Лабораторна робота №6

Тема: Мікрофреймворк Flask. Створення додатку “Список завдань”.

Мета роботи: отримати навички створення Веб додатку.

ТЕОРЕТИЧНА ЧАСТИНА

* для виконання роботи потрібно мати базові знання з HTML+CSS, та Python.

** дана роботи на 100% виконана та перевірена на лінукс подібній операційній системи

Flask - це web фреймворк та модуль Python, який дозволяє легко створювати веб додатки. Flask має простий синтаксис, та дозволяє лего розширювати ваші додатки. Мікрофреймворк не має ORM (*Object Relational Manager*), тобто вам не потрібно думати про функціонал високорівневих процесів, протоколів, менеджеру потоків і т.д.

Робота мікрофреймворку базується на WSGI (*Web Server Gateway Interface)* взаємодією програми написаною мовою Python, що виконується на стороні серверу та самим веб сервером, наприклад Apache. Застосунок повинен відповідати певним вимогам, бути об'єктом що вивкликажться, приймати параметри, повертати ітератор.

Фактично вся розроблювана програма в цій ЛР це інструмент для роботи з інтерфейсом WSGI через Middleware компоненти, що надають інтерфейс, як серверу, так і застосунку. Python має багато бібліотек та компонентів, саме такий інтерфейс (посередник) робить можливою взаємодію та роботу вашого застосунку та серверу.

Офіційна документація за адресою:
https://flask.palletsprojects.com/en/1.1.x/quickstart/

ПОРЯДОК ВИКОНАННЯ

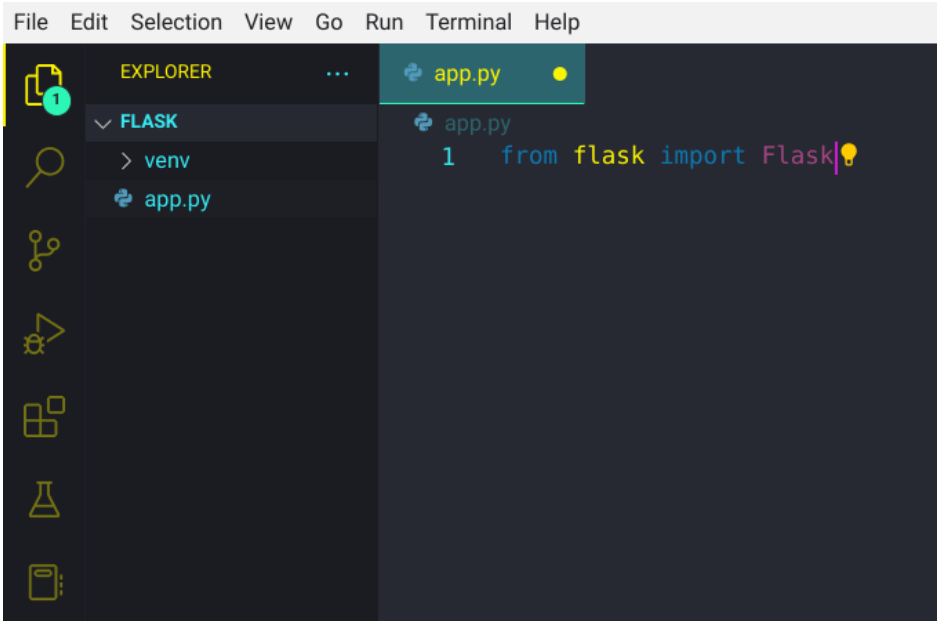
Частина 1. Мінімальний функціонал - до 75 балів

1. Встановіть Python, якщо ще не мажте його на своєму комп’ютері.
2. Створіть папку для роботи. Рекомендована назва “flask”.
3. Використовуючи термінал, перейдіть в папку та виконайте команду “*pip install virtualenv”* що встановить віртуальне середовище.
4. Створіть віртуальне середовище командою “*virtualenv venv*” це створить папку “venv” де будуть зберігатися усі файли проекту.
5. Активуйте віртуальне середовище командою “*source venv/bin/activate*”.
6. Встановіть микрофреймворк flask та базу даних командою “*pip install flask flask-sqlalchemy*”.

7. Створіть файл “*app.py*”.
8. Перші інструкції програми:

```
from flask import Flask
```

9. На даному етапі вміст вашого проекту у папці “*flask*” має виглядати ось так:



10. Додайте текст програми:

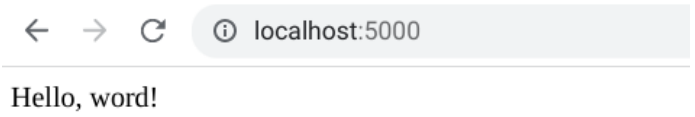
```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, word!"

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

11. Запустіть додаток командою “*python app.py*”, має запуститися локальний сервер. Це виглядатиме так:

```
(venv) nstekh@penguin:~/Python_Lessons/flask$ python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 106-895-313
```

Інструкція “*debug=True*” вмикає сервер в режимі розробки, не залишайте його в цьому режимі, якщо розміщуєте вже готовий сервер в інтернет.

12. Відкрийте вікно браузера та введіть адресу: “*http://localhost:5000*”. Ви побачите у вікні наступне:

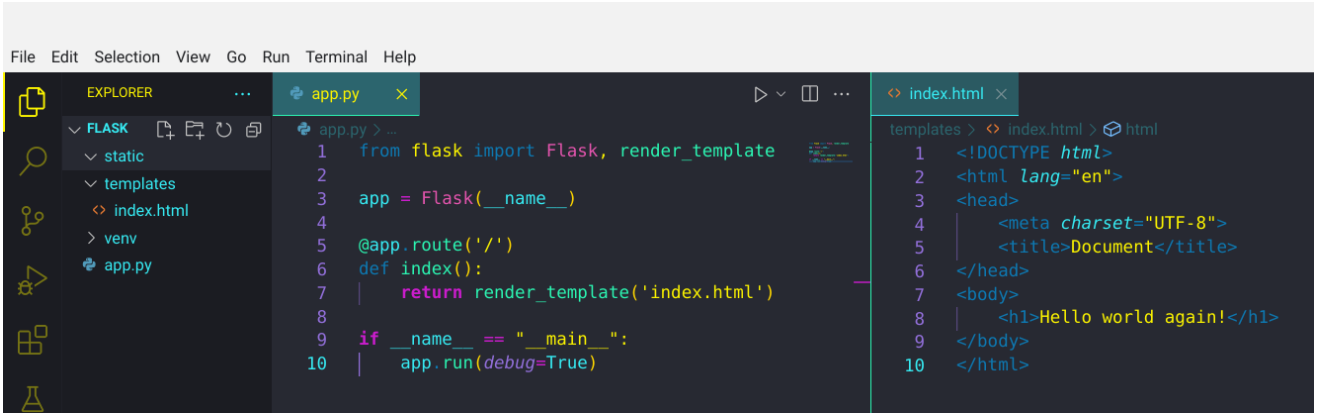


13. Наступний крок - створити дві папки: “*static*” та “*templates*”. Створіть документ “*index.html*” в папці “*templates*” Імпортуємо модуль “*render_template*”, він знадобиться для того, щоб програма повертала функцією “*index()*” документ “*index.html*”. “*Hello world again!*” - текст для файлу “*index.html*”

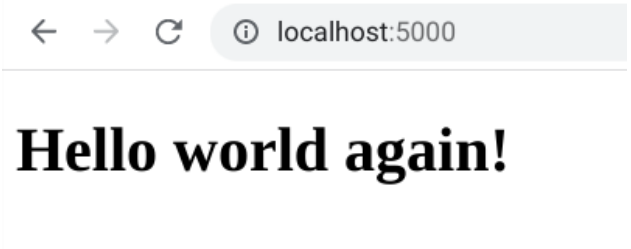
* встановіть розширення emmet для VSCode, а документ “index.html” заповніть шаблоном натиснувши “!” і потім “TAB”.

14. Внесіть зміни в програму `return render_template('index.html')`

Як тільки ви зберігайте файл - ваша сторінка в браузері повинна оновитися, якщо цього не відбулося, оновіть вручну. Ваше дерево проекту та вміст файлів повинні виглядати так:



Виведення на екран:



15. Шаблони потрібні для того, щоб ваш додаток використовував їн на різних сторінках. Створіть файл base.html. В цьому шаблоні ми будемо використовувати елементи написані на Jinja2, спеціальній мові для шаблонів у Flask. Основні аспекти використання мови:

{%....%} - для виразів

{{....}} - вирази що виводяться в шаблон

{#....#} - коментарі, що не виводяться

#....## - вирази що виводяться в рядок

Вміст файлу “*base.html*” та “*index.html*” після внесених змін:

16. Додаємо оформлення до шаблону. Створіть “*static/css/main.css*”. Вміст файлу:

```
body{
    margin: 0;
    background-color: rgb(255, 244, 244);
    display: flex;
    justify-content: center;
}
```

17. Підключаємо в шаблон файл стилів, впишіть в тег <head>:

```
<link rel="stylesheet" href="static/css/main.css">
```

Цей варіант робочий, але нам потрібно додати файл стилів іншим способом, цього вимагає мікрофреймворк для правильної роботи.

Імпортуйте модуль “*url_for*” в “*app.py*” та замініть рядок підключення файлу стилів на:

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
```

Функція створить посилання на файл стилів та передасть його в тег “*link*”

*{%...%} безпосередньо виконує пайтон код в цих скобках, а {{...}} повертає результат виконання.

18. Наступним кроком імпортуємо модуль “SQLALCHEMY” та вкажемо шлях до БД “sqlite”. “////” - абсолютний шлях, “///” - відносний шлях. Наш додаток потребуватиме доступу до поточного часу, тож додамо модуль “*datetime*”.

Об’єкт, що відповідатиме за створення бази даних подану в готовому вигляді, роботу з БД розглянемо в окремій практичній роботі. В середині класу є функція, що буде повертати id створеного завдання (та доданого в БД).

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
db = SQLAlchemy(app)

class Todo(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.String(200), nullable=False)
    data_created = db.Column(db.DateTime, default=datetime.utcnow)
```

```
    def __repr__(self) -> str:
        return '<Task %r>' % self.id
```

Файл “app.py” виглядатиме наступним чином:

19. Налаштуємо базу даних. Віртуальне оточення має бути активоване. Усі дії ми виконуємо в середовищі інтерпретатора:

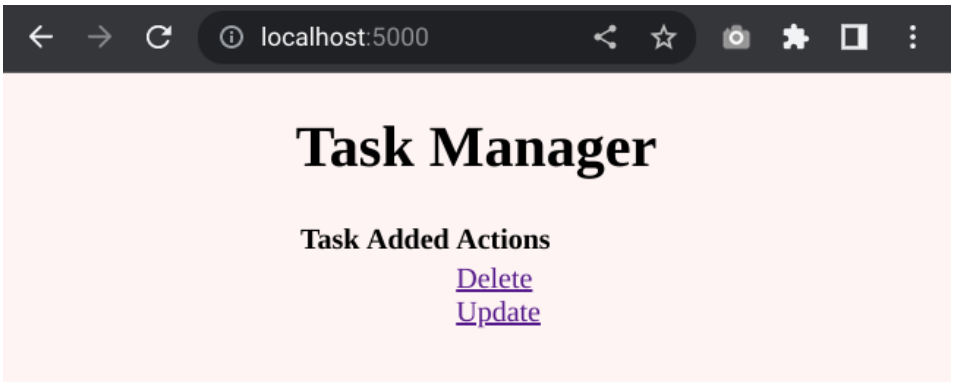
- python
- from app import db #увага, ми імортували db з файлу app.py! 7 рядок
- db.create_all() #клас із файлу щойно виконав усі інструкції, якщо не виконав - вам на stackoveflow або додайте “`app.app_context().push()`” у 8 рядок, перезапустіть все та виконайте заново
- exit() #БД занаходиться в папці “*instance*” або в корневій директорії проекту

20. Перейдемо до створення додатку. У файлі index.html внесіть наступні зміни:

```
{% extends 'base.html' %}

{% block head %}
<title>Task Manager</title>
{% endblock %}

{% block body %}
<div class="content">
    <h1>Task Manager</h1>
    <table>
```

```
        <tr>
            <th>Task</th>
            <th>Added</th>
            <th>Actions</th>
        </tr>
        <tr>
            <td></td>
            <td></td>
            <td>
                <a href="">Delete</a>
                <br>
                <a href="">Update</a>
            </td>
        </tr>
    </table>
</div>

{% endblock %}
```

Запустіть сервер командою “python app.py”, та сам сайт “*localhost:5000*”. Має виглядати так:



21. Допрацємо додаток “app.py”, додаємо два методи відправки даних, які може прийняти “route”:

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

22. Дані ми надсилатимемо формою у з “*index.html*”, додаємо її після таблиці:

```
    <form action="/" method="POST">
        <input type="text" name="content" id="content">
        <input type="submit" value="Add Task">
    </form>
```

23. Додамо блок в “app.py”, який реагуватиме на запити з форми (які вона робитиме на ту ж сторінку на якій знаходиться) відповідний функціонал і бібліотеку “request”. “if” повертає повідомлення у випадку якщо форма

підтверджена, а “else” повертає рендер нашого шаблону, у випадку якщо не було запиту на відправку даних з сайту.

```
@app.route('/', methods=['POST', 'GET'])
def index():
    if request.method == 'POST':
        return “Form submitted!”
    else:
        return render_template('index.html')
```

Спробуйте відправити форму.

24. Додаємо логіку обробки відправлених даних в блок “if”. Імпортуємо “redirect”, та за допомогою об’єкту створеного у п.18 ми зберігаємо дані у змінну “task_content” та відправляємо в БД та повертаємо в сесію програми редірект на головну сторінку.

* зверніть увагу! “” в `request.form['content']` це атрибут тега “input” в “index.html”

```
Todo.query.order_by(Todo.date_created).all()
```

цей запит поверне з БД записи відсортовані за часом створення.

```
return render_template('index.html', tasks=tasks)
```

А ось тут ми повертаємо не просто головну сторінку в браузер, а і дані з БД у змінній (спистку) “tasks”

Змінений блок “if…else”:

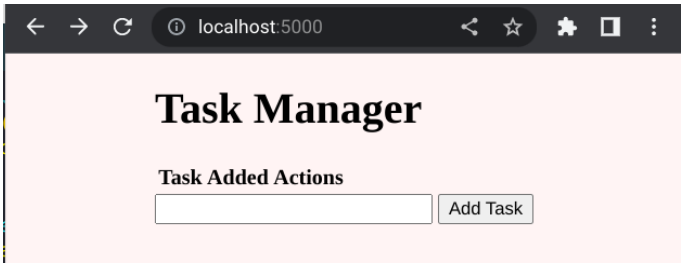
```
    if request.method == 'POST':
        task_content = request.form['content']
        new_task = Todo(content=task_content)

        try:
            db.session.add(new_task)
            db.session.commit()
            return redirect('/')
        except:
            return 'There was an issue adding your task'

    else:
        tasks = Todo.query.order_by(Todo.data_created).all()
        return render_template('index.html', tasks=tasks)
```

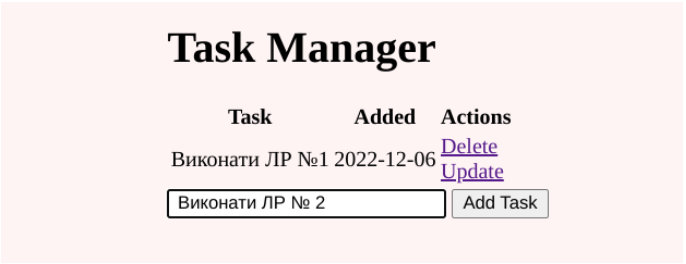
25. Внесемо зміни у файл “index.html”, так щоб він почав виводити рядки таблиці в циклі. Для цього скористаємося синтаксисом Jinja2:

```
    <table>
        <tr>
            <th>Task</th>
            <th>Added</th>
```

```
            <th>Actions</th>
        </tr>
        {% for task in tasks %}
            <tr>
                <td>{{ task.content }}</td>
                <td>{{ task.data_created.date() }}</td>
                <td>
                    <a href="">Delete</a>
                    <br>
                    <a href="">Update</a>
                </td>
            </tr>
        {% endfor %}
    </table>
```

На веб сторінці не буде відображатися таблиця при оновленні, поки ви не відправите в неї першу інформацію:





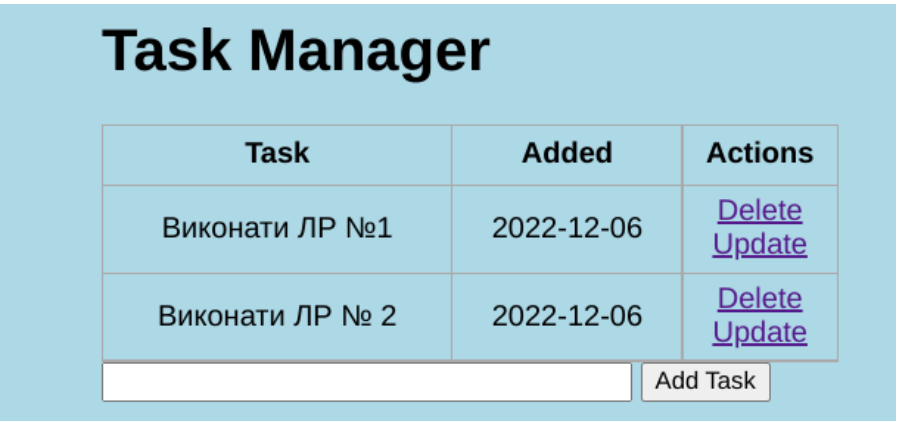
26. Додайте кілька завдань.

Частина 2. Повний функціонал - до 90 балів

27. Оновіть файл main.css

```
body, html {
    margin: 0;
    font-family: sans-serif;
    background-color: lightblue;
}

h1{
    text-align: center;
}
```

```css
.content {
    margin: 0 auto;
    width: 400px;
}

table, td, th {
    border: 1px solid #aaa;
}

table {
    border-collapse: collapse;
    width: 100%;
}

th {
    height: 30px;
}

td {
    text-align: center;
    padding: 5px;
}

.form {
    margin-top: 20px;
}

#content {
    width: 70%;
}
```



28. Додаємо можливість видалення контенту. Створюємо новий роут “delete” і прив’яжемо інформацію про рядок, що має бути видалено до унікального поля в кожному рядку, до “id”.

```
Todo.query.get_or_404(id)
```

Запит в базу даних видалить завдання , або поверне помилку 404, за це відповідає функція “*get_or_404*”

Повний лістинг блоку програми:

```
@app.route('/delete/<int:id>')
def delete(id):
    task_to_delete = Todo.query.get_or_404(id)

    try:
        db.session.delete(task_to_delete)
        db.session.comit()
        return redirect('/')
    except:
        return "Couldn't delete that task."
```

29. У файл “*index.html*” додамо посилання на “*id*” елемента який ми хочемо видалити:

```
<a href="/delete/{{ task.id }}">Delete</a>
```

Подивіться на даний шлях, в документ html скрипт Jinja2 відкриває “id” який передає в роут “delete” файу “app.py”.

30. Перевірте чи видаляється завдання з БД. Виправте граматичну помилку у функції “delete”.

31. Зробимо роут для останнього посилання в таблиці “Update”. Єдина різниця у тому, що для оновлення (зміни) даних в БД, нам потрібно трохи по іншому відобразити матеріал. Вивести інформацію відразу в “input” форму. Роут з функцією виглядатиме так:

```
@app.route('/update/<int:id>', methods=['GET', 'POST'])
def update(id):
    task = Todo.query.get_or_404(id)

    if request.method == 'POST':
        task.content = request.form['content']

        try:
            db.session.commit()
            return redirect('/')
        except:
            return "Can't update task!"
    else:
        return render_template('update.html', task=task)
```

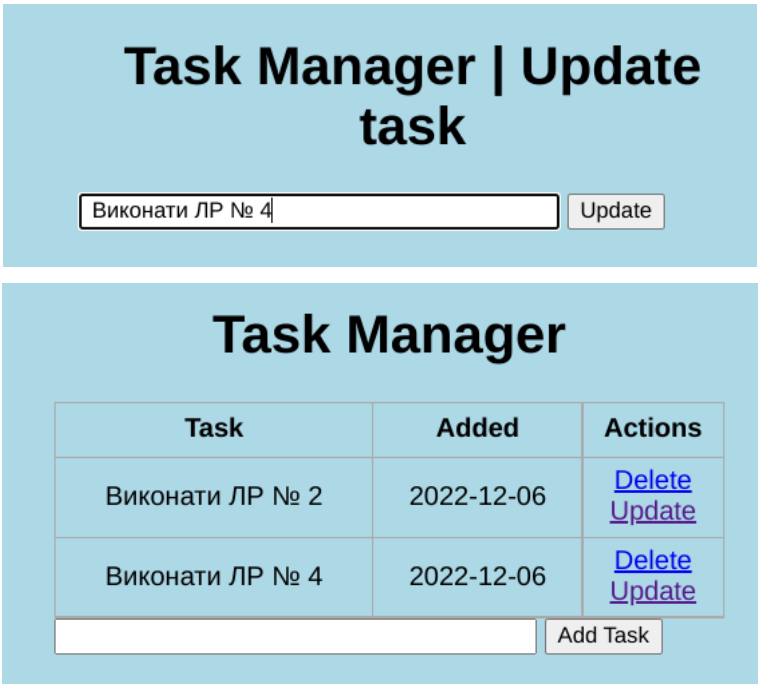
Також внесемо зміни в “index.html”:

```
    <a href="/update/{{ task.id }}">Update</a>
```

32. Створимо ще файл “update.html” в який скіпіюємо вміст “index.html” без таблиці. Додамо атрибут “value” в “input”. Завдання цього отрибута пояснено тут: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/data#attr-value

```
    {% extends 'base.html' %}

    {% block head %}
    <title>Task Manager</title>
    {% endblock %}

    {% block body %}
    <div class="content">
        <h1>Task Manager | Update task</h1>

        <form action="/update/{{ task.id }}" method="POST">
            <input type="text" name="content" id="content" value="{{
task.content }}">
            <input type="submit" value="Update">
        </form>
    </div>

    {% endblock %}
```

33. Ось що маємо в результаті:



Частина 3. 90+ Оформіть веб додаток використавши Bootstrap

Мова йде про підключення стилів, оформлення таблиці та всієї сторінки згідно принципу роботи сітки.