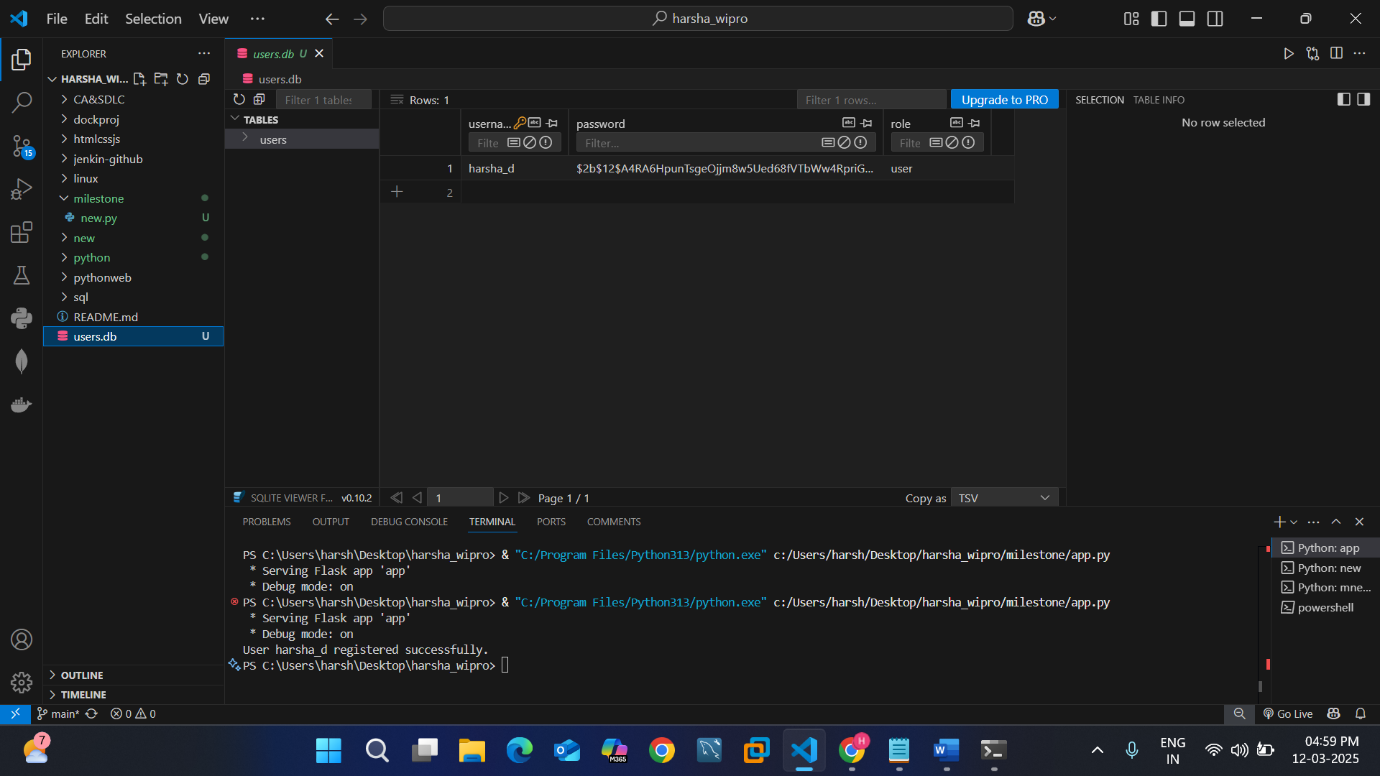
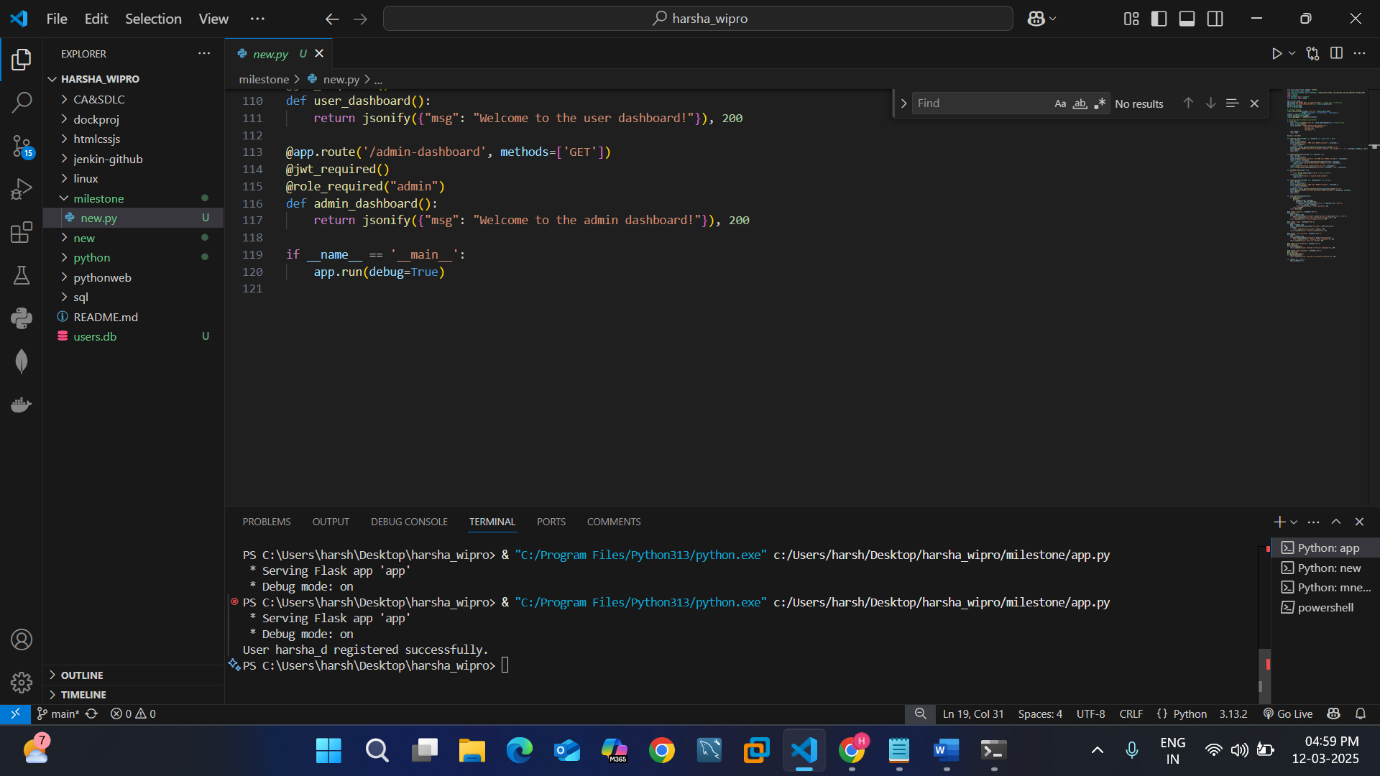
Milestone Assessment4\_CPP

Harshavardhan Devarakonda Resume Id: 29147302  
#solution  
  
from flask import Flask, request, jsonify

from flask\_bcrypt import Bcrypt

from flask\_jwt\_extended import JWTManager, create\_access\_token, jwt\_required, get\_jwt\_identity, decode\_token

import sqlite3

import logging

from datetime import timedelta

from functools import wraps

app = Flask(\_\_name\_\_)

app.config['JWT\_SECRET\_KEY'] = 'supersecretkey'  # Change this in production

app.config['JWT\_ACCESS\_TOKEN\_EXPIRES'] = timedelta(minutes=30)

bcrypt = Bcrypt(app)

jwt = JWTManager(app)

# Configure logging

logging.basicConfig(filename='auth.log', level=logging.INFO,

                    format='%(asctime)s - %(levelname)s - %(message)s')

console = logging.StreamHandler()

console.setLevel(logging.INFO)

logging.getLogger('').addHandler(console)

# Initialize SQLite database (persistent)

def init\_db():

    conn = sqlite3.connect('users.db', check\_same\_thread=False)  # Persist data

    cursor = conn.cursor()

    cursor.execute('''CREATE TABLE IF NOT EXISTS users (

                        username TEXT PRIMARY KEY,

                        password TEXT,

                        role TEXT)''')

    conn.commit()

    return conn

db\_conn = init\_db()

def register\_user(username: str, password: str, role: str) -> bool:

    conn = db\_conn

    cursor = conn.cursor()

    cursor.execute("SELECT \* FROM users WHERE username=?", (username,))

    if cursor.fetchone():

        return False

    hashed\_pw = bcrypt.generate\_password\_hash(password).decode('utf-8')

    cursor.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)", (username, hashed\_pw, role))

    conn.commit()

    return True

def authenticate\_user(username: str, password: str):

    conn = db\_conn

    cursor = conn.cursor()

    cursor.execute("SELECT password, role FROM users WHERE username=?", (username,))

    result = cursor.fetchone()

    if not result or not bcrypt.check\_password\_hash(result[0], password):

        logging.info(f"Failed authentication attempt for user: {username}")

        return False

    logging.info(f"Successful authentication for user: {username}")

    return create\_access\_token(identity={"username": username, "role": result[1]})

def validate\_token(token: str):

    try:

        return decode\_token(token)["sub"]  # Extract identity

    except Exception:

        logging.info("Invalid or expired token attempt")

        return False

def reset\_password(username: str, new\_password: str) -> bool:

    conn = db\_conn

    cursor = conn.cursor()

    cursor.execute("SELECT \* FROM users WHERE username=?", (username,))

    if not cursor.fetchone():

        return False

    hashed\_pw = bcrypt.generate\_password\_hash(new\_password).decode('utf-8')

    cursor.execute("UPDATE users SET password=? WHERE username=?", (hashed\_pw, username))

    conn.commit()

    return True

def role\_required(required\_role):

    def decorator(f):

        @wraps(f)

        def wrapper(\*args, \*\*kwargs):

            identity = get\_jwt\_identity()

            if identity and identity.get("role") in [required\_role, "admin"]:

                return f(\*args, \*\*kwargs)

            return jsonify({"msg": "Access denied"}), 403

        return wrapper

    return decorator

@app.route('/register', methods=['POST'])

def register():

    data = request.json

    if register\_user(data["username"], data["password"], data.get("role", "user")):

        return jsonify({"msg": "User registered successfully"}), 201

    return jsonify({"msg": "User already exists"}), 400

@app.route('/login', methods=['POST'])

def login():

    data = request.json

    token = authenticate\_user(data["username"], data["password"])

    if token:

        return jsonify({"access\_token": token}), 200

    return jsonify({"msg": "Invalid credentials"}), 401

@app.route('/reset-password', methods=['POST'])

def reset():

    data = request.json

    if reset\_password(data["username"], data["new\_password"]):

        return jsonify({"msg": "Password reset successful"}), 200

    return jsonify({"msg": "User not found"}), 404

@app.route('/user-dashboard', methods=['GET'])

@jwt\_required()

def user\_dashboard():

    return jsonify({"msg": "Welcome to the user dashboard!"}), 200

@app.route('/admin-dashboard', methods=['GET'])

@jwt\_required()

@role\_required("admin")

def admin\_dashboard():

    return jsonify({"msg": "Welcome to the admin dashboard!"}), 200

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)