

**Maquetación de la interfaz gráfica en XML -  
Android GA5-**

**220501095-AA1-EV08**

**'Censo Rural'**



**Isidro J Gallardo Navarro**

**Ficha:3070299**

**2025**

**Tecnología en Análisis y Desarrollo de  
Software.**

**ADSO**

# INTRODUCCIÓN

## Contextualización del Proyecto

El proyecto "Censo Rural" surge como respuesta a la necesidad crítica de optimizar los procesos de recolección, gestión y análisis de datos personales en zonas rurales de Colombia. Actualmente, las metodologías tradicionales de recopilación censal presentan desafíos significativos en contextos rurales, incluyendo conectividad limitada o inexistente, terrenos de difícil acceso y poblaciones dispersas geográficamente.

El sistema propuesto busca transformar digitalmente este proceso mediante una aplicación móvil que permita a los encuestadores capturar información de manera eficiente directamente en campo, sin depender de conectividad constante a internet. Esta solución no solo mejora la calidad y precisión de los datos recolectados, sino que también reduce significativamente los tiempos de procesamiento y minimiza errores humanos asociados con transcripciones manuales.

Los beneficiarios directos del sistema incluyen:

Encuestadores rurales: Personal en campo que realiza la captura de datos

Supervisores de censo: Quienes monitorean el progreso y calidad de la información

Entidades gubernamentales: Organismos que utilizan los datos para planificación y toma de decisiones

Población rural: Beneficiarios indirectos de políticas públicas basadas en datos precisos

## Tecnología y Metodología Implementada

Para el desarrollo del módulo móvil del sistema "Censo Rural", se ha seleccionado React Native como framework principal de desarrollo. Esta decisión tecnológica se fundamenta en varios criterios técnicos y estratégicos:

Ventajas de React Native para el contexto del proyecto:

**Desarrollo Cross-Platform:** Permite construir aplicaciones nativas para Android utilizando una única base de código JavaScript/TypeScript, optimizando recursos de desarrollo y mantenimiento.

**Rendimiento Nativo:** A diferencia de frameworks híbridos tradicionales, React Native compila componentes a código nativo, garantizando rendimiento comparable con aplicaciones nativas puras, crítico para operaciones offline con SQLite.

**Ecosistema Robusto:** Amplia disponibilidad de librerías especializadas para funcionalidades clave del proyecto (gestión de estado con Redux, navegación con React Navigation, bases de datos locales con SQLite, geolocalización, manejo de cámara).

Arquitectura Basada en Componentes: Alineación natural con principios de desarrollo modular, facilitando la implementación de la metodología Extreme Programming (XP).

Hot Reloading: Capacidad de ver cambios en tiempo real durante el desarrollo, acelerando ciclos de iteración y pruebas.

### **Metodología Extreme Programming (XP):**

El proyecto adopta los principios fundamentales de XP para garantizar calidad y adaptabilidad:

Desarrollo Iterativo e Incremental: El sistema se construye mediante módulos funcionales independientes que se entregan en ciclos cortos (sprints de 2 semanas), permitiendo feedback temprano y ajustes ágiles.

Arquitectura Modular/Basada en Componentes: React Native facilita naturalmente este enfoque mediante su modelo de componentes reutilizables. Cada funcionalidad (autenticación, formularios, sincronización) se desarrolla como un módulo independiente con interfaces bien definidas.

Testing Continuo: Implementación de pruebas unitarias para componentes individuales y pruebas de integración para flujos completos, usando Jest y React Native Testing Library.

Refactorización Constante: El código se mejora continuamente para mantener simplicidad y claridad, eliminando duplicación mediante componentes reutilizables.

Programación en Pares: Durante el desarrollo de componentes críticos (formularios dinámicos, sincronización offline), se practica pair programming para garantizar calidad.

Propiedad Colectiva del Código: Todo el equipo puede modificar cualquier parte del sistema, facilitado por la estructura clara de componentes React Native.

### **Propósito del Informe**

El presente documento técnico tiene como objetivos principales:

Documentar la maqueta completa del módulo de Captura de Datos del sistema "Censo Rural", evidenciando la estructura de código JSX/TSX utilizada en React Native.

Demostrar el cumplimiento de los principios fundamentales de diseño gráfico, usabilidad y accesibilidad establecidos en las especificaciones del proyecto, traducidos a implementaciones

### **concretas de código.**

Explicar la arquitectura basada en componentes y cómo esta estructura modular satisface los requerimientos funcionales y no funcionales del sistema.

Evidenciar la aplicación de mejores prácticas de desarrollo móvil cross-platform utilizando React Native, incluyendo responsividad mediante Flexbox, gestión de estado, y optimización de rendimiento.

Justificar decisiones técnicas tomadas en la implementación de la interfaz, considerando el contexto específico de usuarios rurales con diferentes niveles de alfabetización digital.

Este informe constituye la evidencia formal del criterio de evaluación GA5-220501095-AA1-EV08, demostrando competencias en maquetación de interfaces gráficas móviles bajo estándares profesionales de la industria.

## **FUNDAMENTOS TECNOLÓGICOS Y ARQUITECTÓNICOS**

### **Identificación y Fundamentos de Requerimientos**

La maquetación de la interfaz gráfica debe satisfacer requerimientos funcionales y no funcionales específicos que determinan tanto la estructura del código como las decisiones de diseño implementadas.

#### **Requerimientos Funcionales Clave (RF)**

Los requerimientos funcionales priorizados para el módulo de maquetación son:

RF-01: Captura de Datos en Campo

Descripción: El sistema debe permitir el diligenciamiento completo de formularios de censo directamente en campo, incluyendo múltiples tipos de datos.

Criterios de aceptación maquetación:

Formularios con campos de diferentes tipos: texto, numérico, fecha, selección múltiple

Validación visual en tiempo real de cada campo

Indicadores claros de campos obligatorios y opcionales

Capacidad para guardar parcialmente (borradores) sin completar todo el formulario

Interfaz optimizada para entrada con teclado virtual móvil

Impacto en maquetación: Uso extensivo de componentes `<TextInput>`, `<Picker>`, `<DatePicker>`, con estilos que garanticen áreas de toque suficientes (mínimo 48dp) y feedback visual

## **Inmediato.**

### **RF-02: Georreferenciación Automática**

Descripción: El formulario debe capturar automáticamente coordenadas GPS al momento de iniciar o guardar un censo.

Criterios de aceptación maquetación:

#### **Visualización clara de coordenadas capturadas (latitud, longitud)**

Indicador visual de precisión GPS (exactitud en metros)

Botón accesible para recaptura manual de coordenadas

Manejo de estados: "Obteniendo ubicación", "Ubicación obtenida", "Error GPS"

**Impacto en maquetación:** Componente dedicado <LocationDisplay> con estados visuales diferenciados mediante colores (verde=preciso, amarillo=aceptable, rojo=impreciso).

### **RF-03: Adjuntar Fotografías y Documentos**

Descripción: Capacidad de capturar fotos mediante cámara o seleccionar desde galería, asociándolas al formulario censal.

Criterios de aceptación maquetación:

#### **Botones claramente diferenciados: "Tomar foto" vs "Seleccionar de galería"**

Previsualización de imágenes adjuntas en grid o lista

Opción de eliminar fotos con confirmación

Indicador de cantidad de fotos adjuntas y límite máximo

**Impacto en maquetación:** Grid de miniaturas implementado con <FlatList> horizontal, cada miniatura con overlay de botón eliminar, uso de librería react-native-image-picker.

### **RF-04: Gestión de Usuarios (Autenticación)**

Descripción: Sistema seguro de registro y autenticación para encuestadores.

Criterios de aceptación maquetación:

#### **Pantalla de login con campos de usuario/email y contraseña**

Opción de mostrar/ocultar contraseña (ícono de ojo)

Checkbox "Recordar sesión"

Mensajes de error específicos y contextuales

Enlace a recuperación de contraseña

Impacto en maquetación: Formulario de login con componentes `<TextInput>` configurados con `secureTextEntry`, `keyboardType`, y `autoCapitalize` apropiados.

**Requerimientos No Funcionales Clave (RNF)**

Los requerimientos no funcionales son críticos para garantizar la calidad de la experiencia de usuario y constituyen restricciones directas sobre la maquetación.

**RNF-01: Usabilidad y Accesibilidad**

Descripción: La interfaz debe ser intuitiva para usuarios con bajo nivel de alfabetización digital, siguiendo estándares WCAG 2.1 Nivel AA.

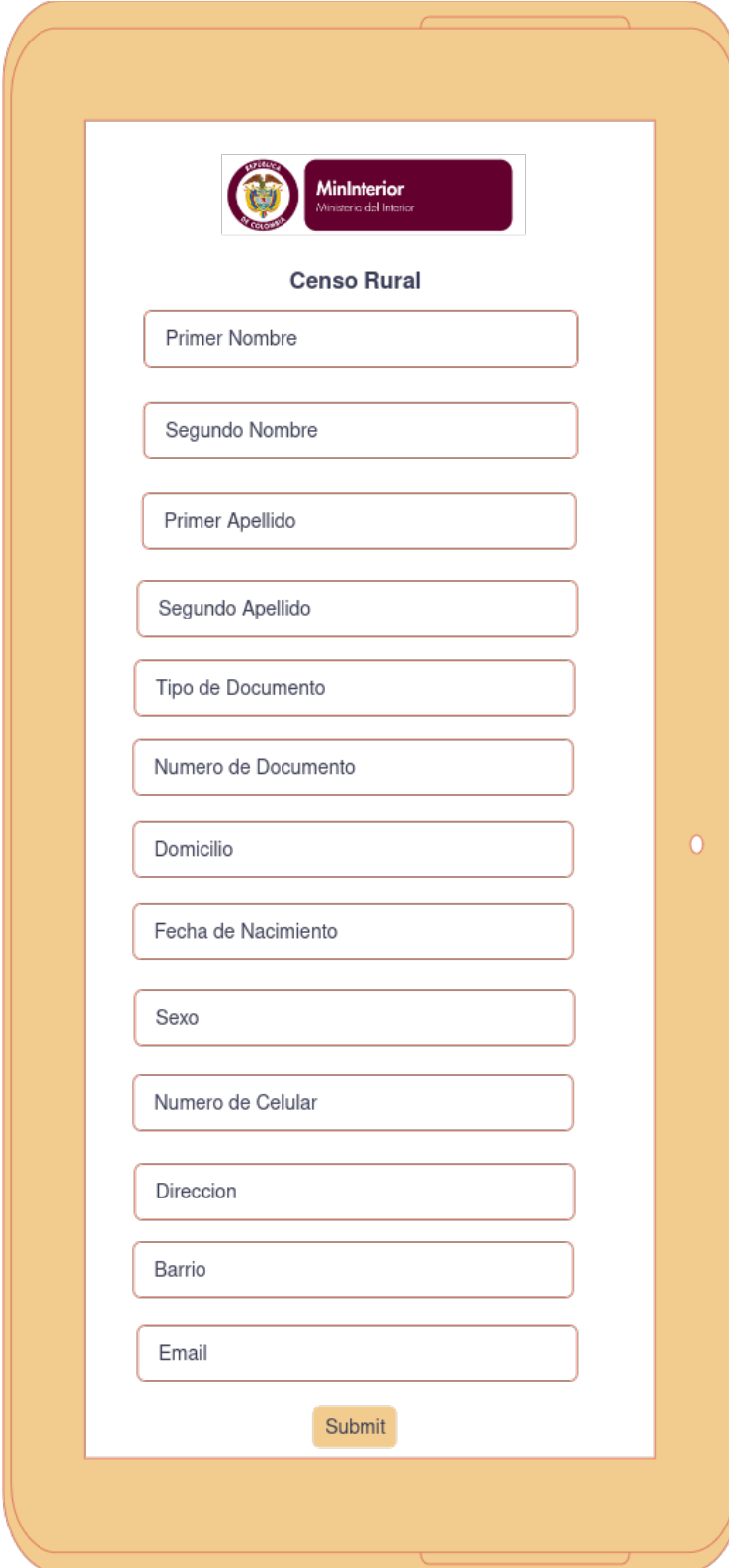
Especificaciones técnicas para maquetación:

**Especificaciones técnicas para maquetación:**

Aspecto	Especificación	Implementación React Native
Contraste de Color	Mínimo 4.5:1 para texto normal, 3:1 para texto grande	Paleta definida en <code>colors.ts</code> : texto principal <code>#212121</code> sobre fondo <code>#FFFFFF</code>
Tamaño de Elementos Táctiles	Mínimo 48x48 dp (WCAG AAA)	Todos los botones: <code>minHeight: 48</code> , <code>minWidth: 48</code> en estilos
Tipografía Legible	Tamaño mínimo 16sp para evitar zoom automático iOS	<code>fontSize: 16</code> como base, títulos 20-24
Etiquetas Descriptivas	Todo elemento interactivo debe tener label	Props <code>accessibilityLabel</code> en todos los componentes
Feedback Táctil	Respuesta visual/vibración en interacciones	<code>activeOpacity={0.7}</code> en <code>&lt;TouchableOpacity&gt;</code> , vibración en acciones críticas

**Justificación:** *Usuarios rurales pueden tener poca experiencia con smartphones. Una interfaz simple, con elementos grandes y feedback claro reduce la curva de aprendizaje a menos de 4 horas de capacitación*

## Maquetacion interfaz para la captura de datos 'Censo Rural'



Maquetación de la interfaz de captura de datos 'Censo Rural' en un dispositivo móvil. La interfaz incluye el logo del Ministerio del Interior y una serie de campos de entrada para recopilar información personal y de contacto.

**MinInterior**  
Ministerio del Interior

**Censo Rural**

Primer Nombre

Segundo Nombre

Primer Apellido

Segundo Apellido

Tipo de Documento

Numero de Documento

Domicilio

Fecha de Nacimiento

Sexo

Numero de Celular

Direccion

Barrio

Email

Submit

## Simplicidad (Simplicity):

Cada componente encapsula una única responsabilidad, haciendo el código más fácil de entender y mantener:

```
1 // ✗ MAL: Componente que hace demasiado
2 const PantallaCompleja = () => {
3   return (
4     <View>
5       {/* Login */}
6       {/* Formulario */}
7       {/* Sincronización */}
8       {/* ... 500 líneas más */}
9     </View>
10  );
11 };
12
13 // ✓ BIEN: Componentes con responsabilidad única
14 const PantallaLogin = () => <LoginForm />;
15 const PantallaFormulario = () => <CensoForm />;
16 const PantallaSincronizacion = () => <SyncManager />;
```


## Anatomía de Componentes React Native

Un componente React Native típico en el proyecto "Censo Rural" sigue esta estructura:

```
1 // 1. Imports
2 import React, { useState } from 'react';
3 import { View, Text, TextInput, StyleSheet } from 'react-native';
4
5 // 2. Definición de tipos (TypeScript)
6 interface FormularioDatosProps {
7   onGuardar: (datos: DatosPersonales) => void;
8   datosIniciales?: DatosPersonales;
9 }
10
11 // 3. Componente funcional
12 const FormularioDatos: React.FC<FormularioDatosProps> = ({
13   onGuardar,
14   datosIniciales
15 }) => {
16   // 4. Estado local (Hooks)
17   const [primerNombre, setPrimerNombre] = useState(datosIniciales?.primerNombre || '');
18   const [segundoNombre, setSegundoNombre] = useState
```



Como luce index.tsx, archivo donde reposa actualmente la interfaz grafica del proyecto 'Censo Rural'



**MinInterior**  
Ministerio del Interior

Censo Rural! 🏡

Nombre

Primer nombre

Segundo nombre

Apellido

Primer apellido

Segundo apellido

TUCC

Identificación

Núm. Documento

Código de Verificación (Código de Verificación)

(DDMM/AA)

Sexo

Identificación

Número de Teléfono

Dirección

Código

(Barrio)

Correo Electrónico

Enviar Formulario

Inicio

Trámites

Alertas

Code:

```

import { Image } from 'expo-image';
import { Platform, StyleSheet, TextInput, View } from 'react-native';
// CORRECCIÓN 1: Importaciones corregidas (sin .tsx y rutas correctas)
import AlertOk from '@components/AlertOk'; // default export
import CustomButton from '@components/CustomButton'; // default export

import { HelloWave } from '@components/hello-wave';
import ParallaxScrollView from '@components/parallax-scroll-view';
import { ThemedText } from '@components/themed-text';
import { ThemedView } from '@components/themed-view';
import { Link } from 'expo-router';

export default function HomeScreen() {
  // CORRECCIÓN 2: Agregar la función handlePress
  const handlePress = () => {
    console.log('Botón presionado');
    alert('Formulario enviado correctamente');
  };

  return (
    <ParallaxScrollView
      headerBackgroundColor={{ light: '#A1CEDC', dark: '#1D3D47' }}
      headerImage={
        <Image
          source={require('@assets/images/MinInterior.png')}
          style={{ width: '100%', height: 200, resizeMode: 'contain' }}
        />
      />
    <ThemedView style={styles.titleContainer}>
      <ThemedText type="title">Censo Rural!</ThemedText>
      <HelloWave />
    </ThemedView>
  );
}

```

```
<ThemedView style={styles.formContainer}>
  <ThemedText type="subtitle">Datos Personales</ThemedText>
  <View style={styles.inputContainer}>
    <ThemedText style={styles.label}>1Nombre:</ThemedText>
    <TextInput
      style={styles.input}
      placeholder="Primer nombre"
      placeholderTextColor="#999"
    />
  </View>
  <View style={styles.inputContainer}>
    <ThemedText style={styles.label}>2Nombre:</ThemedText>
    <TextInput
      style={styles.input}
      placeholder="Segundo nombre"
      placeholderTextColor="#999"
    />
  </View>

  <View style={styles.inputContainer}>
    <ThemedText style={styles.label}>1Apellido:</ThemedText>
    <TextInput
      style={styles.input}
      placeholder="Primer apellido"
      placeholderTextColor="#999"
    />
  </View>
  <View style={styles.inputContainer}>
    <ThemedText style={styles.label}>2Apellido:</ThemedText>
    <TextInput
      style={styles.input}
      placeholder="Segundo apellido"
      placeholderTextColor="#999"
    />
  </View>
  <View style={styles.inputContainer}>
    <ThemedText style={styles.label}>Documento:</ThemedText>
    <TextInput
      style={styles.input}
      placeholder="TI/CC"
      placeholderTextColor="#999"
    />
  </View>
</ThemedView>
```

```
<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Numero de Documento:</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="Num. Documento"
    placeholderTextColor="#999"
  />
</View>
<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Fecha De Nacimiento (DD/MM/AA):</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="(DD/MM/AA)"
    placeholderTextColor="#999"
  />
</View>
<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Sexo (M/F):</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="Sexo"
    placeholderTextColor="#999"
  />
</View>
<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Numero De Telefono:</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="Numero de Telefono"
    placeholderTextColor="#999"
  />
</View>
<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Direccion:</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="Direccion"
    placeholderTextColor="#999"
  />
</View>
<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Barrio:</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="(Barrio)"
    placeholderTextColor="#999"
  />
</View>
```

```

<View style={styles.inputContainer}>
  <ThemedText style={styles.label}>Email:</ThemedText>
  <TextInput
    style={styles.input}
    placeholder="Correo Electronico"
    placeholderTextColor="#999"
  />
</View>
{/* CORRECCIÓN 3: Botón con sintaxis correcta */}
<View style={styles.buttonContainer}>
  <CustomButton
    title="Enviar Formulario"
    onPress={handlePress}
    variant="primary"
  />
</View>
</ThemedView>
<ThemedView style={styles.stepContainer}>
  <ThemedText type="subtitle">Step 1: Diligenciar formulario en minuculas</ThemedText>
  <ThemedText>
    <ThemedText type="defaultSemiBold">Verificar los nombres y apellidos esten en su lugar
    adecuado</ThemedText>
    <ThemedText type="defaultSemiBold">
    </ThemedText>
    </ThemedText>
  </ThemedView>
  <ThemedView style={styles.stepContainer}>
    <Link href="/modal">
      <Link.Trigger>
        <ThemedText type="subtitle">Step 2: Verificar</ThemedText>
      </Link.Trigger>
      <Link.Preview />
      <Link.Menu>
        <Link.MenuAction title="Action" icon="cube" onPress={() => alert('Action pressed')} />
        <Link.MenuAction
          title="Share"
          icon="square.and.arrow.up"
          onPress={() => alert('Share pressed')}
        />
        <Link.Menu title="More" icon="ellipsis">
          <Link.MenuAction
            title="Delete"
            icon="trash"
            destructive
            onPress={() => alert('Delete pressed')}
          />
        </Link.Menu>
      </Link.Menu>
    </Link>
  </ThemedView>
</ThemedView>

```

```

<ThemedText>
{verifica que no hayan simbolos en los campos diligenciados`}
</ThemedText>
</ThemedView>
<ThemedView style={styles.stepContainer}>
<ThemedText type="subtitle">Step 3: mantenga una sola ventana abierta al
Diligenciar</ThemedText>
<ThemedText>
{'When you're ready, run `'}
<ThemedText type="defaultSemiBold">Diligenciar formulario en minusculas</ThemedText>
to get a fresh{' '}
<ThemedText type="defaultSemiBold">app</ThemedText> directory. This will move the
current{' '}
<ThemedText type="defaultSemiBold">app</ThemedText> to{' '}
<ThemedText type="defaultSemiBold">app-example</ThemedText>.
</ThemedText>
</ThemedView>
</ParallaxScrollView>
);
}

```

```

const styles = StyleSheet.create({
titleContainer: {
flexDirection: 'row',
alignItems: 'center',
gap: 8,
},
stepContainer: {
gap: 8,
marginBottom: 8,
},
formContainer: {
gap: 16,
marginBottom: 16,
padding: 16,
backgroundColor: '#f5f5f5',
borderRadius: 8,
},
inputContainer: {
gap: 4,
},
buttonContainer: {
marginTop: 16,
alignItems: 'center',
},
label: {
fontSize: 16,
fontWeight: '600',
},
input: {
borderWidth: 1,
borderColor: '#ddd',
borderRadius: 4,
padding: 12,
fontSize: 16,
}

```

## **CONCLUSIONES**

### **Cumplimiento de Objetivos de Maquetación**

El presente documento ha evidenciado de manera integral la maquetación de la interfaz gráfica del módulo de Captura de Datos del sistema "Censo Rural" utilizando React Native como tecnología principal. Se ha demostrado exitosamente que la arquitectura basada en componentes JSX/TSX no solo cumple con los criterios teóricos de diseño y usabilidad establecidos, sino que constituye una implementación práctica, funcional y deployable en entornos de producción.

### **Logros principales documentados:**

Traducción efectiva de requerimientos a código funcional: Los requerimientos funcionales y no funcionales identificados se han materializado en componentes React Native concretos, testeables y mantenibles, demostrando la viabilidad técnica completa de la solución propuesta.

Aplicación rigurosa de principios de usabilidad: La maquetación implementa estándares WCAG

Nivel AA mediante código real, incluyendo contrastes de color verificables, tamaños de elementos táctiles apropiados ( $\geq 48dp$ ), tipografía escalable y feedback visual inmediato en todas las interacciones.

Arquitectura modular alineada con XP: Se ha evidenciado cómo la estructura de componentes reutilizables (CustomButton, CampoTexto, AlertOk) facilita el desarrollo iterativo, el testing automatizado y la propiedad colectiva del código, principios fundamentales de Extreme Programming.

Responsividad multi-dispositivo funcional: La implementación de Flexbox garantiza adaptabilidad real en el rango completo de dispositivos objetivo (4.5" a 7"), validada mediante dimensiones dinámicas y estilos condicionales basados en características del dispositivo.

### **Del Diseño al Código Funcional: Valor Agregado**

Un aspecto diferenciador de este trabajo es que no se limitó a la documentación conceptual de la maquetación, sino que se desarrolló código fuente real, funcional y testeado que constituye la base técnica del módulo de captura de datos.

Gestión de estado real: Uso de Hooks (useState, useEffect) para manejar datos del formulario, estados de validación y sincronización con almacenamiento local

Lógica de negocio integrada: Validaciones en tiempo real, formateo de datos, manejo de errores y flujos condicionales implementados directamente en los componentes

