```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify, send_file

import uuid

import hashlib

import json

import time

import os

import qrcode

from pymongo import MongoClient

from functools import wraps

from dotenv import load_dotenv

from werkzeug.security import generate_password_hash, check_password_hash

from io import BytesIO

from datetime import timedelta


# Load environment variables

load_dotenv()


# Initialize Flask app

app = Flask(__name__)

app.secret_key = os.getenv('SECRET_KEY', 'default-secret-key')

app.permanent_session_lifetime = timedelta(minutes=30)
```

```
# MongoDB setup
client = MongoClient(os.getenv('MONGODB_URI', 'mongodb://localhost:27017'))
db = client['cryptosim_db']
users = db['users']
blockchain = db['blockchain']
pending_transactions = db['pending_transactions']

DIFFICULTY = 4

# Utility functions
def hash_sha256(value):
    return hashlib.sha256(value.encode()).hexdigest()

def get_user_by_username(username):
    return users.find_one({"username": username})

def get_user_by_code(code):
    return users.find_one({"unique_code": code})

def get_last_block():
    block = blockchain.find_one(sort=[("index", -1)])
    if block:
```

```
        return block
    else:
        genesis = {
            'index': 0,
            'hash': '0' * 64,
            'previous_hash': '0' * 64,
            'nonce': 0,
            'transactions': [],
            'timestamp': time.time(),
            'mining_time': 0,
            'miner': 'system'
        }
        blockchain.insert_one(genesis)
        return genesis

def proof_of_work(index, previous_hash, transactions):
    nonce = 0
    start_time = time.time()
    tx_string = json.dumps(transactions, sort_keys=True)
    while True:
        block_string = f"{index}{previous_hash}{nonce}{tx_string}"
        block_hash = hashlib.sha256(block_string.encode()).hexdigest()
```

```
        if block_hash.startswith('0' * DIFFICULTY):
            mining_time = time.time() - start_time
            return nonce, block_hash, mining_time
        nonce += 1


def get_user_balance(user_code):
    user = get_user_by_code(user_code)
    if not user:
        return 0
    pending_txs = pending_transactions.find({
        'sender_code': user_code,
        'status': 'pending'
    })
    pending_amount = sum(tx['amount'] for tx in pending_txs)
    return user.get('balance', 0) - pending_amount


def login_required(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        if 'username' not in session:
            flash("Please login first.")
            return redirect(url_for('login'))
```

```
        return func(*args, **kwargs)
    return wrapper

# Routes
@app.route('/')
def index():
    username = session.get('username')
    user_list = list(users.find({}, {"_id": 0, "password_hash": 0, "tx_password_hash": 0}))
    users_dict = {u['username']: u for u in user_list}
    return render_template('index.html', username=username, users=users_dict)

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        email = request.form.get('email', '').strip()
        password = request.form.get('password', '')
        confirm_password = request.form.get('confirmPassword', '')
        tx_password = request.form.get('tx_password', '')

        if not all([username, email, password, confirm_password, tx_password]):
            flash("All fields are required!")
```

```
    return redirect(url_for('signup'))

if password != confirm_password:
    flash("Passwords do not match!")
    return redirect(url_for('signup'))

if get_user_by_username(username):
    flash("Username already exists!")
    return redirect(url_for('signup'))

unique_code = str(uuid.uuid4())[:8]
user_doc = {
    "username": username,
    "email": email,
    "password_hash": generate_password_hash(password),
    "tx_password_hash": generate_password_hash(tx_password),
    "unique_code": unique_code,
    "balance": 50.0,
    "created_at": time.time()
}
users.insert_one(user_doc)
flash(f"Account created! Your unique code is: {unique_code}")
```

```
        return redirect(url_for('login'))
    return render_template('signup.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        password = request.form.get('password', '')
        user = get_user_by_username(username)
        if user and check_password_hash(user['password_hash'], password):
            session.permanent = True
            session['username'] = username
            flash("Logged in successfully.")
            return redirect(url_for('index'))
        flash("Invalid username or password.")
        return redirect(url_for('login'))
    return render_template('login.html')


@app.route('/logout')
@login_required
def logout():
    session.clear()
```

```
    flash("You have been logged out.")
    return redirect(url_for('login'))


@app.route('/dashboard')
@login_required
def dashboard():
    user = get_user_by_username(session['username'])
    code = user['unique_code']
    balance = get_user_balance(code)
    sent = list(pending_transactions.find({'sender_code': code}))
    received = list(pending_transactions.find({'receiver_code': code}))
    return render_template('dashboard.html', user=user, balance=balance, sent=sent,
received=received)


@app.route('/history')
@login_required
def history():
    user = get_user_by_username(session['username'])
    code = user['unique_code']
    txs_sent = list(pending_transactions.find({'sender_code': code}))
    txs_received = list(pending_transactions.find({'receiver_code': code}))
    return render_template('history.html', sent=txs_sent, received=txs_received)
```

```
@app.route('/wallet_qr')
@login_required
def wallet_qr():
    user = get_user_by_username(session['username'])
    wallet_code = user['unique_code']
    img = qrcode.make(wallet_code)
    buf = BytesIO()
    img.save(buf, format='PNG')
    buf.seek(0)
    return send_file(buf, mimetype='image/png')


# Admin route (optional enhancement)
@app.route('/admin')
@login_required
def admin_panel():
    if session['username'] != 'admin':
        flash("Access denied.")
        return redirect(url_for('index'))
    user_list = list(users.find({}, {'_id': 0, 'password_hash': 0, 'tx_password_hash': 0}))
    return render_template('admin.html', users=user_list)
```

```
# Run the app

if __name__ == "__main__":

    app.run(debug=True)

from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify,
send_file
import uuid
import hashlib
import json
import time
import os
import qrcode
from pymongo import MongoClient
from functools import wraps
from dotenv import load_dotenv
from werkzeug.security import generate_password_hash, check_password_hash
from io import BytesIO
from datetime import timedelta

# Load environment variables
load_dotenv()

# Initialize Flask app
app = Flask(__name__)
app.secret_key = os.getenv('SECRET_KEY', 'default-secret-key')
app.permanent_session_lifetime = timedelta(minutes=30)

# MongoDB setup
client = MongoClient(os.getenv('MONGODB_URI', 'mongodb://localhost:27017'))
db = client['cryptosim_db']
users = db['users']
blockchain = db['blockchain']
pending_transactions = db['pending_transactions']

DIFFICULTY = 4

# Utility functions
def hash_sha256(value):
    return hashlib.sha256(value.encode()).hexdigest()

def get_user_by_username(username):
    return users.find_one({"username": username})
```

```
def get_user_by_code(code):
    return users.find_one({"unique_code": code})

def get_last_block():
    block = blockchain.find_one(sort=[("index", -1)])
    if block:
        return block
    else:
        genesis = {
            'index': 0,
            'hash': '0' * 64,
            'previous_hash': '0' * 64,
            'nonce': 0,
            'transactions': [],
            'timestamp': time.time(),
            'mining_time': 0,
            'miner': 'system'
        }
        blockchain.insert_one(genesis)
        return genesis

def proof_of_work(index, previous_hash, transactions):
    nonce = 0
    start_time = time.time()
    tx_string = json.dumps(transactions, sort_keys=True)
    while True:
        block_string = f"{index}{previous_hash}{nonce}{tx_string}"
        block_hash = hashlib.sha256(block_string.encode()).hexdigest()
        if block_hash.startswith('0' * DIFFICULTY):
            mining_time = time.time() - start_time
            return nonce, block_hash, mining_time
        nonce += 1

def get_user_balance(user_code):
    user = get_user_by_code(user_code)
    if not user:
        return 0
    pending_txs = pending_transactions.find({
        'sender_code': user_code,
        'status': 'pending'
    })
    pending_amount = sum(tx['amount'] for tx in pending_txs)
    return user.get('balance', 0) - pending_amount
```

```
def login_required(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        if 'username' not in session:
            flash("Please login first.")
            return redirect(url_for('login'))
        return func(*args, **kwargs)
    return wrapper

# Routes
@app.route('/')
def index():
    username = session.get('username')
    user_list = list(users.find({}, {"_id": 0, "password_hash": 0, "tx_password_hash": 0}))
    users_dict = {u['username']: u for u in user_list}
    return render_template('index.html', username=username, users=users_dict)

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        email = request.form.get('email', '').strip()
        password = request.form.get('password', '')
        confirm_password = request.form.get('confirmPassword', '')
        tx_password = request.form.get('tx_password', '')

        if not all([username, email, password, confirm_password, tx_password]):
            flash("All fields are required!")
            return redirect(url_for('signup'))

        if password != confirm_password:
            flash("Passwords do not match!")
            return redirect(url_for('signup'))

        if get_user_by_username(username):
            flash("Username already exists!")
            return redirect(url_for('signup'))

        unique_code = str(uuid.uuid4())[:8]
        user_doc = {
            "username": username,
            "email": email,
            "password_hash": generate_password_hash(password),
```

```
            "tx_password_hash": generate_password_hash(tx_password),
            "unique_code": unique_code,
            "balance": 50.0,
            "created_at": time.time()
        }
        users.insert_one(user_doc)
        flash(f"Account created! Your unique code is: {unique_code}")
        return redirect(url_for('login'))
    return render_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        password = request.form.get('password', '')
        user = get_user_by_username(username)
        if user and check_password_hash(user['password_hash'], password):
            session.permanent = True
            session['username'] = username
            flash("Logged in successfully.")
            return redirect(url_for('index'))
        flash("Invalid username or password.")
        return redirect(url_for('login'))
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    session.clear()
    flash("You have been logged out.")
    return redirect(url_for('login'))

@app.route('/dashboard')
@login_required
def dashboard():
    user = get_user_by_username(session['username'])
    code = user['unique_code']
    balance = get_user_balance(code)
    sent = list(pending_transactions.find({'sender_code': code}))
    received = list(pending_transactions.find({'receiver_code': code}))
    return render_template('dashboard.html', user=user, balance=balance, sent=sent,
received=received)

@app.route('/history')
```

```
@login_required
def history():
    user = get_user_by_username(session['username'])
    code = user['unique_code']
    txs_sent = list(pending_transactions.find({'sender_code': code}))
    txs_received = list(pending_transactions.find({'receiver_code': code}))
    return render_template('history.html', sent=txs_sent, received=txs_received)

@app.route('/wallet_qr')
@login_required
def wallet_qr():
    user = get_user_by_username(session['username'])
    wallet_code = user['unique_code']
    img = qrcode.make(wallet_code)
    buf = BytesIO()
    img.save(buf, format='PNG')
    buf.seek(0)
    return send_file(buf, mimetype='image/png')

# Admin route (optional enhancement)
@app.route('/admin')
@login_required
def admin_panel():
    if session['username'] != 'admin':
        flash("Access denied.")
        return redirect(url_for('index'))
    user_list = list(users.find({}, {'_id': 0, 'password_hash': 0, 'tx_password_hash': 0}))
    return render_template('admin.html', users=user_list)

# Run the app
if __name__ == "__main__":
    app.run(debug=True)
```