TASK-01

SECURE USER AUTHENTICATION

Implement a user authentication system with secure login and registration functionality. Users should be able to sign up for an account, log in securely, and access protected routes only after successful authentication. Use standard mechanisms to handle password hashing, session management, and user role-based access control. Protected routes should restrict unauthorized access to sensitive functionalities.

from flask import Flask, render\_template, redirect, url\_for, request

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

# Password hashing

import bcrypt

# Database (replace with your database model)

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

# Configure database connection (replace details)

engine = create\_engine('sqlite:///users.db')

Base = declarative\_base()

# User model

class User(UserMixin, Base):

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key=True)

username = Column(String(80), unique=True, nullable=False)

password\_hash = Column(String(128), nullable=False)

email = Column(String(120), unique=True, nullable=False)

roles = Column(String(80))

def verify\_password(self, password):

return bcrypt.checkpw(password.encode(), self.password\_hash)

# Create database tables (run once)

Base.metadata.create\_all(engine)

# Session management

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def get\_db():

db = SessionLocal()

yield db

db.close()

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

app.config['SECRET\_KEY'] = 'your\_strong\_secret\_key'

login\_manager = LoginManager()

login\_manager.init\_app(app)

@login\_manager.user\_loader

def load\_user(user\_id):

with get\_db() as db:

return db.query(User).filter\_by(id=user\_id).first()

# Registration

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

def register():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

email = request.form['email']

roles = ['user'] # Default role for new users

# Validate user input (e.g., username length, email format)

# ...

# Hash password

password\_hash = bcrypt.hashpw(password.encode(), bcrypt.gensalt())

# Create user object

new\_user = User(username=username, password\_hash=password\_hash, email=email, roles=roles)

with get\_db() as db:

db.add(new\_user)

db.commit()

return redirect(url\_for('login'))

return render\_template('register.html')

# Login

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

with get\_db() as db:

user = db.query(User).filter\_by(username=username).first()

if user and user.verify\_password(password):

login\_user(user)

return redirect(url\_for('dashboard'))

else:

# Login failed message

pass

return render\_template('login.html')

# Protected route (requires login)

@app.route('/dashboard')

@login\_required

def dashboard():

# Access user information for display

user = current\_user

# This route is only accessible to logged-in users

return render\_template('dashboard.html', user=user)

# Logout

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

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

app.run(debug=True)

# TASK-02

EMPLOYMENT MANAGEMENT SYSTEM

Develop a web application that allows administrators to perform CRUD (Create, Read, Update, Delete) operations on employee records. Users should be able to add new employee view existing employee details, upda employee information, and delete employee records. Ensure the system has proper validation and authentication mechanisms to protect sensitive employee data.

from flask import Flask, render\_template, redirect, url\_for, flash, request

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField, EmailField

from wtforms.validators import DataRequired, Email

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from sqlalchemy import create\_engine, Column, Integer, String, DateTime

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

import datetime

# Database connection details (replace with yours)

engine = create\_engine('sqlite:///employees.db')

Base = declarative\_base()

# User model

class User(UserMixin, Base):

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key=True)

username = Column(String(80), unique=True, nullable=False)

password\_hash = Column(String(128), nullable=False)

email = Column(String(120), unique=True, nullable=False)

roles = Column(String(80))

def set\_password(self, password):

self.password\_hash = generate\_password\_hash(password)

def verify\_password(self, password):

return check\_password\_hash(self.password\_hash, password)

# Employee model

class Employee(Base):

\_\_tablename\_\_ = 'employees'

id = Column(Integer, primary\_key=True)

name = Column(String(80), nullable=False)

email = Column(String(120), unique=True, nullable=False)

department = Column(String(50))

position = Column(String(50))

created\_at = Column(DateTime, default=datetime.utcnow)

updated\_at = Column(DateTime, onupdate=datetime.utcnow)

# Employee form (using WTForms)

class EmployeeForm(FlaskForm):

name = StringField('Name', validators=[DataRequired()])

email = EmailField('Email', validators=[DataRequired(), Email()])

department = StringField('Department')

position = StringField('Position')

submit = SubmitField('Submit')

# Flask app setup

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

app.config['SECRET\_KEY'] = 'your\_strong\_secret\_key'

login\_manager = LoginManager()

login\_manager.init\_app(app)

@login\_manager.user\_loader

def load\_user(user\_id):

session = sessionmaker(bind=engine)()

user = session.query(User).get(user\_id)

session.close()

return user

# Create database tables (run once)

Base.metadata.create\_all(engine)

# Database session management

def get\_db():

db = sessionmaker(bind=engine)()

yield db

db.close()

# Login route

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

session = get\_db()

user = session.query(User).filter\_by(username=username).first()

session.close()

if user and user.verify\_password(password):

login\_user(user)

return redirect(url\_for('index'))

else:

flash('Invalid username or password')

return render\_template('login.html')

# Logout route

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

# Index route (displays employee list for admins, limited view for others)

@app.route('/')

@login\_required

def index():

session = get\_db()

if current\_user.roles == "admin":

employees = session.query(Employee).all()

else:

employees = session.query(Employee).filter(Employee.id != current\_user.id).all() # Hide user's own record (optional)

session.close()

return render\_template('index.html', employees=employees)

# Create employee route (admin only)

@app.route('/create', methods=['