PYTHON CODE

import tkinter as tk

from tkinter import messagebox, ttk

import hashlib

import datetime

import json

# File to persist blockchain

BLOCKCHAIN\_FILE = 'blockchain.json'

# Blockchain Components

class Block:

    def \_\_init\_\_(self, index, timestamp, description, amount, previous\_hash, hash\_value=None):

        self.index = index

        self.timestamp = timestamp

        self.description = description

        self.amount = amount

        self.previous\_hash = previous\_hash

        self.hash = hash\_value or self.calculate\_hash()

    def calculate\_hash(self):

        data\_string = f"{self.index}{self.timestamp}{self.description}{self.amount}{self.previous\_hash}"

        return hashlib.sha256(data\_string.encode()).hexdigest()

    def to\_dict(self):

        return {

            "index": self.index,

            "timestamp": self.timestamp,

            "description": self.description,

            "amount": self.amount,

            "previous\_hash": self.previous\_hash,

            "hash": self.hash

        }

    @staticmethod

    def from\_dict(data):

        return Block(

            data["index"],

            data["timestamp"],

            data["description"],

            data["amount"],

            data["previous\_hash"],

            data["hash"]

        )

class Blockchain:

    def \_\_init\_\_(self):

        self.chain = [self.create\_genesis\_block()]

    def create\_genesis\_block(self):

        return Block(0, datetime.datetime.now().isoformat(), "Genesis Block", 0.0, "0")

    def get\_latest\_block(self):

        return self.chain[-1]

    def add\_block(self, description, amount):

        latest\_block = self.get\_latest\_block()

        new\_block = Block(

            len(self.chain),

            datetime.datetime.now().isoformat(),

            description,

            amount,

            latest\_block.hash

        )

        self.chain.append(new\_block)

    def is\_chain\_valid(self):

        for i in range(1, len(self.chain)):

            current\_block = self.chain[i]

            previous\_block = self.chain[i - 1]

            if current\_block.hash != current\_block.calculate\_hash():

                return False

            if current\_block.previous\_hash != previous\_block.hash:

                return False

        return True

    def calculate\_balance(self):

        balance = 0.0

        for block in self.chain[1:]:  # Skip the genesis block

            balance += block.amount

        return balance

    def to\_dict(self):

        return [block.to\_dict() for block in self.chain]

    @staticmethod

    def from\_dict(data):

        blockchain = Blockchain()

        blockchain.chain = [Block.from\_dict(block) for block in data]

        return blockchain

# Save and Load Blockchain from File

def save\_blockchain(blockchain):

    with open(BLOCKCHAIN\_FILE, 'w') as file:

        json.dump(blockchain.to\_dict(), file, indent=4)

def load\_blockchain():

    try:

        with open(BLOCKCHAIN\_FILE, 'r') as file:

            data = json.load(file)

        return Blockchain.from\_dict(data)

    except FileNotFoundError:

        return Blockchain()

# GUI Application

def setup\_gui():

    global desc\_entry, amount\_entry, tree, balance\_label, blockchain

    blockchain = load\_blockchain()

    def add\_transaction():

        description = desc\_entry.get().strip()

        amount\_str = amount\_entry.get().strip()

        if not description:

            messagebox.showerror("Input Error", "Please enter a description.")

            return

        try:

            amount = float(amount\_str)

        except ValueError:

            messagebox.showerror("Input Error", "Please enter a valid number for amount.")

            return

        blockchain.add\_block(description, amount)

        save\_blockchain(blockchain)  # Save blockchain after adding a block

        load\_transactions()

        update\_balance()

        desc\_entry.delete(0, tk.END)

        amount\_entry.delete(0, tk.END)

        messagebox.showinfo("Success", "Transaction added successfully!")

    def load\_transactions():

        # Clear the existing rows in the Treeview

        for item in tree.get\_children():

            tree.delete(item)

        # Loop through blockchain and add non-Genesis blocks

        for block in blockchain.chain[1:]:  # Skip the Genesis Block (index 0)

            tree.insert(

                '',

                tk.END,

                values=(block.index, block.timestamp, block.description, block.amount, block.previous\_hash, block.hash)

            )

    def update\_balance():

        balance = blockchain.calculate\_balance()

        balance\_label.config(text=f"Balance: Rs. {balance:.2f}")

    def refresh\_transactions():

        global blockchain

        blockchain = load\_blockchain()  # Reload the blockchain from the file

        load\_transactions()

        update\_balance()

        messagebox.showinfo("Refresh", "Transactions reloaded successfully.")

    def reset\_transactions():

        confirm = messagebox.askyesno("Confirm Reset", "Are you sure you want to delete all transactions?")

        if confirm:

            global blockchain

            blockchain = Blockchain()  # Create a new blockchain with only the Genesis Block

            save\_blockchain(blockchain)  # Save the reset blockchain

            load\_transactions()  # Reload the GUI

            update\_balance()

            messagebox.showinfo("Reset", "All transactions have been deleted.")

    root = tk.Tk()

    root.title("Blockchain Expense and Balance Tracker")

    root.geometry("800x600")

    root.resizable(False, False)

    # Frame for adding transactions

    add\_frame = tk.Frame(root, padx=10, pady=10)

    add\_frame.pack(fill=tk.X)

    tk.Label(add\_frame, text="Description:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

    desc\_entry = tk.Entry(add\_frame, width=30)

    desc\_entry.grid(row=0, column=1, padx=5, pady=5)

    tk.Label(add\_frame, text="Amount:").grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)

    amount\_entry = tk.Entry(add\_frame, width=30)

    amount\_entry.grid(row=1, column=1, padx=5, pady=5)

    tk.Label(add\_frame, text="(Use negative for expenses)").grid(row=1, column=2, padx=5, pady=5, sticky=tk.W)

    add\_button = tk.Button(add\_frame, text="Add Transaction", command=add\_transaction)

    add\_button.grid(row=2, column=0, columnspan=3, pady=10)

    # Frame for displaying transactions

    display\_frame = tk.Frame(root, padx=10, pady=10)

    display\_frame.pack(fill=tk.BOTH, expand=True)

    columns = ('Index', 'Timestamp', 'Description', 'Amount', 'Previous Hash', 'Hash')

    tree = ttk.Treeview(display\_frame, columns=columns, show='headings')

    tree.heading('Index', text='Index')

    tree.heading('Timestamp', text='Timestamp')

    tree.heading('Description', text='Description')

    tree.heading('Amount', text='Amount')

    tree.heading('Previous Hash', text='Previous Hash')

    tree.heading('Hash', text='Hash')

    tree.column('Index', width=50, anchor=tk.CENTER)

    tree.column('Timestamp', width=150)

    tree.column('Description', width=150)

    tree.column('Amount', width=100, anchor=tk.E)

    tree.column('Previous Hash', width=200)

    tree.column('Hash', width=200)

    tree.pack(fill=tk.BOTH, expand=True)

    # Frame for balance

    balance\_frame = tk.Frame(root, padx=10, pady=10)

    balance\_frame.pack(fill=tk.X)

    balance\_label = tk.Label(balance\_frame, text="Balance: Rs. 0.00", font=("Helvetica", 14))

    balance\_label.pack()

    # Add Buttons

    button\_frame = tk.Frame(root, padx=10, pady=10)

    button\_frame.pack(fill=tk.X)

    refresh\_button = tk.Button(button\_frame, text="Refresh", command=refresh\_transactions)

    refresh\_button.pack(side=tk.LEFT, padx=5)

    reset\_button = tk.Button(button\_frame, text="Reset", command=reset\_transactions)

    reset\_button.pack(side=tk.RIGHT, padx=5)

    # Load initial transactions

    load\_transactions()

    update\_balance()

    root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

    setup\_gui()