Simple P2P Market for the Regeneration Credit

Introduction

This report documents the implementation of a simple peer-to-peer (P2P) marketplace for the Regeneration Credit (RC) token. The solution was developed as part of the broader Regeneration Credit ecosystem, which aims to facilitate financial transactions that support environmental regeneration projects.

The primary objective of this implementation was to create a lightweight, non-custodial marketplace where RC token holders can list their tokens for sale directly to other users. Unlike traditional exchanges that hold funds in custody, this P2P market serves as a listing and coordination layer—the actual token transfers occur off-chain between the parties involved.

The implementation consists of three main components:

1. **Smart Contract (RCMarket.sol)** - Deployed on the Sintrop blockchain, handles offer creation, cancellation, and sale confirmation
2. **Mobile Application Integration** - A new MarketScreen added to the Regeneration Credit mobile app
3. **Backend Infrastructure** - Contract wrappers and hooks for seamless interaction

This first version prioritizes simplicity and transparency over advanced features. The goal is to provide a functional foundation that can be enhanced in future iterations based on user feedback and real-world usage patterns.

Contract Address

RCMarket Smart Contract: `0xA59CE87F2637084f13057357c9Ee49A47232A769`

Deployed on: Sintrop Mainnet RPC Endpoint: https://rpc.sintrop.com

---

Usage Tutorial

Overview

The P2P market allows RC token holders to list their tokens for sale without intermediaries. Buyers can browse available offers, contact sellers directly, and complete transactions outside the blockchain.

Step-by-Step Example

1. Accessing the Market

Open the Regeneration Credit mobile app and tap the "Mercado" button on the home screen. This will navigate to the MarketScreen where you can view all active offers.

2. Creating an Offer

To sell RC tokens:

1. Tap the "Publicar Oferta" button
2. Fill in the form:
3. **Quantidade de RC**: The amount of tokens you want to sell (e.g., "1000")
4. **Preço Unitário**: Your price per token (e.g., "R$0,10")
5. **Método de Pagamento**: How you want to receive payment (e.g., "pix:12345678901@nubank")
6. **Descrição**: Additional details (e.g., "Vendo 1000 RC por R$100,00. Transferência via Pix. Meu WhatsApp: (11) 99999-9999")
7. Tap "Publicar Oferta"

The smart contract will verify that you have sufficient RC balance and create the listing.

3. Buying RC Tokens

To purchase RC tokens:

1. Browse the list of active offers
2. Tap "Comprar" on the offer you want
3. An alert will appear: "O mercado é apenas informativo, você deverá entrar em contato com vendedor pessoalmente. Veja na descrição da oferta as instruções do vendedor."
4. Contact the seller using the information in the description
5. Complete the payment (Pix, bank transfer, etc.) outside the app
6. Request the seller to confirm the transaction

4. Completing a Sale (Seller)

After receiving payment:

1. Go to your offer in the MarketScreen
2. Tap "Concluir"
3. Enter the buyer's wallet address
4. The smart contract will record the completion and update your reputation score

---

Mobile App Implementation

New Files Added

The following files were added to the mobile application:

1. **src/contracts/RCMarket.json** - ABI and contract address
2. **src/domain/RCMarket/rcMarketContract.ts** - Contract read functions
3. **src/domain/RCMarket/useCases/useCreateOffer.ts** - Create offer hook
4. **src/domain/RCMarket/useCases/useCancelOffer.ts** - Cancel offer hook
5. **src/domain/RCMarket/useCases/useConfirmSale.ts** - Confirm sale hook
6. **src/screens/MarketScreen/MarketScreen.tsx** - Main market screen
7. **src/screens/MarketScreen/components/OfferItem.tsx** - Individual offer display

8. **src/screens/MarketScreen/components/CreateOfferModal.tsx** - Offer creation form

Modified Files

1. **src/screens/index.ts** - Added MarketScreen export
2. **src/screens/HomeScreen/HomeScreen.tsx** - Added "Mercado" menu button
3. **src/routes/AppStack.tsx** - Added MarketScreen route
4. **src/contracts/index.ts** - Added RCMarket export

Future Development

While this first version provides essential functionality, there are several areas identified for future enhancement:

Recommended Improvements

1. **In-App Messaging System**
2. Implement a secure messaging feature within the app
3. Allow buyers and sellers to communicate without sharing personal contact information initially
4. This reduces exposure to scams while maintaining usability
5. **Integrated Payment Processing**
6. Explore integration with payment processors for Pix transactions
7. Implement escrow functionality for higher-value transactions
8. Consider multi-signature wallets for disputed transactions
9. **Reputation System**
10. Expand the basic sellerSalesCount into a full reputation profile
11. Add buyer reviews and ratings
12. Implement trust scores based on transaction history
13. Consider badge system for verified sellers
14. **Advanced Matching**

15. Add search and filter capabilities
16. Implement price alerts

17. Support for partial fills (selling/buying in installments)

18. **Security Enhancements**

19. Two-factor authentication for high-value transactions
20. Transaction limits based on verification level

21. Fraud detection mechanisms

22. **Legal Compliance**

23. Consult with legal professionals regarding financial regulations
24. Implement KYC/AML procedures if required
25. Add necessary disclaimers and terms of service

The current implementation deliberately avoids storing sensitive data or processing on-chain transactions to minimize legal and operational complexity. Future versions can add more sophisticated features as the ecosystem matures and regulatory clarity improves.

Problems and Limitations

No Private Database

This implementation intentionally does not include a private database. This design choice aligns with the core principles of the Regeneration Credit ecosystem:

1. **Transparency**: All offers and transaction records are publicly accessible on the blockchain
2. **Decentralization**: No single point of failure or control
3. **Censorship Resistance**: No entity can remove or modify historical records
4. **Trust Minimization**: Parties interact directly without intermediaries

However, this approach comes with trade-offs:

- No ability to hide offers or make private listings
- All pricing information is publicly visible

- Transaction history cannot be modified or deleted

Legal and Operational Considerations

Operating a P2P marketplace for token trading involves several legal and operational challenges:

Legal Risks

1. **Securities Regulation**: Depending on jurisdiction, RC tokens may be considered securities. Facilitating their trade could trigger registration requirements.
2. **Money Transmission**: If the platform processes payments, it may require money transmitter licenses.
3. **Consumer Protection**: Sellers and buyers have limited recourse if transactions go wrong.
4. **Tax Implications**: Capital gains from token sales may have tax consequences.

Operational Challenges

1. **Dispute Resolution**: No intermediary to resolve conflicts between buyers and sellers.
2. **Fraud Risk**: No verification system beyond wallet addresses.
3. **Price Manipulation**: Anyone can list at any price.
4. **Scam Awareness**: Users must be vigilant about common cryptocurrency scams.

Data Exposure Warning

All data published on the blockchain is public and permanent.

Users should be aware that:

- Wallet addresses can be traced across all transactions
- Payment information (e.g., Pix keys) published in descriptions becomes permanently visible
- Amounts and prices are publicly accessible
- Once published, information cannot be removed

Users should never publish sensitive personal information such as: - Full names (except as business name) - Home addresses - Government ID numbers - Banking credentials

---

Smart Contract Summary

Contract Overview

The RCMarket contract is a non-custodian marketplace deployed on the Sintrop blockchain. It serves as a registry for P2P RC token offers without handling actual token transfers.

Key Functions

createOffer(uint256 amountRC, string unitPrice, string paymentMethod, string description)

Creates a new listing for selling RC tokens.

- **Parameters**:
- amountRC: Number of RC tokens to sell
- unitPrice: Price per token (e.g., "R$0,10")
- paymentMethod: Payment instructions (e.g., "pix:12345678901@bank")
- description: Additional details about the offer
- **Requirements**:
- Seller must have sufficient RC token balance
- All string fields must be non-empty and within length limits
- Returns the new offer ID

cancelOffer(uint256 offerId)

Cancels an active offer.

- **Requirements**:
- Only the original seller can cancel

- Offer must still be active

confirmSale(uint256 offerId, address buyer)

Records a completed sale.

- **Parameters**:
- offerId: The offer being completed
- buyer: The buyer's wallet address
- **Requirements**:
- Only the seller can confirm
- Offer must be active
- **Note**: This does not transfer tokens on-chain. The actual transfer happens off-chain between parties.

getOffersCount() → uint256

Returns the total number of offers created.

getOffer(uint256 offerId) → Offer

Returns detailed information about a specific offer.

getSellerOfferIds(address seller) → uint256[]

Returns all offer IDs belonging to a seller.

getSellerSalesCount(address seller) → uint256

Returns the number of completed sales for a seller (reputation).

Offer Structure

```
struct Offer {
    uint256 id;
    address seller;
```

```
    uint256 amountRC;
    string unitPrice;
    string paymentMethod;
    string description;
    bool active;
    uint256 createdAt;
    address buyer;
    uint256 completedAt;
}
```

Example Usage

Creating an Offer

```
rcMarket.createOffer(
    1000,                          // amountRC: Sell 1000 RC tokens
    "R$0,10",                      // unitPrice: R$0.10 per token
    "pix:12345678901@nubank",      // paymentMethod: Pix key
    "Vendo 1000 RC por R$100,00. Transfira o valor via Pix para a ch
);
```

Response

The function returns a new offer ID (e.g., `1` for the first offer).

Browsing Offers

```
// Get total number of offers
uint256 count = rcMarket.getOffersCount();

// Get each offer in reverse order (newest first)
for (uint256 i = count; i >= 1; i--) {
    Offer memory offer = rcMarket.getOffer(i);
```

```
    // Display offer.id, offer.amountRC, offer.unitPrice, etc.
}
```

Complete Smart Contract Code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

/**
 * @title RCMarket
 * @dev P2P marketplace for RC tokens without custody
 *
 * Sellers create listing offers declaring quantity, unit price and
 * Off-chain payment (Pix/transfer) happens outside blockchain.
 * Seller confirms sale, transferring tokens to buyer.
 *
 * This contract does NOT hold tokens - it only registers offers and
 * Transfer verification is done off-chain via Transfer events.
 */
contract RCMarket {

    // --- Constants ---
    uint256 public constant MAX_DESCRIPTION_LENGTH = 500;
    uint256 public constant MAX_PAYMENT_METHOD_LENGTH = 200;
    uint256 public constant MAX_PRICE_LENGTH = 50;

    // --- State Variables ---
    IERC20 public rcToken;
    uint256 private _offersCount;

    // Mappings
    mapping(uint256 => Offer) public offers;
    mapping(address => uint256[]) public sellerOfferIds;
    mapping(address => uint256) public sellerSalesCount;
```

```
    // --- Structs ---
    struct Offer {
        uint256 id;
        address seller;
        uint256 amountRC;      // Total amount of RC tokens for sale
        string unitPrice;      // Price per unit (ex: "R$0,10" or "0
        string paymentMethod;  // Payment info (ex: "pix:cpf@bank")
        string description;    // Offer description
        bool active;
        uint256 createdAt;
        address buyer;
        uint256 completedAt;
    }

    // --- Events ---
    /**
     * @dev Emitted when a new offer is created
     * @param offerId Unique identifier of the offer
     * @param seller Address of the seller
     * @param amountRC Amount of RC tokens for sale
     * @param unitPrice Unit price for the tokens
     */
    event OfferCreated(
        uint256 indexed offerId,
        address indexed seller,
        uint256 amountRC,
        string unitPrice
    );

    /**
     * @dev Emitted when an offer is cancelled
     * @param offerId Unique identifier of the cancelled offer
     * @param seller Address of the seller who cancelled
     */
    event OfferCancelled(
        uint256 indexed offerId,
        address indexed seller
```

```
    );

    /**
     * @dev Emitted when a sale is completed
     * @param offerId Unique identifier of the completed offer
     * @param seller Address of the seller
     * @param buyer Address of the buyer
     * @param amountRC Amount of RC tokens transferred
     */
    event OfferCompleted(
        uint256 indexed offerId,
        address indexed seller,
        address indexed buyer,
        uint256 amountRC
    );

    // --- Errors ---
    error ZeroAmount();
    error InvalidPrice();
    error InvalidPaymentMethod();
    error InvalidDescription();
    error OfferNotActive(uint256 offerId);
    error NotSeller(uint256 offerId);
    error InsufficientBalance(uint256 required, uint256 available);
    error ZeroAddress();
    error OfferNotFound(uint256 offerId);

    // --- Constructor ---
    constructor(address _rcTokenAddress) {
        if (_rcTokenAddress == address(0)) revert ZeroAddress();
        rcToken = IERC20(_rcTokenAddress);
    }

    // --- External Functions ---

    /**
     * @dev Creates a new offer for selling RC tokens
```

```
     * @param amountRC Amount of RC tokens to sell
     * @param unitPrice Price per unit (short string, ex: "R$0,10")
     * @param paymentMethod Payment method (ex: "pix:cpf@bank")
     * @param description Description of the offer
     * @return offerId The unique ID of the created offer
     *
     * Requirements:
     * - amountRC must be greater than 0
     * - unitPrice must not be empty (max 50 chars)
     * - paymentMethod must not be empty
     * - description must not be empty
     * - seller must have sufficient RC token balance
     */
    function createOffer(
        uint256 amountRC,
        string calldata unitPrice,
        string calldata paymentMethod,
        string calldata description
    ) external returns (uint256) {
        if (amountRC == 0) revert ZeroAmount();
        if (bytes(unitPrice).length == 0 || bytes(unitPrice).length
            revert InvalidPrice();
        if (bytes(paymentMethod).length == 0 || bytes(paymentMethod)
            revert InvalidPaymentMethod();
        if (bytes(description).length == 0 || bytes(description).len
            revert InvalidDescription();

        // Check seller has sufficient balance
        uint256 balance = rcToken.balanceOf(msg.sender);
        if (balance < amountRC) revert InsufficientBalance(amountRC,

        _offersCount++;
        uint256 offerId = _offersCount;

        offers[offerId] = Offer({
            id: offerId,
            seller: msg.sender,
```

```
            amountRC: amountRC,
            unitPrice: unitPrice,
            paymentMethod: paymentMethod,
            description: description,
            active: true,
            createdAt: block.timestamp,
            buyer: address(0),
            completedAt: 0
        });

        sellerOfferIds[msg.sender].push(offerId);

        emit OfferCreated(offerId, msg.sender, amountRC, unitPrice);

        return offerId;
    }

    /**
     * @dev Cancels an active offer
     * @param offerId ID of the offer to cancel
     *
     * Requirements:
     * - Offer must be active
     * - Only seller can cancel
     */
    function cancelOffer(uint256 offerId) external {
        Offer storage offer = offers[offerId];

        if (!offer.active) revert OfferNotActive(offerId);
        if (offer.seller != msg.sender) revert NotSeller(offerId);

        offer.active = false;

        emit OfferCancelled(offerId, msg.sender);
    }

    /**
```

```
     * @dev Confirms a sale and records the buyer
     * @param offerId ID of the offer being completed
     * @param buyer Address of the buyer
     *
     * Note: Token transfer is done off-chain. This function only re
     * the completion for transparency and reputation tracking.
     *
     * Requirements:
     * - Offer must be active
     * - Only seller can confirm
     * - buyer address must not be zero
     */
    function confirmSale(uint256 offerId, address buyer) external {
        if (buyer == address(0)) revert ZeroAddress();

        Offer storage offer = offers[offerId];

        if (!offer.active) revert OfferNotActive(offerId);
        if (offer.seller != msg.sender) revert NotSeller(offerId);

        offer.active = false;
        offer.buyer = buyer;
        offer.completedAt = block.timestamp;

        sellerSalesCount[msg.sender]++;

        emit OfferCompleted(offerId, msg.sender, buyer, offer.amount
    }

    // --- View Functions ---

    /**
     * @dev Returns the total number of offers created
     */
    function getOffersCount() external view returns (uint256) {
        return _offersCount;
    }
```

```
    /**
     * @dev Returns a specific offer by ID
     * @param offerId ID of the offer to retrieve
     * @return Offer struct with the offer details
     */
    function getOffer(uint256 offerId) external view returns (Offer
        if (offerId == 0 || offerId > _offersCount) revert OfferNotF
        return offers[offerId];
    }

    /**
     * @dev Returns all offer IDs for a specific seller
     * @param seller Address of the seller
     * @return Array of offer IDs belonging to the seller
     */
    function getSellerOfferIds(address seller) external view returns
        return sellerOfferIds[seller];
    }

    /**
     * @dev Returns the total number of completed sales for a seller
     * @param seller Address of the seller
     * @return Number of completed sales
     */
    function getSellerSalesCount(address seller) external view retur
        return sellerSalesCount[seller];
    }
}

// Minimal IERC20 interface
interface IERC20 {
    function balanceOf(address account) external view returns (uint2
}
```

Mobile Application Files

The following files must be added to the Regeneration Credit mobile project.

1. src/contracts/RCMarket.json

```
{
  "abi": [...],
  "address": "0xA59CE87F2637084f13057357c9Ee49A47232A769"
}
```

(Full ABI available in the operating-system repository)

2. src/domain/RCMarket/rcMarketContract.ts

```
import Web3 from "web3";
import { RCMarket } from "@contracts";

export interface Offer {
  id: number;
  seller: string;
  amountRC: string;
  unitPrice: string;
  paymentMethod: string;
  description: string;
  active: boolean;
  createdAt: number;
  buyer: string;
  completedAt: number;
}

interface RCMarketContractProps {
  rpc: string;
}

export async function getOffersCount({ rpc }: RCMarketContractProps)
  const provider = new Web3(new Web3.providers.HttpProvider(rpc));
```

```
  const contract = new provider.eth.Contract(RCMarket.abi, RCMarket.
  const count = await contract.methods.getOffersCount().call() as nu
  return Number(count);
}

export async function getOffer({ rpc, offerId }: { rpc: string; offe
  const provider = new Web3(new Web3.providers.HttpProvider(rpc));
  const contract = new provider.eth.Contract(RCMarket.abi, RCMarket.
  const offer = await contract.methods.getOffer(offerId).call() as a
  return {
    id: Number(offer.id),
    seller: offer.seller,
    amountRC: offer.amountRC,
    unitPrice: offer.unitPrice,
    paymentMethod: offer.paymentMethod,
    description: offer.description,
    active: offer.active,
    createdAt: Number(offer.createdAt),
    buyer: offer.buyer,
    completedAt: Number(offer.completedAt),
  };
}
```

3. src/screens/MarketScreen/MarketScreen.tsx

```
import { useEffect, useState } from "react";
import { View, FlatList, ListRenderItemInfo } from "react-native";
import { Screen, Text, Button } from "@components";
import { useUserContext } from "@hooks";
import { Offer, getOffersCount, getOffer } from "@domain";
import { OfferItem } from "./components/OfferItem";
import { CreateOfferModal } from "./components/CreateOfferModal";

const RPC = "https://rpc.sintrop.com";

export function MarketScreen() {
  const { address } = useUserContext();
```

```
  const [offers, setOffers] = useState<Offer[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [showCreateModal, setShowCreateModal] = useState(false);

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

  async function loadOffers() {
    try {
      setIsLoading(true);
      const count = await getOffersCount({ rpc: RPC });
      const loadedOffers: Offer[] = [];
      for (let i = count; i >= 1; i--) {
        try {
          const offer = await getOffer({ rpc: RPC, offerId: i });
          loadedOffers.push(offer);
        } catch (e) {}
      }
      setOffers(loadedOffers);
    } catch (error) {
      console.error("Error loading offers:", error);
    } finally {
      setIsLoading(false);
    }
  }

  function renderOffer({ item }: ListRenderItemInfo<Offer>) {
    return <OfferItem offer={item} currentUserAddress={address} onRe
  }

  return (
    <Screen title="Mercado" showBackButton>
      <View className="flex-1">
        <View className="p-4 border-b border-gray-700">
          <Button onPress={() => setShowCreateModal(true)} className
            Publicar Oferta
```

```
          </Button>
        </View>
        <FlatList
          data={offers}
          keyExtractor={(item) => item.id.toString()}
          renderItem={renderOffer}
          contentContainerClassName="p-4"
          ListEmptyComponent={
            isLoading ? (
              <Text className="text-gray-400 text-center mt-10">Carr
            ) : (
              <Text className="text-gray-400 text-center mt-10">Nenh
            )
          }
        />
      </View>
      <CreateOfferModal
        isVisible={showCreateModal}
        onClose={() => setShowCreateModal(false)}
        onSuccess={() => {
          setShowCreateModal(false);
          loadOffers();
        }}
      />
    </Screen>
  );
}
```

4. src/screens/MarketScreen/components/OfferItem.tsx

```
import { View, Text, TouchableOpacity, Alert } from "react-native";
import { Offer, useCancelOffer, useConfirmSale } from "@domain";

interface OfferItemProps {
  offer: Offer;
  currentUserAddress?: string;
  onRefresh: () => void;
```

```
}

export function OfferItem({ offer, currentUserAddress, onRefresh }:
  const { cancelOffer } = useCancelOffer();
  const { confirmSale } = useConfirmSale();
  const isOwner = currentUserAddress?.toLowerCase() === offer.seller
  const isCompleted = !offer.active && offer.completedAt > 0;

  function handleBuy() {
    Alert.alert(
      "Comprar",
      "O mercado é apenas informativo, você deverá entrar em contato
      [{ text: "OK" }]
    );
  }

  function handleCancel() {
    Alert.alert("Cancelar Oferta", "Tem certeza?", [
      { text: "Não", style: "cancel" },
      { text: "Sim", onPress: () => { cancelOffer({ offerId: offer.i
    ]);
  }

  function handleConfirm() {
    Alert.prompt("Confirmar Venda", "Endereço do comprador:", [
      { text: "Cancelar", style: "cancel" },
      { text: "Confirmar", onPress: (buyer?: string) => {
        if (buyer && buyer.startsWith("0x")) {
          confirmSale({ offerId: offer.id, buyer });
          onRefresh();
        } else {
          Alert.alert("Erro", "Endereço inválido");
        }
      }}
    ]);
  }
```

```
  let statusText = "Ativa";
  let statusColor = "text-green-400";
  if (isCompleted) { statusText = "Concluída"; statusColor = "text-b
  else if (!offer.active) { statusText = "Cancelada"; statusColor =

  return (
    <View className="bg-gray-800 rounded-lg p-4 mb-4">
      <View className="flex-row justify-between items-center mb-2">
        <Text className={`font-bold ${statusColor}`}>{statusText}</T
        <Text className="text-gray-400 text-sm">#{offer.id}</Text>
      </View>
      <View className="flex-row justify-between mb-2">
        <Text className="text-white font-bold text-lg">{offer.amount
        <Text className="text-emerald-400 font-bold">{offer.unitPric
      </View>
      <Text className="text-gray-400 text-sm mb-2">Vendedor: {offer.
      <Text className="text-gray-300 text-sm mb-2">Pagamento: {offer
      <Text className="text-gray-300 text-sm mb-4">{offer.descriptio
      <View className="flex-row gap-2">
        {!isCompleted && offer.active && (
          <>
            <TouchableOpacity className="flex-1 bg-emerald-600 py-2
              <Text className="text-white text-center font-bold">Com
            </TouchableOpacity>
            {isOwner && (
              <>
                <TouchableOpacity className="bg-red-600 px-4 py-2 ro
                  <Text className="text-white text-center">Cancelar<
                </TouchableOpacity>
                <TouchableOpacity className="bg-blue-600 px-4 py-2 r
                  <Text className="text-white text-center">Concluir<
                </TouchableOpacity>
              </>
            )}
          </>
        )}
      </View>
```

```
    </View>
  );
}
```

Conclusion

This first version of the P2P market provides a functional foundation for RC token trading without intermediaries. The implementation prioritizes simplicity, transparency, and decentralization—core values of the Regeneration Credit ecosystem.

While the current version has limitations (no in-app messaging, no integrated payments, basic reputation system), it serves as a solid starting point. Future iterations can add more sophisticated features as the user base grows and the ecosystem matures.

Users are reminded to exercise caution when trading:

- Always verify the other party's reputation
- Never share sensitive personal information
- Complete payments through trusted channels
- Be aware that all blockchain data is permanent and public

The Regeneration Credit ecosystem continues to evolve, and this marketplace is one step toward enabling financial flows that support environmental regeneration around the world.

Document Version: 1.0 Created: March 2026 Regeneration Credit Ecosystem