**A blue and white sign with white text

Description automatically generated**

**Secure Software Development**

**Student number: 2410161**

**MODULE Leader: Dr. Xin Lu**

**MODULE Code: COM7033**

|  |  |
| --- | --- |
| **Assessment Number** | 1 |
| **Assessment Type (and weighting)** | Research Report (70%) |
| **Assessment Name** | COM7033 Secure Software Development |
| **Assessment Submission Date** | 29 November 2024 |

**# North Manchester Hospital - Stroke Prediction System web application.**

The dataset includes the following attributes:

• id: unique identifier

• gender: “Male”, “Female” or “other”

• age: age of the patient

• hypertension: 0 if the patient doesn’t have hypertension, 1 if the patient has

hypertension

• ever\_married: “No” or “Yes”

• work\_type: "Children", "Govt\_jov", "Never\_worked", "Private" or "Self

employed"

• Residence\_type: "Rural" or "Urban"

• avg\_glucose\_level: average glucose level in blood

• bmi: body mass index

• smoking\_status: "Formerly smoked", "Never smoked", "Smokes" or

"Unknown"\*

• stroke: 1 if the patient had a stroke or

NB: Username: admin

Password: Admin@123

from flask import UserMixin  
from typing import Literal  
from extensions import db  
from datetime import datetime  
from werkzeug.security import generate\_password\_hash, check\_password\_hash  
import hmac  
from flask import current\_app, request  
  
  
class User(UserMixin, db.Model):  
 \_\_tablename\_\_ = 'users'  
  
 id = db.Column(db.Integer, primary\_key=True)  
 username = db.Column(db.String(80), unique=True, nullable=False)  
 email = db.Column(db.String(120), unique=True, nullable=False)  
 password\_hash = db.Column(db.String(256), nullable=False)  
 role = db.Column(db.String(20), nullable=False)  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
 last\_login = db.Column(db.DateTime)  
 login\_attempts = db.Column(db.Integer, default=0)  
 locked\_until = db.Column(db.DateTime)  
 reset\_token = db.Column(db.String(100), unique=True)  
 reset\_token\_expiry = db.Column(db.DateTime)  
 active = db.Column(db.Boolean, default=True)  
 status = db.Column(db.String(20), default='active') # active, suspended, deleted  
 last\_password\_change = db.Column(db.DateTime, default=datetime.utcnow)  
 force\_password\_change = db.Column(db.Boolean, default=False)  
 security\_questions = db.Column(db.JSON)  
 last\_failed\_login = db.Column(db.DateTime)  
 ip\_address = db.Column(db.String(45)) # Store last login IP  
  
 def set\_password(self, password):  
 *"""Hash and set the user's password using stronger algorithm."""* try:  
 if not password:  
 raise ValueError("Password cannot be empty")  
  
 # Use consistent method and salt length for password hashing  
 self.password\_hash = generate\_password\_hash(  
 password,  
 method='pbkdf2:sha256',  
 salt\_length=16  
 )  
 current\_app.logger.debug(f"Password hash generated successfully for user {self.username}")  
 except Exception as e:  
 current\_app.logger.error(f"Error setting password for user {self.username}: {str(e)}")  
 raise  
  
 def check\_password(self, password):  
 *"""Verify the user's password with timing-safe comparison and enhanced security."""* try:  
 current\_app.logger.debug(f"Starting password verification for user {self.username}")  
  
 # Input validation with detailed logging  
 if not password:  
 current\_app.logger.warning(  
 f"Password verification failed for user {self.username}: "  
 "Missing password input"  
 )  
 return False  
  
 if not self.password\_hash:  
 current\_app.logger.error(  
 f"Critical security error for user {self.username}: "  
 "Missing password hash in database"  
 )  
 return False  
  
 # Verify hash format  
 if not self.password\_hash.startswith('pbkdf2:sha256:'):  
 current\_app.logger.error(  
 f"Invalid password hash format for user {self.username}: "  
 "Hash does not match expected format"  
 )  
 return False  
  
 # Log attempt details  
 current\_app.logger.debug(  
 f"Attempting password verification for user {self.username} "  
 f"(Attempt #{self.login\_attempts + 1})"  
 )  
  
 try:  
 # Use constant-time comparison for password hash  
 is\_valid = check\_password\_hash(self.password\_hash, password)  
  
 # Use hmac.compare\_digest for timing-safe comparison of the result  
 result = hmac.compare\_digest(str(is\_valid), str(True))  
  
 if result:  
 current\_app.logger.info(  
 f"Password verification successful for user {self.username} "  
 f"from IP: {request.remote\_addr if request else 'Unknown'}"  
 )  
 self.reset\_login\_attempts()  
 else:  
 current\_app.logger.warning(  
 f"Password verification failed for user {self.username}. "  
 f"Attempts: {self.login\_attempts + 1}, "  
 f"IP: {request.remote\_addr if request else 'Unknown'}"  
 )  
 self.increment\_login\_attempts()  
  
 return result  
  
 except ValueError as e:  
 current\_app.logger.error(  
 f"Password hash format error for user {self.username}: {str(e)}"  
 )  
 return False  
  
 except Exception as e:  
 current\_app.logger.error(  
 f"Unexpected error during password verification for user {self.username}: "  
 f"{str(e)}"  
 )  
 return False  
  
 def increment\_login\_attempts(self):  
 *"""Increment failed login attempts."""* try:  
 self.login\_attempts += 1  
 db.session.commit()  
 except Exception as e:  
 current\_app.logger.error(f"Error incrementing login attempts for user {self.username}: {str(e)}")  
 db.session.rollback()  
  
 def reset\_login\_attempts(self):  
 *"""Reset login attempts after successful login."""* try:  
 self.login\_attempts = 0  
 self.last\_login = datetime.utcnow()  
 db.session.commit()  
 except Exception as e:  
 current\_app.logger.error(f"Error resetting login attempts for user {self.username}: {str(e)}")  
 db.session.rollback()  
  
 def is\_account\_locked(self) -> bool:  
 *"""Check if user account is locked."""* return bool(self.locked\_until and datetime.utcnow() <= self.locked\_until)  
  
 @property  
 def is\_active(self) -> Literal[True]:  
 *"""Required by Flask-Login's UserMixin, always returns True.  
 Account status is checked separately via is\_account\_locked()."""* return True  
  
 @property  
 def account\_status(self) -> bool:  
 *"""Check actual account active status."""* return not self.is\_account\_locked() and self.active  
  
 @property  
 def is\_admin(self):  
 *"""Check if user has admin role."""* return self.role == 'admin'

**Popular Third-Party Options:**

• **pytest: A powerful and flexible testing framework with many features.**

**Password test.**

import pytest  
from utils.auth import Auth  
from datetime import datetime  
  
  
def test\_password\_change\_success(test\_db):  
 *"""Test successful password change."""* auth = Auth()  
 username = "password\_test\_user"  
 old\_password = "OldPass123!"  
 new\_password = "NewPass123!"  
  
 # Create test user  
 auth.\_execute\_query(  
 """  
 INSERT INTO users (username, email, password\_hash, role)  
 VALUES (%s, %s, %s, %s)  
 """,  
 (username, 'test@example.com', auth.hash\_password(old\_password), 'user'),  
 commit=True  
 )  
  
 # Test password change  
 success, message = auth.change\_password(username, old\_password, new\_password)  
 assert success  
 assert "success" in message.lower()  
  
 # Verify new password works  
 success, \_ = auth.login(username, new\_password, "127.0.0.1")  
 assert success  
  
  
def test\_password\_change\_invalid\_old\_password(test\_db):  
 *"""Test password change with invalid old password."""* auth = Auth()  
 username = "invalid\_pass\_user"  
 password = "TestPass123!"  
  
 # Create test user  
 auth.\_execute\_query(  
 """  
 INSERT INTO users (username, email, password\_hash, role)  
 VALUES (%s, %s, %s, %s)  
 """,  
 (username, 'test2@example.com', auth.hash\_password(password), 'user'),  
 commit=True  
 )  
  
 # Test with wrong old password  
 success, message = auth.change\_password(username, "WrongPass123!", "NewPass123!")  
 assert not success  
 assert "invalid" in message.lower()  
  
  
def test\_password\_change\_weak\_new\_password(test\_db):  
 *"""Test password change with weak new password."""* auth = Auth()  
 username = "weak\_pass\_user"  
 old\_password = "TestPass123!"  
  
 # Create test user  
 auth.\_execute\_query(  
 """  
 INSERT INTO users (username, email, password\_hash, role)  
 VALUES (%s, %s, %s, %s)  
 """,  
 (username, 'test3@example.com', auth.hash\_password(old\_password), 'user'),  
 commit=True  
 )  
  
 # Test with weak new passwords  
 weak\_passwords = [  
 "short", # Too short  
 "12345678", # Only numbers  
 "abcdefgh", # Only lowercase  
 "ABCDEFGH", # Only uppercase  
 "Pass word" # Contains space  
 ]  
  
 for weak\_password in weak\_passwords:  
 success, message = auth.change\_password(username, old\_password, weak\_password)  
 assert not success  
 assert "requirements" in message.lower()