Aplicación móvil desarrollada en React Native para la gestión de créditos financieros.

Objetivos de la aplicación móvil.
La aplicación debe recabar toda la información necesaria para realizar una evaluación de crédito

Flujo de pantallas de la aplicación móvil:

- Obtener toda la información necesaria para crear una solicitud de crédito

  La información para el crédito debe ser:

  - **Información del cliente**
    - Datos Basicos
    - Domicilio (Long y Lat de vivienda)
    - Datos laborales
    - Datos del empleo o negocio (Incluye long y lat)
    - Datos adicionales
    - Fotos de identificación vista frontal y vista trasera
  - **Información de la solicitud**
  - **Evaluación financiera**
    - Evaluación Financiera
    - Situación Patrimonial
  - **Referencias Personales**
    - Pueden ser de tipo Personales, Comerciales y Bancarias
  - **Fotos del negocio**
    - Fotografías del negocio
    - Documentos como (Permiso Operación, Estado Financiero)
  - **Avales/Garantías de la solicitud**

La Aplicación móvil debe contener un drawer menu con las siguientes opciones:

- **Gestión de solicitudes**: al dar click debe llevar a la sección de solicitudes en proceso en el cual aparecerá una lista de las solicitudes pendientes de enviar al comité de crédito para la discusión de aprobación.
- **Datos Almacenados**: Se mostrará un componente que cuente con tabs donde se vea toda la información que hay almacenada por tabla en la base de datos local.
- Solicitudes enviadas: Similar a la gestión de solicitudes solo que esta validará el estado de la solicitud para mostrar solo las enviadas y no podrán ser editadas a menos que cuente con un parámetro que diga que necesita ser actualizada.
- **Opción de cerrar sesión**.

Login para inicio de sesión

El usuario debe colocar el usuario y la contraseña y se enviará al servidor una solicitud de login y esta responderá con un success true o false, si es true se debe invocar un endpoint donde se le va a enviar al servidor un payload con la información de los catálogos que ya están cacheados en la aplicación móvil, y se validaron por fecha si la fecha del servidor es superior a

la de la aplicación móvil ese catálogo en la tabla se va a actualizar y se hará esto por cada tabla de catálogo que se va a definir.

Una vez logueado y el usuario accede a la app no va a requerir volverse a loguear pero se va a consultar al servidor si hay nuevos datos en los catálogos para ser actualizados en una sección.

Los catálogos inicialmente serán

- Datos del usuario
- Tipo Cliente
- Tipo Identificación
- Paises
- Sector Economico
- Ocupaciones
- Grupo Economico
- Estado Civil
- Ciudades
- Departamentos
- Agencias
- Aldeas
- Clases
- Destinos
- Tipo de Operaciones
- Actividad Financiada
- Fondos
- Programas
- Tipo de Garantia
- Lugar Poblado
- Clase Fondo Programa
- Clases Plazo
- TpObligacion
-

También se va a visualizar información de los registros Almacenados en local en la app

- Cliente
- Solicitudes Locales
- Solicitudes
- Avales
- Garantias
- Referencias Locales
- SLC Destinos
- Img Solicitud
- Secciones completas

Información Técnica de la Aplicación Móvil

- Version de Node JS: v16.16.0
- Version de React JS: "18.2.0"
- Version de React Native: "0.71.4"
-

Cómo va a manejar la los formularios para recolectar la información

- Los formularios de van a manejar con Formik en la versión, se usarán validate de formik customizados para hacer validaciones dinámicas según cambios en algunos campos, pero no se va a utilizar YUP.
- Cuando se le de click en el submit de formik, se deberá crear una copia de ese archivo en localStorage

Como se van a manejar los estilos

- Los estilos van a ser generados con la librería “tailwind-react-native-classnames” estos estilos son creados con las clases de Tailwind CSS y cuando se genere un nuevo componente se debe incluir el estilo de esta forma

  “

  ```
  // Se importa
  import { className } from '../../../../utils/className' // Esta función abstrae la librería

  // Ejemplo de cómo se aplica un estilo
  // Los estilos en dark mode se escriben a si “dark:bg-gray-900” este es un ejemplo
  <Text style=className "AQUÍ DEBEN IR LAS CLASES DE TAILWIND CSS"  >
          </Text>
  ```

  ”

Como se debe escribir el código

- El código debe crear en componentes aislados y las funciones deben comentarse siguiendo las mejores prácticas con JSDocs, los comentarios deben ser cortos y claros,
- Existirá un archivo llamado functionsHelpers y aqui se van a ir creando como módulos de javascript aquellas funciones que puedan ser repetitivas.

Ruta de Gestión de Solicitudes

- Cards de información de centrales de riesgo
- Datos Generales (Botón de enlace)
- Solicitud (Boton de enlace)
- Avales Garantías (Card con listado y botón de agregar)
- Evaluacion Financiera (Botón de enlace)
- Referencias card con listado y botón de agregar
- Fotos del Negocio (Botón de enlace)

- **Datos Generales**

ButtonTab	**Datos Basicos**			
Nombre Visual	**Tipo de**	**Componente**	**FormikPropName**	**Validate**

	Dato			
Tipo de Persona	string	CustomDropdown	personType	
Nombres	string	TextField	firstName	
Apellidos	string	TextField	secondName	
Seudonimo	string	TextField		
Tipo de Identificación	string	CustomDropdown	identificationType	
No. DNI	string	TextField	identifcation	
Emitida En	string	TextField		
Nacionalidad		CustomDropdownSearch	country	
Lugar de Nacimiento	string	Textarea	birthPlace	
Reg tributario (opcional)	string			
Fecha de nacimiento	string	Date	birthDate	
Genero	string	CustomDropdown	gender	
Estado Civil	string	CustomDropdown	civilState	
Nivel Educativo	string	CustomDropdown	educationLevel	
No. Dependientes	int	TextField		
Hijos Dependientes	int	TextField		

- **Domicilio (Long y Lat de vivienda)**

ButtonTab	**Domicilio**			
Nombre del campo	**Tipo de Dato**	**Componente**	**FormikValue**	**Validate**
Estado/Departamento	string	CustomDropdownSearch		
Ciudad/Municipio	string	CustomDropdownSearch		
Barrio/Colonia	string	CustomDropdownSearch		
Aldea/Barrio	string	CustomDropdownSearch		
Dir Completa	string	Textarea		
Tipo de Residencia	string	CustomDropdown		
Tiempo de Residir	string	CustomDropdown		
Telefono Residencia	string	Textfield		
Teléfono Móvil	string	Textfield		

Modal		Modal mapa para seleccionar un punto de ubicación de la vivienda del cliente		

- **Datos laborales**

ButtonTab	**Datos Laborales**			
Nombre del campo	**Tipo de Dato**	**Componente**	**FormikValue**	**Validate**
Ocupación	string	CustomDropdown		
Sabe Leer:	bool	Switch		
Sabe Escribir:	bool	Switch		
Tipo de Empleado	string	CustomDropdown		
Ocupación	string	CustomDropdown		
Salario/Ingreso Mensual	int	TextField		
Otros Ingresos	int	TextField		
Fuente Otros Ingresos	int	TextField		

- **Datos del empleo o negocio (Incluye long y lat)**

-

ButtonTab	**Datos Laborales**			
Nombre del campo	**Tipo de Dato**	**Componente**	**FormikValue**	**Validate**
Nombre del Negocio/Empresa	string	TextField		
Dirección Completa	string	TextField Multiline 3		
Cargo o Puesto	string	TextField		
Fecha de inicio a trabajar - Tipo Fecha	Date	TextField Date Format		
Telefono	string	TextField		
Telefono Fax	string	TextField		
Card con Resumen Salarial (Semanal, Catorcenal, Mensual)				

-

- **Datos adicionales**
  - Grupo Económico - Tipo Select
  - Sector Economico - Tipo Search
  - Medio de comunicación principal: Tipo Select
  - Email
- **Fotos de identificación vista frontal y vista trasera**
  - Card para agregar Foto Identificación - Parte Frontal
  - Card agregar foto Identificación - Parte Trasera

- **Sección Solicitud**

Page	**Sección de Solicitud**			
Nombre del campo	**Tipo de Dato**	**Componente**	**FormikValue**	**Validate**
Fondo	string	CustomDropdown		
Programa	string	CustomDropdown		
Clase de Colocaciones (Tipo Producto)	string	CustomDropdown		
Tipo / Operaciòn	string	CustomDropdown		
Moneda	string	CustomDropdown		
Tipo de Crédito	string	CustomDropdown		
Actividad Financiada	string	CustomDropdown		
Destino	string	CustomDropdownSearch		
Tipo Renegociación	string	CustomDropdown		
Nivel Riesgo - Renegociación	string	CustomDropdown		
Tipo de Plan	string	CustomDropdown		
Frecuencia de Pago	string	CustomDropdown		
Card Información del Producto Seleccionado		Card Informativa		
Monto Original	int	TextField		
Prima / Anticipo	int	TextField		
Monto Solicitado	int	TextField		
Tasa Anual	float	TextField		
Plazos	float	TextField		
Valor de la Cuota	float	TextField		

Interes	float	TextField		
Forma de Pago	string	CustomDropdown		

- **Sección Crear/Editar Referencia**

Page	**Sección de Solicitud**			
Nombre del campo	**Tipo de Dato**	**Componente**	**FormikValue**	**Validate**
Tipo Referencia	string	CustomDropdown		
Nombre del Referente	string	TextField		
Es Familiar	bool	Switch		
Dirección Actual	string	TextField		
Tipo de Residencia	string	CustomDropdown		
Tiempo de Residir	string	CustomDropdown		
Teléfono de Residencia	string	TextField		
Teléfono Móvil	string	TextField		
Telefono de Trabajo	string	TextField		
Correo Electronico	string	TextField		
Relacion	string	TextField		
Observacion	string	TextField Ampliado		

- **Sección Garantía Prendaria**
  - Hola
- **Sección Garantía Hipotecaria**
  - Hola
- **Seccion Garantia Fiduciaria**
  - Hola
- **Sección Otras Garantías**
  - Hola
- **Seccion Evaluacion Financiera**
  - **Evaluación Financiera**
    - Card Ventas
      - Buena
      - Regular
      - Baja
      - Rec CxC
      - Total Mensual

- **Otros Ingresos**
    - Conyuge
    - Otros Negocios
    - Remesas
    - Salarios
    - Total
- **Gastos Operativos**
    - Alquiler
    - Aguar
    - Electricidad
    - Telefono
    - Impuestos
    - Transporte
    - Salarios
    - Mant y Renta
    - Gas/Leña
    - CTAS. PRESTAMOS
    - Otros
    - Total
- **Gastos Familiares**
    - Alimentacion
    - Vestuario
    - Vivienda
    - Salud
    - Educacion
    - Transporte
    - Agua
    - Electricidad
    - Gas
    - Telefono
    - Otros
    - Total
- **ESTADO DE GANANCIAS Y PÉRDIDAS**
    - Ingreso por ventas
    - Menos Costo de Ventas
    - Utilidad Bruta
    - Menos Gastos Operativos
    - Utilidad Operativa
    - Menos Gastos Familiares
    - Más Otros Ingresos
    - Ganancias Netas
    - Margen de Confiabilidad 10%
- **GANANCIAS AJUSTADAS**
    - Mensual

        - Catorcenal
        - Semanal
        - Ingresos
        - Egresos
        - Ventas
        - Util. Neta
- **Situación Patrimonial**
    - **Activos**
        - Efectivo
        - Ahorros
        - Cuentas por Cobrar
        - Inventario
        - Maquinaria y equipo
        - Terrenos
        - Vivienda
        - Vehiculos
        - Telefonos
        - Total Activo
    - **Pasivos**
        - Bancos
        - Cooperativa
        - Casas de Empeño
        - Microfinancieras
        - Prestamista
        - Programa de Gobierno
        - Casas Comerciales
        - Anticipos Por Mercancía
        - Familiares
        - Total Pasivos
    -

Cad para cada Aval/Garantia - Estos datos se muestran en la lista de las garantias
- Tipo Garantia
- Cobertura %
- Sueldo/Avaluo
- Tipo de Obligacion
- No. Identidad del aval

Botones: Ir, Guardar, Borrar

FIDUCIARIA - Tab Navigation (Resumen, C.R, Persona Natural, Domicilio, Fotos)
- Tab Resumen
    - Tipo
    - Nombre

    - No. de Cliente
    - Cobertura %
    - Sueldo/Avaluo

- Tab Centrales de Riesgo

    - Boton Consultar Central de Riesgo
    - Nombre completo
    - No. Identidad
    - Fecha de Nacimiento
    - Direccion
    - Fecha de consulta
    - Card Calificacion de riesgo
    - Botones para ir a ver los PDFs

- Tab Persona Natural

    - Tipo de Identificacion
    - No. de Identificacion
    - Fecha de Nacmiento, Edad
    - RTN Opcional
    - Nacionalidad
    - Emitida en
    - Departamento Nacmiento
    - Ciudad Nac
    - Lugar Poblado Nac
    - Genero
    - Estado Civil
    - Relacion
    - Tipo de Generador de Divisas
    - Sector Economico
    - Profesion, Ocupacion u Oficio
    - Ingreso
    - Especificar fuente de ingresos
    - Datos Laborales
    - Giro o Actividad Desempeñada
    - Tenencia Negocio
    - Informacion del Conyugue

- Tab Domicilio

    - Departamento
    - Ciudad/Municipio
    - Col. Bo/Col
    - Direccion completa (Ciudad, Estado, Cod. Postal, Barrio Col, Calle, etc.)
    - Telefono
    - Telefono Movil

- Tab Fotos

    - 10 cards, Ninguna obligatoria

PRENDARIA - BUTTON TAB NAV

- ButtonTab Generales

- Id Registro
- Tipo de Bien
- Descripicon del bien
- Tipo de garantia mobiliaria
- Clasificacion de garantia
- Comentarios, Opcional

- ButtonTab Garantia

- Vin. No.
- Marca
- Modelo
- Año
- Color
- Tipo
- Motor No.
- Chasis No.
- Cilindraje
- Numero de Placa
- Fecha de Adquisicion

- ButtonTab Ubicacion

- Pais
- Estado/Departamento
- Ciudad/Municipio
- Aldea/Barrio
- Direccion completa

- ButtonTab Avaluo

- Valor de Avaluo
- Nombre del Valuador
- Tipo de Identificacion
- Numero de Identificacion
- Informacion Adicional

- ButtonTab Fotos, card 10 fotos, 5 obligatorias

GARANTIA HIPOTECARIA

- ButtonTab Garantia

- Id Registro
- Fecha de adquisicion
- Descripicon
- Tipo de Garantia Select 1
- Tipo de Garantia Select 2
- Registro Propiedad Toma
- Registro Propiedad Folio

- ButtonTab Ubicacion

- Estado/Departamento
- Ciudad/Municipio
- Aldea/Barrio
- Direccion Completa

- ButtonTab Avaluo

- Tipo de Identificacion
- Numero de Identificacion
- Nombre del Valuador
- Valor del Avaluo
- Informacion Adicional

- ButtonTab Fotos, card 10 fotos, 5 obligatorias

OTRAS GARANTIAS

- ButtonTab Generales

- Id Registro
- Descripcion
- Valor
- Informacion Adicional
- Fecha de adquisicion

- ButtonTab Fotos, card 10 fotos, 1 obligatorias

Fotos Negocio ButtonTabNav

- ButtonTab Cliente (Permiso Operacion, Estado Financiero)
- ButtonTab Fotos Negocio, Agregar descripcion de cada imagen

Las cards de las fotos deben tenen en la parte inferior un campo para colocar la descripcion, un boton para tomar foto, seleccionar de la galeria y eliminar

Vista de perfil del usuario

- Datos Generales

- Nombre
- Fecha de Nacimiento
- Profesion

- Datos Laborales

- Roles
- Agencias
- Años en la institucion
- Logros
- Tipo de empleado

- Datos del Contrato

- Tipo
- Fecha Inicio

- Configuracion

- Idioma
- Actualizar contrasena
- Actualizar Informacion

- Ayuda y Asistencia

- Ayuda
- Sugerencias
- Buzon de ayuda
- Informacion
- Informar de un problema

- Seguridad

- Devices (Lista de dispositivos en los que ha iniciado sesion)
- Organizacion
- My sign-inst / (Device, Location, Date)
- Actividad Reciente (Mapa, App, Fecha, Ip, Device, Account)

Sistemas de puntuacion

- Al final un encargado que revisa la solicitud del credito da estrellitas al asesor y deja comentarios de feedback
-

Tipos de Datos

- Tipo Select: Un Select para listas con pocos registros
- Tipo Search: Un select con la posibilidad de buscar un registro por el nombre
- Switch Bool: Switch activar o desactivar guardar 1 o 0
- Tipo Fecha: Usará valores solo de fecha
- Tipo FechaHora: Generará valores con fecha y hora
- Los campos que sean enteros o numéricos no deben permitir valores menores a 0

Gestion de Garantias y Avales

Las garantias y avales son necesarias ya que sirven como respaldo al ente que va a dar el credito, las garantias permitidas son Prendarias, Fiduciarias, Hipoterarias y otros

Pagina Index para administrar las garantias

Una lista de cards, cada una debe llevar los campos

- Tipo de Garantia - CustomDropdown
- Cobertura, no menor a 0 o mayor que 100, entero ya que es porcentaje
- Sueldo/Avaluo TextField Int
- Tipo de Obligacion - CustomDropDown
- No. Identificacion del Aval, TextField string, este solo de debe mostrar si el tipo de garantia es Fiduciaria

Cada card debe tener un boton de Ir, Guardar y Eliminar, para las acciones de eliminar se debe mostrar una alerta de confirmacion, El boton de ir solo debe estar habilitado una vez se haya guardado el registro.dfff

Campos para una Garantía Prendaria

Page	**Garantía Prendaria**			
Nombre del campo	**Tipo de Dato**	**Componente**	**FormikValue**	**Validate**
	string	CustomDropdown		
	string	TextField		
	bool	Switch		
	string	TextField		
	string	CustomDropdown		
	string	CustomDropdown		
	string	TextField		
	string	TextField		
	string	TextField		
	string	TextField		
	string	TextField		
	string	TextField Ampliado		

Componentes según el tipo de campo

Cuando crees un campo Tipo Select debes guiarte con este ejemplo

“

```
const options = [
    { name: 'Honduras', value: '95' },
    { name: 'Option 2', value: 'option2' },
    { name: 'Option 3', value: 'option3' },
    { name: 'Another Option', value: 'anotherOption' },
 ];

<CustomDropdown
        options={options}
        selectedValue={selectedValue}
        onValueChange={handleChange('country')} // Cuando se utilice con Formik
  />
```

”

Cuando crees un campo Tipo Search debes guiarte con este ejemplo

“

```
const options = [
    { name: 'Honduras', value: '95' },
    { name: 'Option 2', value: 'option2' },
    { name: 'Option 3', value: 'option3' },
    { name: 'Another Option', value: 'anotherOption' },
 ];

<CustomDropdownSearch
        options={options}
        selectedValue={selectedValue}
        onValueChange={handleChange('country')} // Cuando se utilice con Formik
  />
```

”

Flujo de datos en estados en la aplicación

- La información se va a ir guardando en localStorage, se creará un estado global con useContext y este se va a encargar de actualizar un objeto global de creditRequest y todos los objetos asociados se van a ir agregando a este en todas las acciones de guardar
- Los campos no van a ser requeridos pero se va a crear una tabla que se llamara secciones completas, con una función se va a evaluar si todos los campos que se agreguen es esta validación se cumplen se cambiara con una bandera se success true/false, y cuando todas las banderas de la solicitud de crédito estén success se va a habilitar el botón de enviar.
- Cuando se creen useEffect se debe evaluar muy bien la lógica de dependencias para prevenir re-renders innecesarios que bajan rendimiento a la aplicación.

Manejo de toma de imágenes de la aplicación móvil.

La aplicación móvil debe ser capaz de tomar, mostrar, almacenar, eliminar o actualizar fotografías en el dispositivo.

Manejo de sincronía de datos con el servidor.

Manejo de errores en la aplicación móvil

- Se va a crear una tabla o archivo donde se van a guardar los errores generados en la aplicación, cuando se inicie la aplicación y se detecte que el dispositivo tiene conexión a internet se enviaran estos logs a el servidor y se van a eliminar los registros de la aplicación.
  Esquema para almacenar los logs de errores
    -

Llamadas a las APIS

- Las peticiones a los servicios se van a manejar utilizando fetch de javascript, todas las peticiones deben incluir el token en el header de la petición.
-

Manejo de Idiomas

- Se va a utilizar la librería i18 para el translate de idiomas
- Todos los textos que se creen deben de llevar este estándar de internacionalización
- Cuando el texto sea en español se generará una nueva propiedad al archivo /es/globlas.json
- Cuando el texto sea en inglés se generará una nueva propiedad al archivo /en/globlas.json

Ejemplo de como implementar la traducción en los componentes

```
import { useTranslation } from "react-i18next";
const { t } = useTranslation();

// Pasandola aún componente
title={`${t("translation:see")}`
```

Nombramiento de componentes, funciones y variables

- Los nombres de los componentes, funciones y variables deben ser en el idioma Ingles, y con jsDoc se debe crear para cada una una descripción breve y clara de lo que hace.

Estilos para campos de texto

Manejo de Formularios

- Los formularios deben crearse utilizando Formik y validaciones custom, no se va a utilizar YUP.
- Deben mantener un estilo visual consistente.
-

Navegación entre pantallas de la aplicación móvil

Para navegar entre pantallas se cuenta con las librerías

```
  "@react-navigation/bottom-tabs": "^6.5.7",
   "@react-navigation/drawer": "^6.6.2",
   "@react-navigation/material-top-tabs": "^6.6.2",
   "@react-navigation/native": "^6.1.6",
   "@react-navigation/native-stack": "^6.9.12",
  "react-native-screens": "^3.20.0",
```

```
   "react-native-tab-view": "^3.5.1",
```

A continuación muestro ejemplos de cómo implementar cada uno de ellos.

Ejemplo de BottomTabNavigation: Este es ideal para pantallas con tabs en la parte inferior

```
‘’’
import React from 'react'
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Tab = createBottomTabNavigator();

const BottomTabNavigation = () => {
  return (
    <Tab.Navigator screenOptions={{
      headerShown: false,
    }}>
      <Tab.Screen name="Home" component={Test} />
      <Tab.Screen name="Settings" component={Test} />
      <Tab.Screen name="Menu" component={Test} />
    </Tab.Navigator>
  )
}

export default BottomTabNavigation

’’’
```

Ejemplo de NaviteStackNavigation

```
‘’’
import { ScrollView, Button } from 'react-native'
import React from 'react'
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

function NativeStackNavigation(): JSX.Element {
   return (
      <Stack.Navigator initialRouteName='Home' screenOptions={{ animation: "slide_from_right"
}}>
         <Stack.Screen name="Home" component={Home} />
         <Stack.Screen name="Notifications" component={Home} />
         <Stack.Screen name="Profile" component={Home} />
         <Stack.Screen name="Settings" component={Home} />
```

```
      </Stack.Navigator>
   )
}

function Home({ navigation }): JSX.Element {
   return (
      <ScrollView>
         <Button title='Go to Home' onPress={() => navigation.navigate('Home')} />
         <Button title='Go to Notifications' onPress={() => navigation.navigate('Notifications')} />
         <Button title='Go to Profile' onPress={() => navigation.navigate('Profile')} />
         <Button title='Go to Settings' onPress={() => navigation.navigate('Settings')} />
      </ScrollView>
   )
}

export default NativeStackNavigation;
‘’’
```

Ejemplo de TopTabsNavigation

```
‘’’import React from 'react'
import { createMaterialTopTabNavigator } from '@react-navigation/material-top-tabs';

const Tab = createMaterialTopTabNavigator();

const TopTabsNavigation = () => {
   return (
      <Tab.Navigator overScrollMode={"always"} screenOptions={{ lazy : true,
tabBarScrollEnabled: true }}>
         <Tab.Screen name="Home1" component={Home1} />
         <Tab.Screen name="Home2" component={Home2} />
         <Tab.Screen name="Home3" component={Home2} />
      </Tab.Navigator>
   )
}

export default TopTabsNavigation

‘’’
```

Ejemplo del DrawerNavigation: Éste es el principal de toda la aplicación

```
‘’’
import { View, Text, Image } from 'react-native'
import React from 'react'
```

```
import { DrawerContentScrollView, DrawerItem, DrawerItemList, createDrawerNavigator } from
'@react-navigation/drawer';
import TopTabsNavigation from './TopTabsNavigation';
import BottomTabNavigation from './BottomTabNavigation';
import NativeStackNavigation from './NativeStackNavigation';
import tw from 'tailwind-react-native-classnames';

const CustomDrawerContent = (props) => {
   return (
      <DrawerContentScrollView {...props} style={tw`bg-gray-100`}>
         <View style={tw`p-4 mb-4 bg-blue-500`}>
            <Image
               source={{ uri: 'https://via.placeholder.com/150' }}
               style={tw`w-20 h-20 rounded-full mb-2`}
            />
            <Text style={tw`text-white text-lg font-bold`}>John Doe</Text>
            <Text style={tw`text-white text-sm`}>john@example.com</Text>
         </View>
         <DrawerItemList {...props} />
         <DrawerItem
            label="Logout"
            onPress={() => {/* Implementa la lógica de logout aquí */ }}
            icon={({ color, size }) => (
               <Text>ola</Text>
            )}
            style={tw`mt-4`}
            labelStyle={tw`text-red-500`}
         />
      </DrawerContentScrollView>
   );
}


const DrawerNavigation = () => {
   const Drawer = createDrawerNavigator();
   return (
      <Drawer.Navigator
         drawerContent={(props) => <CustomDrawerContent {...props} />}
         screenOptions={{
            drawerActiveBackgroundColor: "#4299e1", // Un azul más suave
            drawerActiveTintColor: "#ffffff",
            drawerInactiveTintColor: "#4a5568",
            drawerLabelStyle: tw`text-base font-medium`,
            headerStyle: tw`bg-blue-500`,
```

```
            headerTintColor: "#ffffff",
            headerTitleStyle: tw`font-bold`,

        }}>
        <Drawer.Screen name="Top Navigation"
            component={TopTabsNavigation}
            options={{
                drawerIcon: ({ color, size }) => (
                    <Text >Hola 2</Text>
                ),
            }}
        />
        <Drawer.Screen name="Bottom Navigation" component={BottomTabNavigation} />
        <Drawer.Screen name="Stack Navigation" component={NativeStackNavigation} />
    </Drawer.Navigator>
  )
}

export default DrawerNavigation
‘’’
```

Ejemplo de un formulario con Formik

Json para generar el formulario

```
“
import CATALOGS from '../constants/CATALOGS.json'

export const GUARANTEE_FIELDS = [
  {
    label: "Agregar Nueva Garantia",
    subLabel: "Nueva Garantia",
    component: "Title",
  },
  {
    label: 'Tipo de Garantía',
    name: 'guaranteeType',
    component: 'CustomDropdown',
    options: CATALOGS.guaranteeTypes,
    columns: 2,
  },
  {
    label: 'Cobertura (%)',
    name: 'coverage',
    component: 'TextField',
```

```
    placeholder: 'Cobertura (%)',
    columns: 2,
    customProps: {
      keyboardType: 'numeric',
    },
  },
  {
    label: 'Sueldo ó Avalúo',
    name: 'salaryAvaluation',
    component: 'TextField',
    placeholder: 'Sueldo ó Avalúo',
    columns: 2,
    customProps: {
      keyboardType: 'numeric',
      maxLength: 10,
    },
  },
  {
    label: 'Tipo de Obligación',
    name: 'obligationType',
    component: 'CustomDropdownSearch',
    columns: 2,
    options: CATALOGS.obligationTypes,
    required: true,
  },
 {
   label: 'Es Familiar',
   name: 'isFamily',
   component: 'Switch',
   placeholder: '¿Es familiar?',
   columns: 2,
  },
  {
    label: 'No. Identificación del Aval',
    name: 'avalIdentification',
    component: 'TextField',
    placeholder: 'No. Identificación del Aval',
    conditionalRender: (values) => values?.guaranteeType === '1',
  },
 {
    title: "Guardar",
    component: "Submit",
  },
 {
```

```
    label: 'Forma de Pago',
    subLabel: 'destinoCredito',
    component: 'Title',
  },
 {
      label: 'Total Mensual',
      name: 'totalMensual',
      component: 'TextField',
      placeholder: 'Total mensual',
      columns: 2,
      customProps: {
         keyboardType: 'numeric',
         editable: false,
         inputMode: 'numeric',
      },
      calculation: {
         type: "sum",
         fields: ["ventasBuena", "ventasRegular", "ventasBaja"],
      },
  },

]
”
```

Implementar en formik

```
“
import React from 'react';
import { View, Text, Modal, TouchableOpacity, TextInput } from 'react-native';
import { Formik } from 'formik';
import { className } from '../../../../../../utils/className';
import CustomField from '../../../../../../components/CustomField';
import ErrorBoundary from '../../../../../../components/ErrorBoundary';
import { GUARANTEE_FIELDS } from '../../../../../../fields/GuaranteeListFields';

const GuaranteeModal = ({ visible, onClose, onSubmit, initialValues }) => {
  return (
    <Modal visible={visible} animationType="fade" transparent={true}>
      <View style={className("flex-1 justify-center items-center bg-black bg-opacity-50")}>
        <View style={className("bg-white p-6 rounded-lg w-11/12 max-w-md")}>
          <Formik
            initialValues={initialValues || {
              guaranteeType: '',
              coverage: '',
              salaryAvaluation: '',
```

```
            obligationType: '',
            avalIdentification: '',
          }}
          onSubmit={onSubmit}
        >
          {({ handleChange, handleSubmit, values, setFieldValue }) => (
            <View style={className("flex flex-row flex-wrap")}>
              <ErrorBoundary>
                {GUARANTEE_FIELDS.map((field, index) => (
                  (!field?.conditionalRender || field?.conditionalRender(values)) && (
                    <CustomField
                      key={index}
                      field={field}
                      handleChange={handleChange}
                      values={values}
                      //setFieldValue={setFieldValue}
                      handleSubmit={handleSubmit}
                    />
                  )
                ))}
              </ErrorBoundary>
            </View>
          )}
        </Formik>
      </View>
    </View>

  </Modal>
 );
};

export default GuaranteeModal;
```

Guardar los datos en localStorage

```
import AsyncStorage from '@react-native-async-storage/async-storage';

/**
 * Almacena un valor en AsyncStorage.
 * @param {string} key - La clave para almacenar el valor.
 * @param {string} value - El valor a almacenar.
 */
export const storeData = async (key, value) => {
```

```
  try {
    await AsyncStorage.setItem(key, value);
    console.log(key + 'Datos almacenados exitosamente');
  } catch (e) {
    console.error('Error al almacenar los datos', e);
  }
};

/**
 * Recupera un valor de AsyncStorage.
 * @param {string} key - La clave del valor a recuperar.
 * @returns {string|null} - El valor almacenado o null si no existe.
 */
export const getData = async (key) => {
  try {
    const value = await AsyncStorage.getItem(key);
    if (value !== null) {
      console.log(key + '-Datos recuperados:', value);
      return value;
    } else {
      console.log('No se encontraron datos para la clave:', key);
      return null;
    }
  } catch (e) {
    console.error('Error al recuperar los datos', e);
    return null;
  }
};

/**
 * Elimina un valor de AsyncStorage.
 * @param {string} key - La clave del valor a eliminar.
 */
export const removeData = async (key) => {
  try {
    await AsyncStorage.removeItem(key);
    console.log(key + 'Datos eliminados exitosamente');
  } catch (e) {
    console.error('Error al eliminar los datos', e);
  }
};

”
```

Ejemplo implementar funciones para guardar y recuperar
“

```
import React, { useEffect, useState } from 'react';
import { View, Text, ScrollView, Button } from 'react-native';
import { Formik } from 'formik';
import { className } from '../../../../../utils/className';
import CustomField from '../../../../../components/CustomField';
import { getData, storeData } from '../../../../../utils/storage';
import { CREDIT_REQUEST_FIELDS } from './../../../../../fields/CreditRequestFields';

/**
 * RequestSection - Componente para la sección de solicitud de crédito.
 * @returns {JSX.Element} Componente de React
 */
const RequestDetailPage = ({ credit }) => {
  const [initialValues, setInitialValues] = useState({
    creditRequestHashId: credit?.hashId,
  });
  const [isLoading, setIsLoading] = useState(true);


  useEffect(() => {
    // Recuperar datos al montar el componente
    const fetchData = async () => {
      const storedData = await getData('credit_request_' + credit?.hashId);
      console.log("Datos recuperados credit_request_: ", storedData);

      if (storedData) {
        // Realiza el merge entre los datos recuperados y los valores iniciales existentes
        setInitialValues((prevValues) => ({
          ...prevValues,
          ...JSON.parse(storedData) // Datos recuperados de AsyncStorage
        }));
      }

      setIsLoading(false); // Datos cargados
    };

    fetchData();
  }, []);


  /**
   * Maneja el envío del formulario.
```

```
 * @param {Object} values - Valores del formulario
 */
const handleSubmit = async (values) => {
  console.log(values);
  // Aquí puedes manejar el envío de datos
  console.log('Form values:', values);

  try {
    // Almacena los datos usando await
    await storeData('credit_request_' + credit?.hashId, JSON.stringify(values));
    console.log('Data stored successfully');
  } catch (error) {
    console.error('Error storing data:', error);
  }
};

return (
  <ScrollView style={className('p-4 bg-white  dark:bg-gray-900')}>
    <Formik
      initialValues={initialValues}
      onSubmit={handleSubmit}
      enableReinitialize={true}
    >
      {({ handleChange, handleSubmit, values }) => (
        <View>
          <Text style={className('text-sm text-gray-600')}>
            {JSON.stringify(values)}
          </Text>
          <View style={className('flex flex-row flex-wrap ')}>
            {CREDIT_REQUEST_FIELDS.map((field, index) => (
              <CustomField
                field={field}
                handleChange={handleChange}
                values={values}
                key={index}
                handleSubmit={handleSubmit}
              />
            ))}
          </View>
        </View>
      )}
    </Formik>
  </ScrollView>
);
```

```
};

export default RequestDetailPage;
”
```

Ejemplo de useContext para la gestion de las garantias, este se va a ir mejorando

```
“
import React, { createContext, useContext, useState, useEffect } from 'react';
import { Alert } from 'react-native';
import { getData, storeData } from '../utils/storage';


const GuaranteesContext = createContext();

export const GuaranteesProvider = ({ children, credit }) => {
  const [guarantees, setGuarantees] = useState([]);
  const [editingGuarantee, setEditingGuarantee] = useState(null);
  const [modalVisible, setModalVisible] = useState(false);
  const [selected, setSelected] = useState({});

  const STORAGE_KEY = `guaranteesData_${credit?.hashId}`;

  useEffect(() => {
    const fetchGuarantees = async () => {
      try {
        const storedGuarantees = await getData(STORAGE_KEY);
        if (storedGuarantees) {
          setGuarantees(JSON.parse(storedGuarantees));
        }
      } catch (error) {
        console.error('Error fetching guarantees:', error);
      }
    };
    fetchGuarantees();
  }, [STORAGE_KEY]);

  const saveGuaranteesToStorage = async (updatedGuarantees) => {
    try {
      await storeData(STORAGE_KEY, JSON.stringify(updatedGuarantees));
      setGuarantees(updatedGuarantees);
    } catch (error) {
      console.error('Error saving guarantees:', error);
    }
```

```
};

const handleSubmit = (values) => {
  let updatedGuarantees;
  if (editingGuarantee) {
    updatedGuarantees = guarantees.map(g =>
      g.id === editingGuarantee.id ? { ...g, ...values } : g
    );
  } else {
    updatedGuarantees = [
      ...guarantees,
      { ...values, id: `dfz-${Date.now()}`, guaranteesNav: values.guaranteeType }
    ];
  }
  saveGuaranteesToStorage(updatedGuarantees);
  setModalVisible(false);
  setEditingGuarantee(null);
};

const handleDelete = (item) => {
  Alert.alert(
    "Eliminar Garantía",
    "¿Está seguro que desea eliminar esta garantía?",
    [
      { text: "Cancelar", style: "cancel" },
      {
        text: "Eliminar",
        onPress: () => {
          const updatedGuarantees = guarantees.filter(g => g.id !== item.id);
          saveGuaranteesToStorage(updatedGuarantees);
        }
      }
    ]
  );
};

const handleEdit = (item) => {
  setEditingGuarantee(item);
  setModalVisible(true);
};

return (
  <GuaranteesContext.Provider value={{
    guarantees,
```

```
            editingGuarantee,
            setEditingGuarantee,
            handleSubmit,
            handleEdit,
            handleDelete,
            modalVisible,
            setModalVisible,
            selected,
            setSelected
        }}>
            {children}
        </GuaranteesContext.Provider>
    );
};

export const useGuarantees = () => useContext(GuaranteesContext);
”
```

CUSTOMFIELDS: Funcion que se utiliza para mostrar los elementos del formulario

“

```
import { View, Text, TextInput, Switch, TouchableOpacity } from 'react-native';
import React from 'react';
import CustomDropdown from './CustomDropdown';
import CustomDropdownSearch from './CustomDropdownSearch';
import { className } from '../utils/className';
import PhotoCard from './PhotoCard';
import ErrorBoundary from './ErrorBoundary';

export default function CustomField({ field, handleChange, values, columns = 1, errors = {},
touched = {}, handleSubmit }) {

  const getWidthClass = () => {
    try {
      switch (field?.columns) {
        case 1: return 'w-1/4 pl-1 pr-1';
        case 2: return 'w-2/4 pl-1 pr-1';
        case 3: return 'w-3/4 pl-1 pr-1';
        case 4: return 'w-full pl-1 pr-1';
        default: return 'w-full pl-1 pr-1';
      }
    } catch (error) {
      console.error("handleTextChange: ", error)
      return 'w-full pl-1 pr-1';
    }
```

```
};

const widthClass = getWidthClass();

const handleTextChange = (text) => {
  try {
    // Elimina cualquier signo negativo y convierte a número
    const numericValue = text?.replace('-', '');

    // Si el campo es numérico, asegúrate de que sea un número positivo
    if (field?.customProps?.keyboardType === 'numeric' ||
      field?.customProps?.keyboardType === 'number-pad') {
      const number = parseFloat(numericValue);
      if (!isNaN(number) && number >= 0) {
        handleChange(field?.name)(numericValue);
      }
    } else {
      // Para campos no numéricos, simplemente actualiza el valor
      handleChange(field?.name)(text);
    }
  } catch (error) {
    console.error("handleTextChange: ", error)
  }
};
const backgroundColor = errors?.[field.name] && touched?.[field.name]
  ? 'bg-red-100 dark:bg-red-900'  // Color de fondo si hay error y el campo fue tocado
  : 'bg-purple-50 dark:bg-gray-700';  // Color normal si no hay error

return (
  <ErrorBoundary>
    <View style={className(`mb-2 ${widthClass}`)}>
      {field?.component === "Switch" && <Text></Text>}
      {!["Switch", "Title"]?.includes(field?.component) &&
        <Text style={className('mb-0 dark:text-white')}>
          {field?.label} {field?.required && "*"}
        </Text>}

      {field?.component === 'TextField' && (
        <ErrorBoundary>
          {console.log("value: ", JSON.stringify(values || {}) )}
          <TextInput
            style={className(`${backgroundColor} p-2 rounded dark:text-white`)}
            onChangeText={handleTextChange}
            value={values?.[field?.name] || ""}
```

```
          {...field?.customProps}
        />
      </ErrorBoundary>
    )}
    {field.component === 'CustomDropdown' && (
      <ErrorBoundary>
        <CustomDropdown
          options={field.options}
          title={field?.label}
          selectedValue={values[field.name]}
          onValueChange={handleChange(field.name)}
        />
      </ErrorBoundary>

    )}
    {field?.component === 'CustomDropdownSearch' && (
      <ErrorBoundary>
        <CustomDropdownSearch
          title={field?.label}
          options={field.options}
          selectedValue={values[field.name]}
          onValueChange={handleChange(field.name)}
        />
      </ErrorBoundary>

    )}

    {field?.component === 'Date' && (
      <TextInput
        style={className('bg-purple-50 p-2 rounded dark:bg-gray-700 text-white')}
        onChangeText={handleChange(field.name)}
        value={values[field.name]}
      />
    )}

    {field?.component === 'Switch' && (
      <ErrorBoundary>
        <View style={className('bg-purple-50 flex flex-row justify-between items-center p-2
rounded dark:bg-gray-700')}>
          <Text>{field.label}</Text>
          <Switch
            onValueChange={(value) => {
              console.log("Value switch: ", value);
              handleChange(field.name)(value);
```

```
            }}
            //value={values[field.name] || false}
            trackColor={{ false: "#767577", true: "#81b0ff" }}
            thumbColor={values[field.name] ? "#f5dd4b" : "#f4f3f4"}
          //value={Boolean(values[field.name]) || false }
          />
        </View>
      </ErrorBoundary>
    )}

    {field?.component === 'Title' && (
      <View>
        <Text style={className('text-lg text-blue-600 font-bold
dark:text-blue-500')}>{field?.label}</Text>
        {
          field?.subLabel && (
            <Text style={className('text-sm')}>{field?.subLabel}</Text>
          )
        }

        {
          field?.showHr != false &&
          <View style={className('w-full bg-gray-300 rounded-md')} >
            <Text style={{ height: 1 }}></Text>
          </View>
        }

      </View>
    )}

    {field?.component === 'PhotoCard' && (
      <PhotoCard
        field={field}
        handleChange={handleChange}
        values={values}
      />
    )}

    {field?.component === 'Submit' && (
      <ErrorBoundary>
        <TouchableOpacity onPress={handleSubmit}>
          <View style={className('bg-green-500 p-3 rounded-lg')}>
            <Text style={className('text-white font-bold text-center')}>{field?.title}</Text>
          </View>
```

```
          </TouchableOpacity>
        </ErrorBoundary>
      )}

      {/** MOSTRAR EL ERROR */}
      {
        errors?.[field.name] && touched?.[field.name] && <View>
          <ErrorBoundary>
            {<Text
style={className(`text-red-500`)}>{errors?.[field.name]}</Text>}{errors?.[field.name] &&
touched?.[field.name] && <Text
style={className(`text-red-500`)}>{errors?.[field.name]}</Text>}
          </ErrorBoundary>
        </View>
      }

    </View>
  </ErrorBoundary>

 );
}
```

”

Ejemplo Manejo de formulario en navegacion

“

```
import React, { useEffect, useState } from 'react';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import tw from 'twrnc';
import FinancialEvaluationPage from './FinancialEvaluationPage';
import FinancialEvaluationSection2 from './FinancialEvaluationSection2';
import { Formik, useFormikContext } from 'formik';
import { getData, storeData } from '../../../../../utils/storage';
import { ActivityIndicator, Text, View } from 'react-native';
import { FINANCIAL_SIT_FINAN } from './FinancialEvaluationFields';
import { className } from '../../../../../utils/className';

const TabFinEv = createBottomTabNavigator();

// Componente para manejar la actualización automática de totalMensual
const AutoUpdateTotalMensual = () => {
  const { values, setFieldValue } = useFormikContext();

  useEffect(() => {
```

```
    const ventasRegular = parseFloat(values.ventasRegular) || 0;
    const ventasBaja = parseFloat(values.ventasBaja) || 0;
    const totalMensual = ventasRegular + ventasBaja;
    setFieldValue('totalMensual', totalMensual.toFixed(2));
  }, [values.ventasRegular, values.ventasBaja]);

  return null;
};

const FinantialEvaluationButtonNav = ({ credit }) => {
  const [initialValues, setInitialValues] = useState({
    creditHashId: credit?.hashId,
    ventasRegular: '0',
    ventasBaja: '0',
    totalMensual: '0',
  });

  const calculateFieldValue = (calculation, values) => {
    if (!calculation) return null;

    switch (calculation.type) {
      case 'sum':
        return calculation.fields.reduce((sum, field) => sum + (parseFloat(values[field]) || 0), 0);
      // Puedes agregar más tipos de cálculos aquí
      default:
        return null;
    }
  };

  const AutoUpdateCalculatedFields = ({ fields }) => {
    const { values, setFieldValue } = useFormikContext();

    useEffect(() => {
      console.log("invoke AutoUpdateCalculatedFields")
      fields.forEach(field => {
        if (field.calculation) {
          const calculatedValue = calculateFieldValue(field.calculation, values);
          setFieldValue(field.name, calculatedValue.toFixed(2));
        }
      });
    }, [values, fields]);

    return null;
  };
```

```
const [isLoading, setIsLoading] = useState(true);

const storageKey = "finantial_evaluation_" + credit?.hashId;

useEffect(() => {
  const fetchData = async () => {
    const storedData = await getData(storageKey);
    console.log("Datos recuperados: ", storedData);

    if (storedData) {
      setInitialValues((prevValues) => ({
        ...prevValues,
        ...JSON.parse(storedData)
      }));
    }

    setIsLoading(false);
  };

  fetchData();
}, []);

const handleSubmit = async (values) => {
  console.log('Form values:', values);
  await storeData(storageKey, JSON.stringify(values));
};

if (isLoading) {
  return (
    <View style={className('h-full p-4 bg-white flex items-center dark:bg-gray-900')}>
      <ActivityIndicator size="large" color="#1E90FF" />
      <Text style={className('dark:bg-white')}>Cargando...</Text>
    </View>
  )
}

return (
  <Formik
    initialValues={initialValues}
    enableReinitialize={true}
    onSubmit={handleSubmit}
  >
```

```
      {({ handleChange, handleSubmit, values, touched }) => (
        <>
          <AutoUpdateCalculatedFields fields={FINANCIAL_SIT_FINAN} />
          <TabFinEv.Navigator
            screenOptions={{
              headerShown: false,
              tabBarBadgeStyle: ``,
              tabBarStyle: tw`bg-white dark:bg-gray-700`
            }}>
            <TabFinEv.Screen name="sit_finan">
              {(props) => (
                <FinancialEvaluationPage
                  {...props}
                  credit={credit}
                  handleChange={handleChange}
                  handleSubmit={handleSubmit}
                  values={values}
                />
              )}
            </TabFinEv.Screen>
            <TabFinEv.Screen name="patrimonio">
              {(props) => (
                <FinancialEvaluationSection2
                  {...props}
                  credit={credit}
                  handleChange={handleChange}
                  handleSubmit={handleSubmit}
                  values={values}
                />
              )}
            </TabFinEv.Screen>
          </TabFinEv.Navigator>
        </>
      )}
    </Formik>
  );
};

export default FinantialEvaluationButtonNav;
”
```

Librerias interesantes
https://github.com/huextrat/react-native-maps-routes
https://github.com/dream-sports-labs/react-native-fast-image

Temas a resolver

- Libreria para Mapas
- Mapa de forma dinamica
- Iconos
- Validaciones de formularios evaluar el maximo de casos posibles
- Screensize
- Get phone information
-