
Software Requirements Specification

for

Vehicle insurance system

Version 1.0 approved

Prepared by

Alcántara Huerta Angel Josué
Leon Ramirez Miguel Angel
Ramirez Navarro Marcos David
Soto García Oscar Gael

Universidad Tecnológica de Tijuana

March 19, 2025

Table of Contents

1. Introduction.....	1
1.1 Purpose.....	1
1.2 Document Conventions.....	1
1.3 Intended Audience and Reading Suggestions.....	1
1.4 Product Scope.....	1
1.5 References.....	1
2. Overall Description.....	2
2.1 Product Perspective.....	2
2.2 Product Functions.....	2
2.3 User Classes and Characteristics.....	2
2.4 Operating Environment.....	3
2.5 Design and Implementation Constraints.....	3
2.6 User Documentation.....	3
2.7 Assumptions and Dependencies.....	5
3. External Interface Requirements.....	5
3.1 User Interfaces.....	5
3.2 Hardware Interfaces.....	5
3.3 Software Interfaces.....	5
3.4 Communications Interfaces.....	6
4. System Features.....	7
4.1 Functional Requirements.....	7
5. Other Nonfunctional Requirements.....	7
5.1 Performance Requirements.....	7
5.2 Safety Requirements.....	8
5.3 Security Requirements.....	8
5.4 Software Quality Attributes.....	8
5.5 Business Rules.....	8
6. Other Requirements.....	8

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

In today's insurance industry, operational efficiency and customer experience are critical to success. This project, the Vehicle Insurance System, aims to transform the way insurance companies manage their day-to-day operations, from policy administration to claims assessment. The main objective is to provide a modern platform that automates routine tasks, facilitates interaction with customers and offers robust management tools for administrators.

The development of this system will be done through the integration of modern technologies such as mobile applications, API services and the potential inclusion of IoT (Internet of Things) to improve interaction and data collection. With a focus on security and scalability, the system is designed to handle large volumes of transactions and data efficiently, ensuring the confidentiality and protection of user information. Data security is a top priority, ensuring that all information is handled with the highest standards of protection.

1.1 Purpose

The purpose of this document is to specify the software requirements for the Insurance System, designed to manage policies, calculate premiums, assess claims and generate reports. The system aims to increase operational efficiency, reduce manual tasks and improve the user experience for both administrators and customers..

1.2 Document Conventions

This SRS uses simple text formatting to ensure easy readability. The following conventions are used:

- **Bold text** is used to highlight key terms or sections.
- *Italics* are used for examples or notes.
- All requirements are listed in normal text and are numbered for easy reference.
- Each requirement has its own priority, meaning the priority level is specified individually for every requirement.

No complex typographical conventions or special symbols are used in this document to keep it simple and clear.

1.3 Intended Audience and Reading Suggestions

This document is intended for several types of readers:

- **Developers:** To understand the functional and technical requirements needed to build the system.
- **Project Managers:** To track project scope and ensure all requirements are met.
- **Testers:** To use the requirements for creating test cases and validating the system.
- **Documentation Writers:** To prepare user manuals based on the system's features.

The document is organized into sections covering an overview of the system, followed by detailed functional and non-functional requirements. Readers should start with the overview sections, like the Purpose and Product Scope, before moving into the more specific requirements.

1.4 Product Scope

This mobile application will have a focus for customers, employees, and administrators of a car insurance company. Its main purpose is to simplify policy management, report claims, and improve efficiency operational by digitizing key processes

1.5 References

- *IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications*

2. Overall Description

2.1 Product Perspective

The proposed Insurance System, is a new product designed to modernize and streamline the operations of insurance providers specializing in vehicle coverage. It is not part of an existing product family, but aims to integrate modern technologies such as mobile applications, IoT (Internet of Things), and APIs services to enhance customer and administrative experiences.

2.2 Product Functions

The Vehicle Insurance Management System performs the following key functions:

- **User Registration and Login:** Allows users to create accounts and manage their profiles securely.
- **Insurance Quotation:** Calculates real-time policy costs based on vehicle details and selected coverage levels.
- **Policy Management:** Displays policy information, including active, expired, or pending policies, with options to download documents in PDF format.
- **Claims Reporting:** Enables users to submit detailed claims, upload supporting photos, and track the resolution status.
- **Claims Handling:** Receive, update, and resolve claims, including tracking the status of each case.

2.3 User Classes and Characteristics

The product targets three distinct user classes (**clients, employees, and admins**), each with specific characteristics and needs:

1. Clients:

- **Characteristics:**
 - Individuals or businesses who purchase vehicle insurance.
 - Limited technical expertise; expect an intuitive and easy-to-use interface.
 - May access the product infrequently, primarily during policy purchases, renewals, or claims.
 - Concerned about security for personal and financial information.
- **Primary Functions Used:**
 - Account management, policy viewing, payment processing, claims submission, and assistance requests.

2. Employees:

- **Characteristics:**
 - Insurance company staff responsible for managing customer policies and claims.
 - Moderate technical expertise, familiar with web-based applications.
 - Regular users, accessing the system daily to process client requests.
- **Primary Functions Used:**
 - Policy and client management, claims handling.

3. Administrators:

- **Characteristics:**
 - Senior staff responsible for overseeing operations, managing employees, and analyzing performance metrics.
 - High technical expertise and decision-making authority.
 - Regular users, requiring access to advanced tools for system management.
- **Primary Functions Used:**
 - Managing policies, configuring pricing, viewing analytical dashboards, and supervising employee actions.

2.4 Operating Environment

These constraints will impact the development of the outsourcing platform:

Hardware Requirements

- Mobile devices (smartphones and tablets) running Android or iOS.
- Desktop or laptop computers with internet connectivity for web access.
- Processor: Dual-core 2.5 GHz or higher.
- RAM: 8GB or higher.
- Storage: SSD with a minimum of 10 GB of free space.

Operating System:

- Android 8.0 (Oreo) or higher for mobile devices.
- iOS 13.0 or higher for Apple devices.
- Windows 10/macOS 10.15 or higher for desktop environments.

2.5 User Documentation

The Vehicle Insurance Management System will include the following user documentation components to ensure a seamless experience for all users:

a. User Manual

Description: Comprehensive guides that cover the system's features and functionality.

Target Audience: Clients, employees, and administrators.

Delivery Format: Downloadable PDF documents.

b. Technical manual

Description: Provides detailed technical information about the system's architecture, configuration, and operation.

Target Audience: Developers and IT support staff.

Delivery Format: Downloadable PDF documents.

2.6 Assumptions and Dependencies

The successful development of the platform relies on several critical factors that, if changed or unmet, could lead to system failures:

- **User Access to Devices:** It is assumed that all users (clients, employees, and administrators) will have access to internet-enabled devices such as smartphones, tablets, or computers to interact with the system.
- **Reliable Internet Connectivity:** The system's functionality, including real-time geolocation and online payment processing, assumes that users will have stable internet access.
- **Third-Party APIs:** The program will include some APIs.

3. External Interface Requirements

3.1 User Interfaces

3.1.1 General Design

We seek to appeal to as many people as possible through a simple, beautiful and eye-pleasing interface.

3.1.2 Main Screen, Menu options

Users will be able to select an option using the keyboard or by clicking on the corresponding button on the graphical interface. Each option will take the user to a new screen or section where they can perform the specific actions related to that category.

3.2 Hardware Interfaces

It will be necessary to have mobile devices that are in good condition and fully functional, it will be required for the execution of the program.

3.3 Software Interfaces

For our product to be fully functional the only other software product that must be installed is in Android, iOS, Windows and macOS devices.

3.4 Communications Interfaces

The server and clients will communicate with each other, using standard protocols on the Internet, whenever possible. For example, existing protocols (FTP or other convenient protocols) should be used to transfer files.

4. System Features

4.1 Functional Requirements

4.1.1 Customers Module

- **Register and access:** Users create their account with their personal data and insured vehicles.
- **Insurance quote:** They can calculate in real time the cost of a policy depending on the characteristics of your vehicle and the level of coverage.
- **Policy management:** Viewing active, expired or pending policies renewal, with the possibility of downloading documents in PDF format.
- **Accident reports:** Allows you to upload photos, describe incidents and track the case status.
- **Secure payments:** Contracting and renewing policies through integrated payment methods such as credit card, debit card or transfers.
- **Roadside assistance:** Function to request services such as tow trucks or support mechanics through geolocation.

4.1.2 Employees Module

- **Customer and vehicle management:** Access to complete customer profiles, including your vehicles and policies.
- **Attention to claims:** Reception of reports, updating of statuses (pending, under review, resolved) and case assignment.
- **Direct communication:** Contact with clients through chat or calls integrated.

4.1.3 Administrative Module

- **Control of rates and plans:** Configuration of prices and terms of the policies.
- **Data analysis:** Dashboard with key metrics such as sales, claims and team performance.
- **Internal management:** Employee supervision, role assignment and monitoring of activities.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- **Response Time:**

The system must respond in less than 2 seconds in 95% of common cases, such as when the user requests a quote, accesses their profile, or searches for information. This performance should be maintained without compromising resource efficiency.

- **Server Response Time:**

The server must process heavy operations (such as payment processing) in less than 1 second, ensuring system reliability and preventing delays or crashes.

- **Scalability:**

The system must be able to automatically scale during periods of high demand (such as payment dates or renewals), without affecting security or operational efficiency.

5.2 Safety Requirements

- **Accidental Access Prevention:**

Automatic session locks should be implemented if the user leaves the app open without interaction for an extended period, to prevent accidental access to sensitive data.

- **Error Handling in the Interface**

The app should provide clear and easy-to-understand error messages so users don't get frustrated or make incorrect decisions, such as entering a policy number or payment details incorrectly.

5.3 Security Requirements

- **Authentication:**

All users must log in with a username and password, ensuring that each access is protected and reliable.

- **Access Control:**

User permissions must be clearly defined so that each user type only accesses what they need, respecting security and privacy principles.

- **Communication Encryption:**

All information transmitted between the user and the server must be encrypted using HTTPS, ensuring that data is protected from potential intruders and securing communication.

5.4 Software Quality Attributes

- **Usability:**

The application must be intuitive and easy to use, with an average learning time estimated at less than 30 minutes for non-technical users. The interface should be simple and efficient so that non-technical users can access its features without difficulty.

5.5 Business Rules

- End users can do basic things like get quotes, check their policies, and update their personal information easily and safely.
- Administrators can manage users, process claims, and review audit reports while keeping the system secure and reliable.
- Insurance agents can add new policies and change coverages.

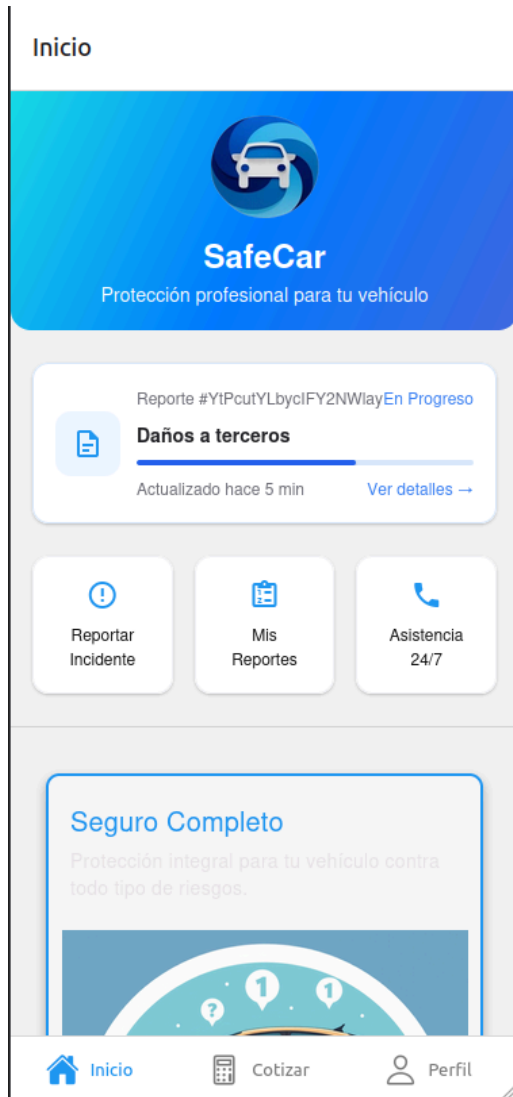
6. Other Requirements

Documentation Requirements: Comprehensive documentation must be provided for both users and developers, including user manuals, and system architecture diagrams.

7. Annexes

Second project progress

Main Screen with his code



```
HomeScreen.js X
screens > JS HomeScreen.js > @ HomeScreen > @ QuickActions

import React, { useState, useEffect } from 'react';
import { ScrollView, TouchableOpacity, Alert, Linking } from 'react-native';
import { VStack, HStack, Box, Text, Heading, Icon, Divider, useTheme, Spinner } from 'native-base';
import { MaterialCommunityIcons, Ionicons } from '@expo/vector-icons';
import Header from '../components/Header';
import CarInsuranceCard from '../components/CarInsuranceCard';
import ReportCard from '../components/ReportCard';
import { useNavigation } from '@react-navigation/native';
import AsyncStorage from 'react-native-async-storage/async-storage';
import { collection, query, where, getDocs, orderBy, limit, doc, getDoc } from 'firebase/firestore';
import { db } from '../config/firebaseConfig';

const HomeScreen = () => {
  const theme = useTheme();
  const navigation = useNavigation();
  const [hasActiveReport, setHasActiveReport] = useState(false);
  const [activeReport, setActiveReport] = useState(null);
  const [userData, setUserData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    getUserData();
  }, []);

  useEffect(() => {
    if (userData) {
      fetchActiveReports();
    }
  }, [userData]);

  const getUserData = async () => {
    try {
      const data = await AsyncStorage.getItem('@user_data');
      if (data) {
        setUserData(JSON.parse(data));
      }
    } catch (error) {
      console.error('Error retrieving user data:', error);
    } finally {
      setLoading(false);
    }
  };

  const fetchActiveReports = async () => {
    try {
      const reportsRef = collection(db, 'log', userData.id, 'reclamosUser');
      const q = query(
        reportsRef,
        where('estadoReclamo', 'in', ['Pendiente', 'En Revisión']),
        orderBy('fechaCreacion', 'desc'),
        limit(1)
      );
      const querySnapshot = await getDocs(q);

      if (!querySnapshot.empty) {
        const reportDoc = querySnapshot.docs[0];
        const reportData = reportDoc.data();

        // Obtener información del vehículo si existe
        let vehicleInfo = null;
        if (reportData.vehiculoId) {
          try {
            const vehicleRef = doc(db, 'log', userData.id, 'carrosUser', reportData.vehiculoId);
            const vehicleSnap = await getDoc(vehicleRef);
            if (vehicleSnap.exists()) {
              vehicleInfo = vehicleSnap.data();
            }
          } catch (error) {
            console.error('Error fetching vehicle info:', error);
          }
        }

        // Mapear el tipo de incidente a un formato más amigable
        const incidentTypes = {
          collision: { name: 'Colisión', icon: 'car-crash' },
          roadside: { name: 'Asistencia vial', icon: 'tow-truck' },
          glass: { name: 'Rotura de cristales', icon: 'car-door' },
          theft: { name: 'Robo', icon: 'shield-alert' },
          terceros: { name: 'Daños a terceros', icon: 'car-multiple' }
        };

        // Crear el objeto de reporte formateado
      }
    }
  };
};
```

Quote Screen with his code

Cotizar

Cotiza tu seguro


Ingresa el VIN y las placas de tu vehículo para comenzar

Buscar Vehículo

Datos del Vehículo

Año	1981
Marca	TOYOTA
Modelo	Corona
Placas	ABC 1234




Planes Disponibles



Cobertura amplia

\$12000 /anual

Cubre daños propios, robo total, responsabilidad civil, gastos médicos y auto sustituto.

 Inicio Cotizar Perfil

```
1 import React, { useEffect, useState } from 'react';
2 import { SafeAreaView } from 'react-native';
3 import { Box, VStack, ScrollView } from 'native-base';
4 import Header from '../components/Header';
5 import QuoteForm from '../components/QuoteForm';
6 import AsyncStorage from '@react-native-async-storage/async-storage';
7
8 const QuoteScreen = () => {
9
10   const [userData, setUserData] = useState(null);
11   const [isOpen, setIsOpen] = useState(false);
12
13   const onClose = () => setIsOpen(false);
14
15   const cancelRef = React.useRef(null);
16
17   useEffect(() => {
18     getUserData();
19   }, []);
20
21   const getUserData = async () => {
22     try {
23       const data = await AsyncStorage.getItem('@user_data');
24       if (data) {
25         setUserData(JSON.parse(data));
26       }
27     } catch (error) {
28       console.error('Error retrieving user data:', error);
29     }
30   };
31
32   console.log("userData:", userData);
33
34   return (
35     <SafeAreaView style={{ flex: 1, backgroundColor: '#f5f5f5' }}>
36       <VStack flex={1}>
37         <ScrollView>
38           <Header />
39           <Box p={4}>
40             <QuoteForm userData={userData}/>
41           </Box>
42         </ScrollView>
43       </VStack>
44     </SafeAreaView>
45   );
46 };
47
48 export default QuoteScreen;
```

Quote Form as complement for Quote Screen

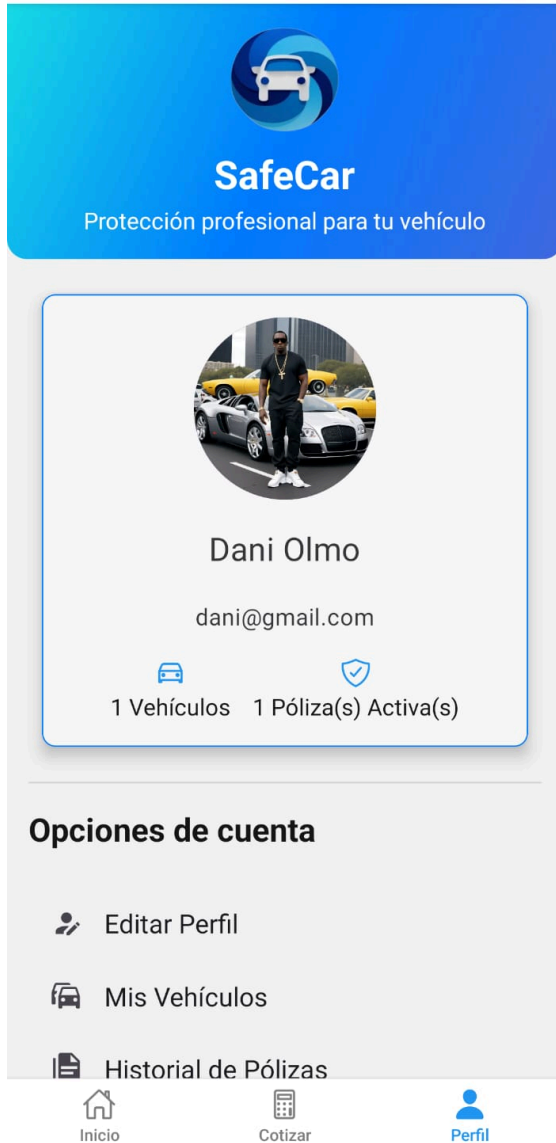
```
1 import React, { useState, useEffect } from "react";
2 import { VStack, Box, Button, Text, Input, HStack, Pressable, Icon, Modal, useToast } from "native-base";
3 import { Car, Shield, ShieldCheck, ShieldPlus } from "lucide-react-native";
4 import CreditCardForm from './CreditCardForm';
5 import { collection, getDocs } from 'firebase/firestore';
6 import { db } from '../config/firebaseConfig';
7
8 const QuoteForm = ({ userData }) => {
9   const [VIN, setVIN] = useState("");
10  const [plates, setPlates] = useState(""); // Placas como estado
11  const [loading, setLoading] = useState(false);
12  const [carData, setCarData] = useState(null);
13  const [planes, setPlanes] = useState([]);
14  const [selectedPlan, setSelectedPlan] = useState(null);
15  const [showPaymentModal, setShowPaymentModal] = useState(false);
16
17  const toast = useToast();
18
19  // Obtener planes de seguro
20  useEffect(() => {
21    const fetchPlanes = async () => {
22      try {
23        const planesSnapshot = await getDocs(collection(db, "polizas"));
24        const planesData = [];
25
26        const iconMap = {
27          basico: Shield,
28          respCivil: ShieldCheck,
29          amplio: ShieldPlus
30        };
31
32        planesSnapshot.forEach((doc) => {
33          planesData.push({
34            id: doc.id,
35            icon: iconMap[doc.id],
36            ...doc.data()
37          });
38        });
39
40        setPlanes(planesData);
41      } catch (error) {
42        console.error("Error al obtener planes:", error);
43        toast.show({
44          description: "No se pudieron cargar los planes disponibles",
45          status: "error"
46        });
47      }
48    };
49    fetchPlanes();
50  }, []);
51
52  const fetchCarDataByVIN = async () => {
53    if (!VIN || VIN.length !== 17) {
54      toast.show({
55        description: "Por favor ingresa un VIN válido de 17 caracteres",
56        status: "warning"
```

Profile Screen

5:48



Perfil



```

1 import React, { useState, useEffect } from "react";
2 import { ScrollView } from "react-native";
3 import { VStack, Box, Text, Divider, Button, AlertDialog } from "native-base";
4 import { List } from "react-native-paper";
5 import AsyncStorage from "@react-native-async-storage/async-storage";
6 import { useNavigation } from "@react-navigation/native";
7 import Header from "../components/Header";
8 import ProfileCard from "../components/ProfileCard";
9
10 const ProfileScreen = ({ handleLogout }) => {
11   const navigation = useNavigation();
12   const [userData, setUserData] = useState(null);
13   const [isOpen, setIsOpen] = useState(false);
14
15   const onClose = () => setIsOpen(false);
16
17   const cancelRef = React.useRef(null);
18
19   useEffect(() => {
20     getUserData();
21   }, []);
22
23   const getUserData = async () => {
24     try {
25       const data = await AsyncStorage.getItem("@user_data");
26       if (data) {
27         setUserData(JSON.parse(data));
28       }
29     } catch (error) {
30       console.error("Error retrieving user data:", error);
31     }
32   };
33
34   const handleLogoutConfirmation = () => {
35     setIsOpen(true);
36   };
37
38   const confirmLogout = () => {
39     onClose();
40     handleLogout();
41   };
42
43   return (
44     <ScrollView>
45       <VStack>
46         <Header />
47         <Box ps(4)>
48           <ProfileCard userData={userData} />
49           <Divider my(4) />
50           <Text fontSize="xl" fontWeight="bold" mb(4)>
51             Opciones de cuenta
52           </Text>
53           <List.Section>
54             <List.Item>
55               title="Editar Perfil"
56               left={
57                 <List.Icon {...props} icon="account-edit" />
58               }
59               onPress={() => navigation.navigate("editPerfil")}
60             </List.Item>
61             <List.Item>
62               title="Mis Vehículos"
63               left={
64                 <List.Icon {...props} icon="car-multiple" />
65               }
66               onPress={() => navigation.navigate("myVehicles")}
67             </List.Item>
68             <List.Item>
69               title="Historial de Pólizas"
70               left={
71                 <List.Icon {...props} icon="file-document-multiple" />
72               }
73               onPress={() => navigation.navigate("Polizes")}
74             </List.Item>
75             <List.Item>
76               title="Configuración"
77               left={
78                 <List.Icon {...props} icon="cog" />
79               }
80               onPress={() => navigation.navigate("Configuration")}
81             </List.Item>
82           </List.Section>
83           <Button mt(4) colorScheme="danger" onPress={handleLogoutConfirmation}>
84             Cerrar Sesión
85           </Button>
86         </Box>
87       </VStack>
88       <AlertDialog leastDestructiveRef={cancelRef} isOpen={isOpen} onClose={onClose}>
89         <AlertDialog.Content>
90           <AlertDialog.CloseButton />
91         </AlertDialog.Content>
92       </AlertDialog>
93     </ScrollView>
94   );
95 }

```

My reports option with is code



```
1 "use client"
2
3 import { useState, useEffect } from "react"
4 import { ScrollView, TouchableOpacity } from "react-native"
5 import {
6   VStack,
7   Box,
8   Text,
9   Heading,
10  HStack,
11  Icon,
12  Divider,
13  Badge,
14  Spinner,
15  Modal,
16  Button,
17  Image,
18  useToast
19 } from "native-base"
20 import { MaterialCommunityIcons, Ionicons } from "@expo/vector-icons"
21 import { collection, getDocs, query, orderBy, doc, getDoc } from "firebase/firestore"
22 import { db } from "../config/firebaseConfig";
23 import AsyncStorage from "react-native-async-storage/async-storage"
24 import Header from "../components/Header"
25 import { LinearGradient } from "expo-linear-gradient"
26
27 // Definición de tipos de incidentes
28 const INCIDENT_TYPES = {
29   collision: {
30     name: "Colisión",
31     icon: "car-emergency"
32   },
33   roadside: {
34     name: "Asistencia vial",
35     icon: "tow-truck"
36   },
37   glass: {
38     name: "Rotura de cristales",
39     icon: "car-door"
40   },
41   theft: {
42     name: "Robo",
43     icon: "shield-alert"
44   },
45   terceros: {
46     name: "Daños a terceros",
47     icon: "car-multiple"
48   }
49 }
50
51 const TrackReportsScreen = ({ navigation }) => {
52   const [reports, setReports] = useState([])
53   const [loading, setLoading] = useState(true)
54   const [userData, setUserData] = useState(null)
55   const [selectedReport, setSelectedReport] = useState(null)
56   const [showModal, setShowModal] = useState(false)
57   const [vehicles, setVehicles] = useState({}) // Cache para vehiculos
58
59   const toast = useToast()
60
61   useEffect(() => {
62     getUserData()
63   }, [])
64
65   useEffect(() => {
66     if (userData) {
67       fetchUserReports()
68     }
69   }, [userData])
70
71   const getUserData = async () => {
72     try {
73       const data = await AsyncStorage.getItem("@user_data")
74       if (data) {
75         setUserData(JSON.parse(data))
76       }
77     } catch (error) {
78       console.error("Error retrieving user data:", error)
79       toast.show({
80         description: "Error al obtener datos del usuario",
81         status: "error"
82       })
83     }
84   }
```


Payment from quote form with is code

Cotizar

Planes Disponibles

Card

5216 2134 5200 2648

Garcia Garfias03/23

Número de Tarjeta

5216 2134 5200 2648

Titular de la Tarjeta

Garcia Garfias

Vencimiento

03/23

CVV

105

Pagar \$3000 MXN

Pagar \$3000 MXN

Inicio Cotizar Perfil

```
1 import React, { useState, useEffect } from 'react';
2 import {
3   VStack,
4   Box,
5   Input,
6   Text,
7   Button,
8   HStack,
9   FormControl,
10  useToast
11 } from 'native-base';
12 import { CreditCard } from 'lucide-react-native';
13 import { collection, addDoc, serverTimestamp } from 'firebase/firestore';
14 import { db } from '../db/firebase'; // Asegúrate que la ruta sea correcta
15
16 const CreditCardForm = ({ amount, onSuccess, onClose, userData, cardData, selectedP
17 const [cardNumber, setCardNumber] = useState('');
18 const [cardHolder, setCardHolder] = useState('');
19 const [expiry, setExpiry] = useState('');
20 const [cvv, setCvv] = useState('');
21 const [errors, setErrors] = useState({});
22 const toast = useToast();
23
24 const formatCardNumber = (text) => {
25   const cleaned = text.replace(/\s+/g, '').replace(/[^0-9/gi, ' ');
26   const matches = cleaned.match(/^(0-9){4,16}/g);
27   const match = (matches && matches[0]) || '';
28   const parts = [];
29   for (let i = 0, len = match.length; i < len; i += 4) {
30     parts.push(match.substring(i, i + 4));
31   }
32   if (parts.length) {
33     return parts.join(' ');
34   } else {
35     return text;
36   }
37 };
38
39 const formatExpiry = (text) => {
40   const cleaned = text.replace(/[^0-9/g, ' ');
41   if (cleaned.length >= 2) {
42     return `${cleaned.slice(0, 2)}/${cleaned.slice(2, 4)}`;
43   }
44   return cleaned;
45 };
46
47 const validate = () => {
48   const newErrors = {};
49
50   if (!cardNumber || cardNumber.replace(/\s+/g, '').length !== 16) {
51     newErrors.cardNumber = 'Número de tarjeta inválido';
52   }
53
54   if (!cardHolder) {
55     newErrors.cardHolder = 'Nombre requerido';
56   }
57
58   if (!expiry || !expiry.includes('/')) {
59     newErrors.expiry = 'Fecha inválida';
60   } else {
61     const [month, year] = expiry.split('/');
62     const currentYear = new Date().getFullYear() % 100;
63     const currentMonth = new Date().getMonth() + 1;
64
65     if (parseInt(month) < 1 || parseInt(month) > 12) {
66       newErrors.expiry = 'Mes inválido';
67     } else if (parseInt(year) < currentYear ||
68       (parseInt(year) === currentYear && parseInt(month) < currentMonth)) {
69       newErrors.expiry = 'Tarjeta vencida';
70     }
71   }
72
73   if (!cvv || cvv.length !== 3) {
74     newErrors.cvv = 'CVV inválido';
75   }
76
77   setErrors(newErrors);
78   return Object.keys(newErrors).length === 0;
79 };
80
81 const saveTransactionData = async () => {
82   try {
83     const userRef = `log/${userData.id}`; // Ruta base del usuario
```