LU06i - Authentifizierung mit Flask-Login

Flask-Login ist eine Erweiterung für Flask, die Benutzersitzungen handhabt und es einfach macht, Funktionen wie `Login`, `Logout` und `Remember Me` zu implementieren. Flask-Login verwendet die Session-Funktionalität von Flask, um den angemeldeten Benutzer zu speichern.

Installation

Zuerst müssen wir Flask-Login installieren, dazu passen wir das requirements.txt an:

```
...
Flask-Login==0.6.3
```

Grundlegende Einrichtung

Nach der Installation müssen wir Flask-Login in unserer Anwendung initialisieren:

```
from flask_login import LoginManager

login_manager = LoginManager()
login_manager.init_app(app)
```

Benutzerklasse

Die Benutzerklasse muss einige spezielle Methoden erfüllen, die von Flask-Login verwendet werden: `get_id()`, `is_authenticated`, `is_active` und `is_anonymous`. Im Allgemeinen kann dies durch die Erweiterung der Benutzerklasse mit der `UserMixin`-Klasse von Flask-Login erreicht werden:

```
from flask_login import UserMixin

class User(UserMixin):
    def __init__(self, user_id, username, email, password):
        self.id = user_id
        self.username = username
        self.email = email
        self.password = password
```

Benutzer-DAO-Klasse

Um den User in der Datenbank zu speichern, müssen wir eine entsprechende DAO-Klasse erstellen:

```
import sqlite3

class UserDao:
```

```
    def __init__(self, db_file):
        self.conn = sqlite3.connect(db_file, check_same_thread=False)
        self.cursor = self.conn.cursor()

    def create_user_table(self):
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER
PRIMARY KEY, username TEXT, email TEXT, password TEXT)''')
        self.conn.commit()

    def add_user(self, user):
        self.cursor.execute("INSERT INTO users (username, email, password)
VALUES (?, ?, ?)", (user.username, user.email, user.password))
        self.conn.commit()

    def get_user_by_id(self, user_id):
        self.cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
        row = self.cursor.fetchone()
        if row:
            return User(row[0], row[1], row[2], row[3])
        return None

    def get_user_by_username(self, username):
        self.cursor.execute("SELECT * FROM users WHERE username = ?",
(username,))
        row = self.cursor.fetchone()
        if row:
            return User(row[0], row[1], row[2], row[3])
        return None

    def delete_user(self, user_id):
        self.cursor.execute("DELETE FROM users WHERE id = ?", (user_id,))
        if self.cursor.rowcount > 0:
            self.conn.commit()
            return True
        return False

    def close(self):
        self.conn.close()
```

Diese DAO-Klasse bietet Methoden zum Erstellen der User-Tabelle, zum Hinzufügen von Benutzern und zum Abrufen von Benutzern anhand ihrer ID oder ihres Benutzernamens.

Benutzer-Loader und DAO-Integration

Die `user_loader`-Funktion ist ein wesentlicher Bestandteil beim Einsatz von Flask-Login. Diese Funktion ist verantwortlich für das Laden eines Benutzerobjekts basierend auf der Benutzer-ID, die in der Session gespeichert ist. Wenn ein Benutzer sich einloggt, wird seine ID in der Flask-Session gespeichert. Bei nachfolgenden Anfragen wird diese ID verwendet, um den zugehörigen Benutzer zu laden und in das `current_user`-Objekt zu speichern. Dadurch weiß die Anwendung während der gesamten Lebensdauer der Session, wer der aktuelle Benutzer ist.

Der `@login_manager.user_loader`-Dekorator markiert die Funktion, die für das Laden des Benutzers zuständig ist. Diese Funktion erhält die Benutzer-ID als Argument, führt die notwendigen Schritte zur Benutzeridentifizierung durch und gibt das entsprechende Benutzerobjekt zurück. In unserem Fall verwendet die `user_loader`-Funktion die `get_user_by_id`-Methode aus der `UserDao`-Klasse.

```
@login_manager.user_loader
def load_user(user_id):
    return user_dao.get_user_by_id(int(user_id))
```

Das Definieren einer `user_loader`-Funktion ist unerlässlich für das Funktionieren der Flask-Login-Erweiterung, da sie nicht selbstständig weiß, wie Benutzerobjekte aus der Session-ID geladen werden sollen.

Login und DAO-Integration

Die Login-Funktion wird die `get_user_by_username`-Methode verwenden, um den User zu finden:

```
from flask_login import login_required, login_user, logout_user

app = Flask(__name__)
app.secret_key = 'supersecretkey' # Add a Secret-Key to the Flask App


@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    user = user_dao.get_user_by_username(data['username'])
    if user and user.password == data['password']:
        login_user(user)
        return jsonify({'success': True}), 200
    return jsonify({'error': 'Invalid username or password'}), 401

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return jsonify({'success': True}), 200
```

Schutz von Endpunkten

Um Endpunkte zu schützen, können wir den `@login_required`-Dekorator verwenden:

```
@app.route('/shoppingitems', methods=['GET'])
@login_required
def get_shopping_items():
    # Implementierung
```

M323-LU06