```python
import multizain
import getpass
import hashlib
import json
import binascii
import subprocess
import random
import string
import tkinter as tk
from tkinter import messagebox
from tkinter import simpledialog

#--------------------------------------------------------------------------------------------#
rpcuser = "multichainrpc"
rpcpassword = "HgqyMSjKBg41DHV7B1tPG5daykNe3GyDHEMQxk85sc6Y"
rpchost = "127.0.0.1"
rpcport = "2792"
chainname = "Information_Security"

# Create a MultiChain client instance
mc = multizain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)
#--------------------------------------------------------------------------------------------#
def hash_password(password):
    # Use a secure hashing algorithm (e.g., SHA-256) to hash the password
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    return hashed_password
#--------------------------------------------------------------------------------------------#

def upload_data(stream_name, data):
    # Iterate through the data and publish to the stream
    for item in data:
        # Use the roll number as the key
        key = item['roll_no']
        # Hash the password before storing it
        hashed_password = hash_password(item['password'])
        # Construct the data to publish as a JSON string, including current_balance
        data_to_publish = json.dumps({'username': item['username'], 'hashed_password':
hashed_password, 'roll_no': item['roll_no']})

        # Convert the data to hexadecimal
        hex_data = binascii.hexlify(data_to_publish.encode()).decode()

        # Build the command to publish
        command = fr"D:\Multichain\multichain-cli.exe {chainname} publish {stream_name} {key}
{hex_data}"

        # Execute the command
        try:
            result = subprocess.check_output(command, shell=True)
            print(result.decode())
            # If upload is successful, call add_user_balance function
            add_user_balance(key)
        except subprocess.CalledProcessError as e:
            print(f"Error code: {e.returncode}")
            print(f"Error message: {e.output.decode()}")

    print(f"Data uploaded to the {stream_name} stream.")
```

```python
#------------------------------------------------------------------------------------------#
def add_user_balance(roll_no):
    try:
        # Prepare the data to publish as a JSON string
        data_to_publish = json.dumps({'fast_coin_balance': 100})  # Initial balance of 100
fast coins

        # Convert the data to hexadecimal
        hex_data = binascii.hexlify(data_to_publish.encode()).decode()

        # Build the command to publish
        command = fr"D:\Multichain\multichain-cli.exe {chainname} publish user_balance
{roll_no} {hex_data}"

        # Execute the command
        result = subprocess.check_output(command, shell=True)
        print(result.decode())
        print("User balance published successfully.")
    except subprocess.CalledProcessError as e:
        print(f"Error code: {e.returncode}")
        print(f"Error message: {e.output.decode()}")
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")



#------------------------------------------------------------------------------------------#
def upload_data_products(stream_name, data):
    # Iterate through the data and publish to the stream
    for item in data:
        # Use the roll number as the key
        key = item['roll_no']
        product_name = item['product_name']
        product_price = item['product_price']

        # Construct the data to publish as a JSON string
        data_to_publish = json.dumps({'product_name': product_name, 'product_price':
product_price})

        # Convert the data to hexadecimal
        hex_data = binascii.hexlify(data_to_publish.encode()).decode()

        # Build the command to publish
        command = fr"D:\Multichain\multichain-cli.exe {chainname} publish {stream_name} {key}
{hex_data}"

        # Execute the command
        try:
            result = subprocess.check_output(command, shell=True)
            print(result.decode())
        except subprocess.CalledProcessError as e:
            print(f"Error code: {e.returncode}")
            print(f"Error message: {e.output.decode()}")

    print(f"Data uploaded to the {stream_name} stream.")

#---------------------------------------------------------------------------------------------#
```

```python
def add_product(roll_no):
    # Take product details as input
    product_name = input("Enter product name: ")
    product_price = input("Enter product price: ")

    # Prepare the data to upload
    data_to_upload = [{'roll_no': roll_no, 'product_name': product_name, 'product_price':
product_price}]
    # Upload the data to the 'products_record' stream
    upload_data_products('products_record', data_to_upload)
    print("Product added successfully.")

#---------------------------------------------------------------------------------------------#
def display_user_products(key):
    stream_name = 'products_record'
    try:
        result = mc.liststreamkeyitems(stream_name, key)  # Ensure key is passed as a string
        if result:
            print("Products:")
            for item in result:
                item_data = binascii.unhexlify(item['data']).decode()
                parsed_data = json.loads(item_data)
                product_name = parsed_data.get('product_name')
                product_price = parsed_data.get('product_price')
                if product_name is not None and product_price is not None:
                    print(f"Product Name: {product_name}, Price: {product_price}")
                else:
                    print("Incomplete product data.")
        else:
            print("No products found for this key in the stream.")
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")

#---------------------------------------------------------------------------------------------#
def fetch_product_data(stream_name):
    try:
        result = mc.liststreamitems(stream_name)
        if result:
            print("Products:")
            for item in result:
                item_data = binascii.unhexlify(item['data']).decode()
                parsed_data = json.loads(item_data)
                print(f"Product Name: {parsed_data['product_name']}, Price:
{parsed_data['product_price']}")
        else:
            print("No products found in this stream.")
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")

#-------------------------------------------------------------------------------------------#
def buy_product(buyer_roll_no):
    # Prompt user to enter the seller's roll number
    seller_roll_no = input("Enter the roll number of the user from whom you want to buy: ")

    # Display the products of the seller
    display_user_products(seller_roll_no)

    # Prompt user to select the product they want to buy
    product_name = input("Enter the name of the product you want to buy: ")

    # Retrieve the selected product's information
    try:
        result = mc.liststreamkeyitems('products_record', seller_roll_no)
        if result:
            for item in result:
                item_data = binascii.unhexlify(item['data']).decode()
                parsed_data = json.loads(item_data)
                if parsed_data.get('product_name') == product_name:
                    product_price = int(parsed_data.get('product_price'))
                    print(f"Product found: {product_name}, Price: {product_price} fastcoins")

                    # Prompt user if they want to proceed with the transaction
                    confirm_transaction = input("Do you want to proceed with the transaction?
(yes/no): ").lower()
                    if confirm_transaction == "yes":
                        # Get the current balance of the buyer
                        buyer_balance = int(get_user_balance(buyer_roll_no) )
                        if buyer_balance is not None:
                            # Deduct the price of the product from the buyer's balance
                            new_buyer_balance = buyer_balance - product_price
                            # Update buyer's balance in the user_balance stream
                            publish_user_balance(buyer_roll_no, new_buyer_balance)
                        else:
                            print("Failed to retrieve buyer's balance.")

                        # Get the seller's current balance
                        seller_balance = int(get_user_balance(seller_roll_no))
                        if seller_balance is not None:
                            # Calculate new balance for the seller after adding the product
                            new_seller_balance = seller_balance + product_price
                            # Update seller's balance in the user_balance stream
                            publish_user_balance(seller_roll_no, new_seller_balance)
                            store_transaction(buyer_roll_no,seller_roll_no)
                        else:
                            print("Failed to retrieve seller's balance.")
                    else:
                        print("Transaction canceled by the user.")
                    return
            print("Specified product not found for sale by the specified user.")
        else:
            print("No products found for sale by the specified user.")
    except Exception as e:  # Handle all exceptions
        print(f"Error occurred: {e}")

    return False
#---------------------------------------------------------------------------------------------#


def generate_transaction_id():
    """Generate a random transaction ID."""
    # Generate a random alphanumeric string of length 10
    transaction_id = ''.join(random.choices(string.ascii_uppercase + string.digits, k=10))
    return transaction_id


def store_transaction(buyer_roll_no, seller_roll_no):
    try:
        # Generate a random transaction ID
        transaction_id = generate_transaction_id()

        # Prepare the data to publish as a JSON string
        data_to_publish = json.dumps({'transaction_id': transaction_id, 'buyer_roll_no':
buyer_roll_no, 'seller_roll_no': seller_roll_no})

        # Convert the data to hexadecimal
        hex_data = binascii.hexlify(data_to_publish.encode()).decode()

        # Build the command to publish
        command = fr"D:\Multichain\multichain-cli.exe {chainname} publish transaction_record
{buyer_roll_no}_{seller_roll_no} {hex_data}"

        # Execute the command
        result = subprocess.check_output(command, shell=True)
        print(result.decode())
        print("Transaction ID successfully stored.")

    except subprocess.CalledProcessError as e:
        print(f"Error code: {e.returncode}")
        print(f"Error message: {e.output.decode()}")
    except Exception as e:
        print(f"Error occurred: {e}")

#-----------------------------------------------------------------------------------------------

def is_roll_no_unique(roll_no):
    # Check if the roll number already exists in the blockchain
    try:
        result = mc.liststreamkeyitems('credentials', roll_no)
        return not result  # If the result is empty, roll number is unique
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")
        return False

#----------------------------------------------------------------------------------------------#

def register():
    roll_no = input("Enter unique roll number: ")
    # Check if the roll number already exists
    if not is_roll_no_unique(roll_no):
        print("Roll number already registered. Please choose a different roll number.")
        return

    username = input("Enter username: ")
    password = getpass.getpass("Enter password: ")

    # Prepare the data to upload
    data_to_upload = [{'username': username, 'password': password, 'roll_no': roll_no}]
    # Upload the data to the 'credentials' stream
    upload_data('credentials', data_to_upload)
    print("Registration successful.")

#----------------------------------------------------------------------------------------------#
```

```python
def authenticate_user(username, roll_no, password):
    # Retrieve data from MultiChain based on roll number
    try:
        result = mc.liststreamkeyitems('credentials', roll_no)
        if result:
            item_data = binascii.unhexlify(result[0]['data']).decode()
            parsed_data = json.loads(item_data)
            stored_username = parsed_data.get('username')
            stored_password = parsed_data.get('hashed_password')

            # Check if the stored username matches the input username
            if stored_username == username:
                # Hash the received password using the same algorithm (e.g., SHA-256)
                hashed_password = hash_password(password)

                # Compare the hashed passwords
                if hashed_password == stored_password:
                    print("Authentication successful")
                    return True
                else:
                    print("Authentication failed: Passwords do not match")
            else:
                print("Authentication failed: Username does not match")
        else:
            print("Authentication failed: Roll number not found")
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")

    return False
#-------------------------------------------------------------------------------------------#

def publish_user_balance(roll_no, fast_coin_balance):
    try:
        # Prepare the data to publish as a JSON string
        data_to_publish = json.dumps({'fast_coin_balance': fast_coin_balance})

        # Convert the data to hexadecimal
        hex_data = binascii.hexlify(data_to_publish.encode()).decode()

        # Build the command to publish
        command = fr"D:\Multichain\multichain-cli.exe {chainname} publish user_balance
{roll_no} {hex_data}"

        # Execute the command
        result = subprocess.check_output(command, shell=True)
        print(result.decode())
        print("User balance published successfully.")
    except subprocess.CalledProcessError as e:
        print(f"Error code: {e.returncode}")
        print(f"Error message: {e.output.decode()}")
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")


#-------------------------------------------------------------------------------------------#
```

```python
def sell_product(seller_roll_no):
    try:
        # Prompt user to enter the buyer's roll number
        buyer_roll_no = input("Enter the roll number of the user to whom you want to sell: ")

        # Display the products of the seller
        display_user_products(seller_roll_no)

        # Prompt user to select the product they want to sell
        product_name = input("Enter the name of the product you want to sell: ")

        # Retrieve the selected product's information
        result = mc.liststreamkeyitems('products_record', seller_roll_no)
        if result:
            for item in result:
                item_data = binascii.unhexlify(item['data']).decode()
                parsed_data = json.loads(item_data)
                if parsed_data.get('product_name') == product_name:
                    product_price = int(parsed_data.get('product_price'
                    print(f"Product found: {product_name}, Price: {product_price} fastcoins")

                    # Prompt user if they want to proceed with the transaction
                    confirm_transaction = input("Do you want to proceed with the transaction?
(yes/no): ").lower()
                    if confirm_transaction == "yes":
                        # Get the current balance of the seller
                        seller_balance = int(get_user_balance(seller_roll_no))
                        if seller_balance is not None:
                            # Deduct the price of the product from the seller's balance
                            new_seller_balance = seller_balance + product_price
                            # Update seller's balance in the user_balance stream
                            publish_user_balance(seller_roll_no, new_seller_balance)
                        else:
                            print("Failed to retrieve seller's balance.")

                        # Get the buyer's current balance
                        buyer_balance = int(get_user_balance(buyer_roll_no))
                        if buyer_balance is not None:
                            # Calculate new balance for the buyer after deducting the product
                            new_buyer_balance = buyer_balance - product_price
                            # Update buyer's balance in the user_balance stream
                            publish_user_balance(buyer_roll_no, new_buyer_balance)
                            # Store the transaction
                            store_transaction(buyer_roll_no, seller_roll_no)
                        else:
                            print("Failed to retrieve buyer's balance.")
                    else:
                        print("Transaction canceled by the user.")
                    return
            print("Specified product not found for sale by the specified user.")
        else:
            print("No products found for sale by the specified user.")
    except Exception as e:  # Handle all exceptions
        print(f"Error occurred: {e}")

    return False
```

```
#---------------------------------------------------------------------------------------------#
def get_user_balance(roll_no):
    try:
        # Retrieve data from the user_balance stream based on the roll number
        result = mc.liststreamkeyitems('user_balance', roll_no, True)  # Set verbose
parameter to True
        if result:
            # Sort the items based on their timestamps (latest first)
            sorted_items = sorted(result, key=lambda x: x['time'], reverse=True)
            # Decode the data and parse the JSON of the latest item
            latest_item_data = binascii.unhexlify(sorted_items[0]['data']).decode()
            parsed_data = json.loads(latest_item_data)
            # Extract and return the fast_coin_balance
            fast_coin_balance = parsed_data.get('fast_coin_balance')
            return fast_coin_balance
        else:
            print("User balance not found for the specified roll number.")
            return None
    except multizain.exceptions.RPCError as e:
        print(f"MultiChain RPC Error: {e}")
        return None


#---------------------------------------------------------------------------------------------#



#---------------------------------------------------------------------------------------------#
def login():
    username = input("Enter username: ")
    roll_no = input("Enter roll number: ")
    password = getpass.getpass("Enter password: ")

    # Authenticate the user
    if authenticate_user(username, roll_no, password):
        print("Login successful. Welcome!")

        while True:
            print("Select an option:")
            print("1. Add Product")
            print("2. Display products")
            print("3. BUY product")
            print("4. Sell products")
            print("5. Display fast coins")
            print("6. Fetch products")
            print("7. Logout")

            choice = input("Enter your choice: ")

            if choice == "1":
                add_product(roll_no)
            elif choice == "2":
                display_user_products(roll_no)
            elif choice == "3":
                buy_product(roll_no)
            elif choice == "4":
                sell_product()
            elif choice == "5":
                resultt= get_user_balance(roll_no)
                print(resultt)

            elif choice == "6":
                fetch_product_data('products_record')
            elif choice == "7":
                print("logging out!")
                break
            else:
                print("Invalid choice. Please enter 1 or 2.")
    else:
        print("Authentication failed. Please check your credentials.")
#___________________________________________________________________________________________#
def main():
    while True:
        print("Select an option:")
        print("1. Register")
        print("2. Login")
        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            register()
        elif choice == "2":
            login()
        elif choice == "3":
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")

if __name__ == "__main__":
    main()
#___________________________________________________________________________________________#
```