ВИСШЕ УЧИЛИЩЕ ПО ТЕЛЕКОМУНИКАЦИИ И ПОЩИ

РЕФЕРАТ

Дисциплина:

Интеграция на системи бази от данни

Тема:

„Web project – ToDo App / УЕБ проект - ToDo App“

Изготвил:

Пламен Георгиев Маринов

III курс, Фак.№ 420017, Специалност КТ

Проверил:

доц. д-р Егн. Йоздикилилер

София 2023 г.

СЪДЪРЖАНИЕ

I. УВОД

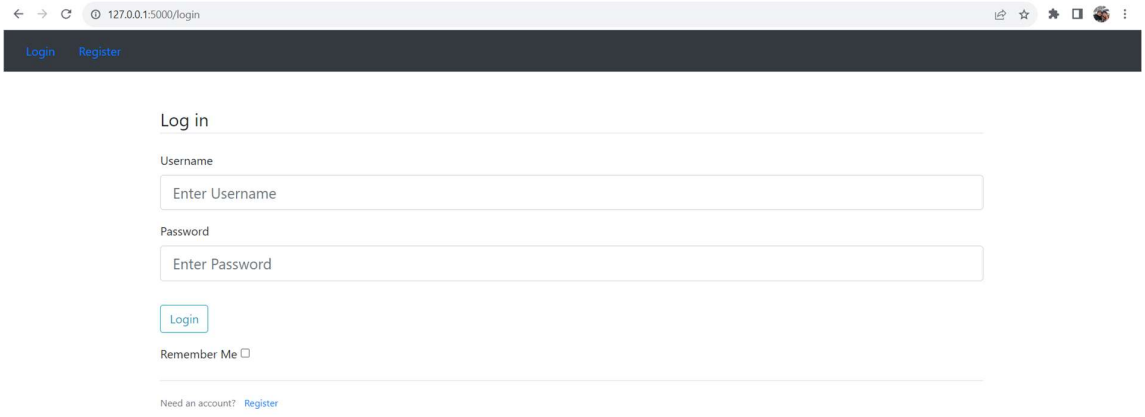
1. КРАТКО ОПИСАНИЕ НА ПРОЕКТА

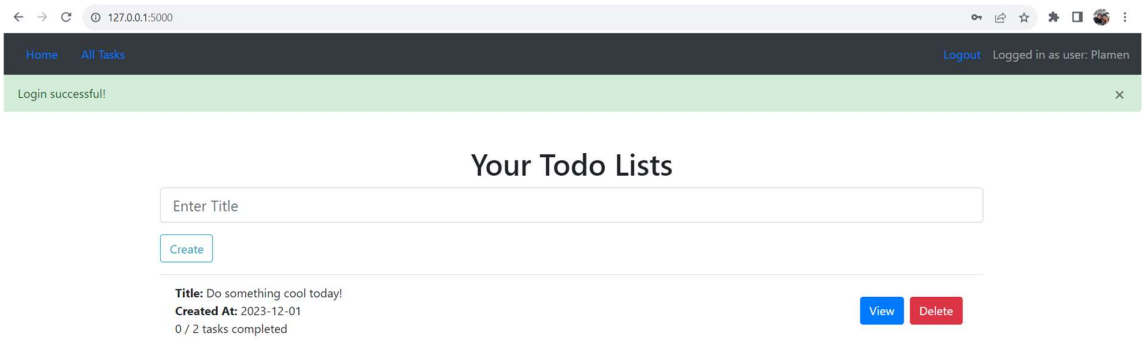
В това „Приложение за задачи (ToDo_App)“, се започва със създаване на потребител, след което може да се създадат списъци с задачи, които могат да бъдат изтривани или разглеждани.

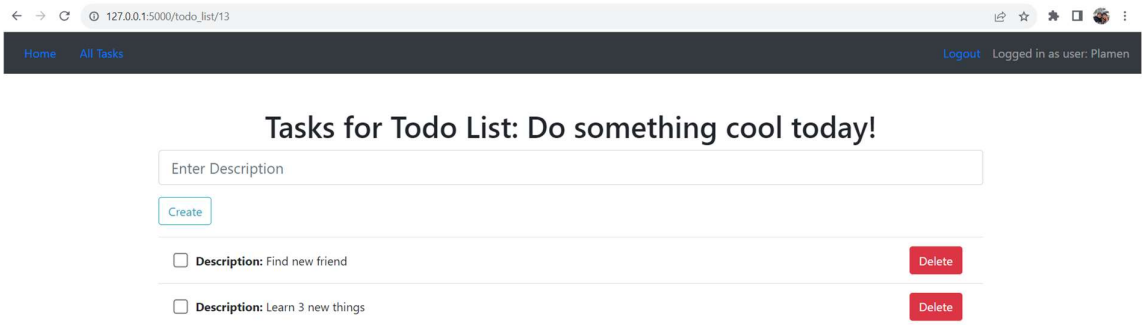
Може да се добавят задачи към списъци с задачи и да се маркират като завършени или да бъдат изтрити.

На началната страница ще се показват всички наши списъци с задачи и колко задачи има за всеки от тях и колко от тях са завършени.

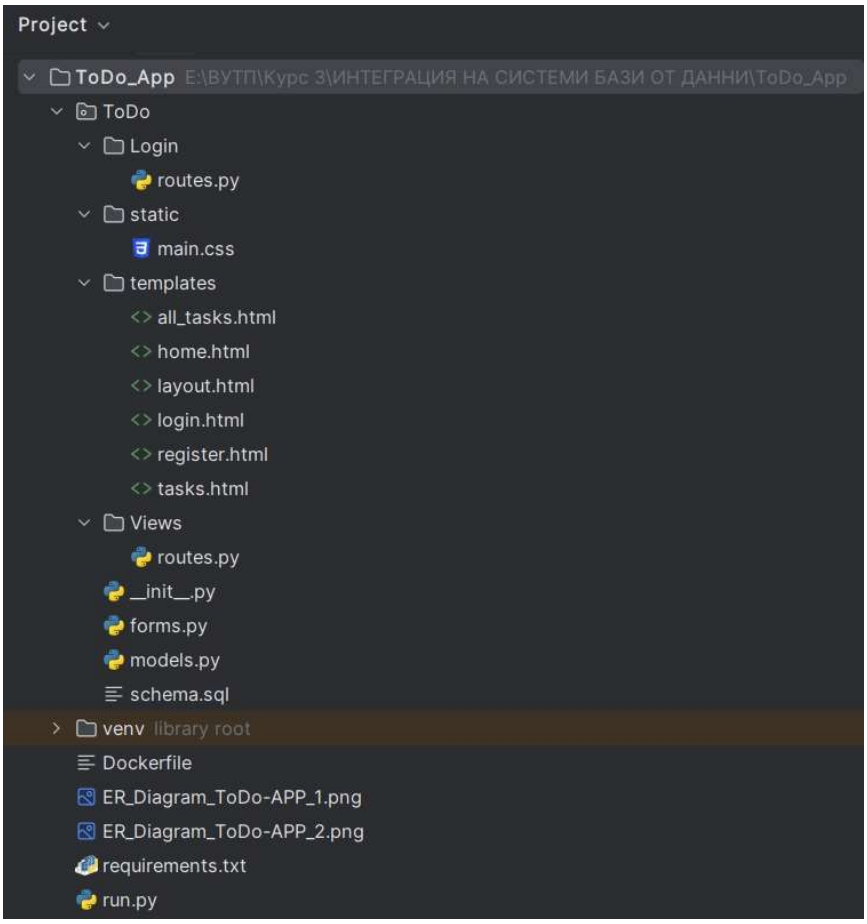
Изтриването на списък с задачи ще изтрие каскадно всички свързани задачи, а изтриването на потребител ще изтрие всички свързани списъци и задачи.







2. ФАЙЛОВА СТРУКТУРА НА ПРОЕКТА



Тази структура следва добрите практики за организация и управление на кода във Flask приложения.

II. ВЪВЕДЕНИЕ

3. УЕБ ПРИЛОЖЕНИЕ, ИНТЕГРАЦИЯ НА БАЗА ОТ ДАННИ И КОНТЕЙНЕРИЗАЦИЯ

3.1 FLASK ФРЕЙМУЪРК

Flask е микро уеб фреймуърк, написан на Python, проектиран да бъде прост и лесен за използване. Въпреки минималистичния си характер, Flask е мощен и разширяем, като го прави идеален избор за уеб разработчици. Той следва стандарта WSGI (Web Server Gateway Interface) и предоставя различни функции, включително маршрутизация, обработка на заявки и рендиране на шаблони.

3.2 POSTGRESQL

PostgreSQL, мощна отворена релационна система за управление на бази данни.

В контекста на този проект за „Приложение за задачи (Todo_App), Flask служи като основа на приложението. Той улеснява създаването на маршрути, които обслужват заявките на потребителите, дефинира структурата на приложението и позволява безпроблемната интеграция с базата данни PostgreSQL.

Flask използва подход, базиран на декоратори, за дефиниране на маршрути, което го прави интуитивен и кратък. В това приложение за задачи, маршрутите се използват за обработка на различни действия на потребителите, като създаване на потребители, управление на списъци с задачи и обработка на задачи.

```python
from flask import blueprints, render_template, flash, redirect, url_for
from flask_login import current_user, login_user, logout_user, login_required

from ToDo.forms import LoginForm, RegisterForm
from ToDo.models import select_users_by_username, select_users_by_email, register_user

Login = blueprints.Blueprint('Login', __name__)


@Login.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        flash("You are already logged in!", category='error')
        return redirect(url_for('Views.home'))

    form = LoginForm()
    if form.validate_on_submit():...
    return render_template('login.html', form=form, user=current_user)

@Login.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        flash("You are already logged in!", category='error')
        return redirect(url_for('Views.home'))

    form = RegisterForm()

    if form.validate_on_submit():...
```

Flask се интегрира безпроблемно с HTML шаблони, позволявайки създаването на динамични и интерактивни потребителски интерфейси. Функцията render_template се използва за рендиране на HTML шаблони и предаване на данни към тях, което позволява представянето на списъци с задачи и задачи на предния план.

3.3 ИНТЕГРАЦИЯ С БАЗАТА ОТ ДАННИ

В този проект се използва библиотеката psycopg2 за свързване с база данни PostgreSQL, заедно с Flask и Flask-Login за управление на потребителските сесии.

```python
import psycopg2
from flask import Flask
from flask_login import LoginManager


app = Flask(__name__)
app.config['SECRET_KEY'] = 'gfjfgtretrehfggh245354342d'

# Database connection
conn = psycopg2.connect(
    host="localhost",
    database="TodoApp",
    user="postgres",
    password="*********"
)

# Login Manager
LoginManager = LoginManager(app)
LoginManager.login_view = 'Login.login'
LoginManager.login_message_category = 'info'

from ToDo.Views.routes import Views
from ToDo.Login.routes import Login

app.register_blueprint(Views)
app.register_blueprint(Login)

```

- Създаване на Flask Приложение:
- Създава инстанция на Flask приложението.
- Конфигуриране на Ключ за Сесия: Задава ключ за криптиране, използван от Flask за сесиите на потребителите.

- Връзка с PostgreSQL База Данни: Установява връзка с базата данни PostgreSQL чрез библиотеката psycopg2. Важно е да се отбележи, че връзката към базата данни се прави при стартирането на приложението.
- Настройка на Flask-Login:
- Инициализира обект от тип LoginManager, който се използва от Flask-Login за управление на сесиите. Задават се основни параметри, като изгледа за вход и категория за съобщения.
- Импортиране на Маршрутите от Различни Модули:
- Импортират се маршрутите (routes) от различни модули. Това е добра практика за организация на кода, като различните части на приложението (например, изгледи и вход) се поддържат в отделни модули.
- Регистриране на Маршрутите:
- Регистрират се Blueprint-ове (модули с маршрути) в приложението. Това позволява по-лесно управление на маршрутите и групиране на функционалностите.

Общо взето, този код представлява базова конфигурация на Flask приложение със съединение към PostgreSQL база данни и използване на Flask-Login за управление на потребителските сесии. Разделението на кода в различни модули улеснява поддръжката и разширяемостта на приложението.

3.4 ТАБЛИЦИ

- Таблица Users:

id (INTEGER): Уникален идентификатор за всеки потребител.

username (VARCHAR): Потребителско име на потребителя, уникално и задължително.

email (VARCHAR): Електронна поща на потребителя, уникална и задължителна.

password (VARCHAR): Парола за аутентикация на потребителя.

- Таблица todo_list:

id (INTEGER): Уникален идентификатор за всяка "списък със задачи" (todo list).

title (VARCHAR): Заглавие на списъка със задачи, задължително.

creation_date (DATE): Дата на създаване на списъка със задачи, задължителна.

user_id (INTEGER): Външен ключ, свързан с Users таблицата чрез id. Указва кой потребител притежава този списък със задачи.

- Таблица tasks:

taskId (SERIAL): Уникален идентификатор за всяка задача.

description (VARCHAR): Описание на задачата, задължително.

completed (BOOLEAN): Поле, което указва дали задачата е завършена (по подразбиране е "false").

todo_list_id (INTEGER): Външен ключ, свързан с todo_list таблицата чрез id. Указва в кой списък със задачи принадлежи тази задача.

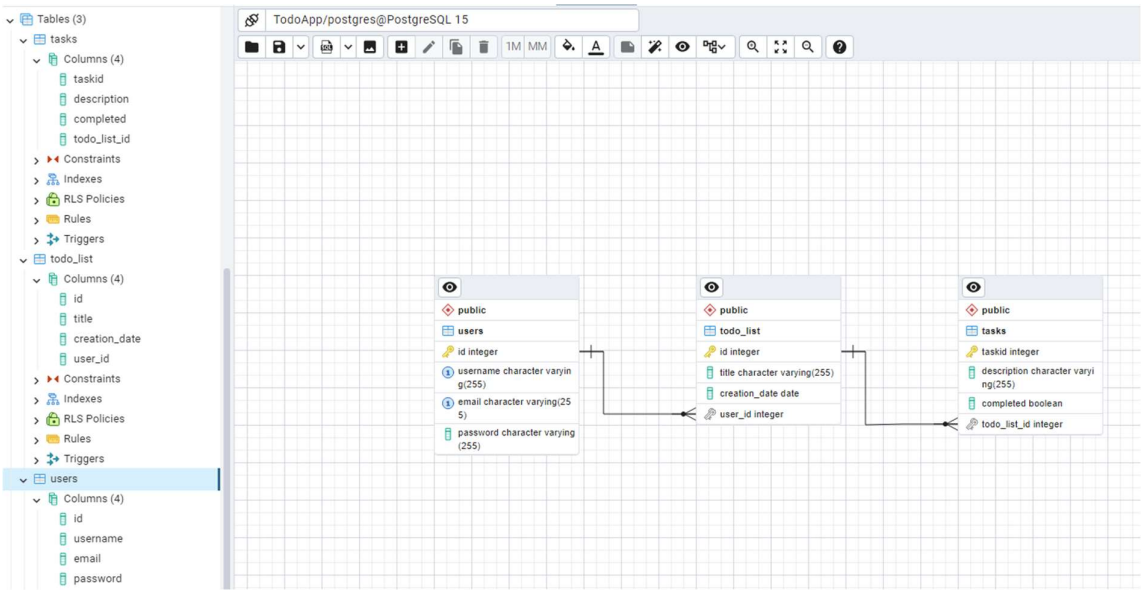
Връзките между таблици се осъществяват чрез външни ключове (FOREIGN KEY), които свързват полетата в една таблица с тези в друга. Това предоставя връзка между записите в различните таблици.

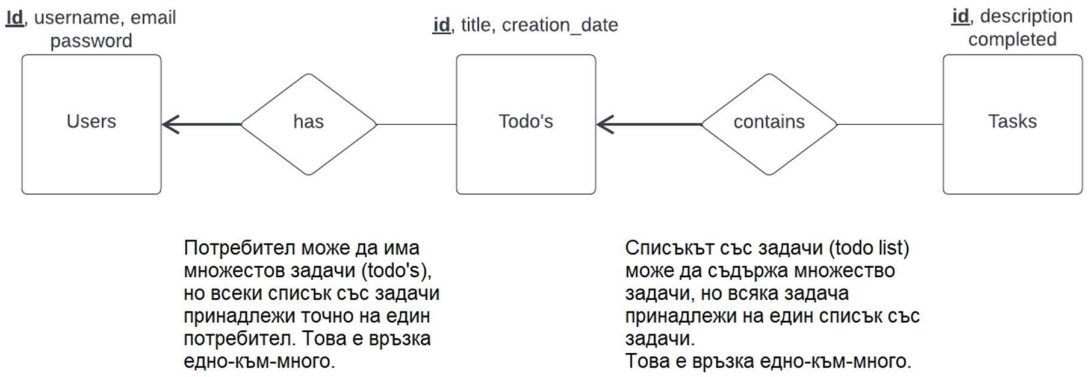
В таблицата todo_list, полето user_id е външен ключ, свързан с id в таблицата Users. Това означава, че всеки списък със задачи принадлежи на определен потребител.

В таблицата tasks, полето todo_list_id е външен ключ, свързан с id в таблицата todo_list. Това означава, че всяка задача принадлежи на определен списък със задачи.

Така, чрез тези връзки, можем да следим кой потребител, кои списъци със задачи притежава и кои задачи са свързани с определен списък.

3.5 ER ДИАГРАМИ





3.6 КОНТЕЙНЕРИЗАЦИЯ

За контейнеризация на този уеб проекта с Flask и PostgreSQL, съм използвал Docker за създаване и управление на контейнери.

Docker контейнер е стандартизиран, лек и изолиран пакет, който включва всичко необходимо за изпълнението на приложение - код, библиотеки, зависимости и настройки на системата. Контейнерите позволяват лесно преносимо и възпроизводимо изграждане и доставка на софтуер, независимо от околната среда.

ПРОЦЕС ПО КОНТЕЙНЕРИЗАЦИЯ

- Дефиниране на Dockerfile

Създавате файл с име "Dockerfile", който съдържа инструкции за създаване на образа на Docker. В този файл се определя основна операционна система, инсталират се необходимите зависимости, копира се кода на приложението и конфигурациите, и се настройва средата за изпълнение.

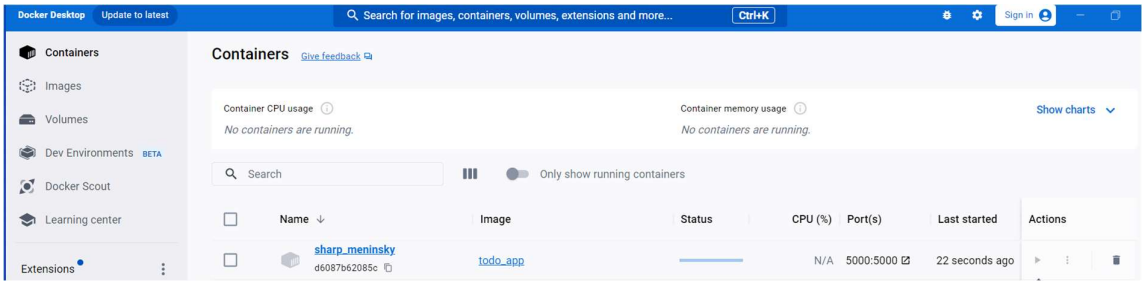
```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "run.py"]
```

- Изграждане на Docker Image

MINGW64:/e/ВУТП/Курс 3/ИНТЕГРАЦИЯ НА СИСТЕМИ БАЗИ ОТ ДАННИ/ToDo_App

```
marinov@itafin MINGW64 /e/ВУТП/Курс 3/ИНТЕГРАЦИЯ НА СИСТЕМИ БАЗИ ОТ ДАННИ/ToDo_A
pp
$ docker build -t todo_app .
#0 building with "default" instance using docker driver

#1 [internal] load .dockerignore
#1 transferring context: 2B 0.0s done
#1 DONE 0.1s

#2 [internal] load build definition from Dockerfile
#2 transferring dockerfile: 445B 0.0s done
#2 DONE 0.1s

#3 [internal] load metadata for docker.io/library/python:3.8-slim
#3 DONE 2.2s

#4 [1/3] FROM docker.io/library/python:3.8-slim@sha256:8e9969d0711a6983ff935dfe2
d68f09dcd82f5af5f6bf472c5674db2d462c486
#4 resolve docker.io/library/python:3.8-slim@sha256:8e9969d0711a6983ff935dfe2d68
f09dcd82f5af5f6bf472c5674db2d462c486 0.1s done
#4 DONE 0.1s

#5 [internal] load build context
#5 transferring context: 1.29MB 0.8s done
#5 DONE 0.9s

#6 [2/3] WORKDIR /app
#6 CACHED

#7 [3/3] COPY . /app
#7 DONE 2.0s

#8 exporting to image
#8 exporting layers
#8 exporting layers 0.7s done
#8 writing image sha256:f2fd911ab305d1f8bbebaa7fb534d6e4d638f82d8dc1573d08807df7
b31b56b6 0.0s done
#8 naming to docker.io/library/todo_app 0.0s done
#8 DONE 0.8s
```

- Стартиране на Docker Container



III. ЗАКЛЮЧЕНИЕ

Проектът, базиран на използването на Flask, PostgreSQL и Docker, представлява успешен модел за създаване на уеб приложение, което е гъвкаво, мощно и лесно преносимо. Flask улеснява разработката на уеб приложения с минимално усилие, предоставяйки нужните инструменти за създаване на маршрути и обработка на заявки.

Интеграцията с PostgreSQL осигурява устойчиво съхранение на данни с възможност за сложни заявки и връзки.

Използването на Docker допринася за лесната преносимост и възпроизводимост на приложението, като осигурява изолация и консистентност в различни околни среди. Контейнеризацията улеснява развитието, тестването и доставката на приложението, като гарантира, че околната среда в контейнера е независима от околната среда на хост системата.

В общи линии, проектът, изграден с тези технологии, демонстрира ефективно използване на съвременни инструменти за разработка, които осигуряват удобство и ефективност при създаване, разширяване и управление на уеб приложения.