Django Starter

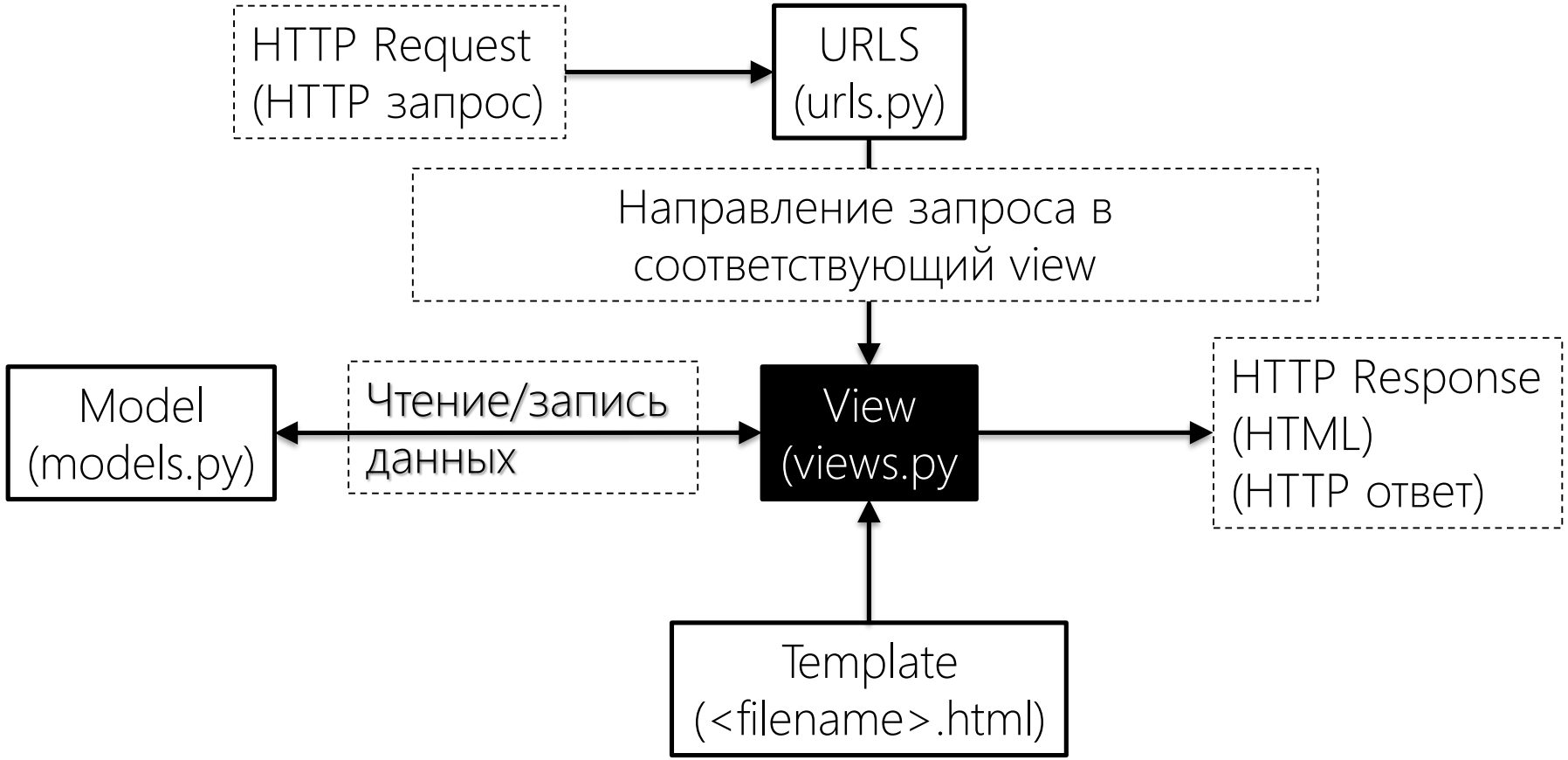
Представления

Представления

План урока

1. Function Based Views
2. Взаимодействие представления и модели
3. Class Based Views

Структура файлов в реализации MVC в Django (MTV)



Function Based Views

view (представление) — это функции/методы в Python, которые отвечают за отображение. Чаще всего одно view соответствует одной странице. Вот некоторые примеры view:

- "домашняя" страница;
- страница комментариев;
- список записей в блоге.

Представления обычно размещаются в файле **views.py**

Function Based Views – это представления на основе функций, в противоположность **Class Based Views** – представлениям на основе методов, являющихся частью класса.

Требования к Function Based Views

Основные требования к функции представления в Django:

- принятие первым аргументом объекта HttpRequest с информацией о запросе;
- возвращение объекта response.

Объект response – это то, что получит клиент, обратившись по этому адресу.

Например, это может быть:

- HTML содержимое веб-страницы;
- перенаправление на другую страницу;
- Ошибка 404;
- XML документ;
- Картинка.

Пример Function Based Views

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world! Вы на странице 3 урока.")
```

index – наша функция представления, **request** – это переменная типа HttpRequest, которая автоматически передается при обращении к данному представлению.

Класс HttpResponse определён в модуле django.http. Он позволяет возвращать ответ в виде строки (и не только). Также есть дополнительные аргументы, с которыми можно создавать объект HttpResponse.

Конструктор HttpResponse

HttpResponse(content=b'', content_type=None, status=200, reason=None, charset=None)

content – непосредственное содержимое в виде строки байтов. Если передать str, то он будет автоматом перекодирован в строку байтов с использованием utf-8.

content_type – используется для определения HTTP Content-Type заголовка. По умолчанию равен "text/html; charset=utf-8".

status - Код состояния HTTP. По умолчанию это код 200 - OK («хорошо»).

reason – текстовое описание кода состояния. Например для кода 200 это "OK".

charset – кодировка ответа. По умолчанию используется переменная DEFAULT_CHARSET из настроек – обычно это 'utf-8'.

Использование атрибутов HttpResponse

У HttpResponse также можно менять параметры возвращаемого объекта с помощью атрибутов, они соответствуют аргументам, которые могут быть указаны в конструкторе:
content, status , reason_phrase, charset. Например:

```
def something_not_found(request):
    respone = HttpResponse("Not found what you want.")
    response.status = 404
    return response
```

Вместо **404** лучше использовать Http.HTTPStatus класс из стандартной библиотеки **Python**: HTTPStatus.NOT_FOUND. Все доступные статусы можно посмотреть по ссылке:
https://docs.python.org/3/library/http.html#http.HTTPStatus

Альтернативы HttpResponse

Есть подклассы HttpResponse, позволяющие проще задавать частые ответы:

HttpResponseRedirect – перенаправление. Первым аргументом указывается **URL**, на который должно быть произведено перенаправление. **Http status** устанавливается в **302**.

FileResponse – оптимизирован для возвращения бинарных файлов. В качестве первого аргумента передается объект типа файл (точнее типа io.BytesIO). После окончания передачи, автоматически вызывается метод close данного объекта, поэтому нет необходимости в использовании контекстного менеджера. Пример использования:

```
def something_not_found(request):
    return FileResponse(open('myfile.png', 'rb'))
```

Взаимодействие представления и модели

ORM (Object-Relation Mapping, Объектно-реляционное отображение) – это связь между объектами и базой данных. Представления используют **ORM** для доступа к **БД**. В Django это обычно это **Django ORM**. С его помощью мы получаем или сохраняем данные в **БД**. Например, так мы можем вывести список объектов из таблицы Question (подробнее в уроке 4):

```
from django.http import HttpResponse
from .models import Question


def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    output = ', '.join([q.question_text for q in latest_question_list])
    return HttpResponse(output)
```

Class Based Views

Представления, основанные на классах, представляют собой альтернативный путь реализации представлений. Вместо функций здесь используются объекты Python.

Основные отличия от представлений, основанных на функциях:

- Организация обработки специфичных для HTTP методов (GET, POST, и т.д.) разнесена по соответствующим методам, вместо того, чтобы писать кучу условий.

- Объектно-ориентированные технологии, такие как миксины (примеси) и множественное наследование позволяют выделять код в компоненты, которые могут повторно использоваться.

Пример реализации Class Based Views

Реализация GET через функцию-представление:

```
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # <view logic>
        return HttpResponse('result')
```

Реализация GET через класс-представление:

```
from django.http import HttpResponse
from django.views.generic import View

class MyView(View):
    def get(self, request):
        # <view logic>
        return HttpResponse('result')
```

Метод as_view у Class Based Views

Поскольку роутинг Django отправляет запрос и ассоциированные с ним аргументы в вызываемую функцию, а не класс, все классы имеют статичный метод as_view(), который указан как точка входа при вызове этого класса. В нём создаётся экземпляр вашего класса и вызывается метод dispatch(), в котором определяется тип запроса (GET, POST и др.) и вызывается соответствующий метод. Если такового не нашлось, то происходит исключение HttpResponseNotAllowed:

```
# urls.py
urlpatterns = [
    url(r'^about/', MyView.as_view()),
]
```

Метод dispatch и HTTP методы у Class Based Views

Метод dispatch() определяет тип запроса и вызывает соответствующий метод. По умолчанию он настроен вызывать методы со следующими именами (если они определены): get, post, put, patch, delete, head, options, trace.

Реализуем с помощью Class-Based View тот же метод, который мы делали на основе представлений:

```
from django.http import HttpResponse
from django.views.generic import View

class MyView(View):
    def get(self, request):
        return HttpResponse("Hello, world! Вы на странице 3 урока.")
```

План урока

1. Function Based Views
2. Взаимодействие представления и модели
3. Class Based Views

Проверка знаний

TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!

Лазорык Михаил
Software developer

tieto

django

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association