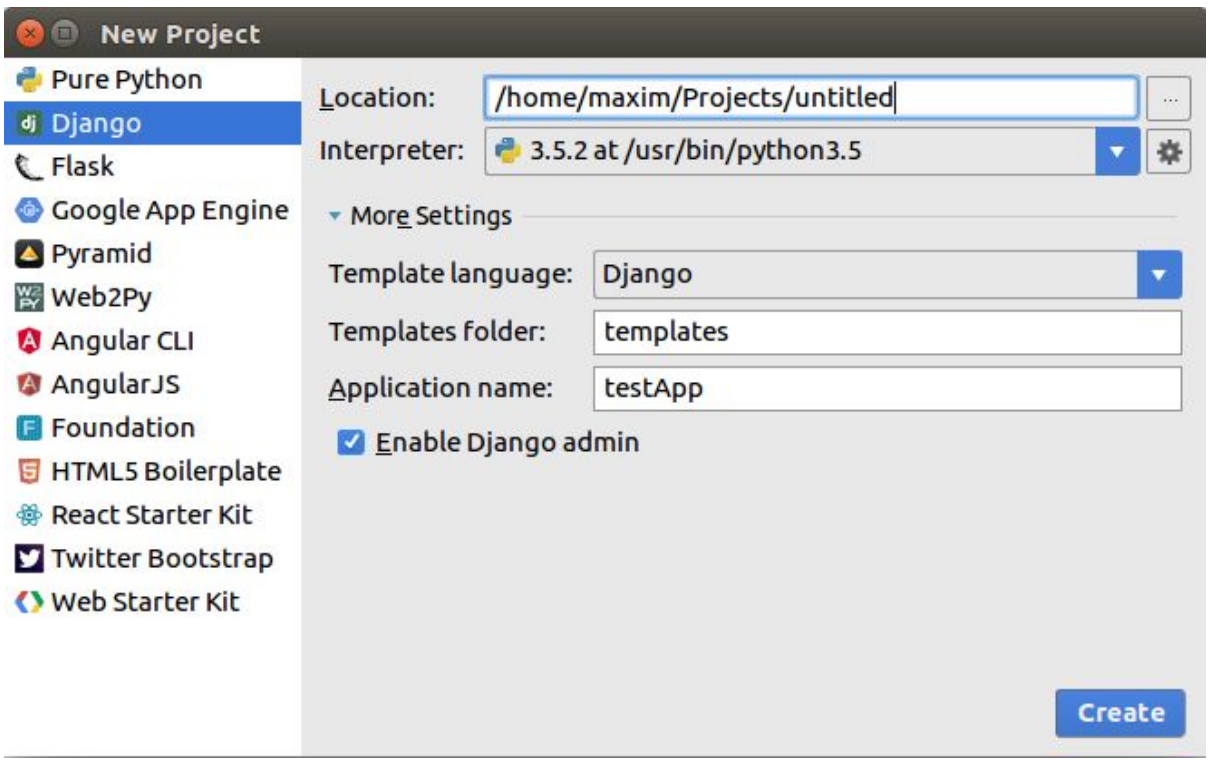
Лабораторная работа №5. Шаблонизация

Задание и порядок выполнения

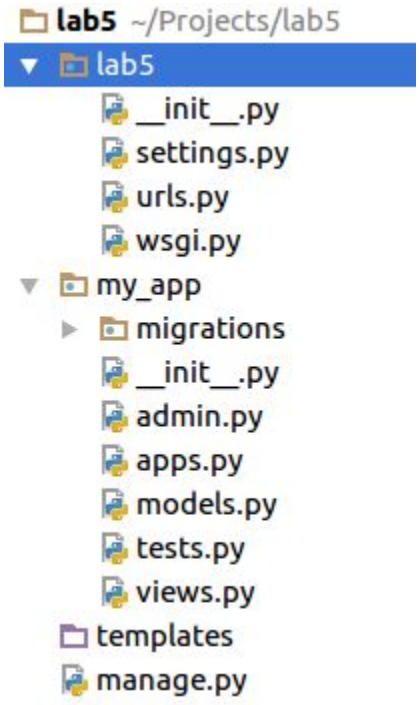
В этой ЛР вы создадите Django-проект, покажете пользователю статичную страницу, познакомитесь с конструкциями шаблонизаторов: переменные, теги, наследование шаблонов.

- Создать проект
- Реализовать view, в которых генерируются html-страницы
- В шаблонах должны быть использованы рассмотренные конструкции: переменные, вложенные значения, циклы, условия
- Все шаблоны должны расширять базовый шаблон
- Для элементов списка использовать тег include
- По нажатии на элемент списка должна открываться страница информации об элементе
- Для верстки необходимо использовать Bootstrap

Создание проекта



Структура проекта



В папке проекта:

settings.py - настройки проекта, в проекте может быть несколько приложений

urls.py - соответствие урлам обработчиков(views).

В пакете *my_app*:

views - обработчики приложения

другие файлы будут рассмотрены в других ЛР

templates - папка для шаблонов (html-файлы)

если не ultimate-версия pycharm:

создать django-проект не выйдет, создаем и запускаем руками
https://docs.djangoproject.com/en/1.10/intro/tutorial01/

для написания кода можно использовать любой pycharm

Пример использования Views

В urls.py

```
url(r'^function_view/', function_view), # functional view
url(r'^class_based_view/', ExampleClassBased.as_view()), #class based view
```

В views.py

```python
def function_view(request):
    return HttpResponse('response from function view')


class ExampleClassBased(View):
    def get(self, request):
        return HttpResponse('response from class based view')
```

Запускаем сервер.
Таким образом, код **function_view** будет вызван при обращении к серверу по урлу **/function_view**
Если использовать **class based views** то переопределяются методы **'get'**, **'post'**, **'put'**, **'patch'**, **'delete'**, **'head'**, **'options'**, **'trace'**
тогда, при обращении по урлу **/class_based_view** будет вызван код метода, которым было произведено обращение.

Шаблонизация

Шаблоном является обычный текстовой файл, чаще всего html, который содержит в себе переменные и теги. Эти конструкции при отрисовке шаблона подменяются на данные, которые пользователь передает в качестве параметров шаблонизации. В конечном итоге у нас получается html-файл, который может быть показан в браузере пользователя.

Для начала будем отдавать пользователю статичную html-страницу, в ней не будет никаких конструкций шаблонизатора.

Создаем html-файл в директории ‘templates’.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    static page
</body>
</html>
```

Чтобы вернуть пользователю созданный файл, использует метод render. Например данный код возвращает страницу example.html, которая была создана в папке templates.

```python
class ExampleView(View):
    def get(self, request):
        return render(request, 'example.html')
```

Переменные

В шаблоне переменная имеет вид: {{ some_variable }}. Когда шаблонизатор рендерит страницу и находит переменную, то вместо нее он подставляет результат, который вычисляется в этой переменной. Добавим в статичную страницу переменные.

```
<body>
    page with variable
    {{ my_variable }}
</body>
```

Чтобы передать значение переменной из кода:

```
render(request, 'example.html', {'my_variable': 'Этот текст подставится вместо переменной'})
```

Если значение переменной не было передано, то она будет заменена пустой строкой. В именах переменных не может быть пробелов или знаков препинания.

В качестве переменной может быть словарь, тогда к вложенным полям можно обращаться через точку.

```
render(request, 'example.html', {'dict': {'inner': 'a'}})
```

```
{{ dict.inner }}
```

Теги

В шаблоне теги выглядят как {% tag %}. С помощью тегов можно реализовывать условия, циклы, свою логику. Большинство тегов должны закрываться:
{% tag %} content {% endtag %}

Допустим, нам нужно вывести список элементов. Для этого воспользуемся **{% for %}**

```
<body>
    <ul>
    {% for element in list %}
        <li>{{ element }}</li>
    {% endfor %}
    </ul>
</body>
```

for итерируется по списку, доступ к элементам можно получать через созданную переменную, в данном случае ‘element’.

Существует конструкция {%for%}...{%empty%}...{%endfor%} - если элементов в списке не оказалось, то будут отрендерены элементы которые находятся после empty.

```
<ul>
{% for element in list %}
    <li>{{ element }}</li>
{% empty %}
    пустой список
{% endfor %}
</ul>
```

{% if variable %} позволяет выводить содержимое блока, если значение переменной “true” (или значение существует, либо если список и он не пустой)
Вместе с этим тегом можно использовать {% elif %}, {% else %}

```
{% for element in list %}
    {% if element == '2' %}
        <li>Двойка</li>
    {% elif element == '3' %}
        <li>Тройка</li>
    {% else %}
        <li>{{ element }}</li>
    {% endif %}
{% empty %}
    пустой список
{% endfor %}
```

В теге if можно использовать:

- and
- or
- not
- операторы сравнения
- in (проверка что значение существует в списке)

В шаблон можно добавлять комментарии с помощью {# comment #}

Остальные теги доступны на
https://docs.djangoproject.com/el/1.10/ref/templates/builtins/#built-in-template-tags-and-filters

Наследование шаблонов

Наследование шаблонов позволяет создать основной шаблон, который содержит общие элементы, а частные места будут переопределять наследники. Места, которые могут быть переопределены помечаются тегами {% block %}

Допустим, есть базовый шаблон base.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <div>Общая часть</div>
    <div>
    {% block body %}Переопределяется в наследниках{% endblock %}
    </div>
</body>
</html>
```

Здесь определена часть, которая будет присутствовать у нас на каждой странице, которая наследуется от него.

Создадим наследника orders.html, который будет выводить список заказов:

```
{# Наследуемся от базового #}
{% extends 'base.html' %}

{# Подставится вместо блока title в базовом шаблоне #}
{% block title %}Заказы{% endblock %}

{% block body %}
    <ul>
        {% for order in orders %}
            <li><a href="{% url 'order_url' order.id %}">{{ order.title }}</a></li>
        {% empty %}
            <li>пустой список</li>
        {% endfor %}
    </ul>
{% endblock %}
```

Тогда рендерим список заказов:

```
class OrdersView(View):
    def get(self, request):
        data = {
            'orders': [
                {'title': 'Первый заказ', 'id': 1},
                {'title': 'Второй заказ', 'id': 2},
                {'title': 'Третий заказ', 'id': 3}
            ]
        }
        return render(request, 'orders.html', data)
```

Сделаем ссылки, которые будут вести на страницы отдельных заказов:

```
{% for order in orders %}
    <li><a href="{% url 'order_url' order.id %}">{{ order.title }}</a></li>
{% empty %}
    <li>пустой список</li>
{% endfor %}
```

В файле urls.py необходимо определить url с именем ‘order_url’, который будет принимать id заказа:

```
url(r'^order/(?P<id>\d+)', OrderView.as_view(), name='order_url'),
```

OrderView рендерит шаблон страницы заказа “order.html”:

```
class OrderView(View):
    def get(self, request, id):
        data = {
            'order': {
                'id': id
            }
        }
        return render(request, 'order.html', data)
```

order.html:

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

{% block title %}Заказ № {{ order.id }}{% endblock %}

{% block body %}Страница заказа №{{ order.id }}{% endblock %}
```

Include

Позволяет рендерить в месте использования тега другой шаблон

```
{% block body %}
    <ul>
        {% for order in orders %}
           {% include 'order_element.html' with element=order %}
        {% empty %}
            <li>пустой список</li>
        {% endfor %}
    </ul>
{% endblock %}
```

with - позволяет сменить контекст. Т.е. в шаблоне *order_element.html* будут доступны не только *order, orders* , но и *element*

order_element.html:

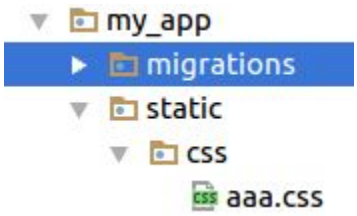
```
<li><a href="{% url 'order_url' element.id %}">{{ element.title }}</a></li>
```

Подключение статических файлов

В settings.py путь до файлов:

```
STATIC_URL = '/static/'
```

В приложении создаем папку и кладем туда файлы:



Перед использованием ссылки на статический файл, в шаблоне:

```
{% load static %}
```

Используем ссылку:

```
<link rel="stylesheet" type="text/css" href="{% static 'css/aaa.css' %}">
```

Вспомогательные материалы

Больше информации по шаблонизации:
https://docs.djangoproject.com/el/1.10/ref/templates/

Контрольные вопросы

1. Что такое шаблон? Что происходит при отрисовке шаблона?
2. Перечислите основные конструкции шаблонов.
3. Каково отличие переменной от тега?
4. Для чего применяется наследование шаблонов? Какие теги при этом используются и для каких целей?
5. Для чего применяется virtualenv
6. Какие действия нужно произвести, чтобы вернуть пользователю строку *“Привет”* по урлу */hello* ?