Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по Домашнему Заданию “Гостевая книга”

Выполнил:
студент группы ИУ5-35Б:
Купцов С.Р.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.В.
Подпись и дата:

Москва, 2024 г.

Задание

Ваша задача - написать приложение-гостевую книгу, позволяющее:

- добавлять сообщения (посты) и комментарии к ним;
- удалять посты и комментарии;
- редактировать посты и комментарии.

Кроме того, ваше приложение обязательно должно сохранять информацию в базе данных.

Добавьте в гостевую книгу возможность регистрации и аутентификации пользователей.

В версии вашей гостевой должны быть следующие возможности:

1) Форма регистрации нового пользователя. При регистрации обязательно должны указываться логин и пароль, по желанию -- дополнительные поля. Должна быть предусмотрена проверка, не занят ли уже такой логин.
2) Форма для входа (аутентификации) пользователя. В форме должны быть предусмотрены поля с логином, паролем и флажок "запомнить меня" (см. пример с урока). После того, как пользователь успешно ввёл логин и пароль, должна появляться возможность выхода (logout).
3) Добавление новых постов и комментариев должно быть возможно только для пользователей, прошедших аутентификацию (анонимные комментарии и посты не допускаем).
4) Пользователю доступно удаление и редактирование только своих постов и комментариев.
5) Для каждого поста и комментария должен быть указан автор.
6) Пароль должен храниться в базе данных в виде хэша
7) Использовать технологию ORM и библиотеку SqlAlchemy

Код программы

db_sessioan.py

```
import sqlalchemy as sa
import sqlalchemy.orm as orm
from sqlalchemy.orm import Session
import sqlalchemy.ext.declarative as dec

SqlAlchemyBase = dec.declarative_base()

session_factory = None


def global_init(db_file):
    global session_factory, sa, orm

    if session_factory:
        return
```

```
    conn_str = f'sqlite:///{db_file.strip()}?check_same_thread=False'
    print("Подключение к БД")

    engine = sa.create_engine(conn_str, echo=False)
    session_factory = orm.sessionmaker(bind=engine)

    from .Models import Users, Posts, Comments

    SqlAlchemyBase.metadata.create_all(engine)


def create_session() -> Session:
    global session_factory
    return session_factory()
```

Models.py

```
import sqlalchemy as sa
import sqlalchemy.orm as orm
from flask_login import UserMixin
from .db_session import SqlAlchemyBase


class Users(SqlAlchemyBase, UserMixin):
    __tablename__ = 'Users'

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True,
nullable=False)
    login = sa.Column(sa.String, nullable=False)
    password = sa.Column(sa.String, nullable=False)

    posts = orm.relationship('Posts', cascade="all, delete")
    comments = orm.relationship('Comments', cascade="all, delete")

    def get_id(self):
        return int(self.id)


class Posts(SqlAlchemyBase):
    __tablename__ = 'posts'

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True,
nullable=False)
    title = sa.Column(sa.String, nullable=False)
    content = sa.Column(sa.String, nullable=False)
    time = sa.Column(sa.String, nullable=False)
    user = sa.Column(sa.Integer, sa.ForeignKey('Users.id',
ondelete="CASCADE"), nullable=False)

    def get_id(self):
        return int(self.id)


class Comments(SqlAlchemyBase):
    __tablename__ = 'comments'

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True,
nullable=False)
    post = sa.Column(sa.Integer, sa.ForeignKey('posts.id',
ondelete='CASCADE'), nullable=False)
    content = sa.Column(sa.String, nullable=False)
    time = sa.Column(sa.String, nullable=False)
    user = sa.Column(sa.Integer, sa.ForeignKey('Users.id',
```

```
ondelete="CASCADE"), nullable=False)

    def get_id(self):
        return int(self.id)
```

LoginForm.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired


class LoginForm(FlaskForm):
    login = StringField("Логин", validators=[DataRequired()])
    password = PasswordField("Пароль", validators=[DataRequired()])
    remember_me = BooleanField("Запомнить меня")
    submit_button = SubmitField("Войти")


class RegisterForm(FlaskForm):
    login = StringField("Логин", validators=[DataRequired()])
    password = PasswordField("Пароль", validators=[DataRequired()])
    password2 = PasswordField("Повторите пароль",
validators=[DataRequired()])
    submit_button = SubmitField("Войти")
```

Blog.py

```
from flask import Flask, request, render_template, redirect
from datetime import datetime
from pymorphy2 import MorphAnalyzer
from flask_login import LoginManager, login_user, login_required,
logout_user, current_user
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.exceptions import HTTPException
from forms.LoginForm import LoginForm, RegisterForm
from random import randint
from os import path
from data import db_session
from data import Models


app = Flask(__name__)
app.config["SECRET_KEY"] = ''.join([chr(randint(20, 128)) for _ in
range(100)])
login_manager = LoginManager()
login_manager.init_app(app)
users = []
months = {1: 'января', 2: 'февраля', 3: 'марта', 4: 'апреля', 5: 'мая', 6:
'июня', 7: 'июля', 8: 'августа', 9: 'сентября', 10: 'октября', 11: 'ноября',
12: 'декабря'}

BASE_DIR = path.abspath(path.dirname(__file__))
db_path = BASE_DIR + "/db/Article.db"
db_session.global_init(db_path)
session = db_session.create_session()


def check(data: Models, bool):
    temp = session.query(data).filter(bool)
    if not temp.all():
```

```
        return None
    return temp.one()


@login_manager.user_loader
def load_user(user_id):
    user = session.query(Models.Users).filter(Models.Users.id ==
user_id).one()
    if user:
        return user
    return None


@app.route("/", methods=["GET"])
def index():
    posts = session.query(Models.Posts).all()
    return render_template("index.html", posts=posts, text_com=f"{len(posts)}
{MorphAnalyzer().parse('Запись')[0].make_agree_with_number(len(posts)).word}:
")


@app.route("/login", methods=["GET", "POST"])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        login = form.login.data
        password = form.password.data
        remember_me = form.remember_me.data
        user = session.query(Models.Users).filter(Models.Users.login ==
login).one()
        if user and check_password_hash(user.password, password):
            users.append(user)
            login_user(user, remember=remember_me)
            return redirect("/")
    return render_template("login.html", form=form)


@app.route("/register", methods=["GET", "POST"])
def register():
    form = RegisterForm()
    if form.validate_on_submit():
        login = form.login.data
        password = form.password.data
        password2 = form.password2.data
        errors = []
        if password != password2:
            errors.append('Пароли не совпадают!')
        elif len(password) < 8:
            errors.append('Пароль слишком короткий!')
        elif password.upper() == password or password.lower() == password or
password.isdigit():
            errors.append('Пароль должен содержать цифры, заглавные и
строчный буквы!')
        elif session.query(Models.Users).filter(Models.Users.login ==
login).all():
            errors.append('Пользователь уже существует!')
        else:
            user = Models.Users(login=login,
password=generate_password_hash(password))
            session.add(user)
            session.commit()
            login_user(user)
            return redirect("/")
        return render_template("register.html", form=form, errors=errors)
```

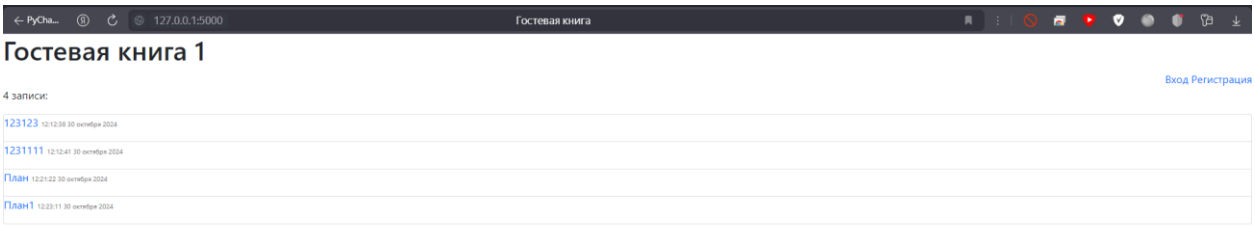
```
    return render_template("register.html", form=form)


@app.route('/logout')
@login_required
def logout():
    id_ = current_user.get_id()
    for user in range(len(users)):
        if users[user].get_id() == id_:
            del users[user]
            break
    logout_user()
    return redirect("/")


@app.route("/add_comment/<int:id_p>", methods=["POST"])
@login_required
def add_comment(id_p):
    if request.form['content'] == '':
        return redirect(f'/post/{id_p}')
    comment = Models.Comments(post=id_p, content=request.form['content'],
time=f'{datetime.now().time().hour}:{datetime.now().time().minute}:{datetime.
now().time().second} {datetime.now().day} {months[datetime.now().month]}
{datetime.now().year}', user=current_user.get_id())
    session.add(comment)
    session.commit()
    return redirect(f'/post/{id_p}')


@app.route("/add_post/", methods=["POST"])
@login_required
def add_post():
    if request.form['content'] == '' or request.form['title'] == '':
        return redirect('/')
    post = Models.Posts(title=request.form['title'],
content=request.form['content'],
time=f'{datetime.now().time().hour}:{datetime.now().time().minute}:{datetime.
now().time().second} {datetime.now().day} {months[datetime.now().month]}
{datetime.now().year}', user=current_user.get_id())
    session.add(post)
    session.commit()
    return redirect('/')


@app.route("/post/<int:id_p>", methods=["GET"])
def get_post(id_p):
    post = check(Models.Posts, Models.Posts.id == id_p)
    if post is None:
        return redirect('/get_error/404')
    comments = session.query(Models.Comments).filter(Models.Comments.post ==
post.id).all()
    comments.sort(key=lambda x: x.time.split()[0].split(':')[0] * 3600 +
x.time.split()[0].split(':')[1] * 60 + x.time.split()[0].split(':')[2],
reverse=True)
    if len(comments) == 0:
        text_com = "Комментариев нет"
    else:
        text_com = str(len(comments)) + " " +
MorphAnalyzer().parse('Комментарий')[0].make_agree_with_number(len(comments))
.word + ":"
    return render_template('post.html', post=post, comments=comments,
text_com=f"{text_com}")
```

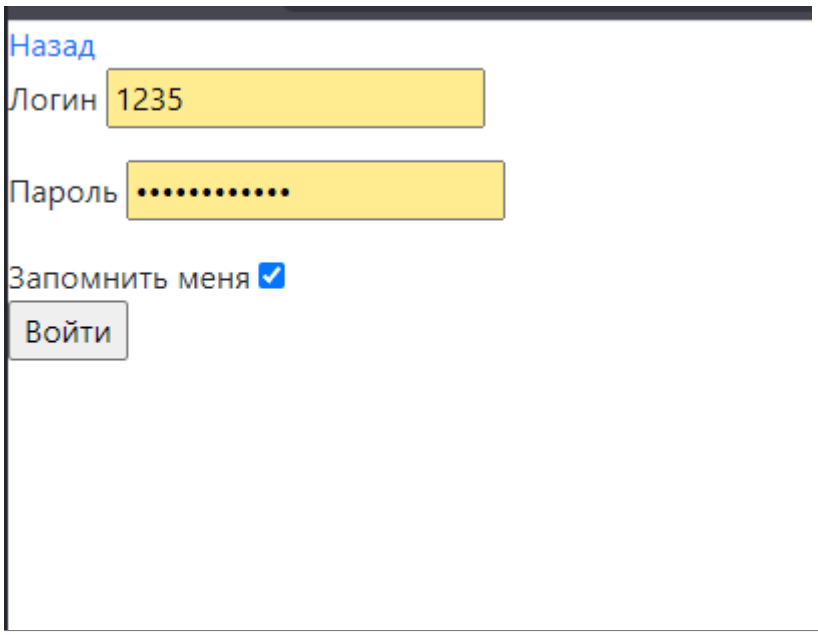
```python
@app.route("/delete_post/<int:id_p>", methods=["GET"])
@login_required
def delete_post(id_p):
    post = check(Models.Posts, Models.Posts.id == id_p)
    if post is None:
        return redirect('/get_error/404')
    if post.user == current_user.get_id():
        comment = session.query(Models.Comments).filter(Models.Comments.post
== id_p).all()
        session.delete(post)
        for com in comment:
            session.delete(com)
        session.commit()
    return redirect('/')


@app.route("/delete_comment/<int:id_p>/<int:id_c>", methods=["GET"])
@login_required
def delete_comment(id_p, id_c):
    comment = check(Models.Comments, Models.Comments.id == id_c)
    if comment is None:
        return redirect('/get_error/404')
    if comment.user == current_user.get_id():
        session.delete(comment)
        session.commit()
    return redirect(f'/post/{id_p}')


@app.route("/edit_post/<int:id_p>", methods=['GET'])
@login_required
def edit_post_g(id_p):
    post = check(Models.Posts, Models.Posts.id == id_p)
    if post is None:
        return redirect('/get_error/404')
    return render_template('edit_post.html', post=post)


@app.route("/edit_post/<int:id_p>", methods=['POST'])
@login_required
def edit_post_p(id_p):
    post = check(Models.Posts, Models.Posts.id == id_p)
    if post is None:
        return redirect('/get_error/404')
    if request.form['content'] == '' or request.form['title'] == '' or
current_user.get_id() != post.user:
        return redirect(f'/edit_post/{id_p}')
    post.title = request.form['title']
    post.content = request.form['content']
    session.add(post)
    session.commit()
    return redirect(f'/post/{id_p}')


@app.route("/edit_comments/<int:id_p>/<int:id_c>", methods=['GET'])
@login_required
def edit_comment_g(id_p, id_c):
    post = check(Models.Posts, Models.Posts.id == id_p)
    comment = check(Models.Comments, Models.Comments.id == id_c)
    if comment is None or post is None:
        return redirect('/get_error/404')
    return render_template('edit_comment.html', post=post, comments=comment)


@app.route("/edit_comments/<int:id_p>/<int:id_c>", methods=['POST'])
```

```python
@login_required
def edit_comment_p(id_p, id_c):
    comment = session.query(Models.Comments).filter(Models.Comments.id ==
id_c).one()
    if request.form['content'] == '' or current_user.get_id() !=
comment.user:
        return redirect(f'/edit_comments/{id_p}/{id_c}')
    comment.content = request.form['content']
    session.add(comment)
    session.commit()
    return redirect(f'/post/{id_p}')


@app.errorhandler(HTTPException)
def errors(errors):
    return render_template('error.html', code=int(str(errors).split()[0])),
str(errors).split()[0]


@app.route("/get_error/<int:id_e>", methods=['GET'])
def get_errors(id_e):
    return render_template('error.html', code=id_e), str(id_e)


if __name__ == '__main__':
    app.run(port=8000)
```

Также использовались файлы базы данных и html-шаблоны, которые не приведены в отчёте, но находятся на репозитории.

Экранные формы с примерами выполнения программы





Гостевая книга 1

1235 Выйти

4 записи:

123123 12:12:38 30 октября 2024

1231111 12:12:41 30 октября 2024

План 12:21:22 30 октября 2024

План1 12:23:11 30 октября 2024

Добавить запись

Название:

Содержание: Отправить

Гостевая книга 1

1235 Выйти

На главную

123123 12:12:38 30 октября 2024

123213

Удалить запись Редактировать

1 комментарий:

12:39:35 1 ноября 2024

1234

Добавить комментарий:

Текст: Aboba Добавить

Гостевая книга 1

На главную

123123 12:12:38 30 октября 2024

123213

Удалить запись Редактировать

2 комментария:

13:46:24 16 декабря 2024 *удалить редактировать*

Aboba

12:39:35 1 ноября 2024

1234

Добавить комментарий:

Текст: Добавить

Гостевая книга 1

На главную

Содержание: Aboba

Отмена Сохранить

Вернуться на главную

Вернуться на главную