Sri Lanka Institute of Information technology

BSc (Hons) in Information Technology – 2022

IE3042 – Secure Software Systems

Cyber Security - Year 3, Semester 2

SECURE WEB APPLICATION DEVELOPMENT

Name - H.K.M. Madutharanga

IT number- IT20657550

❖ Annotated Research Articles On Secure Web Application Development

1) "A Survey on Web Application Security" by M. Asif, A. Mehmood, and T. Umer: This research paper provides an overview of web application security, including common vulnerabilities and attack vectors.

2) "Secure Coding Standards for Web Application Development" by R. K. Jaiswal and R. K. Pandey: This research paper proposes a set of secure coding standards for web application development, including guidelines for input validation and error handling.

3) "Security Testing of Web Applications: A State-of-the-Art Review" by S. S. Ahmed and S. Fatima: This research paper reviews different security testing techniques for web applications, including penetration testing, fuzz testing, and vulnerability scanning.

4) "Secure Web Application Development: Best Practices and Guidelines" by R. K. Jaiswal and R. K. Pandey: This research paper discusses best practices and guidelines for secure web application development, including secure coding practices and secure deployment strategies.

5) "A Study of Web Application Security Standards and Guidelines" by H. Alhaqbani, M. A. Al-Turki, and H. Alhaqbani: This research paper reviews different web application security standards and guidelines, including OWASP Top 10 and SANS Top 25.

6) "Secure Web Application Development: A Systematic Literature Review" by R. K. Jaiswal and R. K. Pandey: This research paper presents a systematic literature review of secure web application development, including analysis of different methodologies and frameworks.

7) "Web Application Security Testing: A Systematic Literature Review" by L. F. Abreu, E. T. de Oliveira, and E. Cirilo: This research paper presents a systematic literature review of web application security testing, including different testing techniques and tools.

8) "Secure Web Application Development: Challenges and Solutions" by S. Qamar, A. Ali, and M. R. Khan: This research paper discusses the challenges of secure web application development, including the need for threat modeling and secure coding practices.
9) "A Survey on Web Application Security Threats and Countermeasures" by N. Kumar, R. Kumar, and M. Singh: This research paper surveys different web application security threats and countermeasures, including XSS, SQL injection, and CSRF.

10) "A Framework for Evaluating Web Application Security" by R. Ali and M. F. Qureshi: This research paper proposes a framework for evaluating web application security, including assessment of different security controls and measures.

11) "A Comparative Analysis of Web Application Security Frameworks" by S. Ali and S. Zia: This research paper presents a comparative analysis of different web application security frameworks, including their features and capabilities.

12) "Security-by-Design in Web Application Development: A Systematic Mapping Study" by M. M. Ali, M. R. Islam, and K. K. K. Chowdhury: This research paper presents a systematic mapping study of security-by-design in web application development, including analysis of different techniques and practices.

13) "A Comparative Study of Security Testing Techniques for Web Applications" by M. A. Shah and A. S. Alghamdi: This research paper presents a comparative study of different security testing techniques for web applications, including penetration testing, static analysis, and dynamic analysis.

14) "A Framework for Secure Web Application Development" by P. V. Mehta and J. K. Patel: This research paper proposes a framework for secure web application development, including guidelines for secure coding practices, vulnerability assessment, and threat modeling.

15) "Secure Web Application Development with OWASP Top 10" by N. Kumar and R. Kumar: This research paper is focused on the Open Web Application Security Project (OWASP) Top 10 list of the most critical web application security risks and provides recommendations for developers on how to address these risks.

16) "Automated Security Testing Techniques for Web Applications" by M. Anwar, A. Mahmood, and N. Baig: This research paper focuses on automated security testing techniques for web applications, including vulnerability scanning, fuzz testing, and penetration testing.

17) "A Review of Secure Coding Practices for Web Applications" by M. A. Almuhanna, A. H. Almuhanna, and A. M. Almuhanna: This research paper reviews different secure coding practices for web applications, including input validation, error handling, and authentication.

18) "Security in Web Application Development: An Analysis of Current Trends and Best Practices" by N. A. Sheikh and A. M. Sheikh: This research paper analyzes current trends and best practices in web application security, including analysis of different frameworks and methodologies.

19) "Secure Web Application Development Using DevOps Practices" by S. Singh, M. H. Khan, and A. Ali: This research paper discusses secure web application development using DevOps practices, including continuous integration, continuous delivery, and automated testing.

20) "Secure Software Development Lifecycle for Web Applications" by M. A. Khan and S. Ahmad: This research paper presents a secure software development lifecycle for web applications, including guidelines for secure coding, security testing, and deployment.

Research papers on secure web application development can provide valuable insights and ideas on how to design and implement a secure web application.

Some key takeaways that developers can consider while developing a secure web application are:

- Instead of adding security as an afterthought, include it from the beginning of the development process.
- Adhere to safe code standards, such as input validation, appropriate error handling, and strong authentication and authorization procedures.
- Conduct security testing often to locate the application's flaws and vulnerabilities.
- Make careful you thoroughly examine frameworks and libraries to make sure they don't add security flaws.
- Use encryption to safeguard sensitive data while it is in transit and at rest.
- To limit access to critical portions of the program, implement access controls and user roles.
- During every stage of the software development lifecycle, including design, development, testing, and deployment, security should be taken into account.
- Patch software and hardware components on a regular basis to fix known vulnerabilities.
- Make user education and awareness a top priority in order to stop social engineering attacks and other security issues.
- To guarantee effective and safe online application development, and promote collaboration between developers, security professionals, and other stakeholders.

By considering these key takeaways, developers may contribute to making sure web applications are secure and protected from online dangers.

❖ Explanation

The process of creating, testing, and deploying online applications with an emphasis on making sure they can withstand hacker assaults and other security risks is known as secure web application development. This calls for a number of methods and procedures, such as secure coding procedures, routine security testing, and the usage of frameworks and libraries that have undergone security vulnerability assessments.

Security-by-design is one of the fundamental tenets of developing safe online applications; it entails taking security needs into account from the beginning of the development process and including security elements into the program from the start. By doing so, it will be possible to prevent security from being tacked on as an afterthought to the program.

The creation of safe web applications should also include secure coding methods. Input validation, error handling, and authentication and authorization protocols are just a few of the strategies used in these procedures to guard against cross-site scripting (XSS) and injection attacks, two prominent security vulnerabilities.

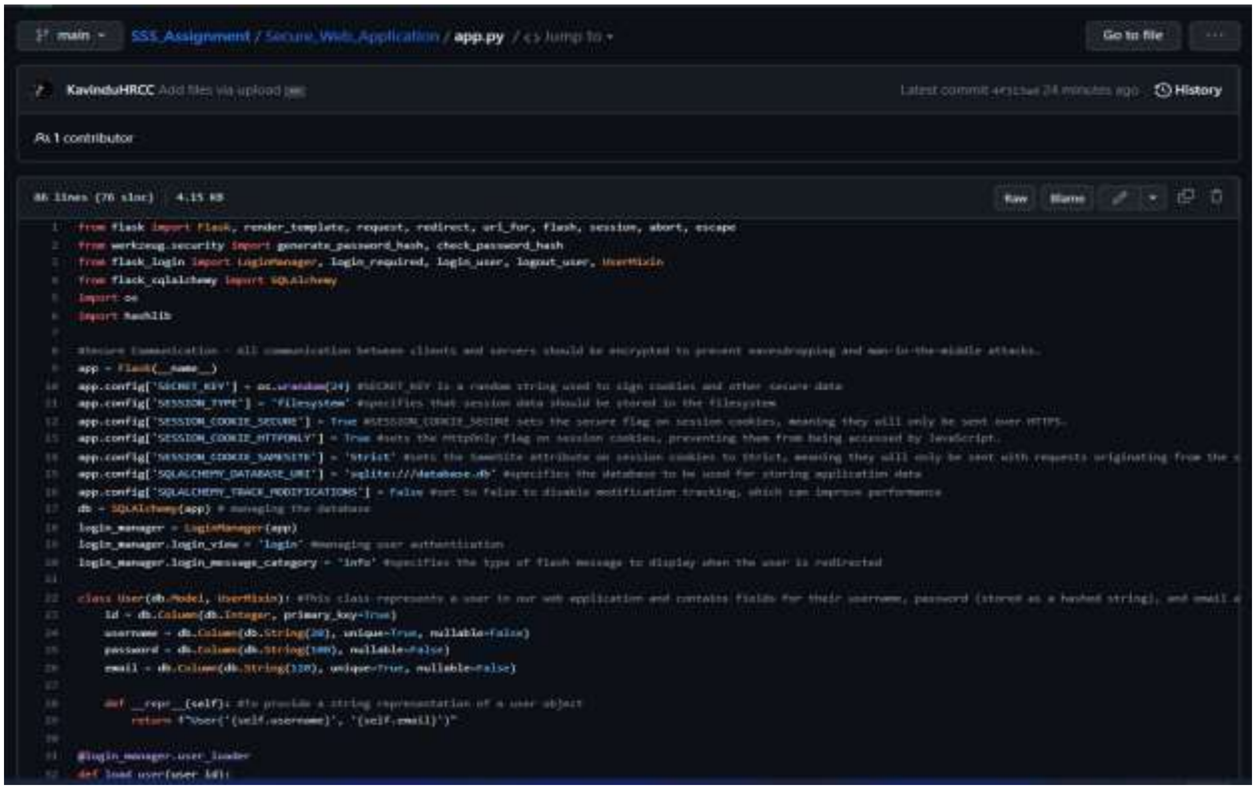
For the application to be secure, regular security testing is also necessary. Penetration testing, vulnerability scanning, and code reviews are just a few of the methods that may be used for this. Developers are able to find and fix flaws before attackers can take advantage of them by routinely testing the program for security vulnerabilities.

Data encryption, controls on access, user education and awareness, and collaboration between developers and security specialists are other crucial elements in developing safe web applications. Organizations may contribute to ensuring that their web applications are safe and secure against online threats by adding these procedures into the development process.

❖ Code Repo's

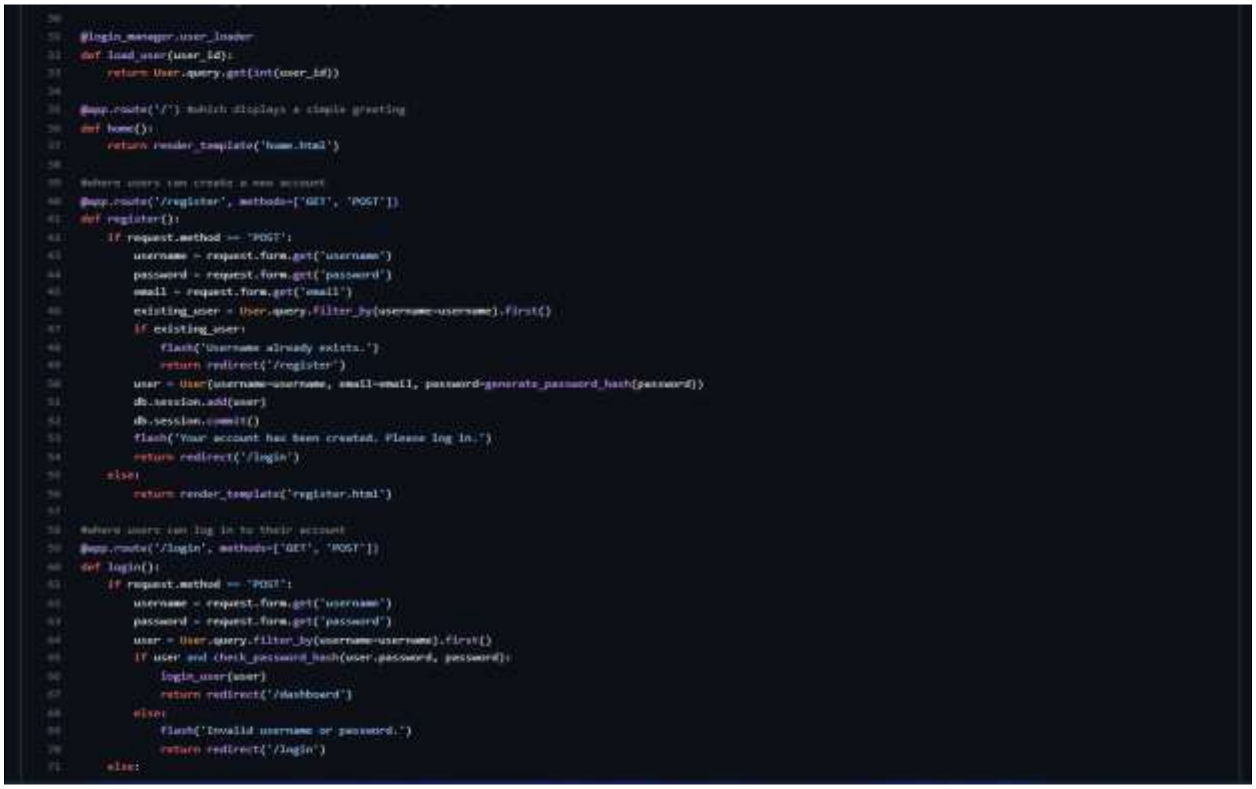
Github Link: [GitHub - KavinduHRCC/SSS_Assignment]()

App.py

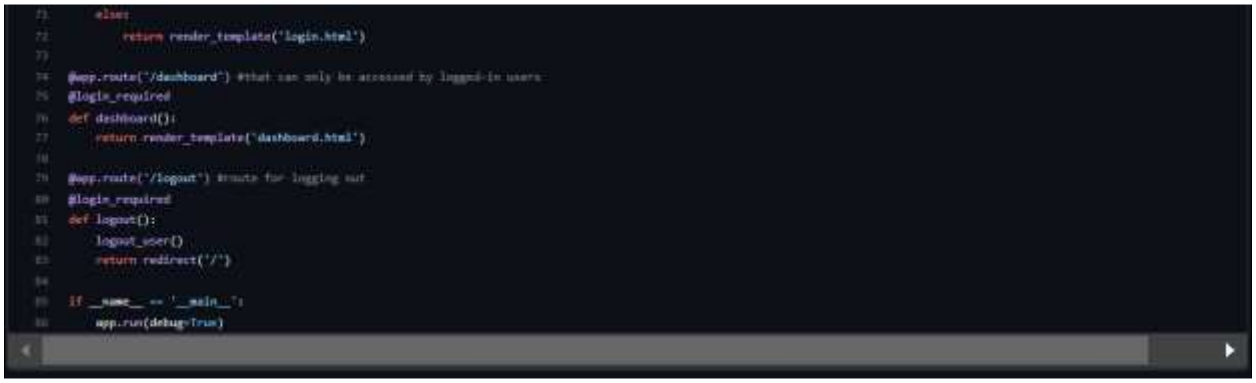


main
SSS_Assignment / Secure_Web_Application / app.py / <> Jump to
Go to file
KavinduHRCC Add files via upload
Latest commit 24 minutes ago
History
1 contributor
86 lines (76 sloc) 4.15 KB
Raw
Blame

```python
from flask import Flask, render_template, request, redirect, url_for, flash, session, abort, escape
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import LoginManager, login_required, login_user, logout_user, UserMixin
from flask_sqlalchemy import SQLAlchemy
import os
import hashlib

#Secure Communication - All communication between clients and servers should be encrypted to prevent eavesdropping and man-in-the-middle attacks.
app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24) #SECRET_KEY is a random string used to sign cookies and other secure data
app.config['SESSION_TYPE'] = 'filesystem' #specifies that session data should be stored in the filesystem
app.config['SESSION_COOKIE_SECURE'] = True #SESSION_COOKIE_SECURE sets the secure flag on session cookies, meaning they will only be sent over HTTPS.
app.config['SESSION_COOKIE_HTTPONLY'] = True #sets the HttpOnly flag on session cookies, preventing them from being accessed by JavaScript.
app.config['SESSION_COOKIE_SAMESITE'] = 'Strict' #sets the SameSite attribute on session cookies to Strict, meaning they will only be sent with requests originating from the s
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db' #specifies the database to be used for storing application data
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False #set to False to disable modification tracking, which can improve performance
db = SQLAlchemy(app) # managing the database
login_manager = LoginManager(app)
login_manager.login_view = 'login' #managing user authentication
login_manager.login_message_category = 'info' #specifies the type of flash message to display when the user is redirected

class User(db.Model, UserMixin): #This class represents a user in our web application and contains fields for their username, password (stored as a hashed string), and email a
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self): #to provide a string representation of a user object
        return f"User('{self.username}', '{self.email}')"

@login_manager.user_loader
def load_user(user_id):
```

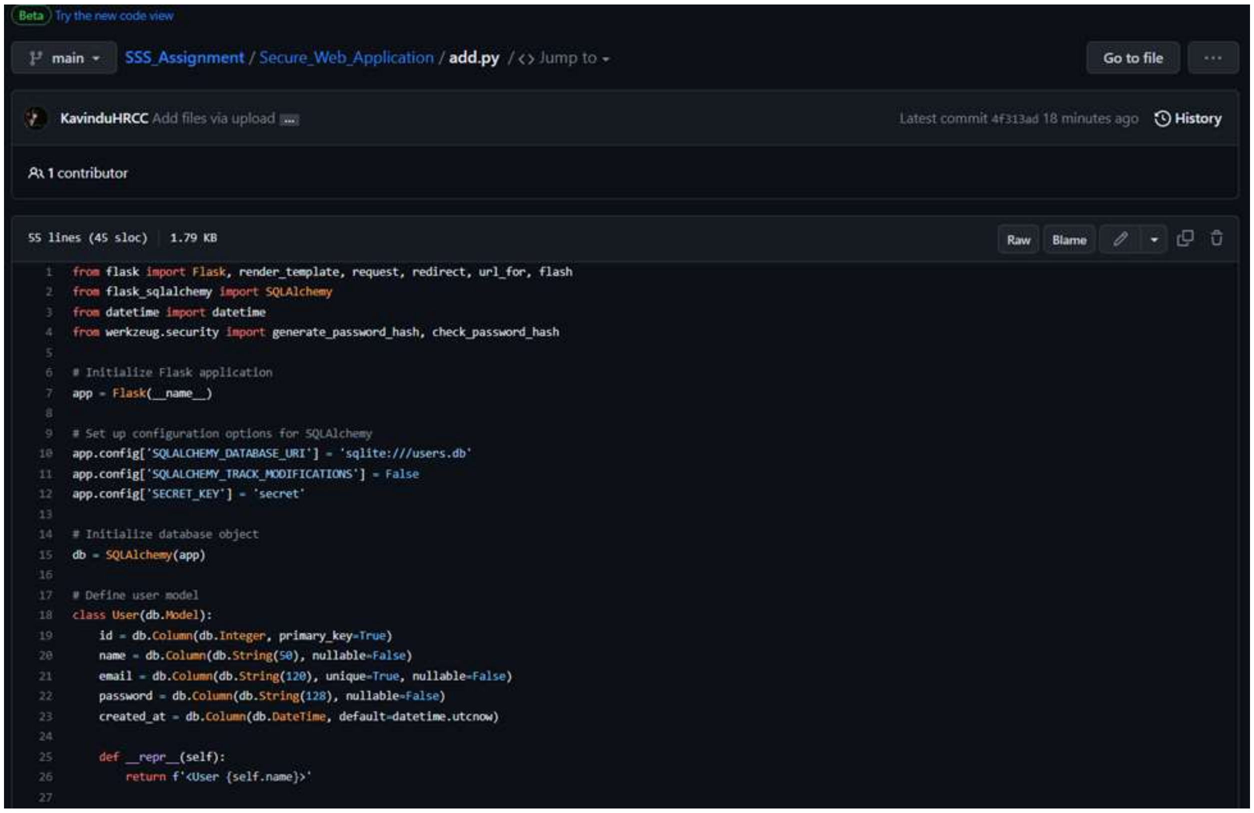


```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/') #which displays a simple greeting
def home():
    return render_template('home.html')

#where users can create a new account
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        email = request.form.get('email')
        existing_user = User.query.filter_by(username=username).first()
        if existing_user:
            flash('Username already exists.')
            return redirect('/register')
        user = User(username=username, email=email, password=generate_password_hash(password))
        db.session.add(user)
        db.session.commit()
        flash('Your account has been created. Please log in.')
        return redirect('/login')
    else:
        return render_template('register.html')

#where users can log in to their account
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        user = User.query.filter_by(username=username).first()
        if user and check_password_hash(user.password, password):
            login_user(user)
            return redirect('/dashboard')
        else:
            flash('Invalid username or password.')
            return redirect('/login')
    else:
```

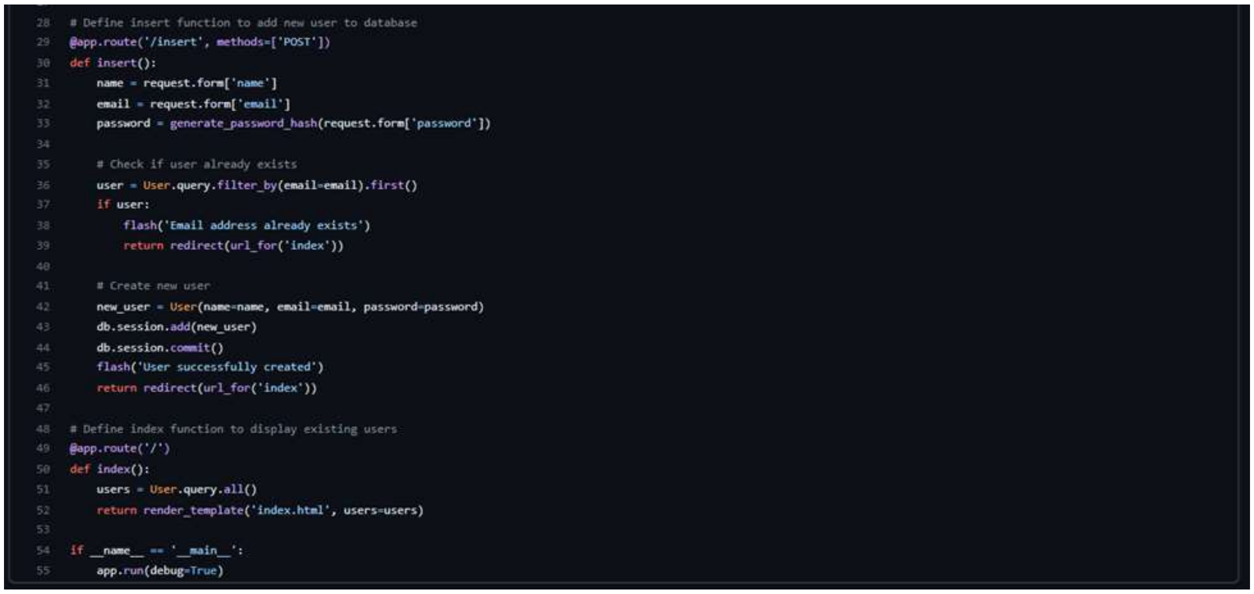


```
    else:
        return render_template('login.html')

@app.route('/dashboard') #that can only be accessed by logged-in users
@login_required
def dashboard():
    return render_template('dashboard.html')

@app.route('/logout') #route for logging out
@login_required
def logout():
    logout_user()
    return redirect('/')

if __name__ == '__main__':
    app.run(debug=True)
```

Add.py

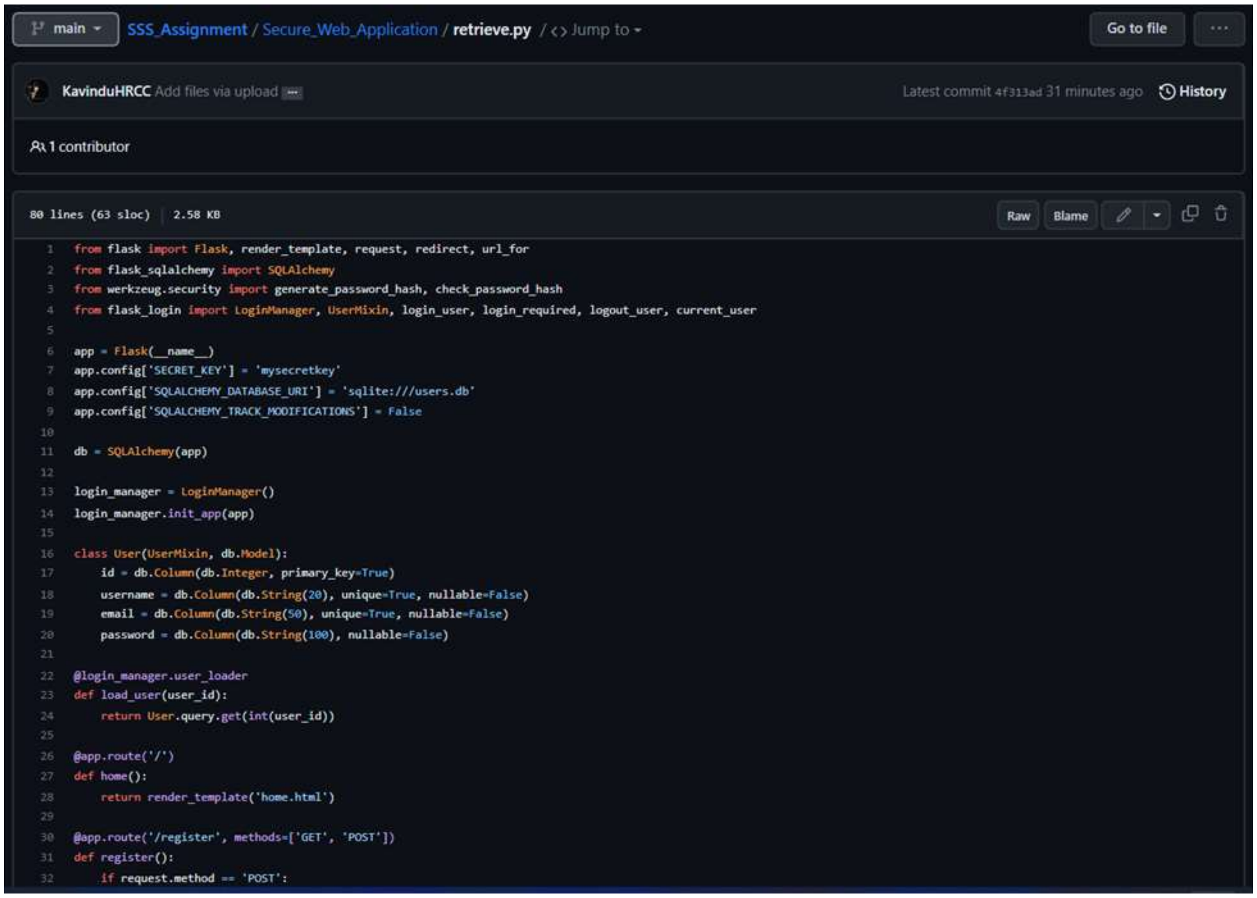

Beta Try the new code view
main
SSS_Assignment / Secure_Web_Application / add.py / <> Jump to
Go to file
KavinduHRCC Add files via upload
Latest commit 4f313ad 18 minutes ago
History
1 contributor
55 lines (45 sloc) | 1.79 KB
Raw
Blame

```python
from flask import Flask, render_template, request, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from werkzeug.security import generate_password_hash, check_password_hash

# Initialize Flask application
app = Flask(__name__)

# Set up configuration options for SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'secret'

# Initialize database object
db = SQLAlchemy(app)

# Define user model
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(128), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    def __repr__(self):
        return f'<User {self.name}>'

```



```python
# Define insert function to add new user to database
@app.route('/insert', methods=['POST'])
def insert():
    name = request.form['name']
    email = request.form['email']
    password = generate_password_hash(request.form['password'])

    # Check if user already exists
    user = User.query.filter_by(email=email).first()
    if user:
        flash('Email address already exists')
        return redirect(url_for('index'))

    # Create new user
    new_user = User(name=name, email=email, password=password)
    db.session.add(new_user)
    db.session.commit()
    flash('User successfully created')
    return redirect(url_for('index'))

# Define index function to display existing users
@app.route('/')
def index():
    users = User.query.all()
    return render_template('index.html', users=users)

if __name__ == '__main__':
    app.run(debug=True)
```

Retrieve.py



main / SSS_Assignment / Secure_Web_Application / retrieve.py / <> Jump to

Go to file

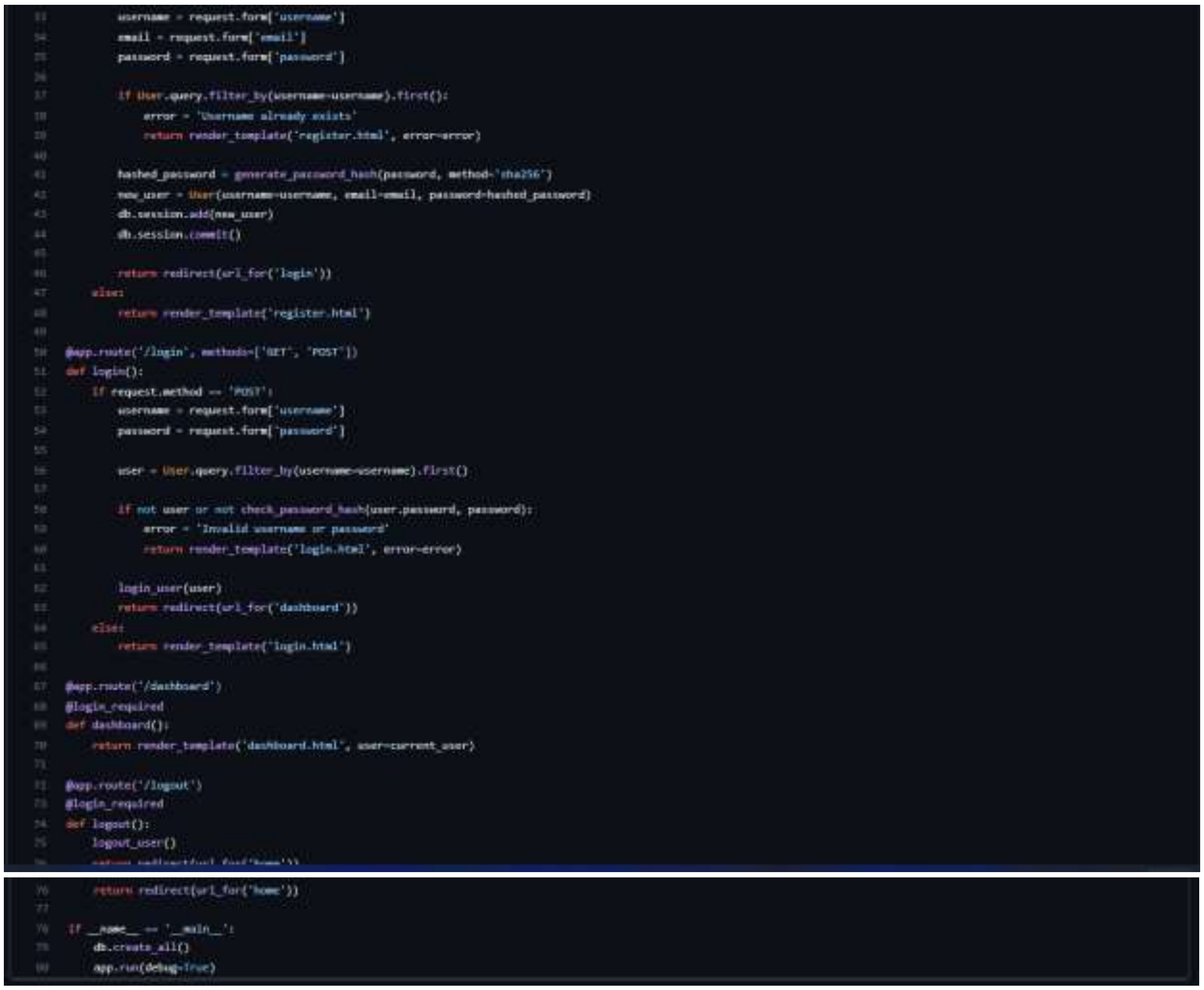
KavinduHRCC Add files via upload ... Latest commit 4f313ad 31 minutes ago History

1 contributor

80 lines (63 sloc) | 2.58 KB

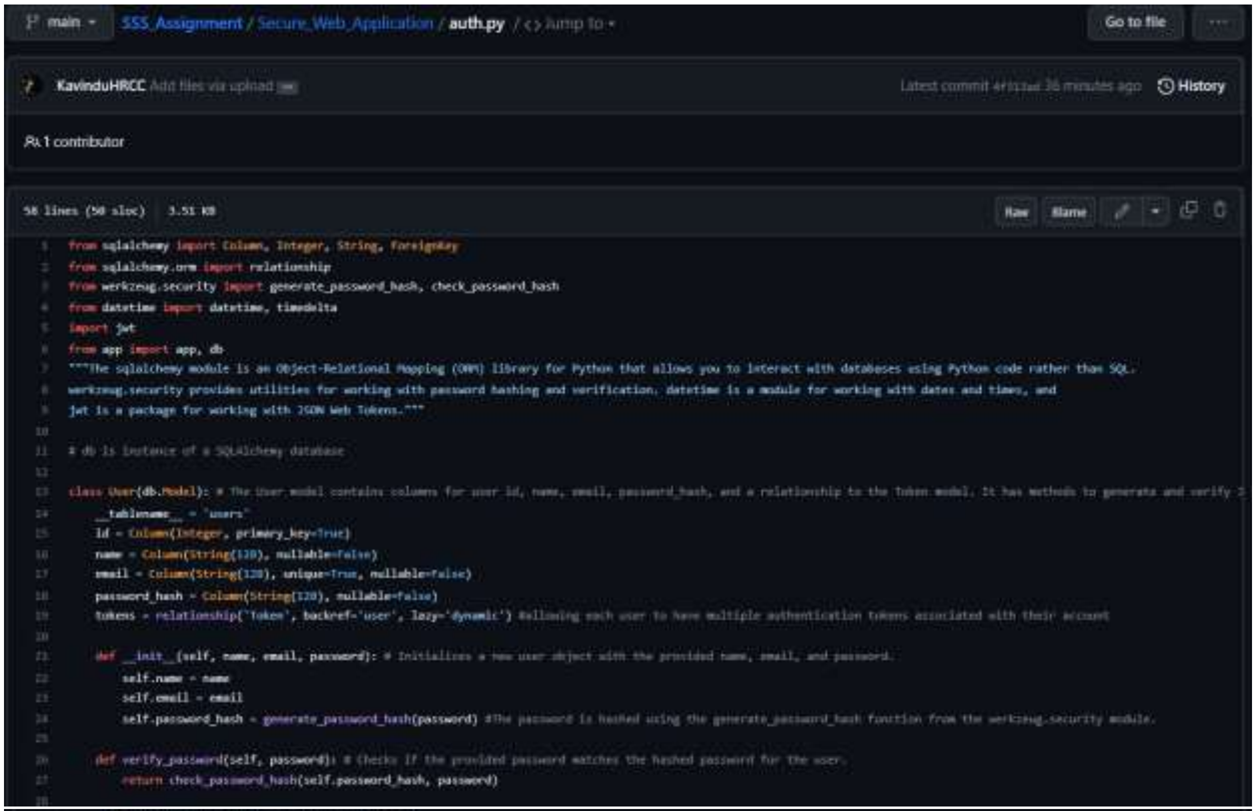
Raw Blame

```python
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user

app = Flask(__name__)
app.config['SECRET_KEY'] = 'mysecretkey'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

login_manager = LoginManager()
login_manager.init_app(app)

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
```



```python
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']

        if User.query.filter_by(username=username).first():
            error = 'Username already exists'
            return render_template('register.html', error=error)

        hashed_password = generate_password_hash(password, method='sha256')
        new_user = User(username=username, email=email, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()

        return redirect(url_for('login'))
    else:
        return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = User.query.filter_by(username=username).first()

        if not user or not check_password_hash(user.password, password):
            error = 'Invalid username or password'
            return render_template('login.html', error=error)

        login_user(user)
        return redirect(url_for('dashboard'))
    else:
        return render_template('login.html')

@app.route('/dashboard')
@login_required
def dashboard():
    return render_template('dashboard.html', user=current_user)

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('home'))

if __name__ == '__main__':
    db.create_all()
    app.run(debug=True)
```

Auth.py



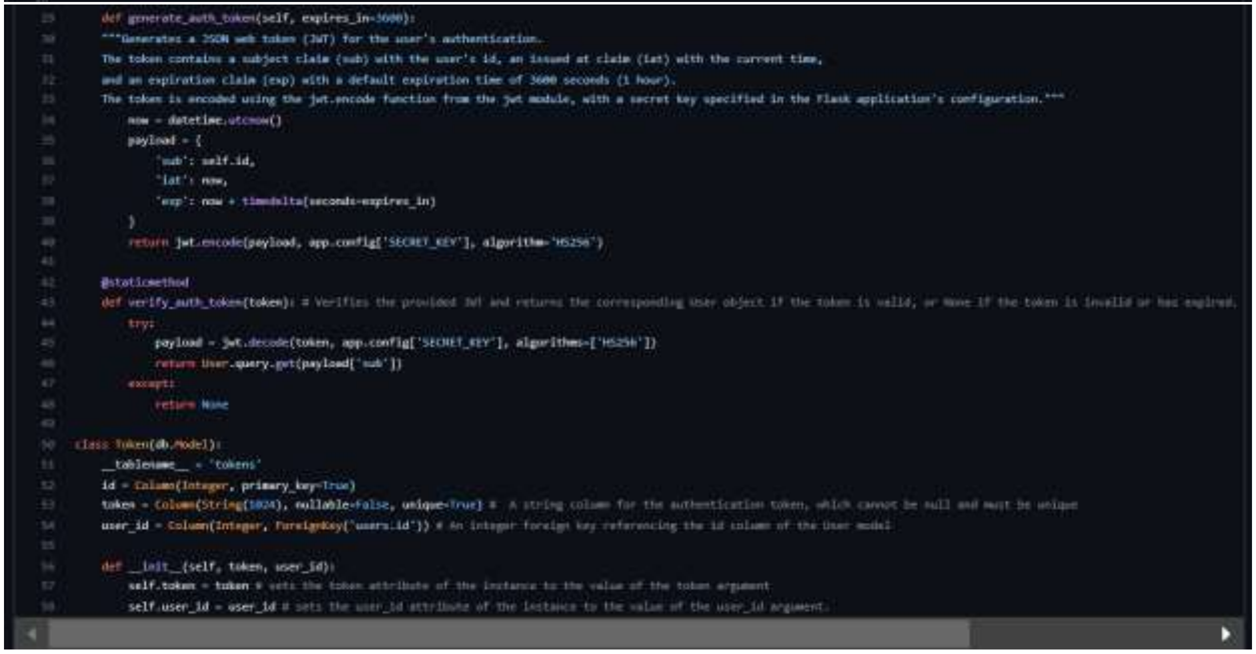
main · SSS_Assignment / Secure_Web_Application / auth.py · Go to file

KavinduHRCC Add files via upload · Latest commit 4f1c1ad 36 minutes ago · History

1 contributor

58 lines (58 sloc) 3.51 KB · Raw · Blame

```python
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime, timedelta
import jwt
from app import app, db
"""The sqlalchemy module is an Object-Relational Mapping (ORM) library for Python that allows you to interact with databases using Python code rather than SQL.
werkzeug.security provides utilities for working with password hashing and verification. datetime is a module for working with dates and times, and
jwt is a package for working with JSON Web Tokens."""

# db is instance of a SQLAlchemy database

class User(db.Model): # The User model contains columns for user id, name, email, password_hash, and a relationship to the Token model. It has methods to generate and verify
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(120), nullable=False)
    email = Column(String(120), unique=True, nullable=False)
    password_hash = Column(String(128), nullable=False)
    tokens = relationship('Token', backref='user', lazy='dynamic') #allowing each user to have multiple authentication tokens associated with their account

    def __init__(self, name, email, password): # Initializes a new user object with the provided name, email, and password.
        self.name = name
        self.email = email
        self.password_hash = generate_password_hash(password) #The password is hashed using the generate_password_hash function from the werkzeug.security module.

    def verify_password(self, password): # Checks if the provided password matches the hashed password for the user.
        return check_password_hash(self.password_hash, password)

```



```python
    def generate_auth_token(self, expires_in=3600):
    """Generates a JSON web token (JWT) for the user's authentication.
    The token contains a subject claim (sub) with the user's id, an issued at claim (iat) with the current time,
    and an expiration claim (exp) with a default expiration time of 3600 seconds (1 hour).
    The token is encoded using the jwt.encode function from the jwt module, with a secret key specified in the Flask application's configuration."""
        now = datetime.utcnow()
        payload = {
            'sub': self.id,
            'iat': now,
            'exp': now + timedelta(seconds=expires_in)
        }
        return jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')

    @staticmethod
    def verify_auth_token(token): # Verifies the provided JWT and returns the corresponding User object if the token is valid, or None if the token is invalid or has expired.
        try:
            payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
            return User.query.get(payload['sub'])
        except:
            return None

class Token(db.Model):
    __tablename__ = 'tokens'
    id = Column(Integer, primary_key=True)
    token = Column(String(1024), nullable=False, unique=True) #  A string column for the authentication token, which cannot be null and must be unique
    user_id = Column(Integer, ForeignKey('users.id')) # An integer foreign key referencing the id column of the User model

    def __init__(self, token, user_id):
        self.token = token # sets the token attribute of the instance to the value of the token argument
        self.user_id = user_id # sets the user_id attribute of the instance to the value of the user_id argument.
```