
Software Requirements Specification

Project: Industrial quality area

Document file

Date	Revision	Author	Verified dep. quality
03/19/2025	1.0	Martinez Lopez Santi Erubey Barragan Hernandez Antonio Lemus Tarango naomi Denisse	

Document validated by the parties on date:

By the client	By the supplying company
Fdo. D./ Dña	Fdo. D./Dña

Software Requirements Specification

Document file

1. Introduction

- 1.1. Purpose
- 1.2. Product Scope
- 1.3. Staff Involved
- 1.4. Definitions and Acronyms
- 1.5. References
- 1.6. Overview

2. General Description

- 2.1. Product Perspective
- 2.2. Product Functionality
- 2.3. User Characteristics
- 2.4. Constraints
- 2.5. Assumptions and Dependencies
- 2.6. Apportioning of Requirements

3. Specific Requirements

- 3.2. Functional Requirement
- 3.3. Non-functional requirements
 - 3.4. Performance Requirements
 - 3.5. Non-functional requirements
 - 3.5.1. Optimization
 - 3.5.2. Security
 - 3.5.3. Availability
 - 3.5.4. Intuitive
- 3.6. System Attributes
- 3.7. External Interface
- 3.8. Technologies and Tools
- [3.9. Use cases](#)
- [3.10. Data base](#)
- [3.11. screen previews](#)

1. Introduction

This document describes the functional and non-functional requirements for the development of a quality management system. The system will enable an organization to efficiently oversee and optimize processes related to quality assurance, certification, and compliance, ensuring adherence to established quality standards across key areas such as process control, product evaluation, operational safety, regulatory compliance, and continuous improvement initiatives..

1.1. Purpose

The purpose of the project is to have a control of the products that are handled in an industrial facility, showing the functional and non-functional requirements will make known what is the purpose of this.

1.2. Product Scope

The system will allow sector workers to check the status of products passing through IoT device scanning and the app's camera. By leveraging IoT devices, the system will automatically collect data on product quality, such as weight, dimensions, temperature, or any other relevant metrics, depending on the type of product. This information will be analyzed and stored in real time.

Through the app's camera, workers will also have the capability to perform manual quality inspections by scanning product labels, QR codes, or barcodes.

1.3. Staff Involved

Name	Martinez Lopez Santi Erubey
Professional Category	TSU-Software Development
Responsibilities	Software programming
Contact Information	032305914@ut-tijuana.edu.mx

Name	Lemus Tarango Naomi Denisse
Professional Category	TSU-Software Development
Responsibilities	documentation and database
Contact Information	0323105950@ut-tijuana.edu.mx

Software Requirements Specification

Name	Barragan Hernandez Antonio
Professional Category	TSU-Software Development
Responsibilities	IOT process
Contact Information	3023105853@ut-tijuana.edu.mx

Software Requirements Specification

1.4. Definitions and Acronyms

Name	Description
User	Person who uses the system to manage the inventory
SRS	Software Requirements Specification
FR	Functional requirement
NFR	Non-Functional requirement

1.5. References

Reference	Title
Standard IEEE 830	IEEE
Fontela, C. (2012). ISBN: 978-987-1609-22-2	UML Software Model for Professionals.

1.6. Overview

In the first part we will show a brief introduction of what the document is about, together with its purpose. In the second part we will show the general description of the project and at the end the requirements.

2. General Description**2.1. Product Perspective**

The system will be a mobile application developed in JAVASCRIPT that will manage activities related to the Quality Management System. Sector workers will be able to realize different actions related to the quality sector.

2.2. Product Functionality

The software project consists of an application focused on the quality process of medical products, specifically single-use devices (disposable products) which are the following:

- Needles
- Syringes
- Catheters
- Medical gloves
- Masks
- Bandages

The program consists of having a quality control of these products, these would be in boxes which will have a QR code and will pass through a conveyor belt that will have a weight sensor, in case it does not have its code or does not comply with the indicated weight, a notice will be given by the application in which the person in charge will have to review the product and make a report of the incident and its QR will be scanned to give a confirmation notice and follow its process.

We will use a database already established by another department, more products cannot be added, only queries can be made.

The following requirements must be met:

1. The QR code must have information about the type of product (product name, product ID, production date, name of the company to which it will be addressed), date, weight.
2. The report will be used to record the error of the product problem and will have a record of the error giving the following data: date, time, name of the person in charge, product, product number, description. To then record your QR code, this being your confirmation that the product was confirmed.
3. The tables are already previously made, the products already registered in the database to be able to make queries about their quality packaging. Once everything is done, another department will be in charge of its distribution.

Software Requirements Specification**2.3. User Characteristics**

The users that will be used will be only one, and it will be the person in charge of the quality area.

User type	admin
Training	Review and monitor the presenting problem.
Activities	monitor issues and resolve them through the app

2.4. Constraints

- The program will be executed in the form of an app.
- Language and technologies of use: HTML, CSS, JavaScript and MySQL.

2.5. Assumptions and Dependencies

- It must be a stable system for the user.
- Only trained people can use it.
- The materials needed must meet the requirements.

2.6. Apportioning of Requirements

- Performance optimization.
- Advanced security.
- Implementation of new areas.
- Fix bugs and problems that arise in the system

3. Specific Requirements

- User Interface: The application must have an intuitive and user-friendly interface.
- Equipment Verification: Users must be able to verify the quality of the product
- Reporting issues: Users should be able to submit a report about problems with a product when passing through the quality verification device.
- Reports: Users must submit a report when the product is verified showing the time, date, and who verified the product..

Software Requirements Specification

3.2. Functional Requirement

Requirement number	RF01
Requirement name	read QR codes
Description of the requirement	Sector employees must be able to manually perform product quality inspections using the mobile app's camera for scanning product labels, QR codes, or barcodes.
NON-functional requirement	•
Priority of the requirement	High

Requirement number	RF02
Requirement name	Automatic system that monitors
Description of the requirement	The system must automatically collect and analyze data from IoT devices to monitor product quality metrics in real time.
NON-functional requirement	•
Functional requirement	
Priority of the requirement	High

Requirement number	RF03
Requirement name	identify faults
Description of the requirement	The system must generate automatic alerts to sector workers if a product fails the quality verification process.
NON-functional requirement	•
Priority of the requirement	High

Requirement number	RF04
Requirement name	make reports
Description of the requirement	The system must allow employees to submit reports documenting product inspection results, including the time, date, and identity of the inspector.
NON-functional requirement	•
Priority of the requirement	High

Requirement number	RF05
Requirement name	inspect for faults
Description of the requirement	The system must provide detailed reports of product inspections, including metrics such as weight, dimensions, and other key quality factors.
NON-functional requirement	•
Priority of the requirement	High

Requirement number	RF06
Requirement name	inspect for faults

Software Requirements Specification

Description of the requirement	The system must provide detailed reports of product inspections, including metrics such as weight, dimensions, and other key quality factors.
NON-functional requirement	•
Priority of the requirement	Hlgh

Requirement number	RF07
Requirement name	Real-time dashboard
Description of the requirement	The system must provide a real-time dashboard for supervisors, displaying quality metrics, alerts, and inspection status summaries.
NON-functional requirement	•
Priority of the requirement	Hlgh

Requirement number	RF08
Requirement name	Customizable inspection templates
Description of the requirement	The app must allow employees to attach multimedia files, such as photos or videos, to inspection reports for better documentation of defects or issues.
NON-functional requirement	•
Priority of the requirement	Hlgh

Requirement number	RF09
Requirement name	Support for multimedia attachments
Description of the requirement	The system must provide detailed reports of product inspections, including metrics such as weight, dimensions, and other key quality factors.
NON-functional requirement	•
Priority of the requirement	Hlgh

Requirement number	RF10
Requirement name	Analytics and insights
Description of the requirement	The system must generate periodic analytical reports, providing insights on quality trends, recurring issues, and process efficiency.
NON-functional requirement	•
Priority of the requirement	Hlgh

3.3. Non-functional requirements

Requirement number	RNF01
Requirement name	Security
Description of the requirement	security is not necessary as the system will be able to function without it.
Priority of the requirement	High

Requirement name	Optimization
Description of the requirement	even if it is not optimized quickly, the system can still be used.
Priority of the requirement	High

Requirement number	RNF02
Requirement name	User interface
Description of the requirement	The app must be simple but understandable for the user.
Priority of the requirement	High

3.4. Performance Requirements

- **RP1: Response Time** The system must be able to process any request (such as product inspections, generating reports, or updating data) within a maximum time of 2 seconds under normal load conditions to ensure real-time operation and efficiency.
- **RP2: Scalability** The system must be able to handle a minimum of 200 concurrent users while maintaining the same level of performance. The system must be capable of managing up to 3,000 active quality inspection records and IoT-connected devices without affecting performance, ensuring smooth operation even with high data volumes.
- **RP3: Resource Consumption** CPU usage on the server must not exceed 75% during normal operations, and memory usage must not exceed 60% under medium load scenarios. The system must be optimized to reduce bandwidth usage, particularly when transmitting large quality inspection reports or data from IoT devices, ensuring that network resources are efficiently utilized.
- **RP4: Availability** The system must have an availability rate of 99%, ensuring it is available to users at all times for product inspections and quality control tasks. The system must implement backup and automatic recovery mechanisms in case of system failures, allowing for quick restoration of services.
- **RP5: Handling Large Volumes of Data** The system must be capable of storing and retrieving historical inspection records and quality data without significantly affecting performance. Queries about the product inspection history or data from IoT devices must execute within a maximum time of 3 seconds, ensuring that users can access critical information quickly.
- **RP6: Analytics and Insights**
The system must be capable of generating analytical reports and insights from large volumes of historical quality data within a maximum time of 5 seconds. Reports must include key trends, recurring issues, and process efficiency metrics, ensuring supervisors and management can make timely decisions.

Software Requirements Specification

- RP7: Real-time Dashboard

The system must update the real-time dashboard with quality metrics, alerts, and inspection summaries within 2 seconds of data being received or processed. The dashboard must remain responsive and provide seamless updates even under high data loads.

- RP8: Customizable Inspection Templates

The system must allow administrators to create, edit, and save customized inspection templates within 3 seconds. Retrieval of pre-existing templates must occur in less than 2 seconds, ensuring an efficient customization workflow.

- RP9: Multimedia Attachments

The system must support uploading multimedia attachments, such as photos and videos, with a maximum file size of 50 MB per file. Uploads must complete within 5 seconds under normal network conditions, and multimedia files must be retrievable within 3 seconds from the inspection records.

3.5. Non-functional requirements

3.5.1. Optimization

The system must be optimized to perform processes immediately without noticeable delays.

3.5.2. Security

Users should only be able to perform actions corresponding to their role within the company.

3.5.3. Availability

The program must be available at any time it is required during working hours.

3.5.4. Intuitive

The web page must have a user-friendly and intuitive interface, using different types of menus and modules that are easy to understand for any user at any given time.

3.6. System Attributes

1. Security

- Confidential data.
- All system activities must be logged and monitored.

2. Usability

- The system must be intuitive and easy to use for people with varying levels of technological skill.
- The training time required for users to operate the system efficiently should be minimized.

3. Portability

- The software must be compatible with multiple operating systems to ensure flexibility in server choice

3.7. External Interface

User Interface

- Description: The system must provide an intuitive and user-friendly graphical interface for employees.
- Technology: Languages such as JavaScript and other materials are being used for its implementation.
- Employees: They must be able to carry out the entire quality management process from the IoT device to verification in the app.
- The interfaces will be as follows:
- Login: The user will be able to log in with their already registered account.
- View reports: The latest reports will be displayed upon logging in.
- View alerts: Selecting the alert option will display the inspection screen. By pressing the inspect option, you will be able to view the issue and make changes, where you will be able to record the QR code and status.
- View history: In the history option, you will be able to view the overall checkout history and a change history where you will only be able to see the changes made.
- Create and view reports: In this section, you will be able to fill out a report on a given issue and view other issues.

3.8. Technologies and Tools

For the development of the quality management system, the following technologies and tools will be used, selected for their compatibility, performance, and ease of implementation in the industrial environment:

Programming Languages

- **JavaScript:** Used for client-side logic and potentially in the backend with Node.js if necessary.
- **HTML, CSS & PHP:** For the structure and design of the web user interface, ensuring an intuitive and accessible experience.

Database

- **MySQL:** Chosen as the database management system (DBMS) due to its stability, scalability, and compatibility with project requirements.

Frameworks and Libraries

- **React Native:** Used for mobile application development, providing a smooth and native experience on Android and iOS devices.
- **Node.js (optional):** For backend management if JavaScript is used on the server.
- **Express.js (optional):** If Node.js is used, this framework will handle API routes and logic.

Devices and Hardware

- **IoT Devices:** Weight sensors, cameras, and QR or barcode scanners to enable automatic data capture for quality control.
- **Servers:** A local or cloud server will be evaluated for data storage and processing.

Security and Access Control

- **User Roles:** Access levels will be implemented to ensure that only authorized personnel can perform specific actions.
- **Data Encryption:** Encryption of sensitive information will be considered if necessary.

Software Requirements Specification

Integrations and APIs

- The possibility of integrating external APIs to enhance system functionality will be evaluated (e.g., authentication services or cloud storage).

This set of tools and technologies ensures that the system is efficient, scalable, and easy to maintain in the industrial environment.

3.9. Use cases

Use Case 1: Login

Name: User Login

Author: Barragan

Date: 02/25/2025

Description: The user will be able to access the platform using their registered account.

Actors: User

Preconditions: The user must have a registered account.

Normal Flow:

The user accesses the login screen.

They enter their credentials (username and password).

They press the "Login" button.

The system validates the data and allows access.

Alternative Flow:

If the data is incorrect, an error message is displayed and a request is made to try again.

Postconditions: The user accesses the system with their active profile.

Use Case 2: View Reports

Name: View Recent Reports

Author: Denisse

Date: 02/25/2025

Description: Upon logging in, the user will be able to view the most recent quality reports registered.

Actors: User

Preconditions: The user must be logged in.

Normal Flow:

The system automatically displays recent reports upon login.

The user can click on each report to view more details.

Alternative Flow:

If there are no registered reports, a message indicating "No reports available" is displayed.

Postconditions: The user views the reports without modifying them.

Use Case 3: View Alerts and Inspect

Name: Alert Management and Product Inspection

Author: Denisse

Date: 02/25/2025

Description: The user will be able to view active alerts and perform the corresponding inspection of the defective product.

Actors: User

Software Requirements Specification

Preconditions: The user must be logged in and there must be active alerts.

Normal Flow:

The user selects the "Alerts" option.

The system displays the products with issues.

The user selects "Inspect."

The problem details are displayed.

The user can scan the QR code again and update the product's status.

Alternative Flow:

If the new QR code is not scanned, the status cannot be updated.

Postconditions: The product is marked as reviewed.

Use Case 4: View History

Name: View the history of revisions and changes.

Author: Barragan

Date: 02/25/2025

Description: The user can view the general history of revisions performed and the history of changes made to the system.

Actors: User

Preconditions: The user must be logged in.

Normal Flow:

The user accesses the "History" section.

Selects between "General History" or "Change History."

The system displays the corresponding information.

Alternative Flow:

If there are no records yet, the system will display "No history available."

Postconditions: The user views the history without modifying it.

Use Case 5: Create and View Incident Reports

Name: Creating and Viewing Quality Reports

Author: Santi

Date: 02/25/2025

Description: The user will be able to create a new report on a detected incident and view other reports already generated.

Actors: User

Preconditions: The user must be logged in and have performed an inspection.

Normal Flow:

The user accesses the "Reports" section.

Selects "Create New Report."

Fills out the form with the incident details.

Save the report.

You can also view other previously recorded reports.

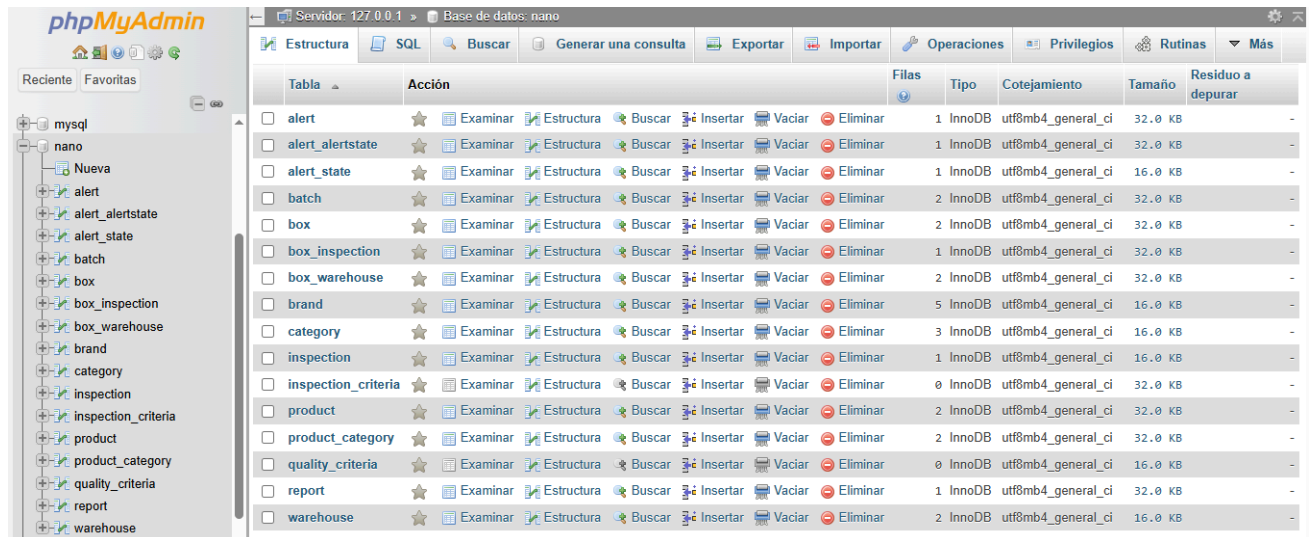
Alternative Flow:

If the form is not completed correctly, the system will display an error and will not allow saving.

Postconditions: The report is registered in the system and available for viewing.

3.10. Data base

In this section the database that will be implemented in the project will be shown:



The screenshot shows the phpMyAdmin interface with the 'nano' database selected. The 'Estructura' (Structure) tab is active, displaying a list of tables and their fields.

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
alert	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	32.0 KB	-
alert_alertstate	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	32.0 KB	-
alert_state	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	16.0 KB	-
batch	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	32.0 KB	-
box	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	32.0 KB	-
box_inspection	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	32.0 KB	-
box_warehouse	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	32.0 KB	-
brand	Examinar Estructura Buscar Insertar Vaciar Eliminar	5	InnoDB	utf8mb4_general_ci	16.0 KB	-
category	Examinar Estructura Buscar Insertar Vaciar Eliminar	3	InnoDB	utf8mb4_general_ci	16.0 KB	-
inspection	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	16.0 KB	-
inspection_criteria	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	32.0 KB	-
product	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	32.0 KB	-
product_category	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	32.0 KB	-
quality_criteria	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	16.0 KB	-
report	Examinar Estructura Buscar Insertar Vaciar Eliminar	1	InnoDB	utf8mb4_general_ci	32.0 KB	-
warehouse	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8mb4_general_ci	16.0 KB	-

Figure 1. Database in xampp.

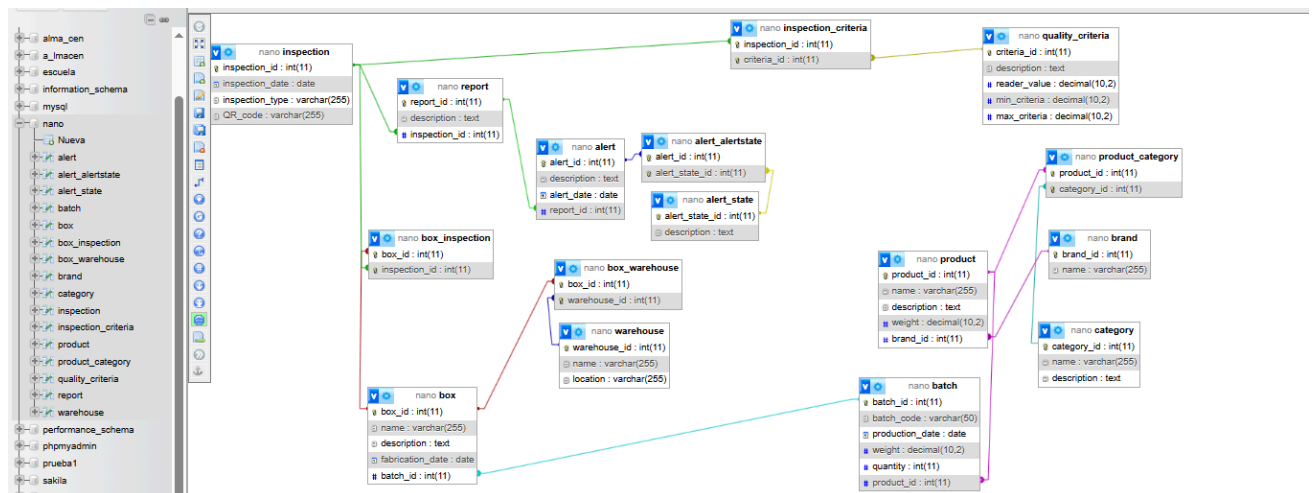


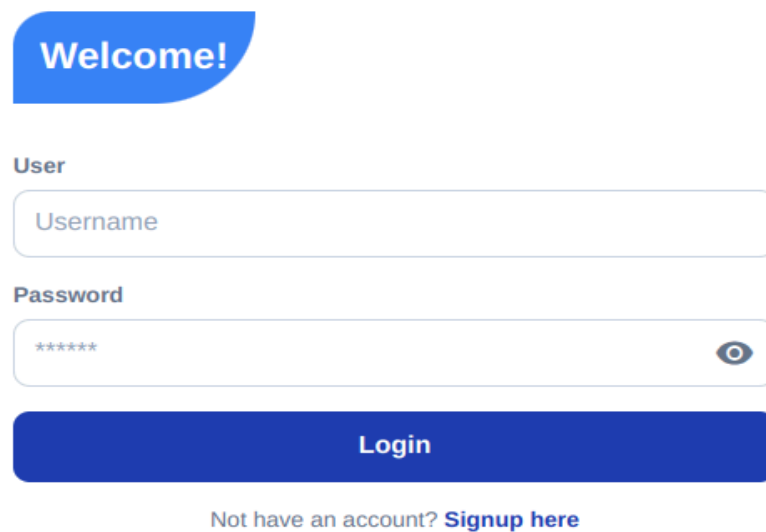
Figure 2. MR of the program.

3.11. screen previews

This section will show the progress made on the screen of a phone app:

In this part you will find the login using a database connection with firebase taking the user's name to validate if it is in the database and verify if the password is correct

Login



The login screen features a blue header with the text "Welcome!". Below this, there are two input fields: "User" with a placeholder "Username" and "Password" with a placeholder "*****". A blue "Login" button is positioned below the password field. At the bottom, there is a link that says "Not have an account? [Signup here](#)".

Figure 3. login screen.

Software Requirements Specification

CODE:

```
import React, { useState } from "react";
import { View, Text, TextInput, TouchableOpacity, Alert } from
"react-native";
import Icon from "react-native-vector-icons/MaterialCommunityIcons";
import { getFirestore, collection, query, where, getDocs } from
"firebase/firestore";
import { db } from "../firebaseConfig";
import { useNavigation } from '@react-navigation/native';

const LoginScreen = () => {
  const [usuario, setUsuario] = useState(''); // Estado para el usuario
  const [password, setPassword] = useState(''); // Estado para la contraseña
  const [passwordVisible, setPasswordVisible] = useState(false);

  const navigation = useNavigation(); // Hook para la navegación
  const handleLogin = async () => {
    try {
      // Verificar si 'usuario' y 'password' están definidos
      if (!usuario || !password) {
        console.log("Nombre o contraseña vacíos");
        Alert.alert("Error", "Por favor ingresa nombre y contraseña.");
        return;
      }
      // Referencia a la colección "Employee"
      const EmployeeRef = collection(db, "Employee");
      // Verificar el valor de 'usuario' antes de la consulta
      console.log("Buscando empleado con el nombre: ", usuario);
      // Realizamos una consulta para encontrar el empleado por el "usuario"
      const q = query(EmployeeRef, where("name", "==", usuario));
      const querySnapshot = await getDocs(q);
      console.log("Documentos encontrados: ", querySnapshot.size); // Ver
cuántos documentos se encontraron
      if (!querySnapshot.empty) {
        // Si encontramos el usuario, verificamos la contraseña
        const EmployeeData = querySnapshot.docs[0].data();
        console.log("Empleado encontrado: ", EmployeeData);
        // Verificar la contraseña
        if (EmployeeData.password === password) {
          // Login exitoso
          Alert.alert("Successfully", "¡Welcome!");
          console.log("Login exitoso");
          // Redirigir a la pantalla Home después del login exitoso
        }
      }
    }
  }
}
```

Software Requirements Specification

```
navigation.navigate("Home"); // Esto redirige a la pantalla Home
} else {
  // Contraseña incorrecta
  Alert.alert("Error", "Incorrect Password.");
  console.log("Contraseña incorrecta");
}
} else {
  // Si no encontramos el usuario
  Alert.alert("Error", "User not found.");
  console.log("Usuario no encontrado");
}
} catch (error) {
  // Manejo de errores generales
  console.error("Error al intentar hacer login: ", error);
  Alert.alert("Error", "Ocurrió un problema al iniciar sesión.");
}
};

return (
  <View style={{ flex: 1, justifyContent: "center", alignItems: "center",
backgroundColor: "white", paddingHorizontal: 24 }}>

    {/* Header con Bienvenida */}
    <View style={{ width: "100%", alignItems: "flex-start", marginBottom: 32
}}>

      <View style={{ backgroundColor: "#3B82F6", padding: 16,
borderTopLeftRadius: 30, borderBottomRightRadius: 80 }}>
        <Text style={{ fontSize: 24, fontWeight: "bold", color: "white"
}}>Welcome!</Text>
      </View>
    </View>

    {/* Campo de Usuario */}
    <Text style={{ alignSelf: "flex-start", fontSize: 14, fontWeight: "600",
color: "#64748B", marginBottom: 8 }}>User</Text>
    <View style={{ width: "100%", flexDirection: "row", alignItems:
"center", borderWidth: 1, borderColor: "#CBD5E1", borderRadius: 10,
paddingHorizontal: 12, marginBottom: 16 }}>
      <TextInput
        placeholder="Username"
        placeholderTextColor="#94A3B8"
        value={usuario}
        onChangeText={(text) => setUsuario(text)} // Actualizar estado del
```

Software Requirements Specification

`usuario`

```
        style={{ flex: 1, paddingVertical: 12, fontSize: 16 }}
      />
    </View>

    { /* Campo de Contraseña */ }
    <Text style={{ alignSelf: "flex-start", fontSize: 14, fontWeight: "600",
color: "#64748B", marginBottom: 8 }}>Password</Text>
    <View style={{ width: "100%", flexDirection: "row", alignItems:
"center", borderWidth: 1, borderColor: "#CBD5E1", borderRadius: 10,
paddingHorizontal: 12, marginBottom: 16 }}>
      <TextInput
        placeholder="*****"
        placeholderTextColor="#94A3B8"
        secureTextEntry={!passwordVisible}
        value={password}
        onChangeText={(text) => setPassword(text)} // Actualizar estado de
la contraseña
        style={{ flex: 1, paddingVertical: 12, fontSize: 16 }}
      />
      <TouchableOpacity onPress={() =>
setPasswordVisible(!passwordVisible)}>
        <Icon name={passwordVisible ? "eye-off" : "eye"} size={24}
color="#64748B" />
      </TouchableOpacity>
    </View>

    { /* Botón de Login */ }
    <TouchableOpacity
      onPress={handleLogin} // Llamar a la función de login cuando se
presiona el botón
      style={{
        width: "100%",
        backgroundColor: "#1E40AF",
        paddingVertical: 14,
        borderRadius: 10,
        alignItems: "center",
        marginBottom: 16,
      }}
    >
      <Text style={{ color: "white", fontWeight: "bold", fontSize: 16
}}>Login</Text>
    </TouchableOpacity>
```

Software Requirements Specification

```
    { /* Enlace para registro */ }  
    <Text style={{ color: "#64748B" }}>Not have an account? <Text style={{  
fontWeight: "bold", color: "#1E40AF" }}>Signup here</Text></Text>  
  
    </View>  
  );  
};  
  
export default LoginScreen;
```


Software Requirements Specification

In this part, which is when you log in, the first thing that is shown to the user is the alerts screen where pending alerts will be shown in case there are any when passing through the band

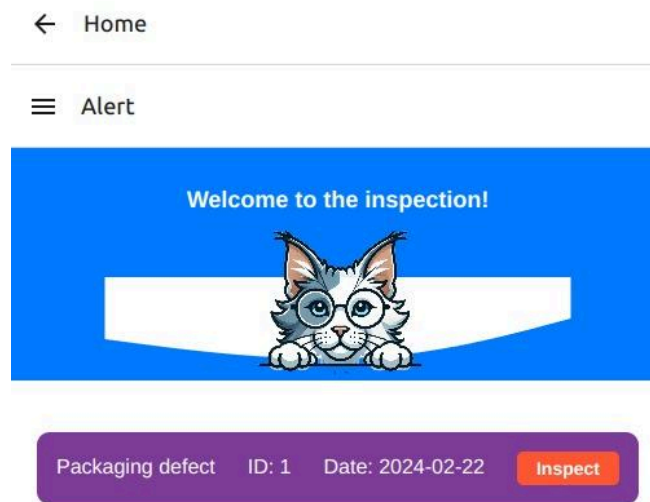


Figure 4. Alert screen.

Software Requirements Specification

CODE:

```
import React, { useState, useEffect } from 'react';
import { View, Text, TouchableOpacity, ScrollView, StyleSheet, Image, Alert
as RNAlert } from 'react-native';
import Svg, { Path } from 'react-native-svg';
import { getFirestore, collection, query, getDocs } from
'firebase/firestore';
import { FirebaseApp } from './firebaseConfig';
import { useNavigation } from '@react-navigation/native';

export default function Alert() {
  const [alerts, setAlerts] = useState([]);
  const navigation = useNavigation(); // Usar el hook de navegación
  useEffect(() => {
    const fetchAlerts = async () => {
      try {
        const db = getFirestore(FirebaseApp);
        const alertRef = collection(db, "Alert");
        const q = query(alertRef);
        const querySnapshot = await getDocs(q);

        if (querySnapshot.empty) {
          console.log("No hay alertas en la base de datos");
          return;
        }

        const fetchedAlerts = [];
        querySnapshot.forEach((doc) => {
          const alertData = doc.data();
          fetchedAlerts.push({
            alert_id: alertData.alert_id,
            alert_date: alertData.alert_date,
            description: alertData.description,
          });
        });

        setAlerts(fetchedAlerts);
      } catch (error) {
        console.error("Error al obtener las alertas: ", error);
        RNAlert.alert("Error", "Hubo un problema al obtener las alertas.");
      }
    };
  });
};
```

Software Requirements Specification

```

    fetchAlerts();
  }, []);

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <Svg height="80" width="100%" viewBox="0 0 1440 320"
style={styles.curve}>
          <Path
            fill="#fff"

d="M0,192L80,202.7C160,213,320,235,480,245.3C640,256,800,256,960,234.7C1120,2
13,1280,171,1360,149.3L1440,128V0H0Z"
          />
        </Svg>
        <Text style={styles.headerText}>Welcome to the inspection!</Text>
        <Image source={require('./assets/gatitoasomandose.png')}
style={styles.image} />
      </View>

      <ScrollView style={styles.alertContainer}>
        {alerts.length > 0 ? (
          alerts.map((alert, index) => (
            <View key={index} style={styles.alertBox}>
              <Text style={styles.alertText}>{alert.description}</Text>
              <Text style={styles.alertText}>ID: {alert.alert_id}</Text>
              <Text style={styles.alertText}>Date: {alert.alert_date}</Text>

              <TouchableOpacity
                style={styles.button}
                onPress={() => navigation.navigate('Inspection', { alertId:
alert.alert_id })}
              >
                <Text style={styles.buttonText}>Inspect</Text>
              </TouchableOpacity>
            </View>
          ))
        ) : (
          <Text style={styles.alertText}>No alerts available</Text>
        )}
      </ScrollView>
    </View>
  );

```

Software Requirements Specification

}

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#fff",
  },
  header: {
    backgroundColor: "#007BFF",
    height: 180,
    justifyContent: "center",
    alignItems: "center",
    position: "relative",
  },
  curve: {
    position: "absolute",
    bottom: 0,
  },
  headerText: {
    fontSize: 18,
    fontWeight: "bold",
    color: "#fff",
    position: "absolute",
    top: 30,
  },
  image: {
    width: 140,
    height: 150,
    position: "relative",
    bottom: -30,
  },
  alertContainer: {
    marginTop: 40,
    paddingHorizontal: 20,
  },
  alertBox: {
    backgroundColor: "#7D3C98",
    padding: 15,
    borderRadius: 10,
    marginBottom: 10,
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
  },
});
```

Software Requirements Specification

```
},
alertText: {
  color: "#fff",
  fontSize: 16,
},
button: {
  backgroundColor: "#FF5733",
  paddingHorizontal: 15,
  paddingVertical: 5,
  borderRadius: 5,
},
buttonText: {
  color: "#fff",
  fontWeight: "bold",
},
});
```

Software Requirements Specification

This section, which is the history section, shows both the history of inspections already carried out and the history of inspections where there were material changes

← Home

≡ History

Welcome to the history!



history

change history

QR Code: QR001

inspection Date: 2024-01-20

inspection ID: 1

inspection Type: Sterility Check

Software Requirements Specification

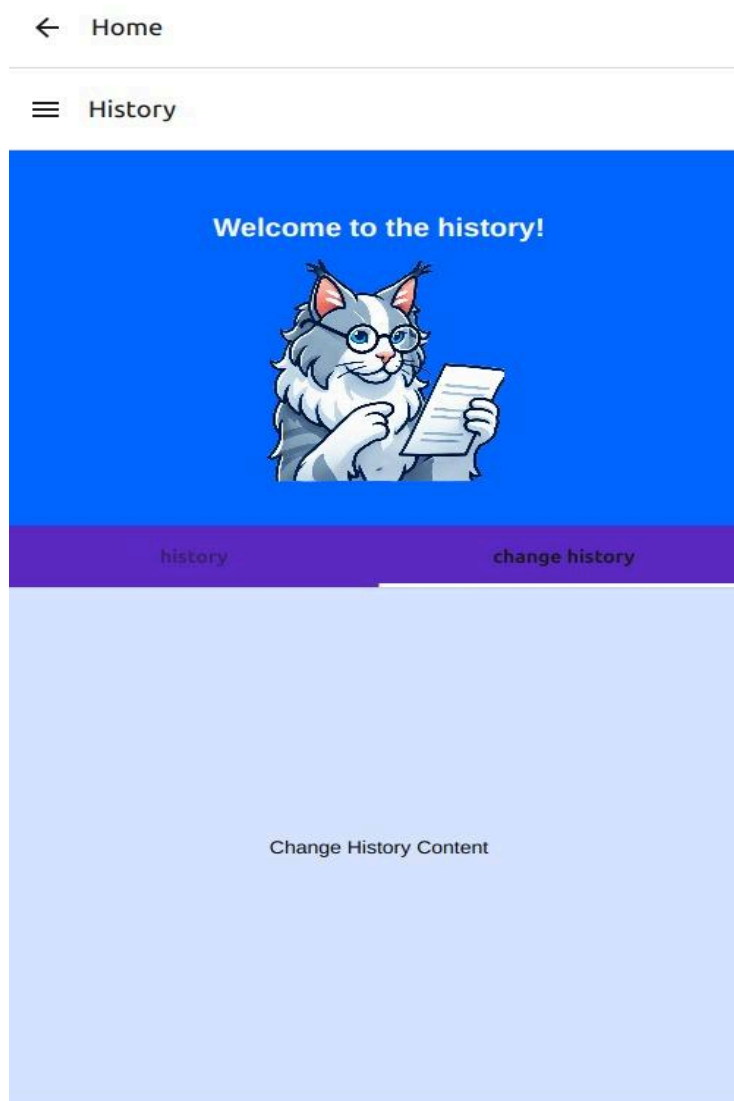


Figure 5. History screen.

Software Requirements Specification

CODE:

```
import React, { useState, useEffect } from "react";
import { View, Text, Image, TouchableOpacity, ScrollView, StyleSheet } from
"react-native";
import { createMaterialTopTabNavigator } from
"@react-navigation/material-top-tabs";
import { getFirestore, collection, query, getDocs } from
"firebase/firestore";
import { FirebaseApp } from './firebaseConfig';
const HistoryScreen = () => {
  const [inspections, setInspections] = useState([]);

  useEffect(() => {
    const fetchInspections = async () => {
      try {
        const db = getFirestore(FirebaseApp);
        const inspectionRef = collection(db, "Inspection");
        const q = query(inspectionRef);
        const querySnapshot = await getDocs(q);

        if (querySnapshot.empty) {
          console.log("No inspections found");
          return;
        }

        const fetchedInspections = [];
        querySnapshot.forEach((doc) => {
          const inspectionData = doc.data();
          fetchedInspections.push({
            inspection_id: inspectionData.inspection_id,
            inspection_date: inspectionData.inspection_date,
            qr_code: inspectionData.QR_code,
            inspection_type: inspectionData.inspection_type,
          });
        });

        setInspections(fetchedInspections);
      } catch (error) {
        console.error("Error fetching inspections:", error);
      }
    };

    fetchInspections();
  });
}
```


Software Requirements Specification

```

}, []);

return (
  <View style={styles.container}>
    {inspections.length > 0 ? (
      inspections.map((inspection, index) => (
        <TouchableOpacity
          key={index}
          style={styles.listItem}
          onPress={() => console.log(`Inspecting:
${inspection.inspection_id}`)}
        >
          <Text style={styles.listItem}>QR Code: {inspection.qr_code}</Text>
          <Text style={styles.listItem}>Inspection Date:
{inspection.inspection_date}</Text>
          <Text style={styles.listItem}>Inspection ID:
{inspection.inspection_id}</Text>
          <Text style={styles.listItem}>Inspection Type:
{inspection.inspection_type}</Text>
        </TouchableOpacity>
      ))
    ) : (
      <Text style={styles.noInspectionsText}>No inspections available</Text>
    )}
  </View>
);

const ChangeHistoryScreen = () => (
  <View style={styles.contentContainer}>
    <Text style={styles.contentText}>Change History Content</Text>
  </View>
);

// 📌 Tabs
const Tab = createMaterialTopTabNavigator();

const HistoryTabs = () => {
  return (
    <Tab.Navigator
      screenOptions={{
        tabBarLabelStyle: { fontSize: 14, fontWeight: "bold" },
        tabBarStyle: { backgroundColor: "#5c2ac2" },
      }}
    >

```

Software Requirements Specification

```
    tabBarIndicatorStyle: { backgroundColor: "white", height: 3 },
  }}
>
  <Tab.Screen name="history" component={HistoryScreen} />
  <Tab.Screen name="change history" component={ChangeHistoryScreen} />
</Tab.Navigator>
);
};

// 📌 Pantalla principal
export default function App() {
  return (
    <View style={styles.container}>
      {/* Encabezado con imagen */}
      <View style={styles.header}>
        <Text style={styles.headerText}>Welcome to the history!</Text>
        <Image source={require('./assets/michihistory.png')}
style={styles.catImage} />
      </View>

      {/* Tabs */}
      <HistoryTabs />

    </View>
  );
}

// 📌 Estilos
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#0066ff",
  },
  header: {
    alignItems: "center",
    paddingTop: 50,
    paddingBottom: 20,
  },
  headerText: {
    color: "white",
    fontSize: 20,
    fontWeight: "bold",
  },
});
```

Software Requirements Specification

```
,
catImage: {
  width: 170,
  height: 200,
  marginTop: 0,
},
listContainer: {
  padding: 20,
},
listItem: {
  height: 80,
  backgroundColor: "#0033cc",
  borderRadius: 10,
  marginBottom: 10,
},
contentContainer: {
  flex: 1,
  alignItems: "center",
  justifyContent: "center",
  backgroundColor: "#d6e4ff",
},
scrollView: {
  paddingBottom: 20,
  width: "100%",
},
inspectionBox: {
  backgroundColor: "#fff",
  padding: 15,
  borderRadius: 10,
  marginBottom: 10,
  width: "90%",
  alignItems: "flex-start",
  shadowColor: "#000",
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.3,
  shadowRadius: 4,
  elevation: 3,
},
inspectionText: {
  fontSize: 16,
  color: "#333",
  marginBottom: 5,
},
```

Software Requirements Specification

```
noInspectionsText: {  
  fontSize: 16,  
  color: "#333",  
  textAlign: "center",  
},  
});
```

Software Requirements Specification

The reports section where the reports made by the employees are shown and there is also the section to create a report where the information to make the report will be filled in.

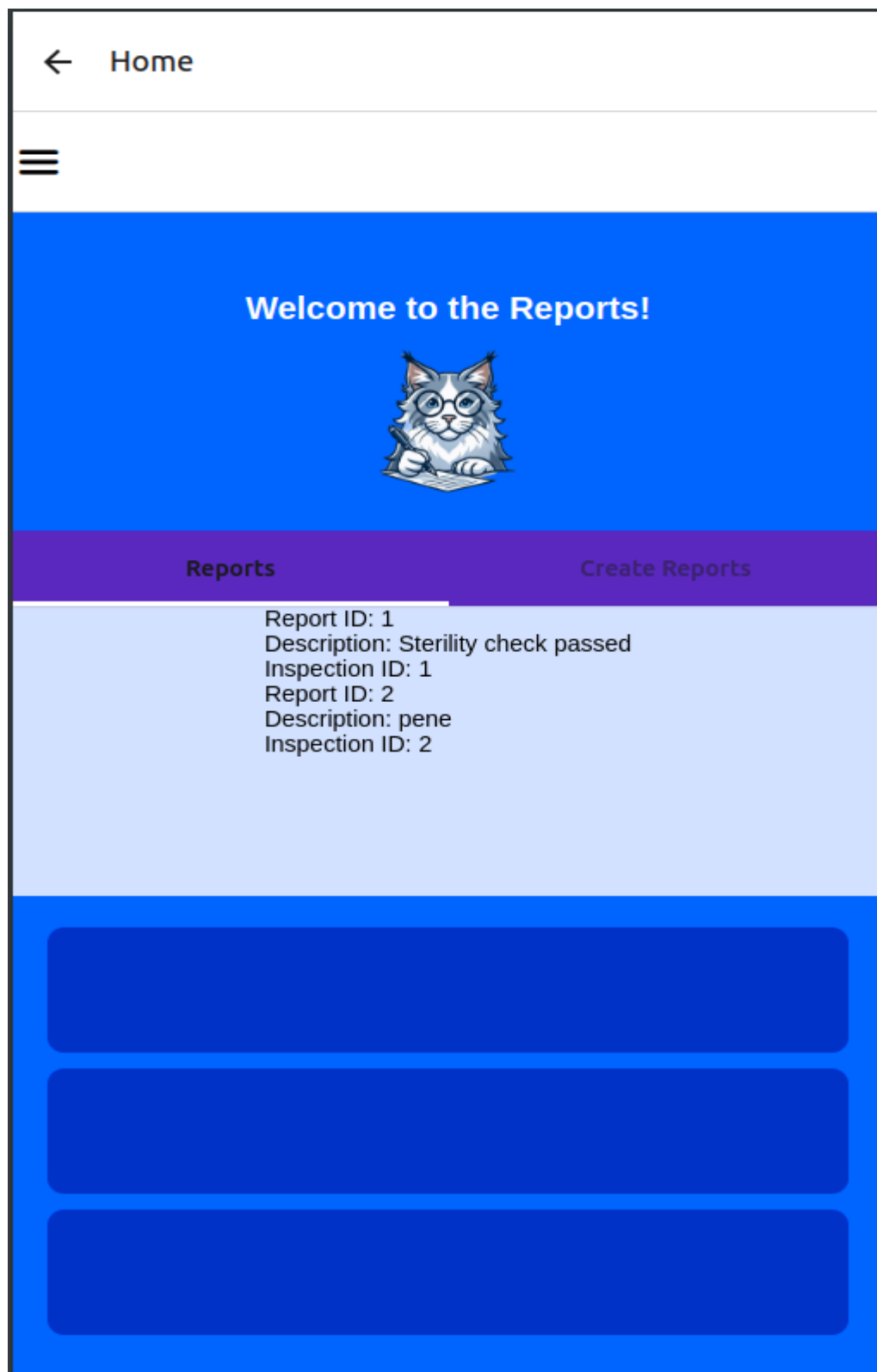



Figure 7. Report screen.

Software Requirements Specification

← Home

≡

Welcome to the Reports!



Reports

Create Reports

Create Report
Description
Inspection ID
Report ID

CREATE REPORT

Figure 8. Report screen.

Software Requirements Specification

CODE:

```
import React, { useEffect, useState } from "react";
import { View, Text, FlatList, ActivityIndicator, StyleSheet, Image,
TouchableOpacity, Button, Alert, TextInput } from "react-native";
import { createMaterialTopTabNavigator } from
"@react-navigation/material-top-tabs";
import { collection, getDocs, addDoc } from "firebase/firestore";

import { db } from "../firebaseConfig";
const ViewReportScreen = () => {
  const [reports, setReports] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchReports = async () => {
      try {
        const querySnapshot = await getDocs(collection(db, "Report"));
        const reportsData = querySnapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        }));
        setReports(reportsData);
      } catch (error) {
        console.error("Error fetching reports:", error);
      } finally {
        setLoading(false);
      }
    };
  });

  fetchReports();
}, []);

if (loading) {
  return (
    <View style={styles.contentContainer}>
      <ActivityIndicator size="large" color="#5c2ac2" />
    </View>
  );
}

return (
  <View style={styles.contentContainer}>
```

Software Requirements Specification

```
<FlatList
  data={reports}
  keyExtractor={({item}) => item.id}
  renderItem={({ item }) => (
    <View style={styles.reportItem}>
      <Text style={styles.title}>Report ID: {item.report_id}</Text>
      <Text>Description: {item.description}</Text>
      <Text>Inspection ID: {item.inspection_id}</Text>
    </View>
  )}
/>
</View>
);
};

const ReportScreen = () => {
  const [description, setDescription] = useState("");
  const [inspectionId, setInspectionId] = useState("");
  const [reportId, setReportId] = useState("");

  const handleCreateReport = async () => {
    if (!description || !inspectionId || !reportId) {
      Alert.alert("Error", "Please fill in all fields.");
      return;
    }

    try {
      // Agregar el reporte a Firestore
      await addDoc(collection(db, "Report"), {
        description,
        inspection_id: inspectionId,
        report_id: reportId,
      });
      Alert.alert("Success", "Report created successfully.");
      // Limpiar campos después de crear el reporte
      setDescription("");
      setInspectionId("");
      setReportId("");
    } catch (error) {
      console.error("Error creating report:", error);
      Alert.alert("Error", "There was an issue creating the report.");
    }
  };
};
```


Software Requirements Specification

```
return (
  <View style={styles.container}>
    <Text style={styles.title}>Create Report</Text>

    <TextInput
      style={styles.input}
      placeholder="Description"
      value={description}
      onChangeText={setDescription}
    />
    <TextInput
      style={styles.input}
      placeholder="Inspection ID"
      value={inspectionId}
      onChangeText={setInspectionId}
      keyboardType="numeric"
    />
    <TextInput
      style={styles.input}
      placeholder="Report ID"
      value={reportId}
      onChangeText={setReportId}
      keyboardType="numeric"
    />

    <Button title="Create Report" onPress={handleCreateReport} />
  </View>
);
};

// 📌 Tabs
const Tab = createMaterialTopTabNavigator();

const ReportTabs = () => {
  return (
    <Tab.Navigator
      screenOptions={{
        tabBarLabelStyle: { fontSize: 14, fontWeight: "bold" },
        tabBarStyle: { backgroundColor: "#5c2ac2" },
        tabBarIndicatorStyle: { backgroundColor: "white", height: 3 },
      }}
    >
```

Software Requirements Specification

```
<Tab.Screen name="Reports" component={ViewReportScreen} />
<Tab.Screen name="Create Reports" component={ReportScreen} />
</Tab.Navigator>
);
};

// 📌 Pantalla principal
export default function App() {
  return (
    <View style={styles.container}>
      {/* Encabezado con imagen */}
      <View style={styles.header}>
        <Text style={styles.headerText}>Welcome to the Reports!</Text>
        <Image source={require('./assets/michireport.png')}
style={styles.catImage} />
      </View>

      {/* Tabs */}
      <ReportTabs />

      {/* Lista de elementos */}
      <View style={styles.listContainer}>
        <TouchableOpacity style={styles.listItem} />
        <TouchableOpacity style={styles.listItem} />
        <TouchableOpacity style={styles.listItem} />
      </View>
    </View>
  );
}

// 📌 Estilos
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#0066ff",
  },
  header: {
    alignItems: "center",
    paddingTop: 50,
    paddingBottom: 20,
  },
  headerText: {
    color: "white",
  },
});
```

Software Requirements Specification

```
    fontSize: 20,
    fontWeight: "bold",
  },
  catImage: {
    width: 100,
    height: 100,
    marginTop: 10,
  },
  listContainer: {
    padding: 20,
  },
  listItem: {
    height: 80,
    backgroundColor: "#0033cc",
    borderRadius: 10,
    marginBottom: 10,
  },
  contentContainer: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center",
    backgroundColor: "#d6e4ff",
  },
  contentText: {
    fontSize: 18,
    fontWeight: "bold",
    color: "#333",
  },
});
```

Software Requirements Specification

In this section, when you select inspect the alert, more information will appear and in turn there will be a button to solve the problem and redirect you to the reports menu

← Inspection

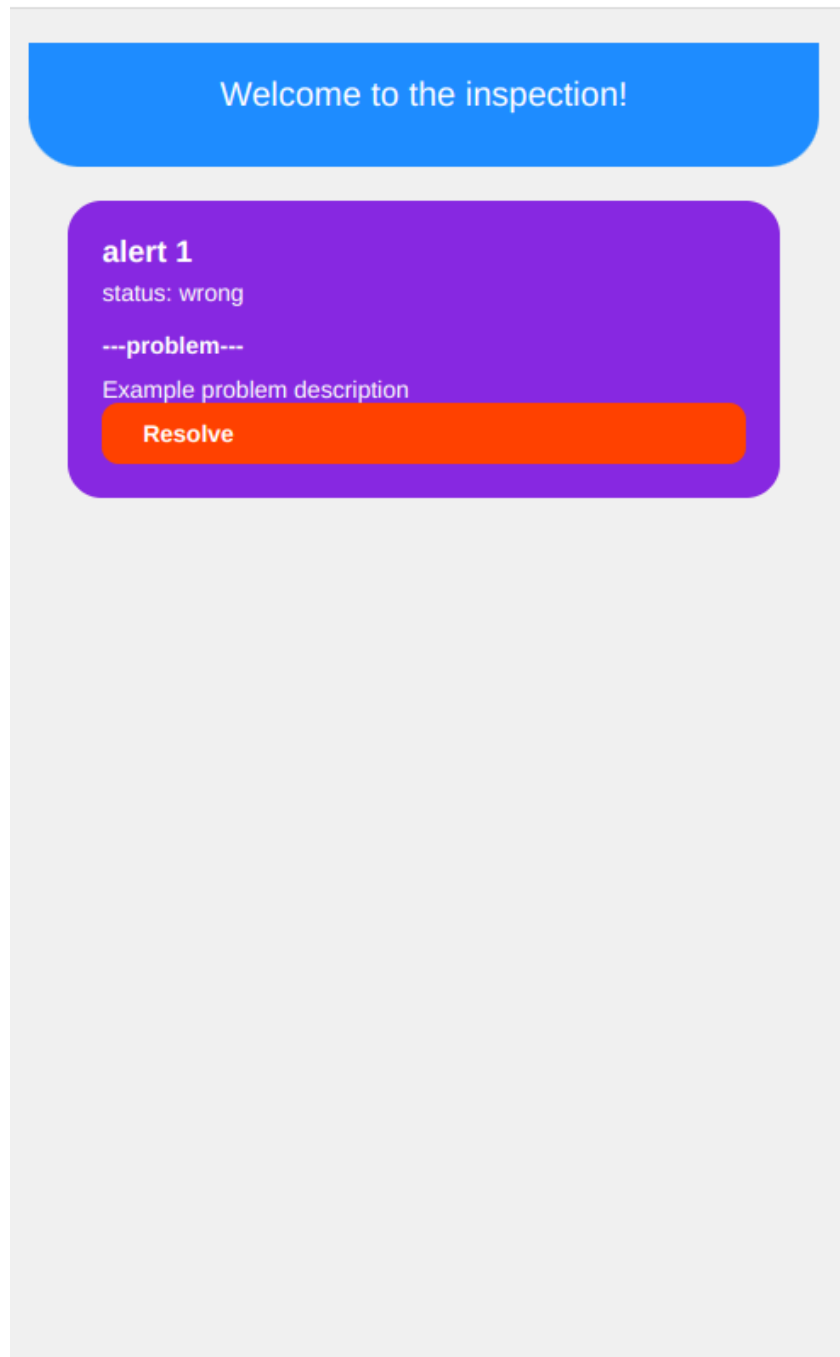


Figure 9. Inspection screen.

Software Requirements Specification

CODE:

```
import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet, TouchableOpacity, Image } from
'react-native';
import { useRoute } from '@react-navigation/native';

export default function Inspection() {
  const route = useRoute();
  const { alertId } = route.params;
  const [alertData, setAlertData] = useState({
    status: 'wrong',
    problem: 'Example problem description'
  });

  useEffect(() => {
    console.log("Inspecting alert with ID:", alertId);
  }, [alertId]);

  return (
    <View style={styles.container}>
      {/* Header */}
      <View style={styles.header}>
        <Text style={styles.headerText}>Welcome to the inspection!</Text>

      </View>

      {/* Alert Card */}
      <View style={styles.alertCard}>
        <Text style={styles.alertTitle}>alert {alertId}</Text>
        <Text style={styles.alertStatus}>status: {alertData.status}</Text>
        <Text style={styles.alertProblem}>---problem---</Text>
        <Text style={styles.alertDescription}>{alertData.problem}</Text>
        <TouchableOpacity style={styles.button}>
          <Text style={styles.buttonText}> Resolve</Text>
        </TouchableOpacity>

      </View>

      {/* Buttons */}
      <View style={styles.buttonContainer}>
      </View>
    </View>
  );
};
```

Software Requirements Specification

}

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#f0f0f0',
    alignItems: 'center',
    padding: 20,
  },
  header: {
    backgroundColor: '#1E90FF',
    width: '100%',
    borderBottomLeftRadius: 30,
    borderBottomRightRadius: 30,
    alignItems: 'center',
    padding: 20,
  },
  headerText: {
    color: 'white',
    fontSize: 20,
    marginBottom: 10,
  },
  catImage: {
    width: 100,
    height: 100,
  },
  alertCard: {
    backgroundColor: '#8A2BE2',
    borderRadius: 20,
    padding: 20,
    marginVertical: 20,
    width: '90%',
  },
  alertTitle: {
    color: 'white',
    fontWeight: 'bold',
    fontSize: 18,
  },
  alertStatus: {
    color: 'white',
    marginVertical: 5,
  },
  alertProblem: {
```

Software Requirements Specification

```
color: 'white',
fontWeight: 'bold',
marginTop: 10,
},
alertDescription: {
  color: 'white',
  marginTop: 10,
},
buttonContainer: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  width: '60%',
},
button: {
  backgroundColor: '#FF4500',
  paddingVertical: 10,
  paddingHorizontal: 20,
  borderRadius: 10,
},
buttonText: {
  color: 'white',
  fontWeight: 'bold',
},
});
```