input.IMEI = codImei
                            input.email = childData.email
                            input.codeModel =
childData.codeModelInit || ""
                            input.description =
childData.deviceDesc || ""

                            //Comprobar el estado de la póliza
(Si esta preasignada, en tramite o activa se puede hacer el
diagnostico)
                            if (isEnable(childData.status)) {
                                //Comprobando si la póliza ya
tiene un device asignado
                                console.log("Comprobando si la
póliza ya tiene un device asignado")
                                if (typeof childData.deviceId
== 'undefined') {
                                    //La poliza no tiene un
device asignado, iniciamos el SDK y realizamos un diagnostico
                                    console.log("La poliza no
tiene un device asignado, iniciamos el SDK y realizamos un
diagnostico")


handleOpenModal().then(resultModal => {
                                        updateLegals(null,
numPoliza, codImei).then(r => {

iniciar_sdk_y_realizar_diagnostico(input)
                                        })
                                    })

                                } else {
                                    //La poliza tiene un
device asignado, comprobamos si coincide con el IMEI
introducido
                                    console.log("La poliza
tiene un device asignado, comprobamos si coincide con el IMEI
introducido")
                                    comprobar_imei(childData,
input, 1)
                                }
                                return true;
                            } else {//Nunca deberia llegar
aqui
                                //No se hace diagnostico
                                console.log('No se hace
diagnostico')
                                if (typeof childData.deviceId
== 'undefined') {
                                    console.log("La poliza no
tiene un device asignado y no se puede realizar diagnóstico")
                                    let msgError = "La poliza
no tiene un dispositivo asignado y no esta activa por lo que
no se puede realizar un diagnóstico"
                                    setMsgError(msgError)
                                    setShowAlertError(true)
                                } else {
                                    console.log("La poliza
tiene un device asignado, comprobamos si coincide con el IMEI
introducido")
                                    comprobar_imei(childData,
input, 0)
                                }
                            }
                        });
                    });
                } else if (existe == 2) {

setMsgError(StrConstants.no_policy_active_text_info)
                    setShowAlertError(true)
                }else if (existe == 1) {

setMsgError(StrConstants.dup_policy_active_text_info)
                    setShowAlertError(true)
                }
            })

        } else {
            clearAll()
        }
    };

    const iniciar_sdk_y_realizar_diagnostico = (input) => {
        setIsLoading(true);
        initSmartDiagnosticsReturn(input).then(res => {
            doDiagnosticsReturn(input).then(response1 => {
                getPolicyActiveByPhone(numPoliza)
                    .then(snapshot => {
                        let policies: any = snapshot
                        policies.forEach(childSnapshot => {
                            let childData: PolicyInterface =
transformPolicyArrayToObject(childSnapshot);

updatePolicyStatusByKey(childData.status, childData.key)
                            storeData('@policy',
childData).then(_r => {
                                suscribePolicy(childData.key)
                                setPolicy(childData);
```

```
getDeviceByDeviceId(childData.deviceId)
                                    .then(snapshot => {
                                        let device:
DeviceInterface = snapshot.val()
                                        storeData('@device',
device).then(r => {

suscribeDevice(childData.deviceId)
                                            setDevice(device);
                                            let input = {} as
useSmartdiagnosticsInterface;
                                            input.policyId =
childData.number

getLastDiagnosticsReturn(input).then(response => {
                                                let result:
DiagnosticResultSDK;
                                                if
(response.code != 200) {
                                                    // Mostrar
un mensaje de error

setMsgError(response.result)

setShowAlertError(true)

                                                }

                                                result =
response

setDiagnostic(result);
                                                //Actualizar
Token y LastLogin en BBDD

retrieveData('@smsIdentification').then(resp => {
                                                    if
(resp != null) {

console.log('smsIdentification: ', resp)

console.log(resp == 'true')

console.log(resp == 'false')

updateTokenAndLastLogin(storedPolicy.deviceId,
storedPolicy.phone,resp == 'true').then(() => {

console.log('Data updated');
setShowAlertDiagnosticDone(true)

//navigation.navigate(StrConstants.mi_poliza);
                                                        }).cat
ch(error => {

setIsLoading(false);

console.error(error);
                                                        });
                                                    }else{

console.log('Response Null');

alertError(StrConstants.loginRequired);
                                                    }
                                                }).catch(error
=> {

console.error(error);
                                                    clearAll()
                                                });


                                            }).catch(error =>
{

console.error(error);

console.error('308');

alertError(error);
                                            });
                                        })
                                            .catch(error => {

console.error(error);

console.error('314');

alertError(errConstants.device_storage_error);
                                            });
                                    })

                                return true;
                            })
                                .catch(error => {
                                    console.error(error);
                                    console.error('327');

alertError(errConstants.policy_storage_error);
                                });
                            return true;
                        })
                    })
            })
                .catch(error => {
                    console.error(error);
                    console.error('336');
                    alertError(error);
                });
        })
            .catch(error => {
                console.error(error);
                console.error('342');
                alertError(error);
            });
    }

    const comprobar_imei = (childData, input, estado) => {

        getDeviceByDeviceId(childData.deviceId)
            .then(snapshot => {
                let device: DeviceInterface = snapshot.val()
                let imeiAsociado = device.uniqueId
                let thisLegals = device.legals

                let remoteDiags = device?.remoteDiags
                let send_ts = remoteDiags?.send_ts
                let isSended = typeof send_ts != 'undefined'


                //Comprobar si hay un device guardado en el
dispositivo
                retrieveData('@device').then(resp => {

                    let localDevice: DeviceInterface =
                        resp != null ? JSON.parse(resp) : null;

                    console.log("IMEI introducido: " + codImei)
                    console.log("IMEI guardado localmente: " +
localDevice?.uniqueId)
                    console.log("IMEI asociado a la poliza: "
+ imeiAsociado)

                    if (imeiAsociado == codImei ||
imeiAsociado == localDevice?.uniqueId) {
```

```
                        //El IMEI asociado a la poliza
coincide con el IMEI introducido
                        console.log("El IMEI asociado a la
poliza coincide con el IMEI introducido o el guardado
localmente");


initSmartDiagnosticsReturn(input).then(res => {

getLastDiagnosticsReturn(input).then(response => {
                                let result:
DiagnosticResultSDK = response
                                if (thisLegals) {
                                    legalsAceptadas(result,
input, response, estado, childData.deviceId, isSended)
                                } else {

                                    handleOpenModal()
                                        .then(resultModal => {
                                            if (resultModal ==
true) {

updateLegals(childData.deviceId, null, null).then(r => {

legalsAceptadas(result, input, response, estado,
childData.deviceId, isSended)
                                                })
                                            }
                                        })
                                        .catch(error => {

setIsLoading(false);
                                        });
                                }



                            })
                                .catch(error => {
                                    console.error(error);
                                    console.error('440');
                                    alertError(error);
                                });

                        }).catch(error => {
                            console.error('450');
                            alertError(error);
                            console.error(error);
                        });
```

```
                    } else {
                        //El IMEI asociado a la poliza no
coincide con el IMEI introducido, hacemos saltar la alerta
                        console.log("El IMEI asociado a la
poliza no coincide con el IMEI introducido")
                        mostrarAlertaPolizaAsignada()
                    }
                }).catch(error => {

alertError(errConstants.local_device_retrieval_error);
                    console.error(error);
                });
            })
            .catch(error => {
                console.error(error);
                console.error('468');
                alertError(errConstants.login_sms_code_error3);
            });
    }

    const legalsAceptadas = (result, input, response, estado,
deviceId, isSended) => {
        if (result.code == 502 || result.code == 200) {
            if (result.code == 502 && (estado == 1
&& !isSended)) {
                doDiagnosticsReturn(input).then(response => {
                    let result: DiagnosticResultSDK;
                    result = response;
                    setDiagnostic(result);
                    getPolicyActiveByPhone(numPoliza)
                        .then(snapshot => {
                            let policies: any = snapshot
                            policies.forEach(childSnapshot =>
{
                                let childData: PolicyInterface
= transformPolicyArrayToObject(childSnapshot);

updatePolicyStatusByKey(childData.status, childData.key)
                                storeData('@policy',
childData).then(_r => {

suscribePolicy(childData.key)
                                    setPolicy(childData);

getDeviceByDeviceId(childData.deviceId)
                                        .then(snapshot => {
                                            let device:
DeviceInterface = snapshot.val()
storeData('@device', device).then(r => {

suscribeDevice(childData.deviceId)

setDevice(device);
                                                let input = {}
as useSmartdiagnosticsInterface;
                                                input.policyId
= storedPolicy.number


onGetLastDiagnosticsReturn(response, childData, 1)

                                            })
                                        })

                                    return true;
                                });
                                return true;
                            })
                        })
                })
                    .catch(error => {
                        console.error('422');
                        alertError(error);
                        console.error(error);
                    });
            } else if (result.code == 502 && (estado == 0 ||
isSended)) {
                //Coger diagnostico de firebase
                console.log('Coger diagnostico de firebase,
deviceId: ', deviceId)
                getDeviceByDeviceId(deviceId).then(snapshot =>
{
                    let device: DeviceInterface =
snapshot.val()
                    console.log('Device to store:')
                    console.log(device)
                    storeData('@device', device).then(r => {
                        suscribeDevice(deviceId)
                        setDevice(device);
                        let result: DiagnosticResultSDK;
                        result =
transformDiagnostics(device.baseDiagnostics);
                        setDiagnostic(result);

                        //Actualizar Token y LastLogin en BBDD
```

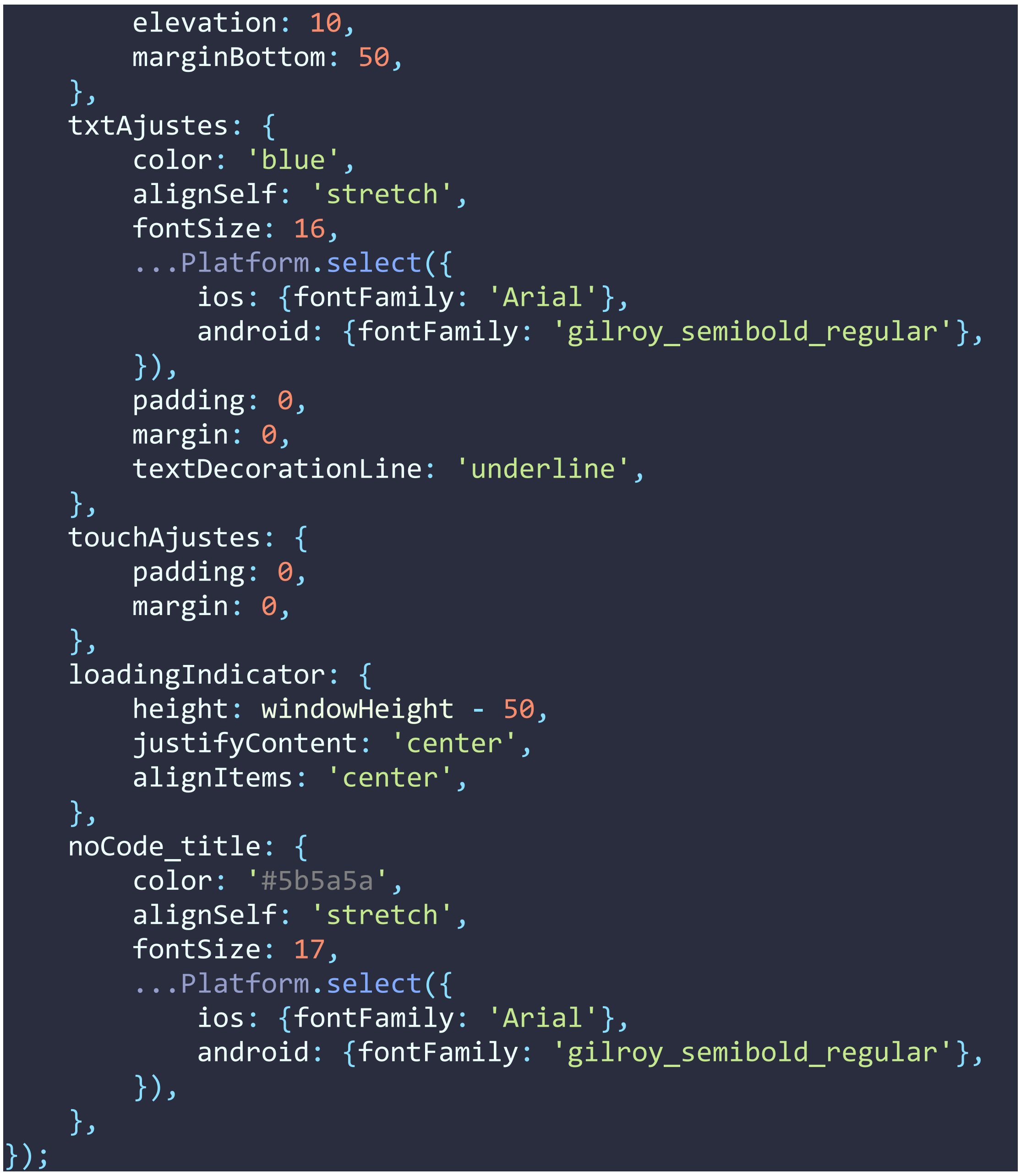
```
retrieveData('@smsIdentification').then(resp => {
                            if (resp != null) {
                                console.log('smsIdentification:
', resp)
                                console.log(resp == 'true')
                                console.log(resp == 'false')

updateTokenAndLastLogin(deviceId, numPoliza, resp ==
'true').then(() => {

getPolicyActiveByPhone(numPoliza)
                                        .then(snapshot => {
                                            let policies: any
= snapshot

policies.forEach(childSnapshot => {
                                                let childData:
PolicyInterface = transformPolicyArrayToObject(childSnapshot);

storeData('@policy', childData).then(_r => {

suscribePolicy(childData.key)

setPolicy(childData);

navigation.navigate(StrConstants.mi_poliza);

setIsLoading(false);

                                                    return
true;
                                                });
                                                return true;
                                            })
                                        })
                                }).catch(error => {
                                    console.error(error);
                                    setIsLoading(false);
                                });
                            }else{
                                console.log('Response Null');

alertError(StrConstants.loginRequired);
                            }
                        }).catch(error => {
                            console.error(error);
                            clearAll()
                        });
```

```
                    })
                })

            } else {

                getPolicyActiveByPhone(numPoliza)
                    .then(snapshot => {
                        let policies: any = snapshot
                        policies.forEach(childSnapshot => {
                            let childData: PolicyInterface =
transformPolicyArrayToObject(childSnapshot);
                            storeData('@policy',
childData).then(_r => {
                                suscribePolicy(childData.key)
                                setPolicy(childData);

getDeviceByDeviceId(childData.deviceId)
                                    .then(snapshot => {
                                        let device:
DeviceInterface = snapshot.val()
                                        storeData('@device',
device).then(r => {

suscribeDevice(childData.deviceId)
                                            setDevice(device);

                                            let input = {} as
useSmartdiagnosticsInterface;
                                            input.policyId =
storedPolicy.number
                                            setIsLoading(true);

onGetLastDiagnosticsReturn(response, childData, 0)

                                        })
                                    })
                            })
                            return true
                        })
                    })
            }
        } else if (response.code != 200) {
            // Mostrar un mensaje de error
            console.error('432');
            alertError(errConstants.last_diag_code_error);

        }
    }
    const onGetLastDiagnosticsReturn = (response, policy, tipo)
=> {
        let result: DiagnosticResultSDK;
        if (response.code != 200) {
            // Mostrar un mensaje de error
            console.error('492');
            alertError(response.result);

        } else {
            result = response
            setDiagnostic(result);

            //Actualizar Token y LastLogin en BBDD
            retrieveData('@smsIdentification').then(resp => {
                if (resp != null) {
                    console.log('smsIdentification: ', resp)
                    console.log(resp == 'true')
                    console.log(resp == 'false')
                    updateTokenAndLastLogin(policy.deviceId,
policy.phone, resp == 'true').then(() => {
                        if (tipo == 1) {
                            setShowAlertDiagnosticDone(true)
                        } else {

navigation.navigate(StrConstants.mi_poliza);
                            setIsLoading(false);
                        }

                    }).catch(error => {
                        console.error(error);
                        setIsLoading(false);
                    });
                }else{
                    console.log('Response Null');
                    alertError(StrConstants.loginRequired);
                }
            }).catch(error => {
                console.error(error);
                clearAll()
            });
        }
    }

    /**
     * Guardar datos en el almacenamiento local
     * @param storage_Key
     * @param storage_value
     */
    const storeData = async (storage_Key, storage_value) => {
        console.log('Guardando ' + storage_Key + ' en
almacenamiento local')
        try {
            const jsonValue = JSON.stringify(storage_value);
            await AsyncStorage.setItem(storage_Key, jsonValue);
        } catch (e) {
            // saving error
            console.error(e)
        }
    };

    const retrieveData = async (storage_Key) => {
        console.log('Recuperando ' + storage_Key + ' del
almacenamiento local')
        return AsyncStorage.getItem(storage_Key);
    };

    // Handle user state changes
    function onAuthStateChanged(user) {
        console.log('------------------Auth State Changed-----
---------')
        console.log(user)
        console.log('-----------------------------------------
---------')
        setUser(user);
        if (initializing) setInitializing(false);
        if (user != null) {
            if (user.phoneNumber != null &&
user.phoneNumber != "") {
                askPermission();
                setNumPoliza(user.phoneNumber)
                console.log('---------------------------Poliza
tipo 1----------------------------')
                console.log(user.phoneNumber)
                onVerificationCompleted(user.phoneNumber);
            }else if (user.email ==
"soporte_sdiag@futurespace.es"){
                retrieveData('@isAuxLogin').then(resp0 => {
                    console.log('Is auxLogin: ', resp0)
                    if (resp0){
                        //Obtener datos en dispositivo

retrieveData('@auxLoginPhone').then(resp1 => {
                            let auxLoginPhone = resp1 != null ?
resp1.replace(/"/g, '') : null;
                            console.log('Numero telefono: ',
auxLoginPhone)
                            if (auxLoginPhone == null){
                                clearAll()
                            }else{
                                setNumPoliza(auxLoginPhone)
                                console.log('-----------------
----------Poliza tipo 2-----------------------------')
                                console.log(auxLoginPhone)

retrieveData('@auxLoginPolicy').then(resp2 => {
                                    console.log(resp2)
                                    let auxLoginPolicy =
resp2 != null ? resp2.replace(/"/g, '') : null;
                                    console.log('Numero poliza:
', auxLoginPolicy)
                                    if (auxLoginPolicy ==
null){

                                        clearAll()
                                    }else{
                                        askPermission();

setCodPoliza(auxLoginPolicy)

onVerificationCompleted(auxLoginPhone);
                                    }
                                }).catch(error => {

alertError(errConstants.retrieve_policy_error);
                                    console.error(error);
                                });
                            }
                        }).catch(error => {

alertError(errConstants.retrieve_phone_error);
                            console.error(error);
                        });
                    }
                }).catch(error => {

alertError(errConstants.retrieve_auxlogin_error);
                    console.error(error);
                });
            }
        }
    }

    const comprobarDatosLocales = () => {
        console.log('---Comprobar Datos Locales---')
        getStoredPolicy.then(async respPolicy => {
            let storedPolicyLocal: PolicyInterface =
respPolicy != null ? JSON.parse(respPolicy) : null;
            if (storedPolicyLocal != null) {
                setNumPoliza(storedPolicyLocal.phone)
                //setCodImei(storedDeviceLocal.uniqueId)
                //setPolizaLoaded(true);
                let polizaStatus = storedPolicyLocal.status;
                //Búscamos la póliza guardada localmente en
Firebase

getPolicyByRef(storedPolicyLocal.key).then((poliza:
PolicyInterface) => {
                    if (poliza == null) {
                        clearAll()
                    } else {
                        controlFlujo(poliza)
                    }
                })

            }
        })


        /*        getStoredDevice.then(async respDevice => {
                    let storedDeviceLocal: DeviceInterface =
respDevice != null ? JSON.parse(respDevice) : null;
                    if (storedDeviceLocal != null) {
                        if (storedDeviceLocal.uniqueId != null
&& storedDeviceLocal != "") {
                            getStoredPolicy.then(async
respPolicy => {
                                let storedPolicyLocal:
PolicyInterface = respPolicy != null ? JSON.parse(respPolicy) :
null;
                                if (storedPolicyLocal != null)
{

setNumPoliza(storedPolicyLocal.phone)

setCodImei(storedDeviceLocal.uniqueId)
                                    setPolizaLoaded(true);
                                }
                            }).catch(error => {
                                console.error(error);
                                clearAll()
                            });
                        } else {
                            clearAll()
                        }
                    }else{
                        clearAll()
                    }
                }).catch(error => {
                    console.error(error);
                    clearAll()
                });*/
    }

    const checkDevice = async (policy) => {
        let isDeviceAsociado = policy.deviceId != null &&
policy.deviceId != "" && typeof policy.deviceId != 'undefined'
        let deviceExists = false
        if (isDeviceAsociado) {
            deviceExists = await
checkDeviceExists(policy.deviceId); // Función para verificar
si ya existe un dispositivo asociado
            if (!deviceExists) {
                //await deleteDeviceIdByKey(policy.key)//TODO
            }
        }
        return isDeviceAsociado && deviceExists
    }

    const resumeInit = (policy) => {

getDeviceByDeviceIdPromise(policy.deviceId).then((device:
DeviceInterface) => {
            setNumPoliza(policy.phone)
            setCodImei(device.uniqueId)
            setPolizaLoaded(true);
        })
    }

    const controlFlujo = async (policy) => {
        console.log('---Control de flujo cuando hay datos
guardados---')
        console.log('Poliza firebase:')
        console.log('Status: ', policy.status)
        console.log('Id: ', policy.key)
        if (policy.status === 0) {
            let isDevice = await checkDevice(policy)
            console.log('isDevice: ', isDevice)
            if (!isDevice) {
                clearAll();
            } else {
                //Si ya existe un dispositivo asociado,
continuar de forma normal.
                resumeInit(policy)
            }
        } else if (policy.status === 1 || policy.status === 2)
```

```
{
            let isDevice = await checkDevice(policy)
            if (!isDevice) {
                //Si no existe un dispositivo asociado,
                // crear uno y pasar la póliza a estado 0
                // (esto indica que algo ha pasado con el
dispositivo anterior y se debe resetear).
                updatePolicyStatusByKeyAndStatus(policy.key, 0)
                clearAll();
            } else {
                //Si ya existe un dispositivo asociado,
continuar de forma normal.
                resumeInit(policy)
            }
        } else if (policy.status > 2) {
            let isDevice = await checkDevice(policy)
            if (!isDevice) {
                //Si no existe un dispositivo asociado, logOut
                clearAll();
            } else {

existsActivePolicyByPhone(policy.phone).then(existe => {
                    if (existe == 0) {
                        clearAll();
                    } else if (existe == 2) {

setMsgError(StrConstants.no_policy_active_text_info)
                        setShowAlertError(true)
                    } else if (existe == 1) {

setMsgError(StrConstants.dup_policy_active_text_info)
                        setShowAlertError(true)
                    }
                })
            }

        }
    }

    const clearAll = async () => {
        console.log('clearAll')
        setTextoMostrar(0)
        setWaitingCode(false)
        setIsLoading(false)
        setCodSms('')
        setCodImei('')
        setNumPoliza('')
        await getSmsAuthTimeout().then(time => {
            console.log('setTimeRemaining: ', time)
            setTimeRemaining(time)
        }).catch((error) => {
            setTimeRemaining(30)
        });
        setCodPoliza('')
        setCodPolizaInput(0)
        await AsyncStorage.removeItem('@policy')
        await AsyncStorage.removeItem('@device')
        await AsyncStorage.removeItem('@auxLoginPolicy')
        await AsyncStorage.removeItem('@auxLoginPhone')
        await AsyncStorage.removeItem('@isAuxLogin')
        if (firebase.auth().currentUser != null) {
            firebase.auth().signOut()
        }
    }

    const checkCodImei = (newCodImei) => {
        try {
            if (isValidIMEI(newCodImei)) {
                setCodImei(newCodImei);//TODO: Replicar
primero el error que han tenido
            } else {
                console.log('El IMEI ingresado no es válido.
Por favor, ingrese un IMEI válido.');
                //TODO: ALerta IMEI no válido
            }
        }catch (e) {
            console.log('El IMEI ingresado no es válido. Por
favor, ingrese un IMEI válido.');
            //TODO: ALerta IMEI no válido
        }
    }

    function isValidIMEI(imei: string): boolean {
        // La expresión regular verifica que el IMEI tenga
exactamente 15 dígitos numéricos
        const imeiRegex = /^\d{15}$/;

        if (!imeiRegex.test(imei)) {
            return false;
        }

        // Calcular y validar el dígito de verificación del
IMEI (algoritmo de Luhn)
        let sum = 0;
        let mul = 2;
        let luhnDigit = 0;

        for (let i = 14; i >= 0; i--) {
            let digit = parseInt(imei.charAt(i), 10);
            let tp = digit * mul;

            if (tp >= 10) {
                sum += tp % 10 + Math.floor(tp / 10);
            } else {
                sum += tp;
            }

            // Alternar multiplicador entre 1 y 2
            mul = mul === 1 ? 2 : 1;
        }

        // El dígito de verificación Luhn es el valor que se
suma al total para hacerlo un múltiplo de 10
        if (sum % 10 !== 0) {
            luhnDigit = 10 - (sum % 10);
        }

        // Comparar el dígito de verificación Luhn con el
último dígito del IMEI
        return parseInt(imei.charAt(14), 10) === luhnDigit;
    }

    const isLastAppVersion = async () => {
        try {
            const currentAppVersion = StrConstants.VERSION_NUM;
            const latestAppVersion = await getLastAppVersion();

            if (latestAppVersion != currentAppVersion) {
                return false;
            } else {
                return true;
            }
        } catch (error) {
            console.error("Error al comprobar la versión de la
aplicación:", error);
            return false;
        }
    };

    useEffect(() => {
        isLastAppVersion().then(isLast =>{
            if (!isLast){
                alertError(errConstants.update_app_error);
            }
        })
    }, []);
    useEffect(() => {
        isLastAppVersion().then(isLast =>{
            if (isLast){
                if (!permissionGranted &&
firebase.auth().currentUser == null) {
                    showAlertPermissions()
                }
                const subscriber =
firebase.auth().onAuthStateChanged(onAuthStateChanged);
                return subscriber; // unsubscribe on unmount
            }
        })
    }, []);

    useEffect(() => {
        isLastAppVersion().then(isLast =>{
            if (isLast){
                navigation.addListener('focus', () => {
                    console.log('---focus---')
                    setPolizaLoaded(false);
                    // Aquí puedes hacer algo cuando la
pantalla esté enfocada
                    SplashScreen.hide();
                    if (firebase.auth().currentUser != null) {
                        comprobarDatosLocales()
                    }

                });
            }
        })
    }, [permissionGranted, navigation]);

    useEffect(() => {
        isLastAppVersion().then(isLast =>{
            if (isLast){
                if (polizaLoaded) {
                    setIsLoading(true);
                    getPolice()
                }
            }
        })
    }, [polizaLoaded, numPoliza]);


    const [modalVisible, setModalVisible] = useState(false);
    const [value, setValue] = useState('');
    const [modalPromise, setModalPromise] = useState(null);

    const handleOpenModal = () => {
        setModalVisible(true);
        return new Promise((resolve, reject) => {
            setModalPromise({resolve, reject});
        });
    };

    const handleCloseModal = (text) => {
        setModalVisible(false);
        if (text) {
            setValue(text);
        }
        if (modalPromise) {
            modalPromise.resolve(text);
            setModalPromise(null);
        }
    };

    async function handleOpenSettings() {
        try {

Linking.sendIntent('android.settings.DEVICE_INFO_SETTINGS')
            Clipboard.setString('')
            setTimeout(async () => {
                Clipboard.getString().then(imei => {
                    if (imei && /^\d+$/.test(imei)) {
                        setCodImei(imei);
                        //TODO: Habilitar input(deshabilitarla
antes)
                    }
                });

            }, 1000);

        } catch (error) {
            console.error(error);
        }
    }

    const [formattedTime, setFormattedTime] =
useState(`${Math.floor(timeRemaining / 60)
        .toString()
        .padStart(2, '0')}:${(timeRemaining %
60).toString().padStart(2, '0')}`);

    const startTimer = () => {
        console.log('Timer started: ');

        let id = setInterval(() => {
            setTimeRemaining((timeRemaining) => {
                if (timeRemaining > 0) {
                    setFormattedTime(
                        `${Math.floor((timeRemaining - 1) / 60)
                            .toString()
                            .padStart(2,
'0')}:${((timeRemaining - 1) % 60).toString().padStart(2,
'0')}`,
                    );
                    return timeRemaining - 1;
                } else {
                    clearInterval(id)
                    return 0;
                }
            });
        }, 1000);
    };

    return (
        <SafeAreaView style={styles.safeAreaView}>

            {isLoading ? (
                <View style={styles.loadingIndicator}>
                    <ActivityIndicator color={'blue'}
size={50}/>
                </View>
            ) : (
                <ScrollView
contentContainerStyle={styles.scrollView}>
                    <View style={styles.containerLogo}>
                        <Image
                            source={imageSource}
                            style={styles.logo}
                            resizeMode="contain"
                            resizeMethod="resize"
                        />
                    </View>
                    {textoMostrar == 0 ? (
                        <View style={styles.containerTxt}>
                            <Text name="welcome_text"
style={styles.txtNumber1}>
                                {StrConstants.welcome_text}
                                <Text name="welcome_text"
style={styles.txtNumber}>

{StrConstants.polizaPlaceholder}
                                </Text>
                                .
                            </Text>
                        </View>
                    ) : (
                        <></>
                    )}
                    {textoMostrar == 1 ? (
                        <View style={styles.containerTxt}>
                            <Text name="welcome_text"
style={styles.txtNumber1}>
                                {StrConstants.login_sms_text}
                            </Text>
                        </View>
                    ) : (
                        <></>
                    )}
                    {textoMostrar == 2 ? (
                        <View style={styles.containerTxt}>
                            <Text name="welcome_text"
style={styles.txtNumber1}>
                                {StrConstants.imei_text1}
                                <TouchableWithoutFeedback
                                    style={styles.touchAjustes}
                                    onPress={() => {
                                        Platform.OS === 'ios'
                                            ?
Linking.openURL('App-Prefs:root=General&path=About')
                                            :
handleOpenSettings();
                                    }}>
                                    <Text
style={styles.txtAjustes}>
                                        {' ' +
StrConstants.ajustes_str}
                                    </Text>
                                </TouchableWithoutFeedback>
                                {StrConstants.imei_text2}
                            </Text>
                        </View>
                    ) : (
                        <></>
                    )}
                    {textoMostrar == 3 ? (
                        <View style={styles.containerTxt}>
                            <Text name="welcome_text"
style={styles.noCode_title}>
                                {StrConstants.noCode_title}
                            </Text>
                            <Text name="welcome_text"
style={styles.txtNumber1}>
                                {StrConstants.noCode_text1}
                            </Text>
                            <Text name="welcome_text"
style={styles.txtNumber1}>
                                {StrConstants.noCode_text2}
                                <TouchableWithoutFeedback
                                    style={styles.touchAjustes}
                                    onPress={() => {
                                        //resetTimer()
                                        setIsReintento(1)
                                        //showAlert()
                                        clearAll()
                                    }}>
                                    <Text
style={styles.txtAjustes}>

{StrConstants.noCode_text2_2}
                                    </Text>
                                </TouchableWithoutFeedback>
                            </Text>

                            <Text style={styles.txtNumber1}>
                                {StrConstants.noCode_text3}
                            </Text>
                                <TextInput
                                    value={codPoliza}
                                    onChangeText={newCodPoliza
=> setCodPoliza(newCodPoliza)}

textContentType="oneTimeCode"
                                    style={styles.inputPoliza}

placeholder={StrConstants.noCode_text5}
                                    keyboardType="numeric"

placeholderTextColor={'#0000AA'}
                                />

                        </View>
                    ) : (
                        <></>
                    )}

                    <View style={styles.containerTxtInput}>
                        {textoMostrar == 0 ? (
                            <TextInput
                                autoComplete="tel"
                                value={numPoliza}
                                onChangeText={newNumPoliza =>
{
                                    setNumPoliza(newNumPoliza)
                                }}
textContentType="telephoneNumber"
                                style={styles.inputPoliza}
placeholder={StrConstants.polizaPlaceholder}
                                keyboardType="phone-pad"
placeholderTextColor={'#0000AA'}
                            />
                        ) : (
                            <></>
                        )}

                        {textoMostrar == 1 ? (
                            <TextInput
                                autoComplete="tel"
                                value={codSms}
                                onChangeText={newCodSms =>
setCodSms(newCodSms)}
                                textContentType="oneTimeCode"
                                style={styles.inputPoliza}
placeholder={StrConstants.set_code_opt}
                                keyboardType="numeric"
placeholderTextColor={'#0000AA'}
                                disabled={waitingCode}
                            />
                        ) : (
                            <></>
                        )}

                        {textoMostrar == 2 ? (
                            <TextInput
                                autoComplete="tel"
                                value={codImei}
                                onChangeText={newCodImei =>
checkCodImei(newCodImei)}
                                style={styles.inputPoliza}
placeholder={StrConstants.set_imei_opt}
                                keyboardType="numeric"
placeholderTextColor={'#0000AA'}
                            />
                        ) : (
                            <></>
                        )}
```

```
                    </View>

                    {waitingCode ? (
                        <View>
                            <ActivityIndicator color={'blue'}
size={50}/>
                        </View>
                    ) : (
                        <></>
                    )}

                    <View
style={styles.loginContinueContainer}>
                        {textoMostrar == 0 ? (
                            <TouchableOpacity
                                disabled={numPoliza == ''}
                                onPress={() => {
                                    if
(!numPoliza.includes('+')) {
                                        console.log('El numero
no incluye un prefijo, añadimos +549(Argentina)')
                                        setNumPoliza('+549' +
numPoliza)
                                    } else {
                                        console.log('El numero
incluye un prefijo')
                                    }
                                    console.log('Número: ',
numPoliza)
                                    _onPressButtonPoliza();
                                }}
                                style={[
                                    styles.loginContinueBtn,
                                    {backgroundColor:
numPoliza == '' ? '#ccc' : StrConstants.main_btn_color}
//Cambiar el color a gris si está deshabilitado
                                ]}>
                                <Text
style={styles.loginContinue}>
                                    {StrConstants.btn_continue}
                                </Text>
                            </TouchableOpacity>
                        ) : (
                            <></>
                        )}
                        {textoMostrar == 1 ? (
                            <TouchableOpacity
                                disabled={codSms == ''}
                                onPress={() => {
                                    _onPressButtonSms();
                                }}
                                style={[
                                    styles.loginContinueBtn,
                                    {backgroundColor: codSms
== '' ? '#ccc' : StrConstants.main_btn_color} //Cambiar el
color a gris si está deshabilitado
                                ]}>
                                <Text
style={styles.loginContinue}>
                                    {StrConstants.btn_continue}
                                </Text>
                            </TouchableOpacity>
                        ) : (
                            <></>
                        )}
                        {textoMostrar == 2 ? (
                            <TouchableOpacity
                                disabled={codImei == ''}
                                onPress={() => {
                                    _onPressButtonImei();
                                }}
                                style={[
                                    styles.loginContinueBtn,
                                    {backgroundColor: codImei
== '' ? '#ccc' : StrConstants.main_btn_color} //Cambiar el
color a gris si está deshabilitado
                                ]}>
                                <Text
style={styles.loginContinue}>
                                    {StrConstants.btn_continue}
                                </Text>
                            </TouchableOpacity>
                        ) : (
                            <></>
                        )}
                        {textoMostrar == 3 ? (
                            <TouchableOpacity
                                disabled={codPoliza == ''}
                                onPress={() => {
                                    _onPressButtonCodPoliza();
                                }}
                                style={[
                                    styles.loginContinueBtn,
                                    {backgroundColor:
codPoliza == '' ? '#ccc' : StrConstants.main_btn_color}
//Cambiar el color a gris si está deshabilitado
                                ]}>
                                <Text
```

```
style={styles.loginContinue}>
                                    {StrConstants.btn_continue}
                                </Text>
                            </TouchableOpacity>
                        ) : (
                            <></>
                        )}
                    </View>

                    {currentFlavor == 'flavor2' ? (
                        <View
style={styles.noSmsCodeTxtContainer}>
                            {textoMostrar == 1 ? (
                                <View
style={styles.containerTxt}>
                                    {timeRemaining === 0 ? (
                                        <Text
name="noSmsCodeTxt" style={
                                            [

styles.noSmsCodeTxt,
                                                {color:
StrConstants.main_color}
                                            ]
                                        }

                                              onPress={() => {

setTextoMostrar(3)
                                              }}>

{StrConstants.noSmsCodeTxt}
                                        </Text>
                                    ) : (
                                        <Text
name="noSmsCodeTxt" style={styles.noSmsCodeTxt}>

{StrConstants.noSmsCodeTxt} {formattedTime}
                                        </Text>
                                    )}
                                </View>
                            ) : (
                                <></>
                            )}
                        </View>
                    ) : (
                        <></>
                    )}
                    <View style={styles.versionContainer}>
                        <Text style={styles.versionTxt}>
                            {StrConstants.version}
{StrConstants.VERSION_NAME}
                        </Text>
                    </View>
                </ScrollView>
            )}

            <View>
                <LegalsModal visible={modalVisible}
onClose={handleCloseModal}/>
            </View>

            <CustomAlert
                visible={showAlertPerm}

message={StrConstants.info_message_doingdiagnostic}
                confirmText={StrConstants.aceptar}
                onConfirm={() => {
                    setShowAlertPerm(false)
                    setIsLoading(false);
                    askPermission();
                }}
            >
            </CustomAlert>

            <CustomAlert
                visible={showAlertError}
                message={msgError}
                confirmText={StrConstants.aceptar}
                onConfirm={() => {
                    clearAll()
                    setShowAlertError(false);
                }}
            >
            </CustomAlert>

            <CustomAlert
                visible={showAlertUpdateApp}
                message={msgError}
                confirmText={StrConstants.actualizar}
                onConfirm={() => {
                    clearAll()
                    openStoreLink()
                    setShowAlertError(false);
                }}
                cancelText={StrConstants.cerrar}
                onCancel={() => {
                    clearAll()
                    //TODO: Close app
                }}
            >
            </CustomAlert>

            <CustomAlert
                visible={showAlertDebug}
                message={msgDebug}
                confirmText={StrConstants.aceptar}
                onConfirm={() => {
                    setShowAlertDebug(false);
                }}
            >
            </CustomAlert>

            <CustomAlert
                visible={showAlertPolizaAsignada}

message={StrConstants.polica_existente_code_error}
                confirmText={StrConstants.aceptar}
                onConfirm={() => {
                    setShowAlertPolizaAsignada(false);
                    clearAll()
                }}
            >
            </CustomAlert>

            <CustomAlert
                visible={showAlertFb}
                message={StrConstants.send_sms.replace('%s',
numPoliza)}
                cancelText={StrConstants.cancelar}
                confirmText={StrConstants.aceptar}
                onCancel={() => setShowAlertFb(false)}
                onConfirm={() => {
                    setShowAlertFb(false);
                    try {
                        //Firebase SignIn With PhoneNumber

signinWithPhoneNumber(numPoliza).then(_r =>

console.log('signInWithPhoneNumber'),
                        );
                    } catch (e) {
                        console.error(e);
                    }
                }}
            >
            </CustomAlert>

            <CustomAlert
                visible={showAlertDiagnosticDone}
                titles={StrConstants.diagnostic_sent_title}
                message={StrConstants.diagnostic_sent}
                confirmText={StrConstants.aceptar}
                onCancel={() =>
setShowAlertDiagnosticDone(false)}
                onConfirm={() => {
                    setShowAlertDiagnosticDone(false)

navigation.navigate(StrConstants.mi_poliza);
                    setIsLoading(false);
                }}
            >
            </CustomAlert>

        </SafeAreaView>
    );
};

const windowHeight = Dimensions.get('window').height;
const windowWidth = Dimensions.get('window').width;


const styles = StyleSheet.create({
    safeAreaView: {
        flexGrow: 1
    },
    scrollView: {
        flexGrow: 1,
        marginHorizontal: 24,
    },
    containerLogo: {
        justifyContent: 'center',
        alignItems: 'center',
        marginTop: 8
    },
    logo: {
        width: windowWidth,
        height: windowHeight * 0.3,
        resizeMode: 'contain'
    },
    containerTxt: {
        marginTop: 0,
        padding: 16,
        paddingTop: 0,
        fontSize: 18,
    },
    txtNumber1: {
        alignSelf: 'stretch',
        fontSize: 16,
        ...Platform.select({
            ios: {fontFamily: 'Arial'},
            android: {fontFamily: 'gilroy_semibold_regular'},
        }),
    },
    txtNumber: {
        alignSelf: 'stretch',
        color: StrConstants.main_color,
        fontSize: 16,
    },
    containerTxtInput: {
        flex: 0,
        flexGrow: 1,
        padding: 16,
        justifyContent: 'center',
        alignItems: 'center',
    },
    inputPoliza: {
        alignSelf: 'stretch',
        marginTop: 10,
        paddingBottom: 0,
        fontSize: 18,
        justifyContent: 'center',
        alignItems: 'center',
        color: StrConstants.main_color,
        borderBottomColor: StrConstants.main_color,
        borderBottomWidth: 1,
    },
    inputPolizaPrev: {
        marginTop: 16,
        alignSelf: 'stretch',
        paddingBottom: 0,
        fontSize: 20,
        justifyContent: 'center',
        alignItems: 'center',
        color: StrConstants.main_color
    },
    loginContinueContainer: {
        flexGrow: 1,
        flex: 1,
        marginTop: 50,
        flexDirection: 'column',
        justifyContent: 'flex-end',
        alignItems: 'center',
        paddingBottom: 16,
    },
    loginContinueBtn: {
        padding: 16,
        backgroundColor: StrConstants.main_btn_color,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        borderRadius: 6,
    },
    loginContinue: {
        color: '#ffffff',
        fontWeight: 'bold',
        fontSize: 18,
    },
    noSmsCodeTxtContainer: {
        flexGrow: 1,
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'flex-end',
        alignItems: 'center',
        paddingBottom: 16,
    },
    noSmsCodeTxt: {
        color: '#ccc',
        alignSelf: 'stretch',
        marginBottom: 16,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        fontSize: 16,
        textAlign: 'center', // <-- Ajusta aquí la propiedad
textAlign
        ...Platform.select({
            ios: {fontFamily: 'Arial'},
            android: {fontFamily: 'gilroy_semibold_regular'},
        }),
    },
    versionContainer: {
        marginBottom: 16,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
    },
    versionTxt: {
        color: StrConstants.main_color,
    },
    headerView: {
        zIndex: 10,
```



```
        elevation: 10,
        marginBottom: 50,
    },
    txtAjustes: {
        color: 'blue',
        alignSelf: 'stretch',
        fontSize: 16,
        ...Platform.select({
            ios: {fontFamily: 'Arial'},
            android: {fontFamily: 'gilroy_semibold_regular'},
        }),
        padding: 0,
        margin: 0,
        textDecorationLine: 'underline',
    },
    touchAjustes: {
        padding: 0,
        margin: 0,
    },
    loadingIndicator: {
        height: windowHeight - 50,
        justifyContent: 'center',
        alignItems: 'center',
    },
    noCode_title: {
        color: '#5b5a5a',
        alignSelf: 'stretch',
        fontSize: 17,
        ...Platform.select({
            ios: {fontFamily: 'Arial'},
            android: {fontFamily: 'gilroy_semibold_regular'},
        }),
    },
});
```