Практическое занятие №6. Шаблоны в FastAPI.

Теперь, когда мы узнали, как обрабатывать ответы на запросы, включая ошибки в предыдущей главе, мы можем приступить к отображению ответов на запросы на веб-странице. В этой главе мы узнаем, как отображать ответы от нашего API на веб-странице, используя шаблоны на основе **Jinja**, который представляет собой язык шаблонов, написанный на Python, предназначенный для облегчения процесса визуализации ответов API.

Шаблонирование — это процесс отображения данных, полученных от API, в различных форматах. Шаблоны действуют как компонент интерфейса в веб-приложениях.

К команду этой работы вы будете владеть знаниями о том, что такое шаблоны и как использовать шаблоны для рендеринга информации из вашего API. В этой главе мырассмотрим следующие темы:

- Понимание Jinja
- Использование шаблонов Jinja2 в FastAPI

1. Понимание Jinja

Jinja — это механизм шаблонов, написанный на Python, предназначенный для облегчения процесса рендеринга ответов API. В каждом языке шаблонов есть переменные, которые заменяются фактическими значениями, переданными им приотображении шаблона, и есть теги, управляющие логикой шаблона.

Механизм шаблонов Jinja использует фигурные скобки { }, чтобы отличить свои выражения и синтаксис от обычного HTML, текста и любой другой переменной вфайле шаблона.

Синтаксис {{ }} называется **блоком переменных**. Синтаксис {% %} содержитуправляющие структуры, такие как **if/else**, **циклы** и **макросы**.

Три общих синтаксических блока, используемых в языке шаблонов Jinja, включают следующее:

- `{% … %}` – Этот синтаксис используется для операторов, таких как управляющие структуры.
- `{{ todo.item }}` – Этот синтаксис используется для вывода значенийпереданных ему выражений.
- `{# This is a great API book! #}` – Этот синтаксис используется принаписании комментариев и не отображается на веб-странице.

Переменные шаблона Jinja могут относиться к любому типу или объекту Python,если их можно преобразовать в строки. Тип модели, списка или словаря можно передать шаблону и отобразить его атрибуты, поместив эти атрибуты во второйблок, указанный ранее.

В следующем разделе мы рассмотрим фильтры. Фильтры являются важной частью каждого механизма шаблонов, и в Jinja фильтры позволяют нам выполнять определенные функции, такие как объединение значений из списка и получение длины объекта, среди прочего.

В следующих подразделах мы рассмотрим некоторые общие функции, используемые в Jinja: фильтры, операторы if, циклы, макросы и наследованиешаблонов.

1.1.Фильтры

Несмотря на сходство синтаксиса Python и Jinja, такие модификации, как объединение строк, установка первого символа строки в верхний регистр и т. д., немогут быть выполнены с использованием синтаксиса Python в Jinja. Поэтому для выполнения таких модификаций у нас в Jinja есть фильтры.

Фильтр отделяется от переменной вертикальной чертой (|) и может содержать необязательные аргументы в круглых скобках. Фильтр определяется в этом формате:

```
{{ variable | filter_name(*args) }}
```

Если нет аргументов, определение становится следующим:

```
{{ variable | filter_name }}
```

Давайте рассмотрим некоторые распространенные фильтры в следующих подразделах.

Фильтр по умолчанию

Переменная фильтра по `умолчанию` используется для замены вывода переданного значения, если оно оказывается `None`:

```
{{ todo.item | default('This is a default todo item') }}
This is a default todo item
```

Эвакуационный фильтр

Этот фильтр используется для отображения необработанного вывода HTML:

```
{{ "<title>Todo Application</title>" | escape }}
<title>Todo Application</title>
```

Фильтры преобразования

Эти фильтры включают фильтры int и float, используемые для преобразования изодного типа данных в другой:

```
{{ 3.142 | int}}3
{{ 31 | float }}
31.0
```

Фильтр объединения

Этот фильтр используется для объединения элементов списка в строку, как в Python:

```
{{ ['Packt', 'produces', 'great', 'books!'] | join(' ') }}
Packt produces great books!
```

Фильтр длины

Этот фильтр используется для возврата длины переданного объекта. Он выполняет туже роль, что и `len()` в Python:

```
Todo count: {{ todos | length }}Todo
count: 4
```

> **Примечание**
>
> Полный список фильтров и дополнительные сведения о фильтрах в Jinja см. на странице `https://jinja.palletsprojects.com/en/3.0.x/templates/#builtin-filters`.

Использование операторов if

Использование операторов `if` в Jinja аналогично их использованию в Python. `if`операторы используются в блоках управления {% %}. Давайте посмотрим на пример:

```
{% if todo | length < 5 %}
    You don't have much items on your todo list!
{% else %}
    You have a busy day it seems!
{% endif %}
```

Циклы

Мы также можем перебирать переменные в Jinja. Это может быть список или общая функция, например, следующая:

```
{% for todo in todos %}
         <b> {{ todo.item }} </b>
{% endfor %}
```

Вы можете получить доступ к специальным переменным внутри цикла `for`, таким как `loop.index`, который дает индекс текущей итерации. Ниже приведен список специальных

переменных и их описания:

Variable	Description
`loop.index`	The current iteration of the loop (1 indexed)
`loop.index0`	The current iteration of the loop (0 indexed)
`loop.revindex`	The number of iterations from the end of the loop (1 indexed)
`loop.revindex0`	The number of iterations from the end of the loop (0 indexed)
`loop.first`	True if first iteration
Variable	**Description**
`loop.last`	True if last iteration
`loop.length`	The number of items in the sequence
`loop.cycle`	A helper function to cycle between a list of sequences
`loop.depth`	Indicates how deep in a recursive loop the rendering currently is; starts at level 1
`loop.depth0`	Indicates how deep in a recursive loop the rendering currently is; starts at level 0
`loop.previtem`	The item from the previous iteration of the loop; undefined during the first iteration
`loop.nextitem`	The item from the following iteration of the loop; undefined during the last iteration
`loop.changed(*val)`	True if previously called with a different value (or not called at all)

Макросы

Макрос в Jinja — это функция, которая возвращает строку HTML. Основной вариант использования макросов — избежать повторения кода и вместо этого использовать один вызов функции. Например, макрос ввода определен для сокращения непрерывного определения тегов ввода вHTML-форме:

```
{% macro input(name, value='', type='text', size=20 %}
    <div class="form">
        <input type="{{ type }}" name="{{ name }}" value="{{
         value|escape }}" size="{{ size }}">
        </div>
    {% endmacro %}
```

Теперь, чтобы быстро создать ввод в вашей форме, вызывается макрос:

```
{{ input('item') }}
```

Это вернет следующее:

```
<div class="form">
    <input type="text" name="item" value="" size="20">
</div>
```

Теперь, когда мы узнали, что такое макросы, мы приступим к изучению того, чтотакое наследование шаблонов и как оно работает в FastAPI.

Наследование шаблонов

Самая мощная функция Jinja — наследование шаблонов. Эта функция продвигаетпринцип **«не повторяйся» (DRY)** и удобна в больших веб-приложениях.

Наследование шаблона — это ситуация, когда базовый шаблон определен, а дочерние шаблоны могут взаимодействовать, наследовать и заменять определенные разделы базового шаблона.

Примечание

Вы можете узнать больше о наследовании шаблонов Jinja на `https://jinja.palletsprojects.com/en/3.0.x/templates/#template-inheritance`.

Теперь, когда вы изучили основы синтаксиса Jinja, давайте научимся использовать шаблоны в FastAPI в следующем разделе.

Использование шаблонов Jinja в FastAPI

Для начала нам нужно установить пакет Jinja и создать новую папку, `templates`, вкаталоге нашего проекта. В этой папке будут храниться все наши файлы Jinja, которые представляют собой файлы HTML, смешанные с синтаксисом Jinja.

Поскольку эта книга не посвящена дизайну пользовательского интерфейса, мы будем использовать библиотеку CSS `Bootstrap` и не будем писать собственныестили.

Библиотека Bootstrap будет загружена из CDN при загрузке страницы. Однако дополнительные активы можно хранить в другой папке. Мы рассмотрим обслуживание статических файлов в следующей главе.

Мы начнем с создания шаблона домашней страницы, на котором будет размещен раздел для создания новых задач. Ниже приведен макет того, как мы хотим, чтобынаш шаблон домашней страницы выглядел:

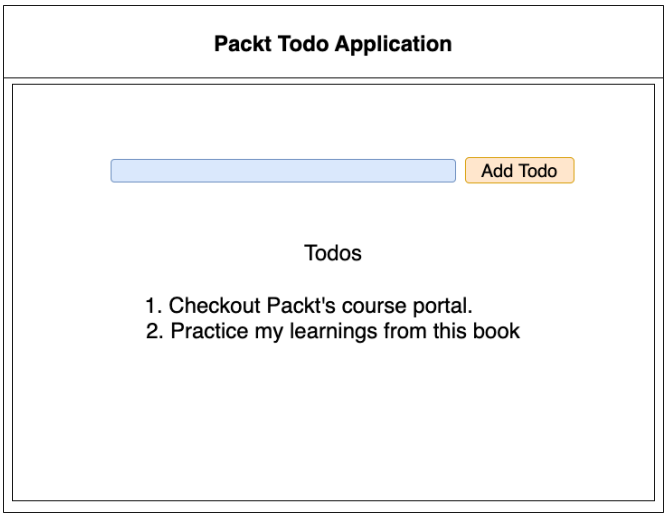

Рисунок 6.1 – Макет нашего шаблона домашней страницы

1. Во-первых, давайте установим пакет Jinja и создадим папку `templates`:

```
(venv)$ pip install jinja2
(venv)$ mkdir templates
```

2. Во вновь созданной папке создайте два новых файла, `home.html` и `todo.html`:

```
(venv)$ cd templates
(venv)$ touch {home,todo}.html
```

В предыдущем командном блоке мы создали два файла шаблона:

- `home.html` для главной страницы приложения
- `todo.html` для страницы задач

В макете на *Рисунке 6.1*, внутреннее поле обозначает шаблон todo, а большее поле — шаблон домашней страницы.

Прежде чем перейти к созданию наших шаблонов, давайте настроим Jinja в нашем приложении FastAPI:

1. Давайте изменим `POST` маршрут компонента API задач, `todo.py`:

```
from fastapi import APIRouter, Path, HTTPException,
status, Request, Depends
from fastapi.templating import Jinja2Templates
from model import Todo, TodoItem, TodoItems
todo_router = APIRouter()
todo_list = []

templates = Jinja2Templates(directory="templates/")

@todo_router.post("/todo")
async def add_todo(request: Request, todo: Todo =
Depends(Todo.as_form)):
    todo.id = len(todo_list) + 1
    todo_list.append(todo)
    return templates.TemplateResponse("todo.html",
    {
        "request": request,
        "todos": todo_list
    })
```

2. Затем обновите маршруты `GET`:

```
@todo_router.get("/todo", response_model=TodoItems)
async def retrieve_todo(request: Request):
    return templates.TemplateResponse("todo.html", {
        "request": request,
        "todos": todo_list
    })


@todo_router.get("/todo/{todo_id}")
async def get_single_todo(request: Request, todo_id: int
= Path(..., title="The ID of the todo to retrieve.")):
```

```
    for todo in todo_list:
        if todo.id == todo_id:
            return templates.TemplateResponse(
            "todo.html", {
                "request": request,
                "todo": todo
            })
    raise HTTPException(
        status_code=status.HTTP_404_NOT_FOUND,
        detail="Todo with supplied ID doesn't exist",
    )
```

В предыдущем блоке кода мы настроили Jinja для просмотра каталога `templates` для обслуживания шаблонов, переданных в `templates`.

Метод `TemplateResponse()`.

Метод для добавления задачи также был обновлен, чтобы включить зависимость от переданного ввода. Зависимости будут подробно обсуждаться далее.

3. В `model.py`, добавьте выделенный код перед классом `Config`:

```
from typing import List, Optional

class Todo(BaseModel):
    id: Optional[int]
    item: str

    @classmethod
    def as_form(
        cls,
        item: str = Form(...)
    ):
        return cls(item=item)
```

Теперь, когда мы обновили наш код API, давайте напишем наши шаблоны. Мы начнем с написания базового шаблона `home.html` на следующем шаге.

4. В `home.html`, мы начнем с объявления типа документа:

```
<!DOCTYPE html>
<html lang="en">
    <head>
```

```
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible"
         content="IE=edge">
        <meta name="viewport" content="width=device-
         width, initial-scale=1.0">
        <title>Packt Todo Application</title>
        <link rel="stylesheet" href=
         "https://stackpath.bootstrapcdn.com/
         bootstrap/4.1.0/css/bootstrap.min.css"
         integrity="sha384-9gVQ4dYFwwWSjIDZ
         nLEWnxCjeSWFphJiwGPXr1jddIhOegi
         u1FwO5qRGvFXOdJZ4" crossorigin="anonymous">
        <link rel="stylesheet" href=
         "https://use.fontawesome.com/releases
         /v5.0.10/css/all.css" integrity="sha384-
         +d0P83n9kaQMCwj8F4RJB66tzIwOKmrdb46+porD/
         OvrJ+37WqIM7UoBtwHO6Nlg" crossorigin=
         "anonymous">
    </head>
```

5. Следующим шагом является написание содержимого для тела шаблона. В тело шаблона мы включим имя приложения под тегом `<header></header>`и ссылку на `todo_container` дочернего шаблона, заключенную в тег блока. Дочерний шаблон будет написан на *шаге 8*.

Включите следующий код сразу после тега `</head>` в файл шаблона `home.html`:

```
    <body>
        <header>
            <nav class="navar">
                <div class="container-fluid">
                    <center>
                        <h1>Packt Todo
                        Application</h1>
                    </center>
                </div>
            </nav>
        </header>
        <div class="container-fluid">
            {% block todo_container %}{% endblock %}
        </div>
    </body>
</html>
</html>
```

Выделенный код сообщает родительскому шаблону, что блок `todo_container` будет определяться дочерним шаблоном. Содержимое дочернего шаблона, содержащего блок `todo_container` и расширяющего родительский шаблон, будетотображаться там.

6. Чтобы увидеть изменения, активируйте виртуальную среду и запустите приложение:

```
$ source venv/bin/activate
(venv)$ uvicorn api:app --host=0.0.0.0 --port 8000
--reload
```

7. Откройте `http://127.0.0.1:8000/todo`, чтобы просмотреть изменения:

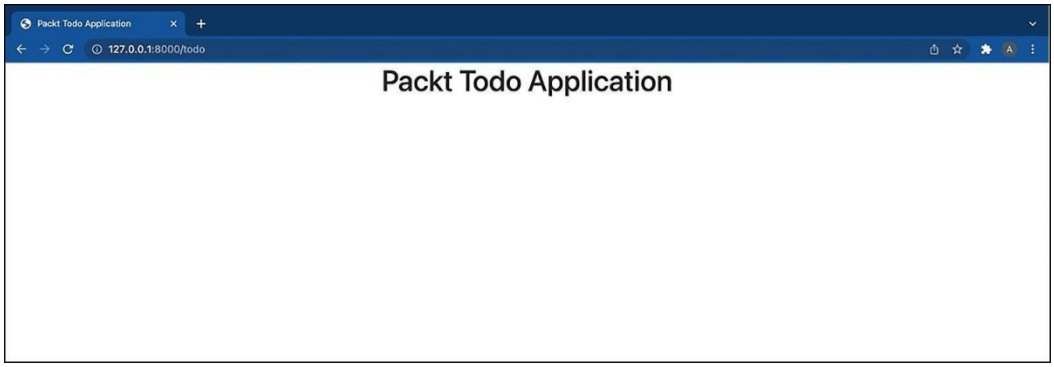

Рисунок 6.2 – Домашняя страница приложения Todo

8. Далее давайте напишем шаблон todo в `todo.html`:

```
{% extends "home.html" %}

{% block todo_container %}
<main class="container">
    <hr>
    <section  class="container-fluid">
        <form method="post">
            <div class="col-auto">
                <div class="input-group mb-3">
                    <input type="text" name="item"
                     value="{{ item }}" class="form-
                     control" placeholder="Purchase
                     Packt's Python workshop course"
                     aria-label="Add a todo"
                     aria-describedby="button-addon2"   />
                    <button class="btn btn-outline-
                     primary" type="submit" id=
                     "button-addon2" data-mdb-ripple-
                     color="dark">
                        Add Todo
```

```
                    </button>
                </div>
            </div>
        </form>
    </section>
    {% if todo %}
        <article class="card container-fluid">
            <br/>
            <h4>Todo ID: {{ todo.id }} </h4>
            <p>
                <strong>
                    Item: {{ todo.item  }}
                </strong>
            </p>
        </article>
    {% else %}
         <section  class="container-fluid">
             <h2 align="center">Todos</h2>
            <br>
            <div class="card">
                <ul class="list-group list-group-
                 flush">
                    {% for todo in todos %}
                        <li class="list-group-item">
                        {{ loop.index }}. <a href=
                        "/todo/{{loop.index}}"> {{
                        todo.item }} </a>
                    </li>
                {% endfor %}
            </ul>
        </div>
    {% endif %}
</section>
</main>
{% endblock %}
```

В предыдущем блоке кода шаблон todo наследует шаблон домашней страницы. Мы также определили блок `todo_container`, содержимое которого будет отображаться в родительском шаблоне.

Шаблон задачи используется как для получения всех задач, так и для одной задачи. В результате шаблон отображает различный контент в зависимости от используемого маршрута.

В шаблоне Jinja проверяет, передается ли переменная todo с помощью блока `{% if todo %}`. Подробная информация о задаче отображается, если передается переменная задачи, в противном случае она отображает содержимое в блоке `{% else %}`, который является списком.

9. Обновите веб-браузер, чтобы просмотреть последние изменения:

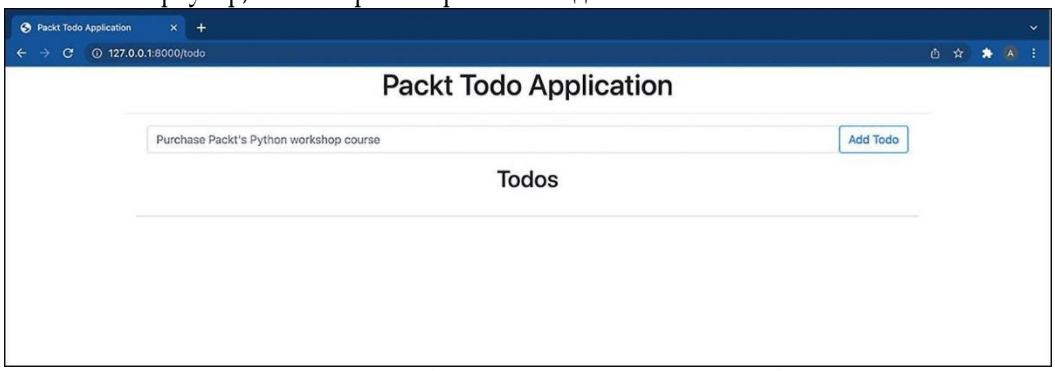

Рисунок 6.3 – Обновите домашнюю страницу todo

10. Давайте добавим задачу, чтобы убедиться, что домашняя страница работает должным образом:

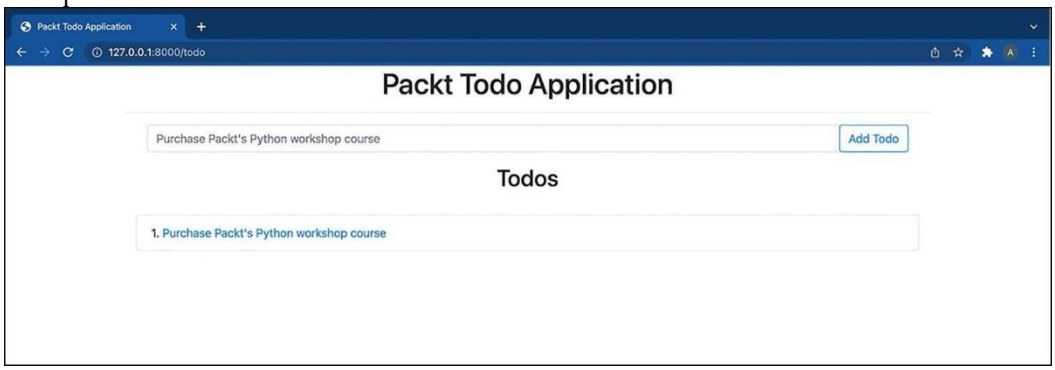

Рисунок 6.4 – Отображаемый список задач

11. Задача кликабельна. Нажмите на todo, и вы должны увидеть следующую страницу:

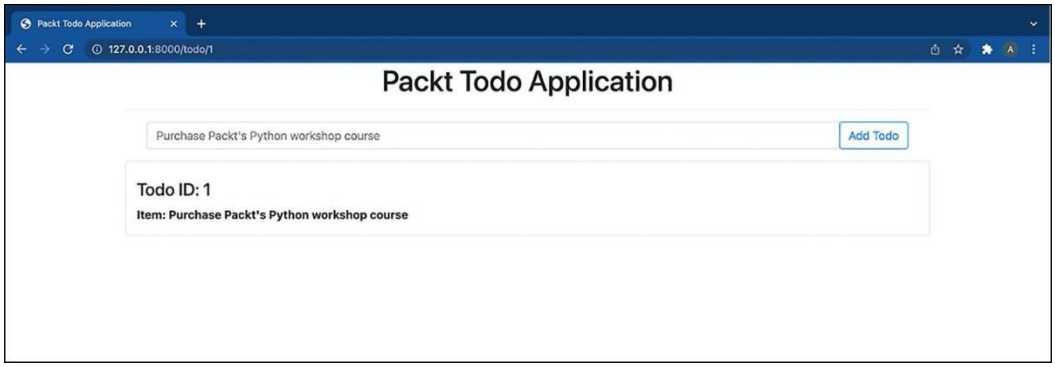

Рисунок 6.5 – Одна страница задач

Мы успешно добавили шаблон в наше приложение FastAPI.

Задание

1. Изучить, что такое шаблоны, основы системы шаблонов Jinja и как использовать ее в FastAPI.
2. Определить, какой контент отображать.
3. Определить как наследовать шаблоны
4. Выполнить наследование шаблонов на примере шаблонов главной страницы и задач.,