Software Requirements Specification

for

RawConnect

Version 1.0 approved

Prepared by:
Becerra Sánchez Brian Alejandro
Medina Becerra Alan Oswaldo
Lara Lopez Josue Isaac
Vazquez Granados Eduardo Antonio

19/03/25

Document information

Date	Review	Author	Verified by quality control.
Monday, January 20 2025	**Review 1**	Becerra Sánchez Brian Alejandro Medina Becerra Alan Oswaldo Lara Lopez Josue Isaac Vazquez Granados Eduardo Antonio	Ray Brunett Parra Galaviz

By the client	By the Supplier Company
Ray Brunett Parra Galaviz	Becerra Sánchez Brian Alejandro Medina Becerra Alan Oswaldo Lara Lopez Josue Isaac Vazquez Granados Eduardo Antonio

Table of Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (ERS) document details the software requirements for RawConnect, a marketplace platform designed to connect raw material producers (mining, industrial, etc.) with direct buyers (industries, distributors or manufacturers).

Scope: This ERS covers the complete RawConnect system, including its web and mobile (Android) applications. It describes the functionality required for both producer and buyer users, covering the entire cycle from account creation to product posting, order management, payment processing and logistics coordination. This includes, but is not limited to, user interface, authentication and security, product catalog management, direct communication and negotiation between users, and shipping and order tracking options. This document does not describe any subsystem in isolation, but provides a comprehensive view of the system as a whole.

1.2 Document Conventions

Format and Sources:

- **Main Title:** Times New Roman font, size 18, bold.
- **Subtitles:** Times New Roman font, size 14, bold.
- **Normal Text:** Times New Roman font, size 12.

Highlighted:

- **Bold:** Used to highlight key terms and headings.
- **Italics:** Used for examples and technical terms.
- **Code:** Any code fragment will be displayed in a monospaced font and enclosed in a code block.

Numbering and Lists:

- **Numbered Lists:** Used for sequential steps.
- **Bulleted Lists:** Used to list items in no specific order.

Identification of Requirements:

- Each requirement is identified with a unique code (e.g. REQ-001) for easy tracking and reference.

1.3 Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) document is intended for the following types of readers, each with specific interests and needs with respect to the content of the RawConnect project:

1. **Developers:**
    - **Interests:** Understand the technical and functional requirements for system implementation.
    - **Suggested Reading:** Detailed sections on functional and non-functional requirements, user interfaces, and technical specifications.
2. **Project Managers:**
    - **Interests:** Oversee project scope, deadlines, and team coordination.
    - **Suggested Reading:** Introduction, project objective, scope, and implementation plan.
3. **Marketing Staff:**
    - **Interests: To** understand the key features and value of the product for promotion and marketing strategies.
    - **Suggested Reading:** Executive summary, main features, and benefits of the project.
4. **Users:**
    - **Interests:** Knowing how the system will work and what benefits they will obtain.
    - **Suggested Reading:** Main functionalities section, examples of use, and user flow.
5. **Testers:**
    - **Interests:** Define and execute test cases to ensure product quality.
    - **Suggested Reading:** Functional and non-functional requirements, acceptance criteria, and test plans.
6. **Documentation editors:**
    - **Interests:** Create user manuals and technical documentation.
    - **Suggested Reading:** Detailed descriptions of functionalities, user interfaces, and operating procedures.

1.4 Product Scope

Software Description: RawConnect is a marketplace platform that facilitates the connection between raw material producers (mining, industrial, etc.) and direct buyers (industries, distributors or manufacturers). The application is available on mobile (Android).

Purpose: The purpose of RawConnect is to eliminate intermediaries and optimize the supply chain by enabling producers to sell their products directly to buyers. This not only improves transaction efficiency, but also promotes transparency and sustainability in the commodities market.

Relevant Benefits:

- **For Producers:**
  - Access to a wider and more diversified market.
  - Increased revenues by eliminating intermediaries.
  - Increased visibility and promotional opportunities.
- **For Buyers:**
  - Direct access to quality products with transparent information.
  - Comparison of prices, qualifications and supplier certifications.
  - Cost reduction by negotiating directly with producers.

Objectives and Goals:

- **Supply Chain Optimization:** Facilitating direct access from buyers to producers, reducing costs and delivery times.
- **Transparency in transactions:** Provide detailed information on products and their origins, ensuring quality and reliability.
- **Producer Empowerment:** Enable producers to have full control over the marketing and sale of their products.
- **Sustainability:** Promote fair and sustainable trade practices, encouraging the certification of organic and ethical products.

Relationship with Corporate Objectives:

- **Market Expansion:** RawConnect is aligned with the corporate strategy of expansion in emerging markets and diversification of the service portfolio.
- **Innovation and Technology:** Promote the use of advanced technologies to improve efficiency and transparency in commercial transactions.
- **Social Responsibility:** Supporting sustainable and fair trade practices, contributing to a balanced and ethical economic development.

2. Overall Description

2.1 Product Perspective

Context: RawConnect is developed in response to the need for a platform that facilitates direct connection between raw material producers and industrial buyers. Currently, raw material markets rely heavily on intermediaries, which can increase costs and reduce transparency. RawConnect seeks to address these challenges by providing a digital ecosystem where transactions can be conducted efficiently and directly.

Source: RawConnect is a new, stand-alone product, designed from the ground up to meet the specific needs of raw material producers and buyers. It is not a member of an existing product family, nor a replacement for current systems. It was developed with the objective of integrating advanced order management, trading, payment and logistics functions into a single platform.

Relationship to Wider Systems: Although RawConnect is a stand-alone product, it integrates with various external services and systems to maximize its functionality and reach. Broader system requirements in relation to RawConnect include:

- **Payment Services:** Integration with payment gateway APIs such as Stripe, PayPal, and Mercado Pago to process transactions securely.

2.2 Product Functions

RawConnect shall perform and enable users to perform the following main functions:

- **For Producers:**
  - **Producer Registration and Profile:** Create and manage your profile, including company information, certifications and production capacity.
  - **Publication of Products:** List products with photos, descriptions, prices and terms of sale.
  - **Order Management:** Accept, reject or negotiate orders and track payments and shipments.
- **For Buyers:**
  - **Browse and Search Products:** Search and filter products by raw material type, location, prices, etc.
  - **Supplier Comparison:** Compare prices, delivery times, qualifications and certifications of suppliers.
  - **Order Management:** Create and manage orders, receive notifications and track order status.
  - **Direct Negotiation:** Direct communication with producers to negotiate specific prices and conditions.

- **General Functionalities:**

- **Rating and Review System: To** rate transactions and ensure the quality and reliability of producers and buyers.
- **Payment Gateway:** Process secure payments through cards, bank transfers or platforms such as PayPal.
- **Product Traceability:** Track the origin and certification of the product to ensure transparency in the supply chain.
- **Integrated Logistics:** Coordinate shipments with logistics providers and provide cost and delivery time estimates.
- **Legal Compliance:** Manage electronic invoicing and mandatory certifications according to the type of raw material.

2.3 User Classes and Characteristics

Producers:

- **Frequency of Use:** High, daily use to manage products and orders.
- **Functions Used:** Publication of products and order management.
- **Technical Expertise:** Varies from basic to advanced depending on the size of the producer.
- **Security Level:** High, requires secure access to product management and payments.
- **Characteristics:** Focused on the sale of raw materials, they seek to optimize their scope and efficiency.

Buyers:

- **Frequency of Use:** Moderate to high, frequent use for searching and ordering.
- **Functions Used:** Product search, supplier comparison, order management, direct negotiation.
- **Technical Expertise:** Generally intermediate, comfortable with advanced searches and online order management.
- **Security Level:** High, especially for payment and shipping data management.
- **Characteristics:** Industrial companies or distributors that need quality raw material.

System Administrators:

- **Frequency of Use:** High, constant use to monitor and maintain the system.
- **Functions Used:** User administration, dispute resolution, system maintenance and upgrades.
- **Technical Expertise:** High, with advanced knowledge in systems and security.
- **Security Level:** Very high, access to all data and system functions.
- **Characteristics:** Technical personnel in charge of the proper operation and security of the platform.

2.4 Operating Environment

Hardware platforms:

- Cloud-based servers for backend (e.g. AWS, Google Cloud).
- Mobile devices (smartphones and tablets) and desktop computers for end users.

Operating Systems:

- **Mobile Applications:** Android 8.0 or higher.
- **Web Application:** Compatible with most recent web browsers (Chrome, Firefox, Safari, Edge).

Software Components:

- **Backend:** Node.js or Django for the server.
- **Frontend:** React.js for the web and React Native for the mobile app.
- **Databases:** Firebase or DynamoDB in the cloud.
- **External Services:** Payment gateway APIs, logistics and authentication services.

2.5 Design and Implementation Constraints

Corporate or Regulatory Policies:

- GDPR compliance for personal data management.
- Local and international regulations related to e-commerce and payment security.

Hardware limitations:

- Time and memory requirements to ensure optimal performance on mobile devices and servers.

Interfaces with Other Applications:

- Integration with external payment gateway APIs (Stripe, PayPal).
- Connection with logistics services (DHL, UPS).

Specific Technologies and Tools:

- Use of specific technologies such as React.js and React Native for frontend development.
- Security tools such as two-factor authentication.

2.6 User Documentation

Documentation Components:

Online Help: Help sections accessible from within the application, providing answers to frequently asked questions.

Tutorials: Step-by-step documentation to help users become familiar with the main functions.

2.7 Assumptions and Dependencies

Assumed Factors:

- Use of external services for payments and logistics that must be kept available and up to date.
- Reliance on cloud platforms for data storage and management.
- Need for continuous updating and maintenance to comply with new regulations and technological improvements.

3. External Interface Requirements

3.1 User Interfaces

- **Logical Features:** Intuitive and easy-to-use screens, with clear and accessible navigation.
- **GUI standards:** Follow user interface style guides to maintain consistency.
- **Common Functionalities:** Standard buttons such as "Help", "Back" and "Confirm".

3.2 Hardware Interfaces

Logical and Physical Features: Compatibility with mobile and desktop devices, support for label printers and scanning devices.

3.3 Software Interfaces

- **Connections:** Communication with external databases and services through APIs.
- **Data Elements:** Identification and management of product, user and transaction data.

3.4 Communications Interfaces

- **Requirements:** Support for communication protocols such as HTTPS to secure data transmission.
- **Security:** Implementation of SSL/TLS encryption to protect data in transit.

4. System Features

This section organizes the functional requirements of the product according to the main features offered by the RawConnect system. These features are designed to meet the main needs of the users.

4.1 System Feature 1

Order Management

Description and Priority:

- Allow users to manage orders from creation to delivery. High priority.

Stimulus/Response Sequences:

- The user creates an order -> The system saves the order and notifies the seller.
- The seller confirms the order -> The system updates the order status.
- The order is shipped -> The system notifies the buyer and updates the shipment status.
- Buyer receives the order -> The system updates the status to "Delivered" and allows the buyer to qualify the transaction.

Functional Requirements:

- **REQ-1:** The system must allow buyers to create and manage orders.
- REQ-2: The system must notify vendors of new orders.
- REQ-3: The system must allow vendors to update the status of orders.
- REQ-4: The system must notify buyers of order status updates.
- REQ-5: The system must allow buyers to qualify the transaction once the order has been delivered.

4.2 System Feature 2

Publication of Products

Description and Priority:

- Allow producers to publish and manage their products. High priority.

Stimulus/Response Sequences:

- The producer uploads information of a new product -> The system saves and displays the product in the catalog.
- Producer updates product information -> System updates product details in real time.
- Producer removes a product -> The system removes the product from the catalog and notifies the interested parties.

Functional Requirements:

- REQ-6: The system must allow producers to publish and edit products.
- REQ-7: The system must display products in the catalog in an organized manner.
- REQ-8: The system must allow producers to delete products and update the catalog in real time.

4.3 System Feature 3

Direct Negotiation

Description and Priority:

- Facilitate communication and negotiation between buyers and producers. Medium priority.

Stimulus/Response Sequences:

- The buyer sends a message to the producer -> The system delivers the message to the producer and notifies the buyer of the delivery.
- The producer responds to the message -> The system delivers the response to the buyer and updates the conversation.

Functional Requirements:

- REQ-9: The system must allow direct communication between buyers and producers.
- REQ-10: The system must notify users of new messages.
- REQ-11: The system must store and display the conversation history.

System Feature 4: Rating and Review System

Description and Priority:

- Provide a rating and review system to evaluate the quality and reliability of producers and buyers. High priority.

Stimulus/Response Sequences:

- The buyer completes an order -> The system invites the buyer to qualify the transaction.
- The buyer leaves a review -> The system saves the review and displays it in the producer's profile.

Functional Requirements:

- REQ-12: The system should allow users to rate and leave reviews on transactions.
- REQ-13: The system should display ratings and reviews in user profiles.
- REQ-14: The system must allow moderation of inappropriate or fraudulent reviews.

4.4 System Feature 4

Rating and Review System

Description and Priority:

- Provide a rating and review system to evaluate the quality and reliability of producers and buyers. High priority.

Stimulus/Response Sequences:

- The buyer completes an order -> The system invites the buyer to qualify the transaction.
- The buyer leaves a review -> The system saves the review and displays it in the producer's profile.

Functional Requirements:

- REQ-12: The system should allow users to rate and leave reviews on transactions.
- REQ-13: The system should display ratings and reviews in user profiles.
- REQ-14: The system must allow moderation of inappropriate or fraudulent reviews.

.

4.5 System Feature 5

Product Traceability

Description and Priority:

- Provide traceability and product certification to ensure transparency. High priority.

Stimulus/Response Sequences:

- The producer uploads product traceability information -> The system saves and displays the information associated with the product.
- The buyer consults the traceability of the product -> The system shows the details of the origin and certification of the product.

Functional Requirements:

- REQ-15: The system must allow producers to upload product traceability information.
- REQ-16: The system must display traceability information associated with each product.

4.6 System Feature 6

Product Listing

Description and Priority: Allow producers to publish and manage their products on the platform. High Priority.

Stimulus/Response Sequences:

- The producer enters the product information -> The system saves and publishes the product information.
- The producer edits the product information -> The system updates and displays the edited information.
- Producer deletes a product -> The system deletes the product information from the platform.

Functional Requirements:

- **REQ-5:** The system must allow producers to add new products with name, description, price, quantity available, and images.
- **REQ-6:** The system must allow producers to edit their product information.
- **REQ-7:** The system must allow producers to remove products from the list.
- **REQ-8:** The system must display the products available to buyers.

4.7 System Feature 7

User Registration and Login

Description and Priority: Allow users to register and access the platform. High Priority.

Stimulus/Response Sequences:

- The user enters his registration data -> The system saves the data and creates an account.
- The user enters his login credentials -> The system verifies the credentials and allows access.

Functional Requirements:

- **REQ-1:** The system must allow users to register with their name, email, password, and phone number.
- **REQ-2:** The system must allow users to log in with their email and password.
- **REQ-3:** The system must verify the authenticity of the e-mail during registration.
- **REQ-4:** The system must provide an option to recover the forgotten password.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The platform must handle at least 1000 concurrent users without significant loss of performance.

5.2 Safety Requirements

These requirements address potential loss, damage, or harm that could result from the use of the RawConnect platform. They outline necessary safeguards, actions to be taken, and actions to be prevented. Additionally, they refer to external policies or regulations impacting the product's design and use, as well as any safety certifications required.

Potential Risks:

- **Data Breach:**
  - Possible unauthorized access to sensitive user data, leading to identity theft or financial loss.
- **Fraudulent Transactions:**
  - Risk of buyers or producers engaging in fraudulent activities, causing financial loss or delivery issues.
- **Unsafe Products:**
  - Potential for products sold on the platform to be unsafe or not meeting industry standards, leading to harm.

Safeguards and Actions:

1. **Data Encryption:**
   - **Action:** Implement end-to-end encryption for all user data transmissions.
   - **Purpose:** Protect user data from unauthorized access during transit.
2. **User Authentication:**
   - **Action:** Require multi-factor authentication (MFA) for all user accounts.
   - **Purpose:** Enhance security by verifying user identities during login.
3. **Transaction Monitoring:**
   - **Action:** Implement real-time monitoring and anomaly detection for transactions.
   - **Purpose:** Identify and prevent fraudulent activities promptly.
4. **Product Verification:**
   - **Action:** Require producers to provide certifications and compliance documents for their products.
   - **Purpose:** Ensure that products meet safety and quality standards.
5. **Regular Audits:**
   - **Action:** Conduct regular security audits and vulnerability assessments.
   - **Purpose:** Identify and mitigate potential security risks proactively.
6. **Incident Response Plan:**
   - **Action:** Develop and implement a comprehensive incident response plan.
   - **Purpose:** Ensure swift and effective action in case of security breaches or other incidents.

Preventive Actions:

1. **Prohibit Unsafe Products:**
   - **Action:** Disallow the listing of products that do not meet safety standards or lack proper certifications.
   - **Purpose:** Protect users from purchasing unsafe products.
2. **Restrict Unauthorized Access:**
   - **Action:** Prevent access to sensitive data and system functionalities by unauthorized users.
   - **Purpose:** Minimize the risk of data breaches and misuse of the platform.
3. **Compliance with Regulations:**

- **Action:** Ensure compliance with relevant data protection and e-commerce regulations (e.g., GDPR, CCPA).
- **Purpose:** Adhere to legal requirements and safeguard user rights.

Safety Certifications:

1. **ISO/IEC 27001:**
   - **Requirement:** Achieve certification for information security management.
   - **Purpose:** Demonstrate commitment to maintaining a robust information security management system (ISMS).
2. **PCI DSS:**
   - **Requirement:** Comply with Payment Card Industry Data Security Standard (PCI DSS).
   - **Purpose:** Ensure the secure handling of payment information.
3. **FDA Compliance (if applicable):**
   - **Requirement:** Ensure that any products requiring FDA approval meet all relevant standards.
   - **Purpose:** Guarantee that regulated products are safe for use.
4. **Other Industry-Specific Certifications:**
   - **Requirement:** Obtain and maintain certifications relevant to the specific industry sectors represented on the platform (e.g., USDA Organic, Fair Trade Certification).
   - **Purpose:** Assure buyers of the quality and safety of products.

5.3 Security Requirements

Protection of Personal Data:

- **REQ-S1:** The system must **comply with personal data protection regulations**, such as the **General Data Protection Regulation (GDPR)** and the **Federal Law on Protection of Personal Data Held by Private Parties (LFPDPPP)** in Mexico.
- **REQ-S2:** Implement **encryption of sensitive data** both in transit and at rest, using secure protocols like **SSL/TLS** for transmission and **256-bit AES encryption** for storage.

User Authentication and Authorization:

- **REQ-S3:** Require **user authentication** using secure credentials (email and password).
- **REQ-S4:** Implement **two-factor authentication (2FA)** to enhance security.
- **REQ-S5:** Manage **user privileges according to their role** (Producer, Buyer, Administrator, etc.), ensuring access only to pertinent functions and data.

Password Management:

- **REQ-S6:** Store passwords using **secure hashing algorithms**, such as **bcrypt** or **Argon2**.
- **REQ-S7:** Enforce **strong password policies**, requiring a minimum length and a combination of characters (uppercase, lowercase, numbers, and symbols).

5.4 Software Quality Attributes

- **Adaptability:** Ability to integrate new functions without interrupting the service.
- **Availability:** Maintain 99.9% annual availability.
- **Ease of Use:** Intuitive interface that minimizes the learning curve for new users.

5.5 Business Rules

- Only authenticated users can perform transactions.
- Promotions can only be created by verified producers.

6. Other Requirements

- **REQ-DB1:** Data must be stored in a secure and structured manner, allowing efficient queries and guaranteeing the integrity of the information.

- **REQ-I18N1:** The system must correctly handle international date, time, currency and number formats according to the user's location.

- **REQ-L1:** The system must comply with all **applicable laws and regulations** in the jurisdictions where it operates, including e-commerce regulations, consumer protection and tax laws.

- **REQ-R1:** The code and components developed must be **modular** and **reusable**, facilitating their use in future projects or expansions of the platform.

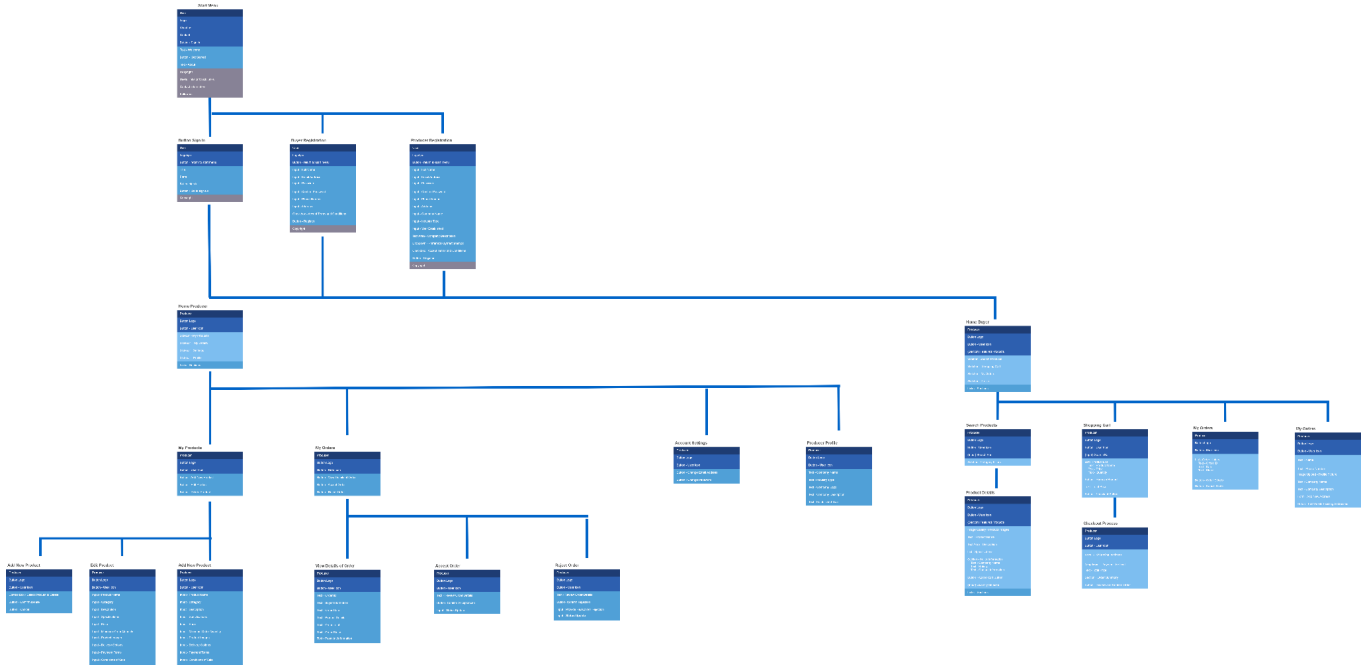
Appendix A: Glossary

1. **API:** Application Programming Interface.
2. **GDPR:** General Data Protection Regulation.
3. **SSL/TLS:** Transport Layer Security Protocols.

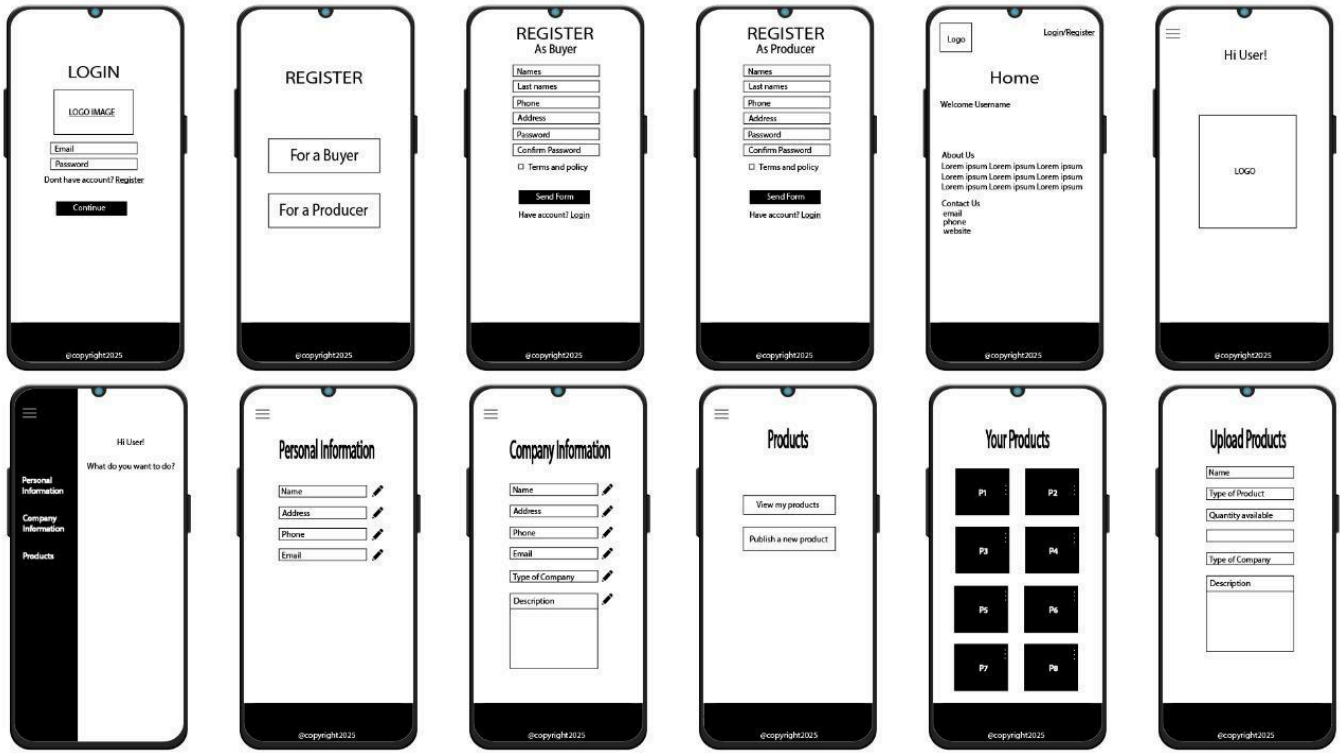
Appendix B: Analysis Models

1.1 Information Architecture Diagram

Description: An Information Architecture Diagram (IAD) is a visual representation of the structure and organization of information within a website, application, or system. It is a crucial component of user experience design, aimed at facilitating the assimilation of information, ensuring easy access to various content blocks, and maintaining consistency and scalability of the content structure.

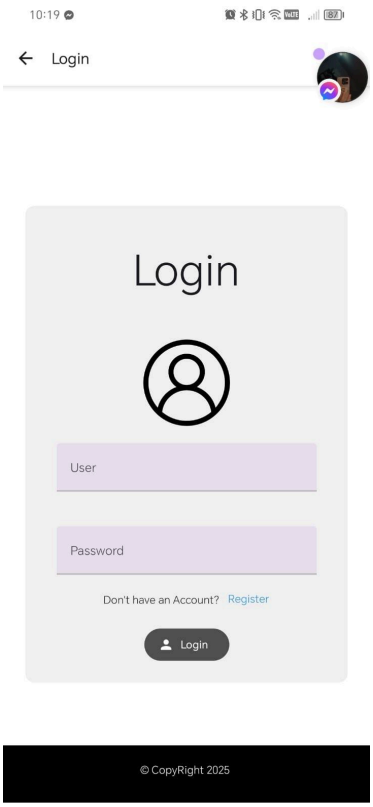


1.2 Mock-ups for the RawConnect Project



1.3 Software Interface functionality

1.3.1 Login interface



Description: This interface allows the user access to his account making a validation if the user is a customer or a producer, this interface ensures that the information of every user is safe, they must enter an email address and a password that previously was registered.

```
import { StatusBar } from 'expo-status-bar';
import { Box, Input, NativeBaseProvider } from 'native-base';
import React, { useState, useEffect } from "react";
import { Image, StyleSheet, TouchableOpacity, View } from 'react-native';
import { Button, Card, Text } from 'react-native-paper';
import Footer from '../components/Footer';
import { auth, db, doc, getDoc, signInWithEmailAndPassword, onAuthStateChanged } from '../config/fb.js';

export default function Login({ navigation }) {
  const [user, setUser] = useState("");
  const [pass, setPass] = useState("");
  const [displayText, setDisplayText] = useState({
    user: "",
    pass: "",
  });
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(true);

  useEffect(() => {

    const unsubscribe = onAuthStateChanged(auth, async (currentUser) => {
      if (currentUser) {

        const docRef = doc(db, "Roles", currentUser.email);
        const docSnap = await getDoc(docRef);

        if (docSnap.exists()) {
          const role = docSnap.data().role;
          if (role === 1) {
            navigation.navigate('MainBuyer');
          } else if (role === 2) {
            navigation.navigate('MainProducer');
          } else {
            setError("Invalid role assigned to this user.");
          }
        } else {
          setError("User not found in roles collection.");
        }
      }
      setLoading(false);
    });


    return () => unsubscribe();
  }, [navigation]);

  const handleLogin = async () => {
    try {
```

```
export default function Login({ navigation }) {
  const handleLogin = async () => {
    try {
      const userCredential = await signInWithEmailAndPassword(auth, user, pass);
      const currentUser = userCredential.user;

      const docRef = doc(db, "Roles", currentUser.email);
      const docSnap = await getDoc(docRef);

      if (docSnap.exists()) {
        const role = docSnap.data().role;

        if (role === 1) {
          navigation.navigate('MainBuyer');
        } else if (role === 2) {
          navigation.navigate('MainProducer');
        } else {
          setError("Invalid role assigned to this user.");
        }
      } else {
        setError("User not found in roles collection.");
      }
    } catch (error) {
      setError("Login failed. Please check your credentials.");
    }
  };

  const handlePredefinedLoginpRroducer = async () => {
    try {

      const userCredential = await signInWithEmailAndPassword(auth, predefinedUserProd.email, predefinedUserPr
      const currentUser = userCredential.user;

      const docRef = doc(db, "Roles", currentUser.email);
      const docSnap = await getDoc(docRef);

      if (docSnap.exists()) {
        const role = docSnap.data().role;

        if (role === 1) {
          navigation.navigate('MainBuyer');
        } else if (role === 2) {
          navigation.navigate('MainProducer');
        } else {
          setError("Invalid role assigned to this user.");
        }
      } else {
        setError("User not found in roles collection.");
```

```
export default function Login({ navigation }) {
  const handlePredefinedLoginpRroducer = async () => {
        setError("User not found in roles collection.");
      }
    } catch (error) {
      setError("Login failed. Please check your credentials.");
    }
  };

  const predefinedUserProd = {
    email: "productor@gmail.com",
    password: "Prod123456&"
  };

  const handlePredefinedLoginpBuyer = async () => {
    try {

      const userCredential = await signInWithEmailAndPassword(auth, predefinedUserBuyer.email, predefinedUserB
      const currentUser = userCredential.user;

      const docRef = doc(db, "Roles", currentUser.email);
      const docSnap = await getDoc(docRef);

      if (docSnap.exists()) {
        const role = docSnap.data().role;


        if (role === 1) {
          navigation.navigate('MainBuyer');
        } else if (role === 2) {
          navigation.navigate('MainProducer');
        } else {
          setError("Invalid role assigned to this user.");
        }
      } else {
        setError("User not found in roles collection.");
      }
    } catch (error) {
      setError("Login failed. Please check your credentials.");
    }
  };

  const predefinedUserBuyer = {
    email: "jorgito@gmail.com",
    password: "Jorgito123456&"
  };
```

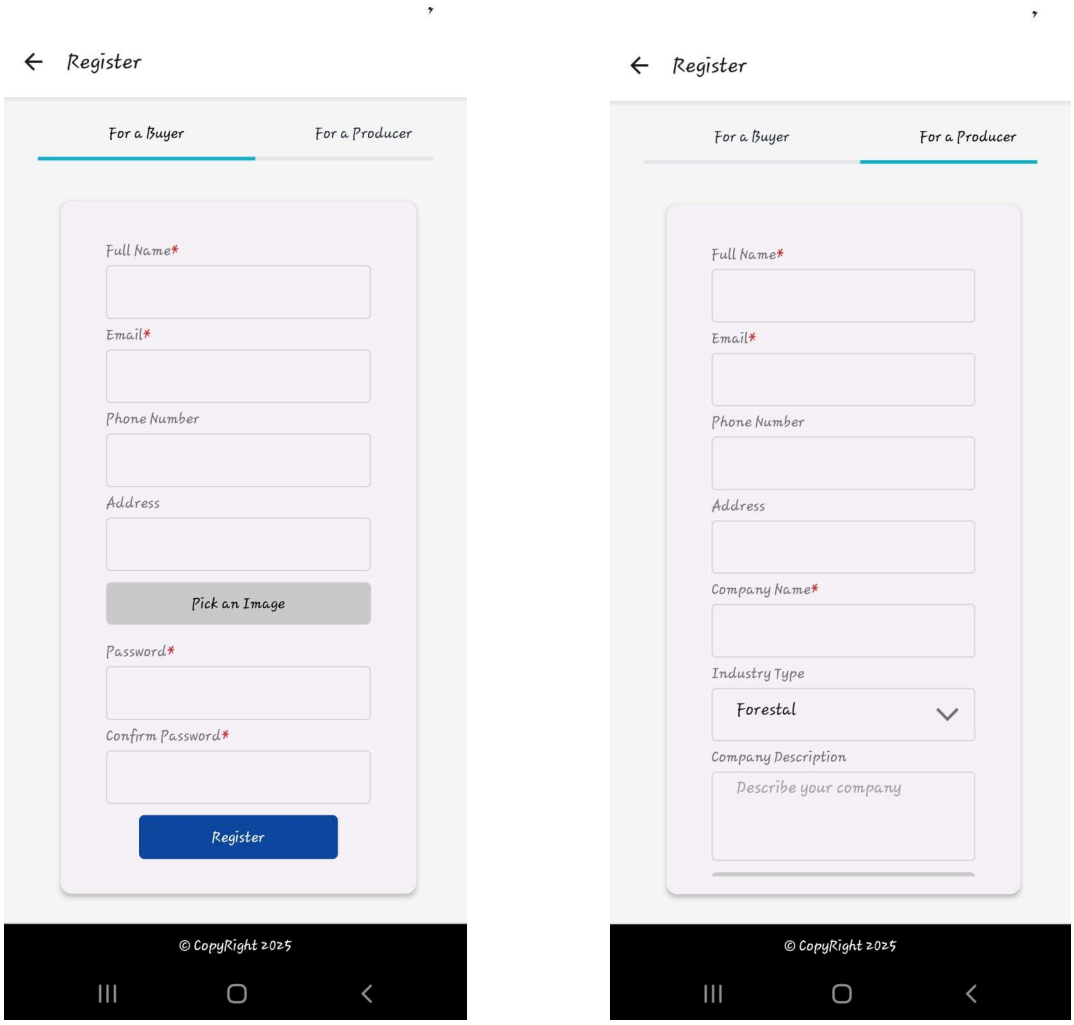
```
export default function Login({ navigation }) {



  if (loading) {
    return <Text>Loading...</Text>;
  }


  return (
    <NativeBaseProvider>
      <View style={styles.container}>
        <Box>
          <Card.Content style={{ boxShadow: '50%', borderRadius: 10, backgroundColor: '#f0f0f0', alignItems: '
            <Text variant="displayLarge" style={{ margin: 40 }}>Login</Text>
            <Card.Content >
              <Image source={require('../assets/user.png')} style={styles.image} />
            </Card.Content>
            <Input
              variant="underlined"
              placeholder="Email"
              value={user}
              onChangeText={setUser}
              style={styles.input}
            />
            <Input
              variant="underlined"
              placeholder="Password"
              value={pass}
              onChangeText={setPass}
              style={styles.input}
              secureTextEntry={true}
            />

            {error ? <Text style={styles.errorText}>{error}</Text> : null}

            <Text style={styles.text}>Don't have an Account?{" "}
              <TouchableOpacity onPress={() => navigation.navigate('Register')}>
                <Text style={styles.link}>Register</Text>
              </TouchableOpacity>
            </Text>

            <Button
              icon="account"
              mode="contained"
              onPress={handleLogin}
              style={{ margin: 25, backgroundColor: '#4f4f4f' }}>
              Login
```

```
export default function Login({ navigation }) {
                style={{ margin: 25, backgroundColor: '#4f4f4f' }}>
                Login
              </Button>

              <Button
                icon="account"
                mode="contained"
                onPress={handlePredefinedLoginpRroducer}
                style={{ margin: 25, backgroundColor: '#4f4f4f' }}>
                Login Producer
              </Button>

              <Button
                icon="account"
                mode="contained"
                onPress={handlePredefinedLoginpBuyer}
                style={{ margin: 25, backgroundColor: '#4f4f4f' }}>
                Login Buyer
              </Button>

            </Card.Content>

            <StatusBar style="auto" />
          </Box>

          <Footer />
        </View>
      </NativeBaseProvider>
    );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  input: {
    margin: 20,
    width: 300,
  },
  image: {
    width: 100,
    height: 100,
    borderRadius: 50,
  },
  linksContainer: {
```

Once the type of user was validated the application redirect the user to the correct screen (MainProducer or MainBuyer)

1.3.2 Register Interface

1.3.3 Form Registration



Description: This interface is a user registration form. It allows users to enter their personal information, such as full name, email address, phone number, address, and password. If the user is a producer, they must also provide their company name, the type of industry they operate in, and a brief company description.

The form includes validations to ensure that all required fields are completed, that the email format is valid, and that passwords match and have a minimum length of six characters.

components > JS formReg.js > Reg

```
1   import { useNavigation } from "@react-navigation/native"
2   import ImageUploader from './ImageUploader';
3   import { createUserWithEmailAndPassword } from "firebase/auth"
4   import { doc, setDoc } from "firebase/firestore"
5   import { Alert, Box, Button, FormControl, Input, Select, Text, TextArea } from "native-base"
6   import { useState } from "react"
7   import { Image, ScrollView, StyleSheet, TouchableOpacity, View } from "react-native"
8   import { auth, db } from "../config/fb.js"
9
10  export default function Reg({ isProducer = false }) {
11    const navigation = useNavigation()
12    const [formData, setFormData] = useState({
13      fullName: "",
14      email: "",
15      phone: "",
16      address: "",
17      password: "",
18      confirmPassword: "",
19      companyName: "",
20      industryType: "Forestal",
21      companyDescription: "",
22    })
23
24    const [image, setImage] = useState(null)
25    const [error, setError] = useState("")
26    const [loading, setLoading] = useState(false)
27    const [successMessage, setSuccessMessage] = useState("")
28
29    const updateFormData = (field, value) => {
30      setFormData({ ...formData, [field]: value })
31    }
32
33    const validateForm = () => {
34      if (!formData.fullName || !formData.email || !formData.password || !formData.confirmPassword) {
```

Additionally, users have the option to upload a profile image using an image uploader component (ImageUploader), the link to which is stored in the database.

When the user submits the form, a Firebase Authentication account is created with the provided email and password. The user's information is then saved to Firestore, including the profile image if uploaded. A role is also assigned in the database, differentiating between buyers and producers.

```
components > JS formReg.js > Reg
export default function Reg({ isProducer = false }) {
  const [error, setError] = useState("")
  const [loading, setLoading] = useState(false)
  const [successMessage, setSuccessMessage] = useState("")

  const updateFormData = (field, value) => {
    setFormData({ ...formData, [field]: value })
  }

  const validateForm = () => {
    if (!formData.fullName || !formData.email || !formData.password || !formData.confirmPassword) {
      setError("Please complete all required fields.")
      return false
    }

    if (formData.password !== formData.confirmPassword) {
      setError("Passwords do not match.")
      return false
    }

    if (formData.password.length < 6) {
      setError("Password must be at least 6 characters.")
      return false
    }

    if (!/\S+@\S+\.\S+/.test(formData.email)) {
      setError("Please enter a valid email address.")
      return false
    }

    if (isProducer && !formData.companyName) {
      setError("Company name is required for producers.")
      return false
    }

    return true
  }

  const handleSubmit = async () => {
    setError("")
    setSuccessMessage("")

    if (!validateForm()) {
      return
    }

    setLoading(true)

    let imageUrl = null
    if (image) {
      imageUrl = await uploadImageToCloudinary(image)
      if (!imageUrl) {
        setError("Image upload failed.")
        setLoading(false)
        return
      }
    }

    try {
      const userCredential = await createUserWithEmailAndPassword(auth, formData.email, formData.password)

      const user = userCredential.user
```

```
components > JS formReg.js > Reg
export default function Reg({ isProducer = false }) {
  const handleSubmit = async () => {
      const user = userCredential.user

      const userDocRef = doc(db, "users", user.email);
      const profileImage = formData.profileImage || null;
      await setDoc(userDocRef, {
        fullName: formData.fullName,
        email: formData.email,
        phone: formData.phone || "",
        address: formData.address || "",
        companyName: isProducer ? formData.companyName : "",
        industryType: isProducer ? formData.industryType : "",
        companyDescription: isProducer ? formData.companyDescription : "",
        profileImage: profileImage,
        createdAt: new Date(),
      })


      const roleDocRef = doc(db, "Roles", user.email)
      await setDoc(roleDocRef, {
        role: isProducer ? 2 : 1,
      })

      setSuccessMessage("User successfully registered")

      setTimeout(() => {
        navigation.navigate("Login")
      }, 2000)
    } catch (error) {
      console.error("Error during registration:", error)
      setError("Error registering the user: " + error.message)
    } finally {
      setLoading(false)
    }
  }

  return (
    <ScrollView>
      <View style={styles.container}>
        <Box width="100%" maxWidth="400px">
          <FormControl isRequired>
            <FormControl.Label>Full Name</FormControl.Label>
            <Input
              value={formData.fullName}
              onChangeText={(text) => updateFormData("fullName", text)}
              style={styles.input}
            />
          </FormControl>

          <FormControl isRequired>
            <FormControl.Label>Email</FormControl.Label>
            <Input
              value={formData.email}
              onChangeText={(text) => updateFormData("email", text)}
              style={styles.input}
              keyboardType="email-address"
              autoCapitalize="none"
            />
          </FormControl>

          <FormControl>
```

If successful, a message confirming registration is displayed, and after a few seconds, the application automatically redirects to the login screen. If an error occurs during the process, an alert message informs you of the problem.

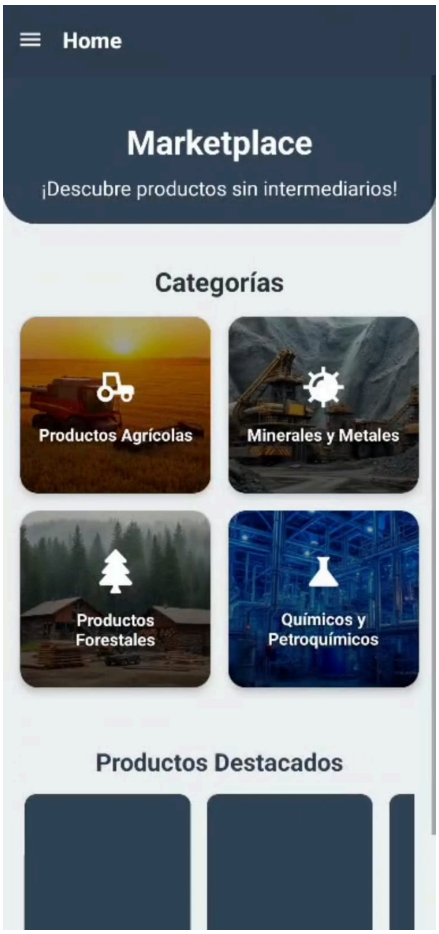
1.3.4 Image Uploader

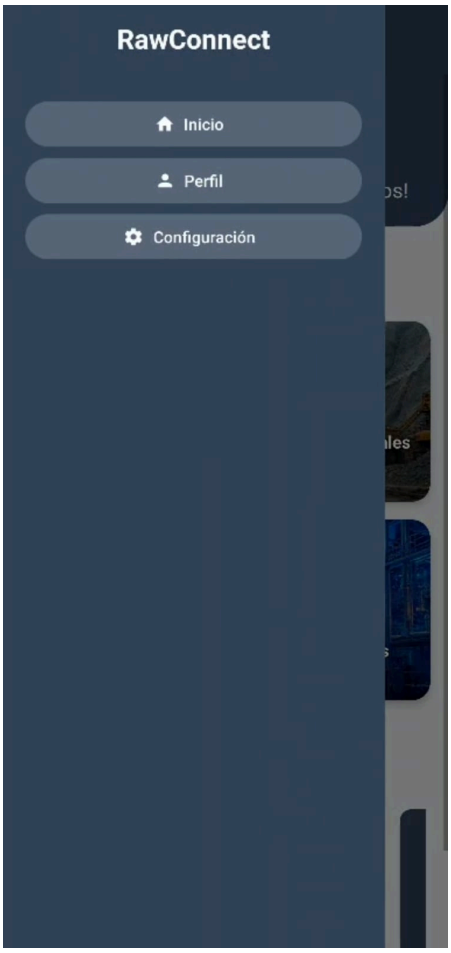
This code implements a functional component in React Native called ImageUploader, which allows users to select an image from their gallery and upload it to Cloudinary. It does this by using useState to manage the selected image's state and upload state, and expo-image-picker to open the device's image gallery.

When the user taps the Pick an Image button, the pickImage function is executed, which opens the image library with certain settings, such as allowing only images, enabling editing, and setting a square (4:4) aspect ratio. If the user selects an image and does not cancel the operation, its URI is saved in the image state, and the uploadImageToCloudinary function is then called to upload the image to Cloudinary.

```
import { useState } from 'react';
import * as ImagePicker from 'expo-image-picker';
import { Image, TouchableOpacity, Text, StyleSheet, View } from 'react-native';

const ImageUploader = ({ uploadPreset, onUploadComplete }) => {
  const [image, setImage] = useState(null);
  const [loading, setLoading] = useState(false);

  const pickImage = async () => {
    const result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [4, 4],
      quality: 1,
    });

    if (!result.canceled) {
      setImage(result.assets[0].uri);
      await uploadImageToCloudinary(result.assets[0].uri);
    }
  };

  const uploadImageToCloudinary = async (imageUri) => {
    setLoading(true);
    const formData = new FormData();
    formData.append('file', {
      uri: imageUri,
      type: 'image/jpeg',
      name: 'profile.jpg',
    });
    formData.append('upload_preset', uploadPreset);
    formData.append('cloud_name', 'df5qzxunp');

    try {
      const response = await fetch('https://api.cloudinary.com/v1_1/df5qzxunp/image/upload', {
        method: 'POST',
        body: formData,
      });
      const data = await response.json();
      setLoading(false);

      if (data.secure_url) {
        onUploadComplete(data.secure_url);
      } else {
        console.error('Failed to upload image.');
      }
    } catch (error) {
      console.error('Error uploading image:', error);
      setLoading(false);
    }
  };

  return (
    <View>
      <TouchableOpacity onPress={pickImage} style={styles.imagePicker}>
        <Text>{loading ? 'Uploading...' : 'Pick an Image'}</Text>
      </TouchableOpacity>
      {image && <Image source={{ uri: image }} style={styles.image} />}
    </View>
  );
};
```

1.3.5 Main Buyer





The interface starts by importing libraries such as react-navigation for navigation, react-native for the user interface, expo-linear-gradient for visual effects and @expo/vector-icons for icons. We also import several custom components such as Agricultural, Chemicals, Forestry, Minerals, ProductDetails and ProfileScreen, which represent different sections of the application and the categories of the products offered by the system.

```
import { FontAwesome5, MaterialCommunityIcons, MaterialIcons } from "@expo/vector-icons";
import { createDrawerNavigator } from "@react-navigation/drawer";
import { LinearGradient } from "expo-linear-gradient";
import { Alert, Dimensions, Image, ScrollView, StyleSheet, TouchableOpacity, View } from "react-native";
import { Button, Text } from "react-native-paper";
import { auth, db, doc, getDoc, setDoc, updateDoc, signOut } from "../config/fb.js";

import Agricultural from "./Components/Agricultural";
import Chemicals from "./Components/Chemicals";
import Forestry from "./Components/Forestry";
import Minerals from "./Components/Minerals";
import ProductDetails from "./Components/ProductDetails";

import ProfileScreen from "./Components/ProfileScreen";
```

Next, the device window width is obtained using Dimensions.get("window") and a DrawerNavigator is created, which will be used for side navigation. A GradientBackground component is defined, which generates a background with a color gradient and is reused in different parts of the interface. CategoryCard is also implemented, a component that represents interactive cards with a background image, an icon, and a title, allowing navigation to different product categories within the app.

```
const { width } = Dimensions.get("window");
const Drawer = createDrawerNavigator();

const GradientBackground = ({ colors, style, children }) => (
  <View style={[styles.gradientContainer, style]}>
    {colors.map((color, index) => (
      <View
        key={index}
        style={[
          StyleSheet.absoluteFill,
          {
            backgroundColor: color,
            opacity: 1 - index / colors.length,
          },
        ]}
      />
    ))}
    {children}
  </View>
);

const CategoryCard = ({ title, icon, imagePrompt, onPress }) => (
  <TouchableOpacity style={styles.categoryCard} onPress={onPress}>
    <Image
      source={{ uri: `https://api.a0.dev/assets/image?text=${encodeURIComponent(imagePrompt)}&aspect=16:9` }}
      style={styles.categoryBackground}
    />
    <LinearGradient colors={["rgba(0,0,0,0.3)", "rgba(0,0,0,0.7)"]} style={styles.categoryGradient}>
      <View style={styles.categoryContent}>
        {icon}
        <Text style={styles.categoryTitle}>{title}</Text>
      </View>
    </LinearGradient>
  </TouchableOpacity>
);
```

The HomeScreen main screen features a structured layout with a header displaying the Marketplace title and a subheader. A category section then opens, with each category represented by a CategoryCard, allowing the user to navigate to sections such as "Agricultural Products," "Minerals and Metals," "Forest Products," and "Chemicals and Petrochemicals." Additionally, the screen includes a "Featured Products" section displayed in a horizontal scrollview.

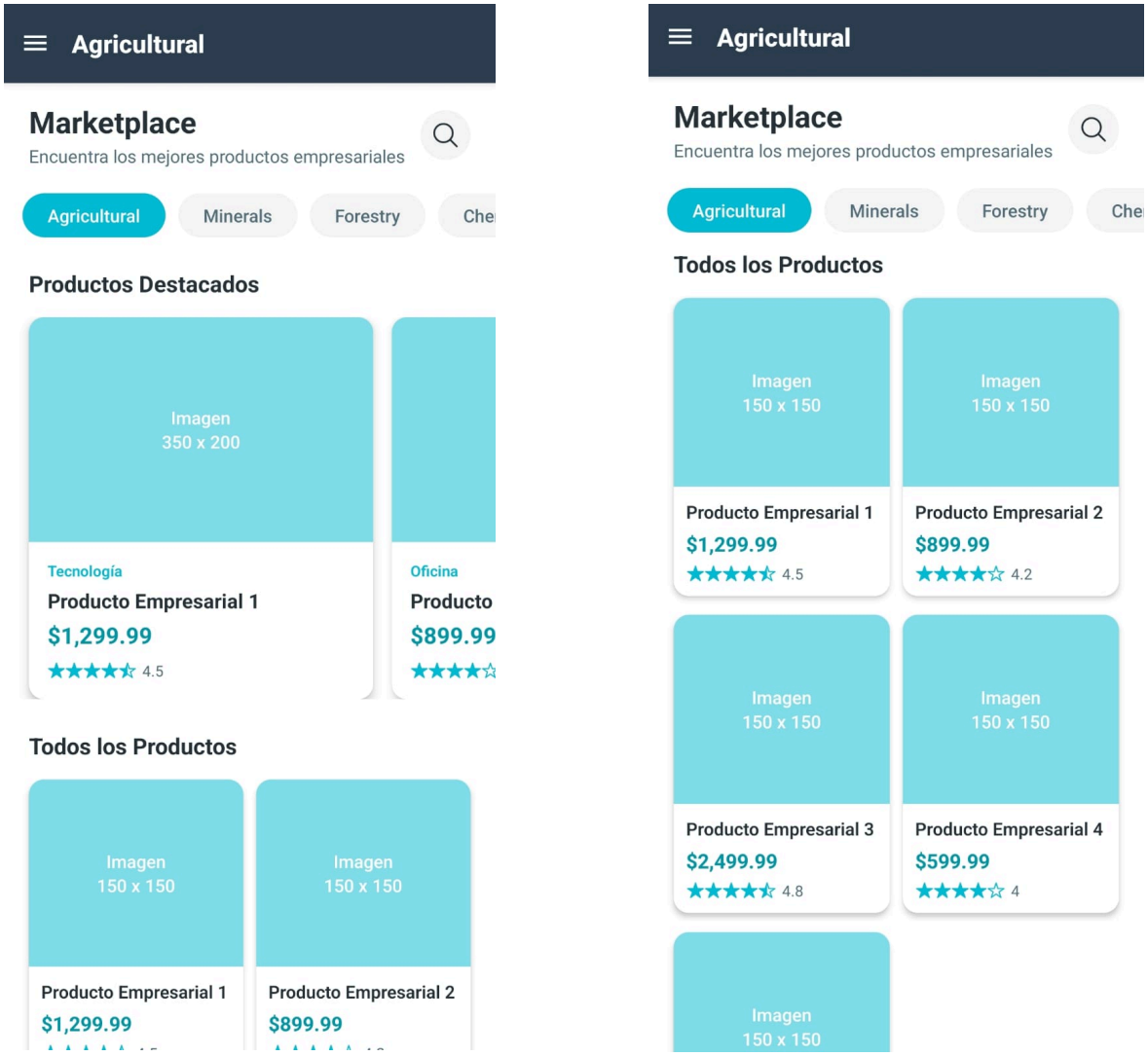
```
const HomeScreen = ({ navigation }) => {
  return (
    <ScrollView style={styles.container}>
      <GradientBackground colors={["#2c3e50", "#34495e"]} style={styles.header}>
        <Text style={styles.headerText}>Marketplace</Text>
        <Text style={styles.subHeaderText}>Descubre productos sin intermediarios!</Text>
      </GradientBackground>

      <View style={styles.categoriesContainer}>
        <Text style={styles.sectionTitle}>Categorías</Text>
        <View style={styles.categoriesGrid}>
          {/* 🔗 Ahora el botón de Productos Agrícolas navega a la pantalla "Categorys" */}
          <CategoryCard
            title="Productos Agrícolas"
            icon={<MaterialCommunityIcons name="tractor" size={40} color="white" />}
            onPress={() => navigation.navigate("Agricultural")}
            imagePrompt="modern agricultural machinery in a vast golden wheat field at sunset, dramatic lighting"
          />
          <CategoryCard
            title="Minerales y Metales"
            icon={<MaterialCommunityIcons name="mine" size={40} color="white" />}
            onPress={() => navigation.navigate("Minerals")}
            imagePrompt="industrial mining operation with massive machinery and raw minerals, dramatic industrial scene"
          />
          <CategoryCard
            title="Productos Forestales"
            icon={<FontAwesome5 name="tree" size={40} color="white" />}
            onPress={() => navigation.navigate("Forestry")}
            imagePrompt="sustainable forestry operation with lumber mill and forest management, morning mist"
          />
          <CategoryCard
            title="Químicos y Petroquímicos"
            icon={<MaterialIcons name="science" size={40} color="white" />}
            onPress={() => navigation.navigate("Chemicals")}
            imagePrompt="modern chemical plant with sophisticated equipment and blue lighting, industrial scene"
          />
        </View>
      </View>

      <View style={styles.featuredSection}>
        <Text style={styles.featuredTitle}>Productos Destacados</Text>
        <ScrollView horizontal showsHorizontalScrollIndicator={false} style={styles.featuredScroll}>
          {[1, 2, 3, 4, 5].map((item) => (
            <GradientBackground key={item} colors={["#34495e", "#2c3e50"]} style={styles.featuredItem}>
              <Text style={styles.featuredItemText}>Producto {item}</Text>
            </GradientBackground>
          ))}
        </ScrollView>
      </View>
    </ScrollView>
  );
};
```

The DrawerContent component defines the content of the Drawer navigation menu, which presents options such as "Home," "Profile," and "Settings," each with a styled button that allows navigation between screens. Finally, the MainBuyer component configures the DrawerNavigator, establishing the different screens available in the side navigation, including the HomeScreen, ProfileScreen, and product categories. Layout styles are also defined using StyleSheet.create(), ensuring a consistent and modern look throughout the application.

```
const DrawerContent = (props) => (
  <GradientBackground colors={["#2c3e50", "#34495e"]} style={styles.drawerContent}>
    <View style={styles.drawerHeader}>
      <Text style={styles.drawerTitle}>RawConnect</Text>
    </View>
    <View style={styles.drawerItems}>
      <Button
        icon="home"
        mode="contained"
        onPress={() => props.navigation.navigate("Home")}
        style={styles.drawerButton}
      >
        Inicio
      </Button>
      <Button
        icon="account"
        mode="contained"
        onPress={() => props.navigation.navigate("Profile")}
        style={styles.drawerButton}
      >
        Perfil
      </Button>
      <Button icon="cog" mode="contained" onPress={() => alert("Configuración")} style={styles.drawerButton}>
        Configuración
      </Button>
      <Button
  icon="logout"
  mode="contained"
  onPress={() => handleSignOut(props.navigation)}
  style={styles.signOutButton}
>
  Sign Out
</Button>

    </View>
  </GradientBackground>
);
```

```
const MainBuyer = () => {
  return (
    <Drawer.Navigator
      drawerContent={(props) => <DrawerContent {...props} />}
      screenOptions={{
        headerStyle: {
          backgroundColor: "#2c3e50",
        },
        headerTintColor: "#fff",
        headerTitleStyle: {
          fontWeight: "bold",
        },
      }}
    >
      <Drawer.Screen name="Home" component={HomeScreen} />
      <Drawer.Screen name="Profile" component={ProfileScreen} options={{ title: "Perfil" }} />
      <Drawer.Screen name="Agricultural" component={Agricultural} options={{ title: "Agricultural" }} />
      <Drawer.Screen name="Chemicals" component={Chemicals} options={{ title: "Chemicals" }} />
      <Drawer.Screen name="Forestry" component={Forestry} options={{ title: "Forestry" }} />
      <Drawer.Screen name="Minerals" component={Minerals} options={{ title: "Minerals" }} />
      <Drawer.Screen name="ProductDetails" component={ProductDetails} options={{ title: "ProductDetails" }} />

    </Drawer.Navigator>
  );
};
```

1.3.6 Agricultural



This code implements the DetailsBuyer screen interface, which is triggered when the user selects a category in the Marketplace app.

First, the screen dimensions are defined with Dimensions.get("window") to make the interface responsive, and a color palette of turquoise, white, and gray is set. Next, the products state is declared, which contains a list of business products with information such as name, price, rating, description, and category. A selectedCategory state is also defined to manage the category selected by the user.

```
const { width } = Dimensions.get("window")

// Paleta de colores
const COLORS = {
    primary: "#00BCD4",
    secondary: "#80DEEA",
    accent: "#0097A7",
    white: "#FFFFFF",
    lightGray: "#F5F5F5",
    gray: "#9E9E9E",
    text: "#263238",
    textLight: "#546E7A",
}
```

```
export default function DetailsBuyer({ navigation }) {
    const [products, setProducts] = useState([
        {
            id: "1",
            name: "Producto Empresarial 1",
            price: "$1,299.99",
            rating: 4.5,
            description: "Descripción breve del producto",
            category: "Tecnología",
        },
        {
            id: "2",
            name: "Producto Empresarial 2",
            price: "$899.99",
            rating: 4.2,
            description: "Descripción breve del producto",
            category: "Oficina",
        },
        {
            id: "3",
            name: "Producto Empresarial 3",
            price: "$2,499.99",
            rating: 4.8,
            description: "Descripción breve del producto",
            category: "Tecnología",
        },
```

The product list is organized into two sections: "Featured Products" and "All Products." The first section displays up to three products in a horizontal scroll view, where each product is represented by a card containing its name, price, category, and star rating. The second section, "All Products," displays a grid of all available products, using more compact cards but with the same visual structure.

```
return (
    <SafeAreaView style={styles.container}>
        <StatusBar backgroundColor={COLORS.white} barStyle="dark-content" />

        {/* Header */}
        <View style={styles.header}>
            <View>
                <Text style={styles.headerTitle}>Marketplace</Text>
                <Text style={styles.headerSubtitle}>Encuentra los mejores productos empresariales</Text>
            </View>
            <TouchableOpacity style={styles.searchButton}>
                <Ionicons name="search-outline" size={24} color={COLORS.text} />
            </TouchableOpacity>
        </View>

        {/* Categorías */}
        <View style={styles.categoriesContainer}>
            <ScrollView
                horizontal
                showsHorizontalScrollIndicator={false}
                contentContainerStyle={styles.categoriesScrollView}
            >
                {categories.map((category, index) => (
                    <TouchableOpacity
                        key={index}
                        style={[styles.categoryButton, selectedCategory === category && styles.categoryButtonActive]}
                        onPress={() => setSelectedCategory(category)}
                    >
                        <Text
                            style={[styles.categoryButtonText, selectedCategory === category && styles.categoryButtonTextActive]}
                            >
                            {category}
                        </Text>
                    </TouchableOpacity>
                ))}
            </ScrollView>
        </View>
```

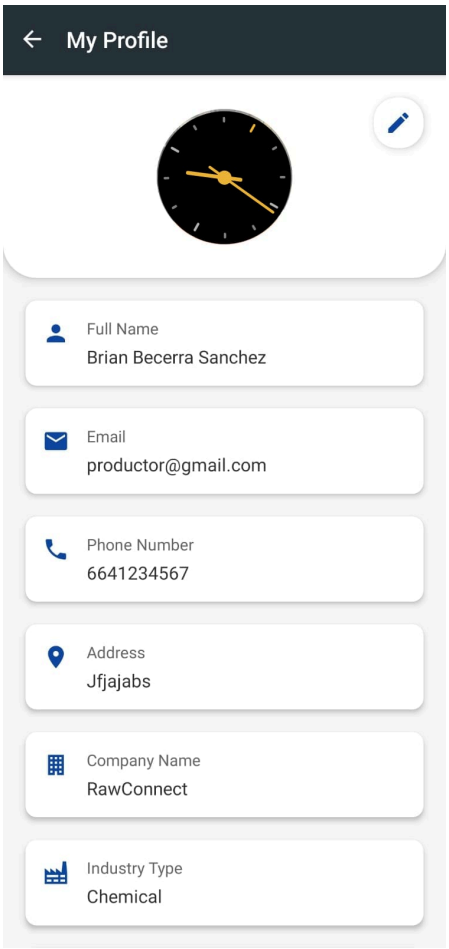
```
{/* Lista de Productos */}
<ScrollView showsVerticalScrollIndicator={false} contentContainerStyle={styles.productsContainer}>
    {/* Productos Destacados */}
    <View style={styles.featuredContainer}>
        <Text style={styles.sectionTitle}>Productos Destacados</Text>
        <ScrollView
            horizontal
            showsHorizontalScrollIndicator={false}
            contentContainerStyle={styles.featuredScrollView}
        >
            {products.slice(0, 3).map((product) => (
                <TouchableOpacity
                    key={product.id}
                    style={styles.featuredProductCard}
                    onPress={() => navigateToProductDetails(product)}
                >
                    {/* AQUÍ IMPLEMENTAR IMAGEN DEL PRODUCTO */}
                    <View style={styles.
                        <Text style={sty
                        <Text style={sty
                    </View>
                    <View style={styles.featuredProductInfo}>
                        <Text style={styles.productCategory}>{product.category}</Text>
                        <Text style={styles.featuredProductName} numberOfLines={1}>
                            {product.name}
                        </Text>
                        <Text style={styles.featuredProductPrice}>{product.price}</Text>
                        {renderStars(product.rating)}
                    </View>
                </TouchableOpacity>
            ))}
        </ScrollView>
    </View>
```

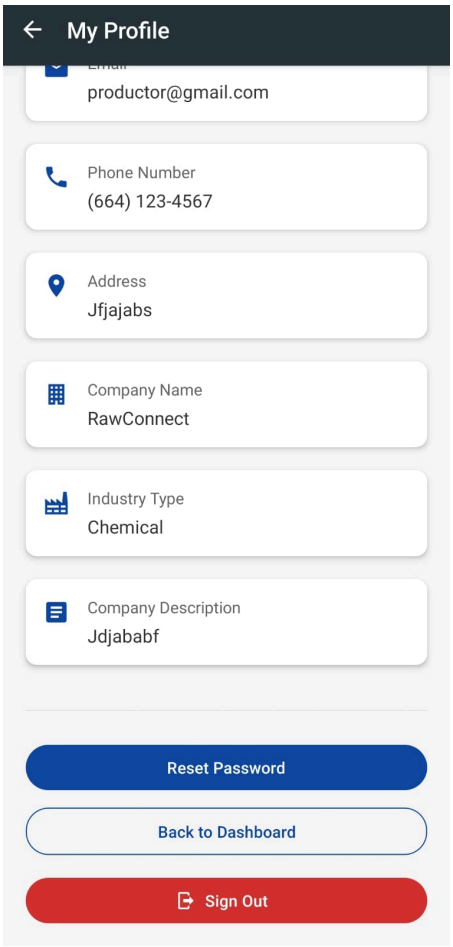
```
(property) featuredProductInfo: {
    padding: number;
}
```

Additionally, the renderStars function is implemented, which dynamically generates a series of stars based on the product's rating, differentiating between full, half, and empty stars. The navigateToProductDetails function is also defined, which allows navigation to the ProductDetails screen, sending the complete data for the selected product.

```
const renderStars = (rating) => {
    const stars = []
    const fullStars = Math.floor(rating)
    const halfStar = rating - fullStars >= 0.5

    for (let i = 0; i < 5; i++) {
        if (i < fullStars) {
            stars.push(<Ionicons key={i} name="star" size={14} color={COLORS.primary} />)
        } else if (i === fullStars && halfStar) {
            stars.push(<Ionicons key={i} name="star-half" size={14} color={COLORS.primary} />)
        } else {
            stars.push(<Ionicons key={i} name="star-outline" size={14} color={COLORS.primary} />)
        }
    }

    return (
        <View style={styles.ratingContainer}>
            {stars}
            <Text style={styles.ratingText}>{rating}</Text>
        </View>
    )
}
```

1.3.7 Profile-screen





This code defines the ProfileScreen profile screen. It is designed to allow users to view and update their personal and company information, as well as allow them to log out or reset their password if necessary.

To handle user information, the code first checks if the data has been passed as parameters from another screen, such as MainProducer. If so, it uses it directly and disables the loading state. Otherwise, it attempts to retrieve the data from Firestore using the authenticated user's email. If the user's document doesn't exist in the database, default values are created, and a new document with this data is saved in Firestore.

```
return (
  <ScrollView style={styles.container}>
    <View style={styles.header}>
      {/* Aquí cambiamos a Avatar.Image si profileImage está disponible */}
      {userData?.profileImage ? (
        <Avatar.Image
          size={120}
          source={{ uri: userData.profileImage }}
        />
      ) : (
        <Avatar.Text
          size={120}
          label={userData?.fullName?.charAt(0) || userData?.email?.charAt(0) || "P"}
          backgroundColor="#0D47A1"
        />
      )}
      <TouchableOpacity style={styles.editButton} onPress={handleEdit}>
        <Icon name={isEditing ? "check" : "pencil"} size={24} color={theme.colors.primary} />
      </TouchableOpacity>
    </View>
    <View style={styles.content}>
      {renderField("account", "Full Name", userData.fullName, "fullName")}
      {renderField("email", "Email", userData.email, "email")}
      {renderField("phone", "Phone Number", userData.phone, "phone")}
      {renderField("map-marker", "Address", userData.address, "address")}
      {renderField("office-building", "Company Name", userData.companyName, "companyName")}
      {renderField("factory", "Industry Type", userData.industryType, "industryType")}
      {renderField("text-box", "Company Description", userData.companyDescription, "companyDescription", true)}

      <Divider style={styles.divider} />
```

The user can edit their information by tapping a button that toggles the editing state. When editing is enabled, text fields become editable using TextInput, allowing users to modify values such as name, phone number, address, and company details. Tapping the confirmation button saves the information to Firestore using updateDoc, ensuring the data remains up-to-date.

1.3.8 Product Details

Description: The interface displays detailed information about a selected product, including its name, category, price, star rating, description, features, specifications, stock availability, and seller. The screen allows users to adjust the desired quantity of the product using increment and decrement buttons and includes an option to add it to the shopping cart. It also includes a button to return to the previous screen and another to mark the product as a favorite.

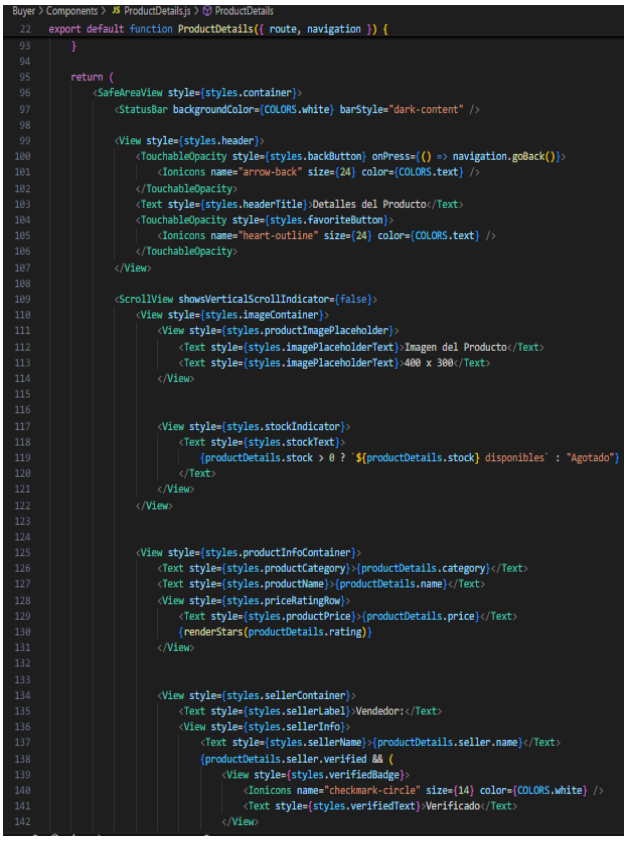
Buyer > Components > JS ProductDetails.js > ProductDetails

```
"use client"

import { Ionicons } from "@expo/vector-icons"
import { useState } from "react"
import { Dimensions, SafeAreaView, ScrollView, StatusBar, StyleSheet, Text, TouchableOpacity, View } from "react-native"


const { width } = Dimensions.get("window")


const COLORS = {
    primary: "#00BCD4",
    secondary: "#80DEEA",
    accent: "#0097A7",
    white: "#FFFFFF",
    lightGray: "#F5F5F5",
    gray: "#9E9E9E",
    text: "#263238",
    textLight: "#546E7A",
}

export default function ProductDetails({ route, navigation }) {

    const { product } = route.params


    const [productDetails, setProductDetails] = useState({
        ...product,
        description:
            "Este es un producto empresarial de alta calidad diseñado para satisfacer las necesidades de su empresa. Fabr
        features: [
            "Característica 1: Alta durabilidad",
            "Característica 2: Eficiencia energética",
            "Característica 3: Fácil integración",
            "Característica 4: Soporte técnico incluido",
            "Característica 5: Garantía extendida",
        ],
        specifications: {
            Dimensiones: "30 x 20 x 10 cm",
            Peso: "1.5 kg",
            Material: "Aluminio y polímeros de alta resistencia",
            Garantía: "2 años",
            Origen: "Importado",
        },
        stock: 15,
        seller: {
            name: "Empresa Proveedora S.A.",
            rating: 4.7,
            verified: true,
        },
    })
```

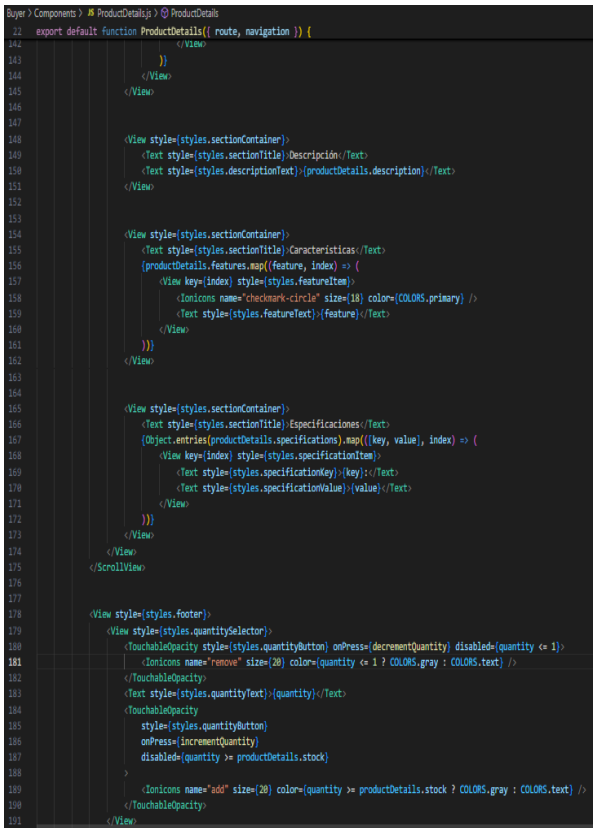
UseState is used to manage the product's state and its selected quantity. Additionally, the interface is structured with a ScrollView to make it scrollable and a SafeAreaView to ensure a suitable layout on different devices. The design is styled with StyleSheet, and Ionicons icons are used to enhance the visual experience.

Buyer > Components > JS ProductDetails.js > ProductDetails

```
22  export default function ProductDetails({ route, navigation }) {
44          },
45          stock: 15,
46          seller: {
47              name: "Empresa Proveedora S.A.",
48              rating: 4.7,
49              verified: true,
50          },
51      })
52
53
54      const [quantity, setQuantity] = useState(1)
55
56      const renderStars = (rating) => {
57          const stars = []
58          const fullStars = Math.floor(rating)
59          const halfStar = rating - fullStars >= 0.5
60
61          for (let i = 0; i < 5; i++) {
62              if (i < fullStars) {
63                  stars.push(<Ionicons key={i} name="star" size={16} color={COLORS.primary} />)
64              } else if (i === fullStars && halfStar) {
65                  stars.push(<Ionicons key={i} name="star-half" size={16} color={COLORS.primary} />)
66              } else {
67                  stars.push(<Ionicons key={i} name="star-outline" size={16} color={COLORS.primary} />)
68              }
69          }
70
71          return (
72              <View style={styles.ratingContainer}>
73                  {stars}
74                  <Text style={styles.ratingText}>{rating}</Text>
75              </View>
76          )
77      }
78
79      const incrementQuantity = () => {
80          if (quantity < productDetails.stock) {
81              setQuantity(quantity + 1)
82          }
83      }
84
85      const decrementQuantity = () => {
86          if (quantity > 1) {
87              setQuantity(quantity - 1)
88          }
89      }
90
91      const addToCart = () => {
92          alert(`Agregado al carrito: ${quantity} unidades de ${productDetails.name}`)
```

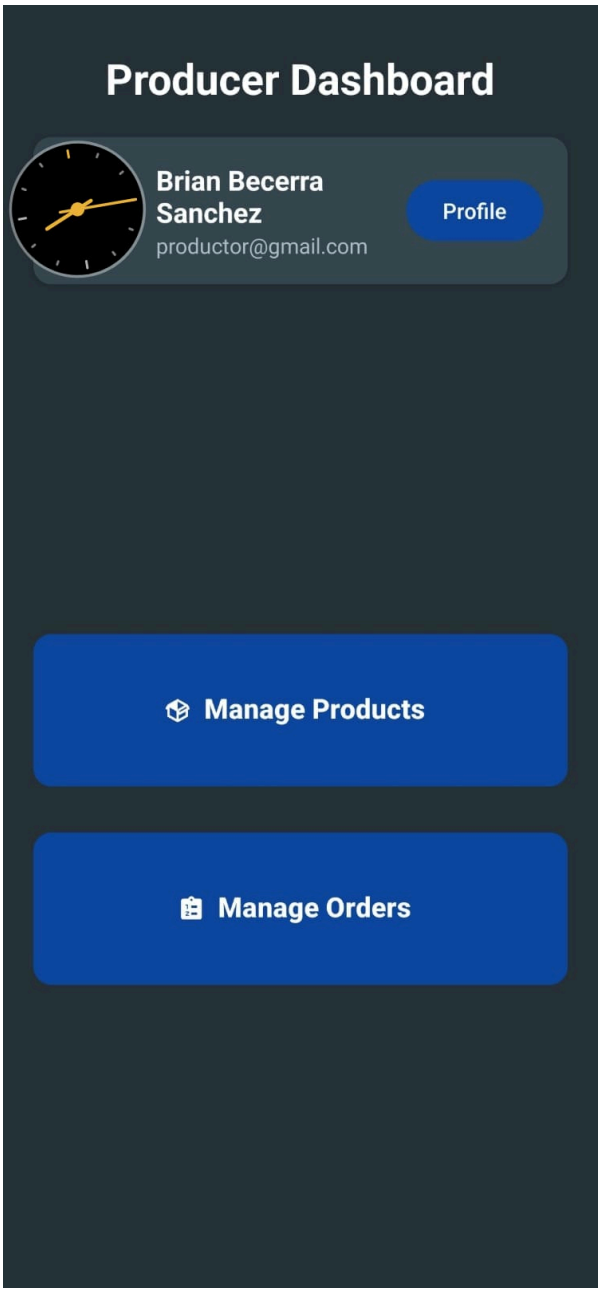


```
Buyer > Components > JS ProductDetails.js > ProductDetails
22  export default function ProductDetails({ route, navigation }) {
93      }
94
95      return (
96          <SafeAreaView style={styles.container}>
97              <StatusBar backgroundColor={COLORS.white} barStyle="dark-content" />
98
99              <View style={styles.header}>
100                 <TouchableOpacity style={styles.backButton} onPress={() => navigation.goBack()}>
101                     <Ionicons name="arrow-back" size={24} color={COLORS.text} />
102                 </TouchableOpacity>
103                 <Text style={styles.headerTitle}>Detalles del Producto</Text>
104                 <TouchableOpacity style={styles.favoriteButton}>
105                     <Ionicons name="heart-outline" size={24} color={COLORS.text} />
106                 </TouchableOpacity>
107             </View>
108
109             <ScrollView showsVerticalScrollIndicator={false}>
110                 <View style={styles.imageContainer}>
111                     <View style={styles.productImagePlaceholder}>
112                         <Text style={styles.imagePlaceholderText}>Imagen del Producto</Text>
113                         <Text style={styles.imagePlaceholderText}>400 x 300</Text>
114                     </View>
115
116
117                     <View style={styles.stockIndicator}>
118                         <Text style={styles.stockText}>
119                             {productDetails.stock > 0 ? `${productDetails.stock} disponibles` : "Agotado"}
120                         </Text>
121                     </View>
122                 </View>
123
124
125                 <View style={styles.productInfoContainer}>
126                     <Text style={styles.productCategory}>{productDetails.category}</Text>
127                     <Text style={styles.productName}>{productDetails.name}</Text>
128                     <View style={styles.priceRatingRow}>
129                         <Text style={styles.productPrice}>{productDetails.price}</Text>
130                         {renderStars(productDetails.rating)}
131                     </View>
132
133
134                     <View style={styles.sellerContainer}>
135                         <Text style={styles.sellerLabel}>Vendedor:</Text>
136                         <View style={styles.sellerInfo}>
137                             <Text style={styles.sellerName}>{productDetails.seller.name}</Text>
138                             {productDetails.seller.verified && (
139                                 <View style={styles.verifiedBadge}>
140                                     <Ionicons name="checkmark-circle" size={14} color={COLORS.white} />
141                                     <Text style={styles.verifiedText}>Verificado</Text>
142                                 </View>
```



```
Buyer > Components > JS ProductDetails.js > ProductDetails
22  export default function ProductDetails({ route, navigation }) {
142                             </View>
143                         ))}
144                     </View>
145                 </View>
146
147
148                 <View style={styles.sectionContainer}>
149                     <Text style={styles.sectionTitle}>Descripción</Text>
150                     <Text style={styles.descriptionText}>{productDetails.description}</Text>
151                 </View>
152
153
154                 <View style={styles.sectionContainer}>
155                     <Text style={styles.sectionTitle}>Características</Text>
156                     {productDetails.features.map((feature, index) => (
157                         <View key={index} style={styles.featureItem}>
158                             <Ionicons name="checkmark-circle" size={18} color={COLORS.primary} />
159                             <Text style={styles.featureText}>{feature}</Text>
160                         </View>
161                     ))}
162                 </View>
163
164
165                 <View style={styles.sectionContainer}>
166                     <Text style={styles.sectionTitle}>Especificaciones</Text>
167                     {Object.entries(productDetails.specifications).map(([key, value], index) => (
168                         <View key={index} style={styles.specificationItem}>
169                             <Text style={styles.specificationKey}>{key}:</Text>
170                             <Text style={styles.specificationValue}>{value}</Text>
171                         </View>
172                     ))}
173                 </View>
174             </View>
175         </ScrollView>
176
177
178         <View style={styles.footer}>
179             <View style={styles.quantitySelector}>
180                 <TouchableOpacity style={styles.quantityButton} onPress={decrementQuantity} disabled={quantity <= 1}>
181                     <Ionicons name="remove" size={20} color={quantity <= 1 ? COLORS.gray : COLORS.text} />
182                 </TouchableOpacity>
183                 <Text style={styles.quantityText}>{quantity}</Text>
184                 <TouchableOpacity
185                     style={styles.quantityButton}
186                     onPress={incrementQuantity}
187                     disabled={quantity >= productDetails.stock}
188                 >
189                     <Ionicons name="add" size={20} color={quantity >= productDetails.stock ? COLORS.gray : COLORS.text} />
190                 </TouchableOpacity>
191             </View>
```

1.3.9 MainProducer



When the component loads, useEffect is executed, which attempts to retrieve user data from Firestore using their email address as an identifier. If the user document exists in the database, the data is stored in the userData state. Otherwise, information available through Firebase authentication, such as email and username, is used. Once the operation is complete, the loading state is deactivated to display the interface.

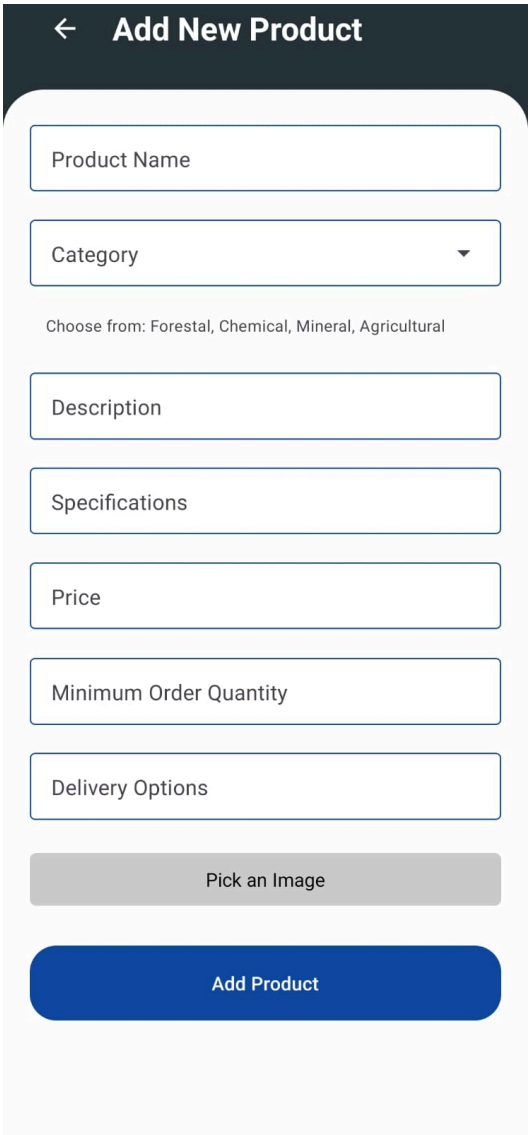
```
const MainProducer = () => {
  const theme = useTheme()
  const navigation = useNavigation()
  const [userData, setUserData] = useState(null)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    const fetchUserData = async () => {
      try {
        const currentUser = auth.currentUser
        if (currentUser) {
          // Get user profile data from Firestore
          const userDocRef = doc(db, "users", currentUser.email)
          const userDoc = await getDoc(userDocRef)

          if (userDoc.exists()) {
            setUserData({
              email: currentUser.email,
              ...userDoc.data(),
            })
          } else {
            setUserData({
              email: currentUser.email,
              fullName: currentUser.displayName || "Producer",
            })
          }
        }
      } catch (error) {
        console.error("Error fetching user data:", error)
      } finally {
        setLoading(false)
      }
    }

    fetchUserData()
  }, [])
```

The screen layout includes a header titled "Producer Dashboard," and if the user data has already been loaded, a card with their information is displayed. Within the card, the user's profile image is checked. If so, it is displayed with Avatar.Image; if not, an image with their first initial or email address is generated using Avatar.Text. The user's name and email address are also displayed, along with a button that allows you to navigate to the profile screen, passing the user data as a parameter.

Below the user card are two main buttons. The first one leads to the product management screen, while the second one takes you to the order management section. Both buttons have representative icons and are stylized to maintain visual consistency within the app.

```
      <View style={styles.buttonContainer}>
        <Button
          mode="contained"
          icon="package-variant"
          contentStyle={styles.buttonContent}
          style={styles.button}
          labelStyle={styles.buttonLabel}
          onPress={() => navigation.navigate("ProductManagement")}
        >
          Manage Products
        </Button>

        <Button
          mode="contained"
          icon="clipboard-list"
          contentStyle={styles.buttonContent}
          style={styles.button}
          labelStyle={styles.buttonLabel}
          onPress={() => navigation.navigate("MyOrders")}
        >
          Manage Orders
        </Button>
      </View>
    </View>
  )
}
```

1.3.10 Add-product-screen



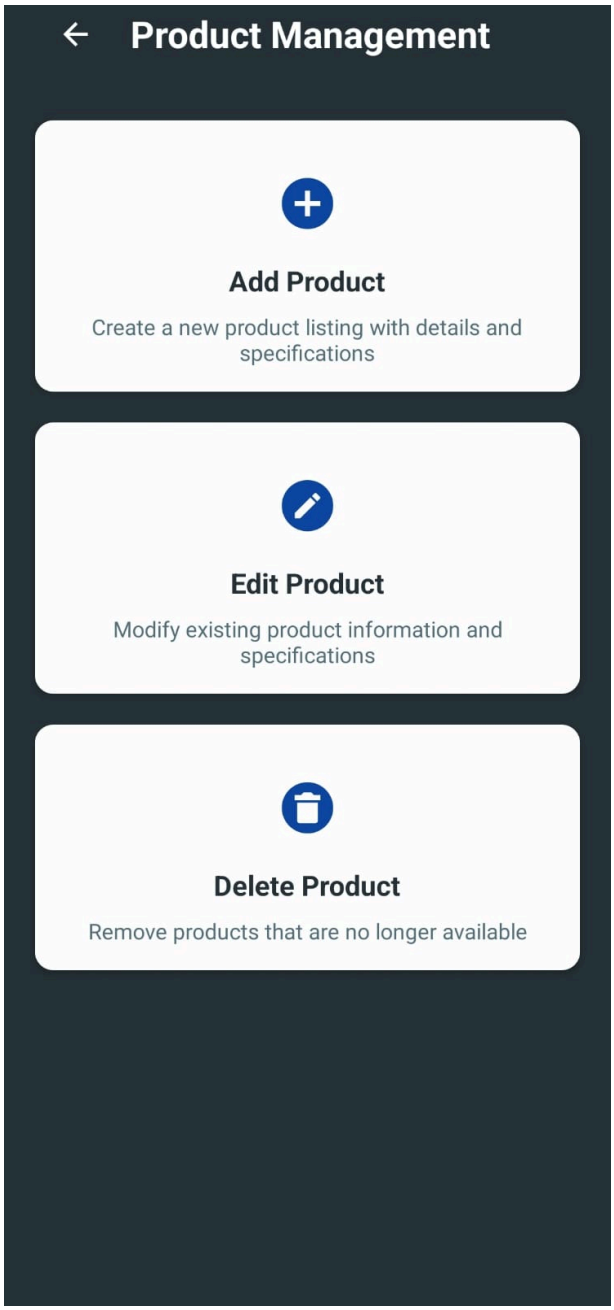
The AddProductScreen component allows users to add a new product to the database. Its core functionality includes capturing product data, validating the form, and storing it in Firestore.

At startup, a product state is defined containing product attributes, such as name, category, description, price, and delivery options. An imageUrl state is also included to store the URL of the uploaded image. The updateProduct function updates the state values as the user enters information into the form fields.

Before submitting the form, the validateForm function verifies that the required fields (name, category, and price) are complete. If any are missing, an alert is displayed to notify the user.

```
<TextInput
    label="Description"
    value={product.description}
    onChangeText={(text) => updateProduct("description", text)}
    mode="outlined"
    style={styles.input}
    outlineColor="#0D47A1"
    activeOutlineColor="#1565C0"
    multiline
    numberOfLines={3}
/>

<TextInput
    label="Specifications"
    value={product.specifications}
    onChangeText={(text) => updateProduct("specifications", text)
    mode="outlined"
    style={styles.input}
    outlineColor="#0D47A1"
    activeOutlineColor="#1565C0"
    multiline
    numberOfLines={3}
/>

<TextInput
    label="Price"
    value={product.price}
    onChangeText={(text) => updateProduct("price", text)}
    mode="outlined"
    style={styles.input}
    outlineColor="#0D47A1"
    activeOutlineColor="#1565C0"
    keyboardType="numeric"
/>

<TextInput
    label="Minimum Order Quantity"
    value={product.minimumOrder}
    onChangeText={(text) => updateProduct("minimumOrder", text)}
    mode="outlined"
```

1.3.11 Product-Management-Screen



The ProductManagementScreen component provides an interface for managing products. It includes interactive cards that allow the user to add, edit, or delete products, each with a representative icon and a brief description. It also features a back button to return to the previous screen.

```
    return (
        <View style={styles.container}>
            <View style={styles.header}>
                <IconButton icon="arrow-left" iconColor="#FFFFFF" size={24} onPress={() => navigation.goBack()} />
                <Text style={styles.title}>Product Management</Text>
            </View>

            <View style={styles.actionCards}>
                <Card style={styles.card} onPress={() => navigation.navigate("AddProduct")}>
                    <Card.Content style={styles.cardContent}>
                        <IconButton icon="plus-circle" iconColor="#0D47A1" size={40} style={styles.cardIcon} />
                        <Text style={styles.cardTitle}>Add Product</Text>
                        <Text style={styles.cardDescription}>Create a new product listing with details and specifications</Text>
                    </Card.Content>
                </Card>

                <Card style={styles.card} onPress={() => navigation.navigate("EditProduct")}>
                    <Card.Content style={styles.cardContent}>
                        <IconButton icon="pencil-circle" iconColor="#0D47A1" size={40} style={styles.cardIcon} />
                        <Text style={styles.cardTitle}>Edit Product</Text>
                        <Text style={styles.cardDescription}>Modify existing product information and specifications</Text>
                    </Card.Content>
                </Card>

                <Card style={styles.card} onPress={() => navigation.navigate("DeleteProduct")}>
                    <Card.Content style={styles.cardContent}>
                        <IconButton icon="delete-circle" iconColor="#0D47A1" size={40} style={styles.cardIcon} />
                        <Text style={styles.cardTitle}>Delete Product</Text>
                        <Text style={styles.cardDescription}>Remove products that are no longer available</Text>
                    </Card.Content>
                </Card>
            </View>
        </View>
    )
}
```