**Джанго**

/Установка Питона/

Просмотр библиотек pip

pip3 install django установка джанго

**django-admin проверка установки**

1.4 Создание проекта на Django

Папка MyDjangoProject

(base) lemon@MacBook-Air-Lemon Джанго % cd /Users/lemon/Desktop/Курсы/Джанго

(base) lemon@MacBook-Air-Lemon Джанго % mkdir MyDjangoProject

(base) lemon@MacBook-Air-Lemon Джанго % ls

MyDjangoProject Джанго курс.docx

~$анго курс.docx

(base) lemon@MacBook-Air-Lemon Джанго % cd mkdir MyDjangoProject

cd: string not in pwd: mkdir

(base) lemon@MacBook-Air-Lemon Джанго % cd MyDjangoProject

(base) lemon@MacBook-Air-Lemon MyDjangoProject %

**Первый проект в папке MyDjangoProject**

django admin

django-admin startproject first

django-admin startproject second

django-admin startproject blog

cd /Users/lemon/Desktop/Курсы/Джанго

mkdir **myBestDjangoProjects**

cd **myBestDjangoProjects**

django-admin startproject **animal\_blog**

django-admin startproject **appleStore**

django-admin startproject **librarryApp**

**ls**

**1.5 Состав проекта Django**

***Установка локального виртуального окружения venv***

*Команд Б Проект сеттингс питон сеттингс селект интерпретатор добавить новый интерпр ОК*

*Если нет Джанго нажать + инсталл джанго*

/Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject

1.6 Запускаем локальный сервер разработки Django

pip3 install django

python3 manage.py

python3 manage.py runserver

Starting development server at <http://127.0.0.1:8000/>

localhost:8000/

^C

python manage.py migrate

127.0.0.1

1.7 Проект состоит из приложений

python manage.py

python manage.py startapp video

python manage.py startapp users

Итог:

Создаем проект, папка Проект, внутри папки будет создана одноименная папка

django-admin startproject blog

python3 manage.py runserver

Создаем приложения внутри папки

python manage.py startapp video

Подключаем приложения в настройках проекта setting

**2.1 Введение в модуль "Urls и Views"**

Новый проект

django-admin

django-admin startproject my\_page

Открываем в пайчарме проект my\_page

Терминал

cd /Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject/my\_page

python manage.py startapp horoscope

python manage.py migrate

python3 manage.py runserver

**Задание 2.1**

Ваша задача по примеру из видео создать новый проект с любым названием (в видео проект назывался **my\_page**).

cd /Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject

ls

django-admin

django-admin startproject my\_page\_home

Открываем в пайчарме проект my\_page\_home

Терминал

cd /Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject/my\_page\_home

python manage.py migrate

python manage.py startapp horoscope

python3 manage.py runserver

<http://127.0.0.1:8000/>

python manage.py startapp **week\_days**

python3 manage.py runserver

<http://127.0.0.1:8000/>

из файлов apps.py в horoscope и week\_days вставляем имена приложений в settings.py папки my\_page\_home в INSTALLED\_APPS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2.2 Главное об URL и View(представление)**

Сайт Доменное имя/корневой URL После ДИ Роут URL

Во View код для каждого URL функция или класс

2.3 Создаем Url и View

cd /Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject/my\_page

python3 manage.py runserver

http://127.0.0.1:8000/ horoscope/leo

Переходим в файл my\_page.urls.py

urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("horoscope/leo/", представление),

Переходим в файл my\_views.py

from django.shortcuts import render  
from django.http import HttpResponse  
  
  
# Create your views here.  
  
def leo(request):  
 return HttpResponse("Знак зодиака Лев")

Переходим в файл my\_page.urls.py

from django.contrib import admin  
from django.urls import path  
from horoscope.views import leo  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("horoscope/leo/", leo),

]

Просмотрели в браузере код страницы

Добавляем скорпиона

файл my\_views.py

def leo(request):  
 return HttpResponse("Знак зодиака Лев")  
  
def scorpio(request):  
 return HttpResponse("Знак зодиака Скорпион")

urls.py

from django.contrib import admin  
from django.urls import path  
from horoscope.views import leo, scorpio  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("horoscope/leo/", leo),  
 path("horoscope/scorpio/", scorpio),  
]

правильная форма

from horoscope import views  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("horoscope/leo/", views.leo),  
 path("horoscope/scorpio/", views.scorpio),

]

**Коды статусов ответа HTTP**

Код статуса представляет собой ответ сервера, который состоит из трёх цифр. Первая определяет класс ответа, а две другие указывают конкретный ответ.

Существует пять основных категорий кодов статусов. Все они приведены ниже:

1. Информационные 1XX
2. Успешные 2XX
3. Перенаправления 3XX
4. Клиентские ошибки 4XX
5. Серверные ошибки 5XX

Более подробно можно почитать [здесь](https://developer.mozilla.org/ru/docs/Web/HTTP/Status)

**Методы HTTP запроса**

HTTP определяет множество **методов запроса**, которые указывают, какое желаемое действие выполнится для данного ресурса. Изначально (в начале 90-х) предполагалось, что клиент может хотеть от ресурса только одно — получить его, однако сейчас по протоколу HTTP можно создавать посты, редактировать профиль, удалять сообщения и многое другое. И эти действия сложно объединить термином «получение».  
  
Для разграничения действий с ресурсами на уровне HTTP-методов и были придуманы следующие варианты:

* GET — получение ресурса (часто используется)
* POST — создание ресурса (часто используется)
* PUT — обновление ресурса
* DELETE — удаление ресурса

Но существуют и другие методы, подробнее о них можно почитать [здесь](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods)

Дом задание

urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("horoscope/leo/", views.leo),  
 path("horoscope/scorpio/", views.scorpio),  
 path("horoscope/aries/", views.aries),  
 path("horoscope/taurus/", views.taurus),  
 path("horoscope/gemini/", views.gemini),  
 path("horoscope/cancer/", views.cancer),  
 path("horoscope/virgo/", views.virgo),  
 path("horoscope/libra/", views.libra),  
 path("horoscope/sagittarius/", views.sagittarius),  
 path("horoscope/aquarius/", views.aquarius),  
 path("horoscope/pisces/", views.pisces),  
 path("horoscope/capricorn/", views.capricorn),  
]

def хххх(request):  
 return HttpResponse("Знак зодиака хххх")

Задание 2.3.7

urls.py

from week\_days import views as views\_week\_days  
  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("todo\_week/monday", views\_week\_days.monday),  
 path("todo\_week/tuesday", views\_week\_days.tuesday),  
]

my\_views.py

from django.http import HttpResponse  
  
  
# Create your views here.  
  
def monday(request):  
 return HttpResponse("Дел много")  
  
  
def tuesday(request):  
 return HttpResponse("Дел еще больше")

cd /Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject/my\_page\_home

python3 manage.py runserver

**2.4** **Создаем собственный URLconf**

В папке horoscope создаем файл питон urls.py

# 2 файла urls.py … не запутаться бы

Изначальный в папке my\_page

from django.contrib import admin  
from django.urls import path, include

urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("horoscope/", include('horoscope.urls')),  
]

Новый в папке horoscope пишем код

from django.urls import path  
from . import views  
  
urlpatterns = [  
 path("leo/", views.leo),  
 path("scorpio/", views.scorpio),  
 path("aries/", views.aries),  
 path("taurus/", views.taurus),  
 path("gemini/", views.gemini),  
 path("cancer/", views.cancer),  
 path("virgo/", views.virgo),  
 path("libra/", views.libra),  
 path("sagittarius/", views.sagittarius),  
 path("aquarius/", views.aquarius),  
 path("pisces/", views.pisces),  
 path("capricorn/", views.capricorn),  
]

Запускаем python3 manage.py runserver, проверяем все ОК

Задание 2.4.4

Теперь когда мы научились создавать свои URLconf, давайте перенесем два [ранее созданных роута](https://stepik.org/lesson/681492/step/7?unit=680271) в конфиг своего приложения **week\_days**. Речь идет о роутах

* todo\_week/monday
* todo\_week/tuesday

в главном urls:

from django.contrib import admin  
from django.urls import path, include  
# from week\_days import views as views\_week\_days  
  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("week\_days/", include('week\_days.urls')),  
]

в my\_page\_home urls:

from django.contrib import admin  
from django.urls import path, include  
# from week\_days import views as views\_week\_days  
  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("week\_days/", include('week\_days.urls')),  
]

**2.5** **Отладка Django в Pycharm**

Установка отладчика Zodiac

**Установка локального виртуального окружения venv**

Команд Б Проект сеттингс питон сеттингс селект интерпретатор добавить новый интерпр ОК

**2.5 .2Отладка Django в Pycharm.2**

Настройки терминал нужно cmd.exe оставил как было /bin/zsh

Выход из режима пайтон exit()

**2.5.4 Проект «Мой блог»**

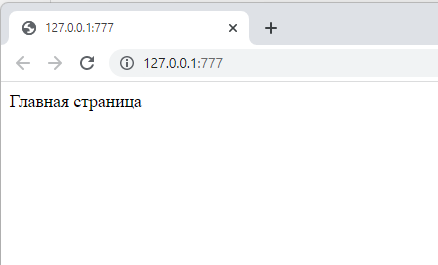
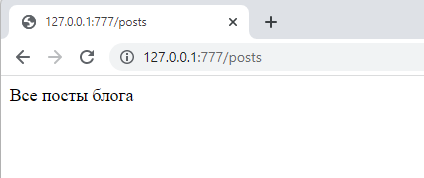
Начиная с этого урока мы будем применять все наши полученные знания для того, чтобы создать приложение-блог.

Для этого создайте новый проект в django, название можете выбрать любое, например my\_site.

Внутри проекта my\_site необходимо создать приложение blog

Необходимо настроить файл конфигурации запуска это проекта так, чтобы он запускался на порту 777. Это мы делаем, чтобы можно было параллельно запускать оба наших проекта. Они будут работать на разных портах и не мешать друг другу

Приложение blog должно обрабатывать два адреса:

* обращение к главной странице. Верните в качестве ответа текст «Главная страница»
* обращение по пути posts. Верните в качестве ответа текст «Все посты блога»

**Решение**

проект в папке MyDjangoProject

В пайчарме

cd /Users/lemon/Desktop/Курсы/Джанго/MyDjangoProject

django-admin

django-admin startproject my\_site

**Установка локального виртуального окружения venv**

Команд Б Проект сеттингс питон сеттингс селект интерпретатор добавить новый интерпр ОК

Открываем новый терминал с venv вначале

Находимся на

(venv) (base) lemon@MacBook-Air-Lemon my\_site %

pip3 install Django OK

python3 manage.py OK

python3 manage.py startapp blog OK

Проверка

python3 manage.py runserver ОК стр в браузере есть

В папке blog создаем файл питон urls.py

В папке my\_site urls.py

from django.contrib import admin  
from django.urls import path, include

urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("", include('blog.urls')),  
]

Переходим в файл my\_views.py

from django.shortcuts import render  
from django.http import HttpResponse  
  
  
# Create your views here.  
  
def posts(request):  
 return HttpResponse("Все посты блога")  
  
def empty(request):  
 return HttpResponse("Главная страница")

В urls.py в папке blog пишем код

from django.urls import path  
from . import views  
  
  
urlpatterns = [  
 path("posts/", views.posts),  
 path("", views.empty),  
]

В терминале:

python manage.py migrate

python3 manage.py runserver

Создаем отладчик Blog\_1

Проверяем отладчик через его запуск ОК

Меняем порт файл конфигурации на 777

777 не проходит на 7777 ОК

**2.6** **Динамический URL Django**

На примере проекта my\_page приложения horoscope

Динамический роут – неизвестно что придет в request – организация приема и обработки любого роута в request

Файл horoscope.urls.py

urlpatterns = [  
 path("leo/", views.leo),]

пишем

path("<sign\_zodiac>/", views. get\_info\_about\_sign\_zodiac),]

<sign\_zodiac> любой роут в <> джанго воспринимает как параметр == аналог переменной,

sign\_zodiac название параметра

Файл views.py

from django.http import HttpResponse, HttpResponseNotFound

def get\_info\_about\_sign\_zodiac(request, sign\_zodiac):  
 if sign\_zodiac == 'leo':  
 return HttpResponse("Лев - пятый знак зодиака, солнце (с 23 июля по 21 августа)")  
 else:  
 return HttpResponseNotFound("Неизвестный знак Зодиака")

Окончательный вид функции

def get\_info\_about\_sign\_zodiac(request, sign\_zodiac):  
 signs = {  
 "aries": "Овен - первый знак зодиака, планета Марс (с 21 марта по 20 апреля).",  
 "taurus": "Телец - второй знак зодиака, планета Венера (с 21 апреля по 21 мая).",  
 "gemini": "Близнецы - третий знак зодиака, планета Меркурий (с 22 мая по 21 июня).",  
 "cancer": "Рак - четвёртый знак зодиака, Луна (с 22 июня по 22 июля).",  
 "leo": "Лев - пятый знак зодиака, солнце (с 23 июля по 21 августа).",  
 "virgo": "Дева - шестой знак зодиака, планета Меркурий (с 22 августа по 23 сентября).",  
 "libra": "Весы - седьмой знак зодиака, планета Венера (с 24 сентября по 23 октября).",  
 "scorpio": "Скорпион - восьмой знак зодиака, планета Марс (с 24 октября по 22 ноября).",  
 "sagittarius": "Стрелец - девятый знак зодиака, планета Юпитер (с 23 ноября по 22 декабря).",  
 "capricorn": "Козерог - десятый знак зодиака, планета Сатурн (с 23 декабря по 20 января).",  
 "aquarius": "Водолей - одиннадцатый знак зодиака, планеты Уран и Сатурн (с 21 января по 19 февраля).",  
 "pisces": "Рыбы - двенадцатый знак зодиака, планеты Юпитер (с 20 февраля по 20 марта)."  
 }  
  
 if sign\_zodiac.lower() in signs:  
 return HttpResponse(signs[sign\_zodiac.lower()])  
 return HttpResponseNotFound(f"Неизвестный знак Зодиака - {sign\_zodiac}")

------------------------------------------

Задание 2.6.2

В  приложении **week\_days**замените ранее созданные роуты на один роут с динамическим URL.

Параметр данного роута должен принимать латинское название дня недели. В случае, если передается значение, не являющееся названием дня, сообщите пользователю об этом

Нет venv, устанавливаем

**Установка локального виртуального окружения venv**

Команд Б Проект:… питон interpret добавить новый интерпр ОК

pip3 install Django OK

python manage.py migrate

python3 manage.py runserver

----------

from django.urls import path  
from . import views  
  
urlpatterns = [  
 path("<day>/", views.get\_day),  
 # path("monday/", views.monday),  
 # path("tuesday/", views.tuesday),  
]

-------

from django.http import HttpResponse, HttpResponseNotFound

def get\_day(request, day):  
 dict\_day = {  
 'monday': 'Составляем план на неделю',  
 'tuesday': 'Пишем план на неделю',  
 'wednesday': 'Редактируем план на неделю',  
 'thursday': 'Согласуем план на неделю',  
 'friday': 'Вешаем план на стенку',  
 'saturday': 'Делаем пометки в плане',  
 'sunday': 'Обдумываем новый план',  
 }  
 if day.lower() in dict\_day:  
 return HttpResponse(dict\_day[day.lower()])  
 return HttpResponseNotFound(f"Нет такого дня в неделе - {day}")

-------------------------------- OK

Задание 2.6.3

### Проект «Мой блог»

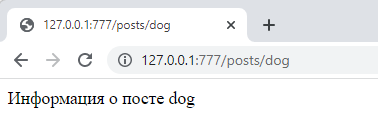
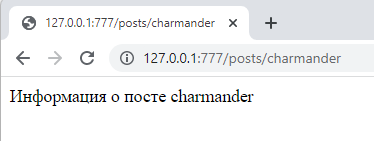
Возвращаемся к нашему блогу. Давайте добавим еще один роут, обрабатывающий адрес:

 posts/<name\_post>

Это динамический роут, который принимает в запросе название поста и возвращает ответ следующего вида:

Информация о посте {name\_post}

Вот примеры его работы

Решение

Главный urls

from django.contrib import admin  
# from django.urls import path  
from django.urls import path, include  
from blog import views

urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("", views.empty),  
 path("posts/", include('blog.urls')),  
]

Дочерний urls

from django.urls import path  
from . import views

urlpatterns = [  
 path("", views.posts),  
 path("<info>/", views.about\_info),  
]

Views

from django.http import HttpResponse, HttpResponseNotFound

def posts(request):  
 return HttpResponse("Все посты блога")  
  
def empty(request):  
 return HttpResponse("Главная страница")  
  
def about\_info(request, info):  
 return HttpResponseNotFound(f"Информация о посте {info}")

---------------------------- OK

**2.7** **Конвертеры роутов в Django**

Расположение конвертеров важно, обработка сверху вниз в списке путей

Стандартные str int и все, но можно создавать собственные конвертеры, отдельная тема вне курса

Проект my\_page horoscope

Главный urls без изменений

Дочерний urls

urlpatterns = [  
 path("<int:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac\_num),  
 path("<str:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac),  
]

Views

signs = {  
 "aries": "Овен - первый знак зодиака, планета Марс (с 21 марта по 20 апреля).",  
 "taurus": "Телец - второй знак зодиака, планета Венера (с 21 апреля по 21 мая).",  
 "gemini": "Близнецы - третий знак зодиака, планета Меркурий (с 22 мая по 21 июня).",  
 "cancer": "Рак - четвёртый знак зодиака, Луна (с 22 июня по 22 июля).",  
 "leo": "Лев - пятый знак зодиака, солнце (с 23 июля по 21 августа).",  
 "virgo": "Дева - шестой знак зодиака, планета Меркурий (с 22 августа по 23 сентября).",  
 "libra": "Весы - седьмой знак зодиака, планета Венера (с 24 сентября по 23 октября).",  
 "scorpio": "Скорпион - восьмой знак зодиака, планета Марс (с 24 октября по 22 ноября).",  
 "sagittarius": "Стрелец - девятый знак зодиака, планета Юпитер (с 23 ноября по 22 декабря).",  
 "capricorn": "Козерог - десятый знак зодиака, планета Сатурн (с 23 декабря по 20 января).",  
 "aquarius": "Водолей - одиннадцатый знак зодиака, планеты Уран и Сатурн (с 21 января по 19 февраля).",  
 "pisces": "Рыбы - двенадцатый знак зодиака, планеты Юпитер (с 20 февраля по 20 марта)."  
}  
  
def get\_info\_about\_sign\_zodiac(request, sign\_zodiac: str):  
 descr = signs.get(sign\_zodiac, None)  
 if descr:  
 return HttpResponse(descr)  
 else:  
 return HttpResponseNotFound(f"Неизвестный знак Зодиака - {sign\_zodiac}")  
  
  
def get\_info\_about\_sign\_zodiac\_num(request, sign\_zodiac: int):  
 return HttpResponse(f"This is the number {sign\_zodiac}")

Задание 2.7.3

### Приложение «Список дел на неделю»

В приложении **week\_days**добавьте еще один динамический URL с параметром целого типа. Если ему поступает число от 1 до 7, то он возвращает ответ "Сегодня {day} день недели", в остальных случаях - "Неверный номер дня - {day}", где day - значение параметра

/todo\_week/3 -> Сегодня 3 день недели

/todo\_week/7 -> Сегодня 7 день недели

/todo\_week/21 -> Неверный номер дня – 21

-------------

Решение

Главный urls без изменений

Дочерний urls

urlpatterns = [  
 path("<int:day>/", views.get\_day\_num),  
 path("<str:day>/", views.get\_day),  
]

Views

def get\_day(request, day:str):  
 dict\_day = {  
 'monday': 'Составляем план на неделю',  
 'tuesday': 'Пишем план на неделю',  
 'wednesday': 'Редактируем план на неделю',  
 'thursday': 'Согласуем план на неделю',  
 'friday': 'Вешаем план на стенку',  
 'saturday': 'Делаем пометки в плане',  
 'sunday': 'Обдумываем новый план',  
 }  
 if day.lower() in dict\_day:  
 return HttpResponse(dict\_day[day.lower()])  
 return HttpResponseNotFound(f"Нет такого дня в неделе - {day}")  
  
  
def get\_day\_num(request, day:int):  
 if 1 <= day <= 7:  
 return HttpResponse(f'Сегодня {day} день недели')  
 else:  
 return HttpResponse(f'Неверный номер дня - {day}')

------------------- ОК

Задание 2.7.4

### Приложение «Геометрия»

Создайте новое приложение geometry. В этом приложении нужно создать urls.py и затем его подключить в главному URLconf вашего проекта по следующему адресу:

# urls.py проекта

path('calculate\_geometry/', include('geometry.urls')),

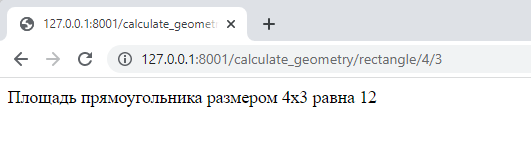
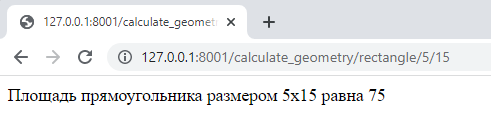
В файле urls.py приложения необходимо обработать три роута:

* rectangle/<ширина>/<высота>
* square/<ширина>
* circle/<радиус>

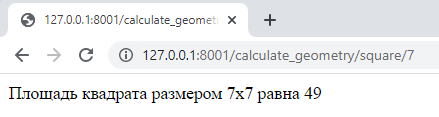
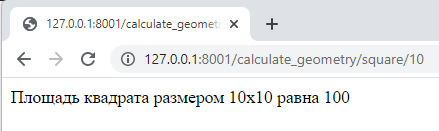
Каждый роут должен находить площадь соответствующий фигуры. Для этого вам потребуется реализовать в файле views.py  следующие функции:

* get\_rectangle\_area (площадь прямоугольника) принимает длину и ширину фигуры
* get\_square\_area (площадь квадрата) принимает размер стороны квадрата
* get\_circle\_area (площадь круга) принимает радиус круга

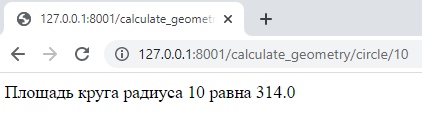
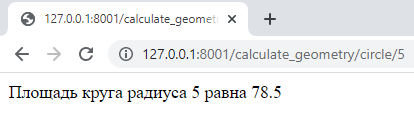
Ниже пример обработки роута прямоугольника

Ниже пример работы роута квадрата

А вот пример обращения к роуту круга

---------------------------

Решение

Создание приложения geometry в папке my\_page

Заходим в проект my\_page

Перемещаемся в терминале в папку my\_page

python3 manage.py

python3 manage.py startapp geometry OK

В geometry создаем папку urls.py ОК

В главный urls копируем путь

path('calculate\_geometry/', include('geometry.urls')),

Из urls гороскопа копируем нужное в urls геометри

from django.urls import path  
from . import views

Создаем список путей

urlpatterns = [  
 path("rectangle/<int:w>/<int:h>/", views.get\_rectangle\_area),  
 path("square/<int:s>/", views.get\_square\_area),  
 path("circle/<int:r>/", views.get\_circle\_area),  
]

В папке views

from django.http import HttpResponse

from math import pi

def get\_rectangle\_area(request, w, h):  
 ar = w \* h  
 return HttpResponse(f'Площадь прямоугольника размером {w}x{h} равна {ar}')  
  
  
def get\_square\_area(request, s):  
 ar = s \*\* 2  
 return HttpResponse(f'Площадь квадрата размером {s}x{s} равна {ar}')  
  
  
def get\_circle\_area(request, r):  
 ar = round(pi \* r \*\* 2, 1)  
 return HttpResponse(f'Площадь круга радиуса {r} равна {ar}')

--------------------------------- OK

Задание 2.7.5

### Проект «Мой блог»

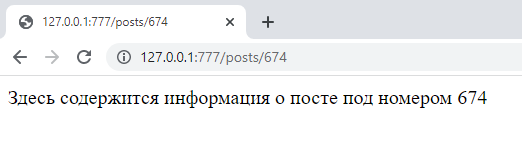
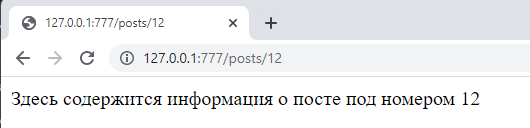
Возвращаемся к нашему блогу. Давайте добавим еще один роут, обрабатывающий адрес:

 posts/<int:number\_post>

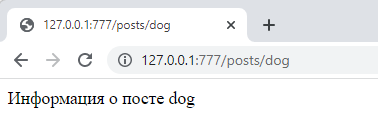
Это динамический роут, который принимает в запросе номер поста и возвращает ответ следующего вида:

Здесь содержится информация о посте под номером {number\_post}

Вот примеры его работы

Но не забывайте, что если указать не числовое значение, должен отображаться ранее созданное представление



Решение

urlpatterns = [  
 path("", views.posts),  
 path("<int:info>/", views.about\_info\_int),  
 path("<info>/", views.about\_info),  
]

def about\_info\_int(request, info):  
 return HttpResponseNotFound(f"Здесь содержится информация о посте под номером {info}")

--------------------------------- ОК

**2.8** **Redirect Url в Django**

Проект my\_page приложение Гороскоп

Views

from django.http import HttpResponse, HttpResponseNotFound, HttpResponseRedirect

def get\_info\_about\_sign\_zodiac\_num(request, sign\_zodiac: int):  
 if not 1 <= sign\_zodiac <= 12:  
 return HttpResponseNotFound(f"Неправильный порядковый номер знака зодиака - {sign\_zodiac}")  
 zodiacs = list(signs)[sign\_zodiac - 1]  
 return HttpResponseRedirect(f'/horoscope/{zodiacs}')

-------------------------------

Задание 2.8.3

from django.http import HttpResponse, HttpResponseNotFound, HttpResponseRedirect

def get\_day\_num(request, day: int):  
 if not 1 <= day <= 7:  
 return HttpResponse(f'Неверный номер дня - {day}')  
 day\_ind = list(dict\_day)[day - 1]  
 return HttpResponseRedirect(f'/week\_days/{day\_ind}')

Задание 2.8.4

Давайте вспомним приложение [«geometry»](https://stepik.org/lesson/681496/step/5?unit=680275). Необходимо добавить три роута

* get\_rectangle\_area/<int:width>/<int:height>
* get\_square\_area/<int:width>
* get\_circle\_area/<int:radius>

В представлении views.py сделайте перенаправление на соответствующий имеющийся из того задания роут

http://127.0.0.1:8000/calculate\_geometry/get\_rectangle\_area/10/5

должен перенаправить на

http://127.0.0.1:8000/calculate\_geometry/rectangle/10/5

http://127.0.0.1:8000/calculate\_geometry/get\_square\_area/10

должен перенаправить на

http://127.0.0.1:8000/calculate\_geometry/square/10

http://127.0.0.1:8000/calculate\_geometry/get\_circle\_area/4

должен перенаправить на

http://127.0.0.1:8000/calculate\_geometry/circle/4

Решение

urlpatterns = [  
 path("rectangle/<int:w>/<int:h>/", views.get\_rectangle\_area),  
 path("square/<int:s>/", views.get\_square\_area),  
 path("circle/<int:r>/", views.get\_circle\_area),  
 path("get\_rectangle\_area/<int:w>/<int:h>/", views.get\_rectangle\_area\_redir),  
 path("get\_square\_area/<int:s>/", views.get\_square\_area\_redir),  
 path("get\_circle\_area/<int:r>/", views.get\_circle\_area\_redir),  
]

def get\_rectangle\_area\_redir(request, w, h):  
 return HttpResponseRedirect(f'/calculate\_geometry/rectangle/{w}/{h}')  
  
  
def get\_square\_area\_redir(request, s):  
 return HttpResponseRedirect(f'/calculate\_geometry/square/{s}')  
  
  
def get\_circle\_area\_redir(request, r):  
 return HttpResponseRedirect(f'/calculate\_geometry/circle/{r}')

----------------------------- ОК

**2.9** **Функция reverse в Django**

Каждому url можно дать имя, зарегистрировать за ним определенное название

В дочернем url

urlpatterns = [  
 path("<int:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac\_num),  
 path("<str:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac, name='horoscope-name')  
]

Во views

from django.urls import reverse

def get\_info\_about\_sign\_zodiac\_num(request, sign\_zodiac: int):  
 if not 1 <= sign\_zodiac <= 12:  
 return HttpResponseNotFound(f"Неправильный порядковый номер знака зодиака - {sign\_zodiac}")  
 zodiacs = list(signs)[sign\_zodiac - 1]  
 redirect\_url = reverse('horoscope-name', args=[zodiacs])  
 return HttpResponseRedirect(redirect\_url)

в args должна передаваться коллекция

Задание 2.9.3

В приложении **week\_days**переделайте перенаправление на другой роут при помощи функции reverse

/todo\_week/1 -> перенаправляет на -> /todo\_week/monday

/todo\_week/7 -> перенаправляет на -> /todo\_week/sunday

/todo\_week/21 -> Неверный номер дня - 21

Не забудьте дать названия роутам

Решение:

urlpatterns = [  
 path("<int:day>/", views.get\_day\_num),  
 path("<str:day>/", views.get\_day, name='week\_day\_name'),  
]

---

def get\_day\_num(request, day: int):  
 if not 1 <= day <= 7:  
 return HttpResponse(f'Неверный номер дня - {day}')  
 day\_ind = list(dict\_day)[day - 1]  
 redirect\_num = reverse('week\_day\_name', args=[day\_ind])  
 return HttpResponseRedirect(redirect\_num)

------------------- OK

Задание 2.9.4

 приложение [«geometry»](https://stepik.org/lesson/681496/step/5?unit=680275) также не обходимо переписать перенаправление при помощи функции reverse

http://127.0.0.1:8000/calculate\_geometry/get\_rectangle\_area/10/5

должен перенаправить на

http://127.0.0.1:8000/calculate\_geometry/rectangle/10/5

http://127.0.0.1:8000/calculate\_geometry/get\_square\_area/10

должен перенаправить на

http://127.0.0.1:8000/calculate\_geometry/square/10

http://127.0.0.1:8000/calculate\_geometry/get\_circle\_area/4

должен перенаправить на

http://127.0.0.1:8000/calculate\_geometry/circle/4

Решение:

urlpatterns = [  
 path("rectangle/<int:w>/<int:h>/", views.get\_rectangle\_area, name='rectangle'),  
 path("square/<int:s>/", views.get\_square\_area, name='square'),  
 path("circle/<int:r>/", views.get\_circle\_area, name='circle'),  
 path("get\_rectangle\_area/<int:w>/<int:h>/", views.get\_rectangle\_area\_redir),  
 path("get\_square\_area/<int:s>/", views.get\_square\_area\_redir),  
 path("get\_circle\_area/<int:r>/", views.get\_circle\_area\_redir),  
]

def get\_rectangle\_area\_redir(request, w, h):  
 re\_dir = reverse('rectangle', args=[w, h])  
 return HttpResponseRedirect(re\_dir)  
  
  
def get\_square\_area\_redir(request, s):  
 re\_dir = reverse('square', args=[s])  
 return HttpResponseRedirect(re\_dir)  
  
  
def get\_circle\_area\_redir(request, r):  
 re\_dir = reverse('circle', args=[r])  
 return HttpResponseRedirect(re\_dir)

--------------- OK

**2.10** **Создаем главное меню**

Проект Гороскоп

Дочерний urls

urlpatterns = [  
 path('', views.index),  
 path("<int:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac\_num),  
 path("<str:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac, name='horoscope-name')  
]

Views

def index(request):  
 zodiacs = list(signs)  
 li\_elements = ''  
 for sign in zodiacs:  
 redirect\_path = reverse('horoscope-name', args=[sign])  
 li\_elements += f"<li> <a href='{redirect\_path}'> {sign.title()} </a> </li>"  
 responce = f'''  
 <ul>  
 {li\_elements}  
 </ul>  
 '''  
 return HttpResponse(responce)

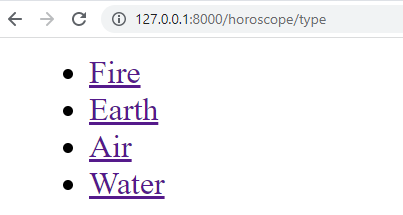
2.11 Итоги модуля. Практическое задание

### Задание 1 (обязательное)

Необходимо создать два роута

1) **/horoscope/type**

По обращению к нему, должна выводиться информация о четырех стихиях (огонь, вода, земля и воздух)



По обращению к любому из этих элементов мы должны переходить к роуту 2 (см. пункт 2)

2) **/horoscope/type/<element>**

Обращаясь по этому пути, мы бы хотели видеть список знаков зодиака, которые относятся к конкретной стихии. Например, /horoscope/type/fire должен показать нам список всех огненных знаков.



Каждый элемент списка должен являться ссылкой на описание самого знака. Также не забудьте обработать ситуацию, когда в адрес передана неправильная стихия

### Задание 2 (необязательное)

Создать роут, в котором принимается два целых числа - месяц и день

horoscope/<int:month>/<int:day>

Параметр month - целое число от 1 до 12

Параметр day - целое число от 1 до 28-31 в зависимости от номера месяца.

По переданной дате необходимо вычислить к какому знаку зодиака она относится

Решение заданий 1 и 2

urlpatterns = [  
 path('', views.index),  
 path('type/', views.index\_type),  
 path('type/<str:sign\_type>/', views.get\_info\_about\_sign\_type, name='type-name'),  
 path('<int:month>/<int:day>/', views.get\_info\_sign\_zodiac),  
 path("<int:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac\_num),  
 path("<str:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac, name='horoscope-name'),  
]

def index\_type(request):  
 li\_elements = ''  
 for sign in sign\_types\_lst:  
 redirect\_path = reverse('type-name', args=[sign])  
 li\_elements += f"<li> <a href='{redirect\_path}'> {sign.title()} </a> </li>"  
 responce = f'''  
 <ul>  
 {li\_elements}  
 </ul>  
 '''  
 return HttpResponse(responce)  
  
  
def get\_info\_about\_sign\_type(request, sign\_type: str):  
 descr = sign\_types.get(sign\_type, None)  
 if descr:  
 li\_elements = ''  
 for sign in descr:  
 redirect\_path = reverse('horoscope-name', args=[sign])  
 li\_elements += f"<li> <a href='{redirect\_path}'> {sign.title()} </a> </li>"  
 responce = f'''  
 <ul>  
 {li\_elements}  
 </ul>   
 '''  
 return HttpResponse(responce)  
 else:  
 return HttpResponseNotFound(f"Неизвестный знак Стихии - {sign\_type}")

def get\_info\_sign\_zodiac(request, month, day):  
 m = zodiacs[10:] + zodiacs[:10]  
 d = (20, 19, 21, 20, 21, 22, 23, 23, 23, 24, 23, 22)  
 month = month-1  
 if day > d[month]:  
 sign = m[month]  
 else:  
 sign = m[month-1]  
 re\_dir = reverse('horoscope-name', args=[sign])  
 return HttpResponseRedirect(re\_dir, f'{sign}')

**2.12 Бонус🚀 Создаем собственный конвертер**

Ссылка на документацию Джанго, отсюда пример

<https://django.fun/ru/docs/django/4.1/topics/http/urls/>

URLS

from django.urls import path, register converter \_   
from . import views, converters  
  
register\_converter(converters.FourDigitYearConverter, 'yyyy')  
register\_converter(converters.My\_Float\_Converter, 'my\_float')  
register\_converter(converters.MyDateConverter, 'my\_date')

urlpatterns = [  
 path('', views.index),  
 path("<my\_date:sign\_zodiac>/", views.get\_my\_date\_converters),  
 path('type/', views.index\_type),  
 path('type/<str:sign\_type>/', views.get\_info\_about\_sign\_type, name='type-name'),  
 path('<int:month>/<int:day>/', views.get\_info\_sign\_zodiac),  
 path("<yyyy:sign\_zodiac>/", views.get\_yyyy\_converters),  
 path("<int:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac\_num),  
 path("<my\_float:sign\_zodiac>/", views.get\_my\_float\_converters),  
 path("<str:sign\_zodiac>/", views.get\_info\_about\_sign\_zodiac, name='horoscope-name'),  
]

VIEWS

def get\_yyyy\_converters(request, sign\_zodiac):  
 return HttpResponse(f'Вы передали число из 4-х цифр - {sign\_zodiac}')  
  
  
def get\_my\_float\_converters(request, sign\_zodiac):  
 return HttpResponse(f'Вы передали вещественное число - {sign\_zodiac}')  
  
  
def get\_my\_date\_converters(request, sign\_zodiac):  
 return HttpResponse(f'Вы передали дату - {sign\_zodiac}')

CONVERTERS

from datetime import datetime  
  
  
class FourDigitYearConverter:  
 regex = '[0-9]{4}'  
  
 def to\_python(self, value):  
 return int(value)  
  
 def to\_url(self, value):  
 return '%04d' % value  
  
  
class My\_Float\_Converter:  
 regex = '[+-]?(\d\*\.)?\d+'  
  
 def to\_python(self, value):  
 return float(value)  
  
 def to\_url(self, value):  
 return str(value)  
  
  
class MyDateConverter:  
 regex = '(((0[1-9])|([12][0-9])|(3[01]))-((0[0-9])|(1[012]))-((20[012]\d|19\d\d)|(1\d|2[0123])))'  
  
 def to\_python(self, value):  
 return datetime.strptime(value, '%d-%m-%Y')  
  
 def to\_url(self, value):  
 return value.strftime('%d-%m-%Y')

---------------------OK

**2.13 Тестирование представлений**

Файл в приложении Гороскоп tests.py

1 Тест horoscope/' все знаки

from django.test import TestCase  
  
class TestHoroscope(TestCase):  
  
 def test\_index(self):  
 response = self.client.get('/horoscope/')  
 self.assertEqual(response.status\_code, 200)

В терминале manage.py test horoscope

Видим ОК

Если поставить 300 а не 200 будет не ОК

Добавляем еще один отладчик Test Zodiac

В параметрах меняем только test horoscope

2 Тест horoscope/libra

def test\_libra(self):  
 response = self.client.get('/horoscope/libra/')  
 self.assertEqual(response.status\_code, 200)  
 self.assertIn('Весы - седьмой знак зодиака, планета Венера (с 24 сентября по 23 октября).',  
 response.content.decode())  
# self.client.get('/horoscope/libra/').content.decode()

3 Тест horoscope/7 horoscope/libra ( редирект)

def test\_libra\_redirect(self):  
 response = self.client.get('/horoscope/7/')  
 self.assertEqual(response.status\_code, 302)  
 self.assertEqual(response.url, '/horoscope/libra/')

## **Типы assert проверок**

Джанго позаимствовала типы проверок из стандартного модуля unittest. В модуле unittest также имеется базовый класс TestCase , который предоставляет несколько assert-методов для проверки и сообщения об ошибках. В следующей таблице перечислены наиболее часто используемые методы

| **Название метода** | **Тип проверки** |
| --- | --- |
| [assertEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertEqual) | a == b |
| [assertNotEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotEqual) | a != b |
| [assertTrue(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertTrue) | bool(x) is True |
| [assertFalse(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertFalse) | bool(x) is False |
| [assertIs(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIs) | a is b |  |
| [assertIsNot(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNot) | a is not b |  |
| [assertIsNone(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNone) | x is None |  |
| [assertIsNotNone(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNotNone) | x is not None |  |
| [assertIn(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIn) | a in b |  |
| [assertNotIn(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotIn) | a not in b |  |
| [assertIsInstance(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsInstance) | isinstance(a, b) |  |
| [assertNotIsInstance(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotIsInstance) | not isinstance(a, b) |  |

**Задание** 1

У нас есть тест test\_libra

class TestHoroscope(TestCase):

def test\_libra(self):

response = self.client.get('/horoscope/libra')

self.assertEqual(response.status\_code, 200)

self.assertIn('Весы - седьмой знак зодиака, планета Венера (с 24 сентября по 23 октября)',

response.content.decode())

Он проверяет на работоспособность один роут знака зодиака весы.

Добавьте новый тест test\_signs, который будет в цикле проверять сразу на все знаки зодиака, которые есть в словаре zodiac\_dict

zodiac\_dict = {

'aries': 'Овен - первый знак зодиака, планета Марс (с 21 марта по 20 апреля).',

'taurus': 'Телец - второй знак зодиака, планета Венера (с 21 апреля по 21 мая).',

'gemini': 'Близнецы - третий знак зодиака, планета Меркурий (с 22 мая по 21 июня).',

'cancer': 'Рак - четвёртый знак зодиака, Луна (с 22 июня по 22 июля).',

'leo': ' Лев - <i>пятый знак зодиака</i>, солнце (с 23 июля по 21 августа).',

'virgo': 'Дева - шестой знак зодиака, планета Меркурий (с 22 августа по 23 сентября).',

'libra': 'Весы - седьмой знак зодиака, планета Венера (с 24 сентября по 23 октября).',

'scorpio': 'Скорпион - восьмой знак зодиака, планета Марс (с 24 октября по 22 ноября).',

'sagittarius': 'Стрелец - девятый знак зодиака, планета Юпитер (с 23 ноября по 22 декабря).',

'capricorn': 'Козерог - десятый знак зодиака, планета Сатурн (с 23 декабря по 20 января).',

'aquarius': 'Водолей - одиннадцатый знак зодиака, планеты Уран и Сатурн (с 21 января по 19 февраля).',

'pisces': 'Рыбы - двенадцатый знак зодиака, планеты Юпитер (с 20 февраля по 20 марта).',

}

Проверять нужно на роут и на наличие текста в контенте страницы

**Задание 2**

  Исправьте тест test\_libra\_redirect

class TestHoroscope(TestCase):

def test\_libra\_redirect(self):

response = self.client.get('/horoscope/7')

self.assertEqual(response.status\_code, 302)

self.assertEqual(response.url, '/horoscope/libra')

так чтобы, он проверял на переходы не только по адресу /horoscope/7, но и на все числа от 1 до 12

**Решение**

def test\_signs(self):  
 for i in views.signs:  
 s = reverse('horoscope-name', args=[i])  
 response = self.client.get(s)  
 self.assertEqual(response.status\_code, 200)  
 self.assertIn(views.signs[i],  
 response.content.decode())  
  
  
def test\_libra\_redirect(self):  
 for i in range(12):  
 s = reverse('int-sign', args=[i + 1])  
 response = self.client.get(s)  
 self.assertEqual(response.status\_code, 302)  
 s2 = views.zodiacs[i]  
 s1 = reverse('horoscope-name', args=[s2])  
 self.assertEqual(response.url, s1)

------------------ OK

**3.1 Что такое шаблон и как его создать**

В папке horoscope создаем папку templates

В папке templates создаем папку horoscope

В новой папке horoscope создаем html файл info\_zodiac

Заполняем инфой info\_zodiac

In VIEWS

from django.template.loader import render\_to\_string

Меняем функцию

def get\_info\_about\_sign\_zodiac(request, sign\_zodiac: str):  
 response = render\_to\_string('horoscope/info\_zodiac.html')  
 return HttpResponse(response)

Запускаем сервер Zodiac отладчик

Horoscope ok

Horoscope/area ошибка

Заходим в файл settings в папке my\_page

TEMPLATES = [  
 {  
 "BACKEND": "django.template.backends.django.DjangoTemplates",  
 "DIRS": [  
 BASE\_DIR / 'horoscope' / 'templates'  
 ],  
 "APP\_DIRS": True,……….

файл info\_zodiac

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Гороскоп</title>  
</head>  
<body>  
 <h1>Приложение Гороскоп</h1>  
 <h2>Близнецы - третий знак зодиака, планета Меркурий (с 22 мая по 21 июня).</h2>  
</body>  
</html>

Способ BASE\_DIR / 'horoscope' / 'templates' где искать шаблоны неудобен (много проектов – каждый раз записывать новый)

Другой способ

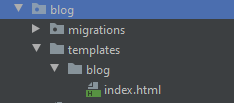
Вписываем horoscope в список в сеттинге

INSTALLED\_APPS = [  
 "django.contrib.admin",  
 "django.contrib.auth",  
 "django.contrib.contenttypes",  
 "django.contrib.sessions",  
 "django.contrib.messages",  
 "django.contrib.staticfiles",  
 "horoscope"  
]

### Задание 3.1.3

### Проект «Мой блог»

В приложении blog создайте папку  templates , в ней - подкаталог blog



Внутри каталога blog создаем шаблон index.html со следующим содержимым

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Home</title>

</head>

<body>

<h2>Главная страница</h2>

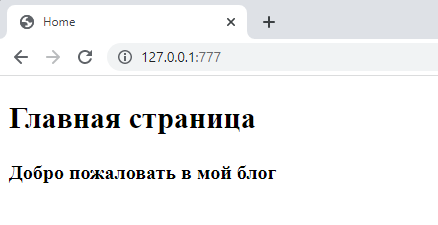
<h4>Добро пожаловать в мой блог</h4>

</body>

</html>

Сделайте так, чтобы содержимое этого шаблона отображалось в браузере, когда обращаемся к главной странице (пустой роут).

Вот что у вас должно получится



### Решение

INSTALLED\_APPS = [  
 "django.contrib.admin",  
 "django.contrib.auth",  
 "django.contrib.contenttypes",  
 "django.contrib.sessions",  
 "django.contrib.messages",  
 "django.contrib.staticfiles",  
 'blog'  
]

path("", views.empty),

def empty(request):  
 response = render\_to\_string('blog/index.html')  
 return HttpResponse(response)

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Home</title>  
</head>  
<body>  
 <h2>Главная страница</h2>  
 <h4>Добро пожаловать в мой блог</h4>  
</body>  
</html>

-------------------------OK

**3.2 Поиск шаблонов в Django. Коллизии в именах шаблонов**

Коллизии – если html файлы размещать в папке templates. Если в разных проектах будут одинаковые по названию html файлы 🡪 будет работать только шаблон верхнего приложения в списке INSTALLED\_APPS, остальные будут его повторами

Новое

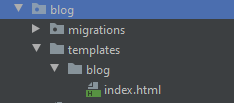
from django.shortcuts import render

def get\_info\_about\_sign\_zodiac(request, sign\_zodiac: str):  
 return render(request, 'horoscope/info\_zodiac.html')

### Задание 3.2.5

### Проект «Мой блог»

В приложении blog у нас уже есть один шаблон index.html



Давайте к нему добавим еще один шаблон  list\_detail.html вот с таким содержимым

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>List post</title>

</head>

<body>

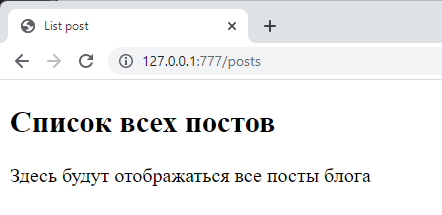
<h2>Список всех постов</h2>

<p>Здесь будут отображаться все посты блога</p>

</body>

</html>

И далее необходимо сделать так, чтобы обращение по пути posts выглядело во так:



### Решение

path("", views.posts),

def posts(request):  
 return render(request, 'blog/list\_detail.html')

### Задание 3.2.3

**Приложение «Список дел на неделю»**

В приложении week\_days создайте шаблон «greeting.html» примерно с таким содержимым

**Добро пожаловать в приложение «Список дел»**

*Здесь будут показаны все ваши дела на ближайшую неделю*

где первая строка - заголовок второго уровня(тег h2), а вторая - параграф (тег p)

Верните содержимое данного шаблона по любому роуту этого приложения

### Решение

def get\_day(request, day: str):  
 return render(request, 'week\_days/greeting.html')

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Хьюстон, у нас нет проблем</title>  
</head>  
<body>  
 <h2>Добро пожаловать в приложение «Список дел»</h2>  
 <p>monday - Составляем план на неделю</p>  
 <p>tuesday - Пишем план на неделю</p>  
 <p>wednesday - Редактируем план на неделю</p>  
 <p>thursday - Согласуем план на неделю</p>  
 <p>friday - Вешаем план на стенку</p>  
 <p>saturday - Делаем пометки в плане</p>  
 <p>sunday - Обдумываем новый план</p>  
</body>  
</html>

### Задание 3.2.4

**Приложение «Геометрия»**

В приложении [«geometry»](https://stepik.org/lesson/681496/step/5?unit=680275) давайте создадим три шаблона для вычисления площадей фигур

1. *rectangle.html*
2. <!DOCTYPE html>
3. <html lang="en">
4. <head>
5. <meta charset="UTF-8">
6. <title>Площадь прямоугольника</title>
7. </head>
8. <body>
9. <h3>Площадь прямоугольника</h3>
10. </body>

</html>

1. *square.html*
2. <!DOCTYPE html>
3. <html lang="en">
4. <head>
5. <meta charset="UTF-8">
6. <title>Площадь квадрата</title>
7. </head>
8. <body>
9. <h3>Площадь квадрата</h3>
10. </body>

</html>

1. *circle.html*
2. <!DOCTYPE html>
3. <html lang="en">
4. <head>
5. <meta charset="UTF-8">
6. <title>Площадь круга</title>
7. </head>
8. <body>
9. <h3>Площадь круга</h3>
10. </body>

</html>

Верните данные шаблоны в качестве ответа.

### Решение

urlpatterns = [  
 path("rectangle/<int:w>/<int:h>/", views.get\_rectangle\_area, name='rectangle'),  
 path("square/<int:s>/", views.get\_square\_area, name='square'),  
 path("circle/<int:r>/", views.get\_circle\_area, name='circle'),  
 path("get\_rectangle\_area/<int:w>/<int:h>/", views.get\_rectangle\_area\_redir),  
 path("get\_square\_area/<int:s>/", views.get\_square\_area\_redir),  
 path("get\_circle\_area/<int:r>/", views.get\_circle\_area\_redir),  
 path("rectangle", views.get\_fige),  
 path("square", views.get\_fige),  
 path("circle", views.get\_fige),  
]

def get\_fige(request):  
 a = request.path.split('/')[-1]  
 return render(request, f'geometry/{a}.html')

**3.3 Django template language. Рендеринг шаблона.**

Файл шаблона html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Гороскоп для {{sign}}</title>  
</head>  
<body>  
 <h1>Приложение Гороскоп</h1>  
 <h2>{{description}}</h2>  
 <h2>Число {{intt}}</h2>  
 <h2>Список {{listt}}</h2>  
 <h2>Словарь {{dictt}}, ключ {{dictt.name}}</h2>  
 <h2>Экземпляр класса {{my\_class}}, {{my\_class.age}} лет</h2>  
</body>  
</html>

VIEWS

from dataclasses import dataclass

@dataclass()  
class Person:  
 name: str  
 age: int  
  
 def \_\_str\_\_(self):  
 return f'{self.name}'  
  
def get\_info\_about\_sign\_zodiac(request, sign\_zodiac: str):  
 descr = signs.get(sign\_zodiac, None)  
 data = {  
 'description': descr,  
 'sign': sign\_zodiac.title(),  
 'intt': 111,  
 'listt': [1, 2, 3],  
 'dictt': {'name': 'Mic', 'age': 40},  
 'my\_class': Person('Bill', 55)  
 }  
 return render(request, 'horoscope/info\_zodiac.html', context=data)

### Задание 3.3.6

Я приготовил для вас шаблон, в котором содержится несколько переменных

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Киа́ну Чарльз Ривз</title>

</head>

<body>

<h1>Киа́ну Чарльз Ривз</h1>

<p> Киа́ну Ривз — канадский актёр, кинорежиссёр, кинопродюсер и музыкант (бас-гитарист).

Родился 2 сентября {{year\_born}} в городе {{city\_born}}, Ливан</p>

<p>Ссылаясь на его игру в фильме «{{movie\_name}}» (1991), критик газеты The New York Times Джанет Маслин хвалит

актёрскую многосторонность Ривза, говоря, что он «проявляет значительную дисциплину и диапазон. Он легко

перемещается между выдержанным поведением, подходящим для полицейско-процедурной истории, и гибкой манерой его

комических ролей»</p>

</body>

</html>

Ваша задача создать представление, которое будет возвращать рендеринг этого шаблона, при этом в контекст вам нужно передать значения для всех переменных шаблона (в качестве значений можете выбрать что угодно). Также не забудьте написать роут для вашего представления.

### Решение

urlpatterns = [  
 path("", views.posts),  
 path("keanu", views.keanu),  
 path("<int:info>/", views.about\_info\_int),  
 path("<info>/", views.about\_info),  
]

def keanu(request):  
 data = {  
 'year\_born': '1964 г.',  
 'city\_born': 'Бейрут',  
 'movie\_name': 'На гребне волны',  
 }  
 return render(request, 'blog/keanu.html', context=data)

### Задание 3.3.7

Ниже представлен текст

Мощный удар

Норвежец Narve Laeret прославился тем, что 9 ноября 2006 г. в Осло на съемках телевизионной передачи

"Senkveld" телеканал TV2 в течение одной минуты сломал рукой 90 бетонных блоков.

Гамбургер

31 июля 2006г. в меню гриль-бара Bob's BBQ & Grill на пляже Паттайя в Таиланде был включен самый

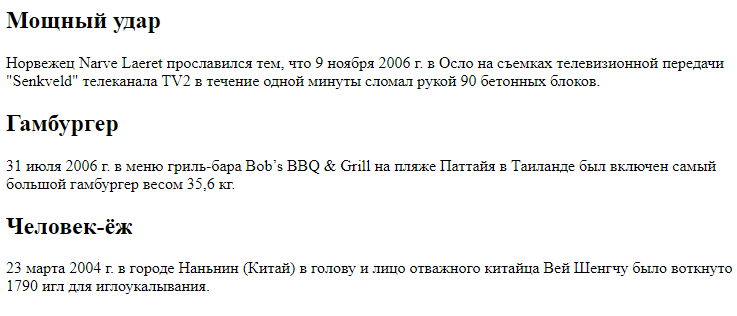
большой гамбургер весом 35,6 кг.

Человек-ёж

23 марта 2004 г. в городе Наньнин (Китай) в голову и лицо отважного китайца Вей Шенгчу было воткнуто

1790 игл для иглоукалывания.

На его основании вам необходимо создать шаблон guinnessworldrecords.html, который будет выводить информацию в  браузере как показано на картинке ниже



Здесь используются только теги <h2> и <p>

Также для вас я приготовил представление.

def get\_guinness\_world\_records(request):

context = {

'power\_man': 'Narve Laeret',

'bar\_name': 'Bob’s BBQ & Grill',

'count\_needle': 1790,

}

return render(request, 'guinnessworldrecords.html', context=context)

Используйте значения из контекста для рендеринга шаблона, их необходимо подставить в нужные места шаблона

### Решение

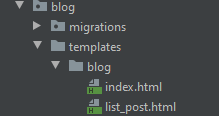
urlpatterns = [  
 path("", views.posts),  
 path("keanu", views.keanu),  
 path("records", views.get\_guinness\_world\_records),  
 path("<int:info>/", views.about\_info\_int),  
 path("<info>/", views.about\_info),  
]

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>World records</title>  
</head>  
<body>  
 <h2>Мощный удар</h2>  
 <p>Норвежец {{power\_man}} прославился тем, что 9 ноября 2006 г. в Осло на съемках телевизионной передачи  
"Senkveld" телеканал TV2 в течение одной минуты сломал рукой 90 бетонных блоков.</p>  
 <h2>Гамбургер</h2>  
 <p>31 июля 2006г. в меню гриль-бара {{bar\_name}} на пляже Паттайя в Таиланде был включен самый  
большой гамбургер весом 35,6 кг.</p>  
 <h2>Человек-ёж</h2>  
 <p>23 марта 2004 г. в городе Наньнин (Китай) в голову и лицо отважного китайца Вей Шенгчу было воткнуто  
 {{count\_needle}} игл для иглоукалывания.</p>  
</body>  
</html>

### Задание 3.3.8

### Проект «Мой блог»

В приложении blog у нас уже есть два шаблон



Создадим еще парочку:

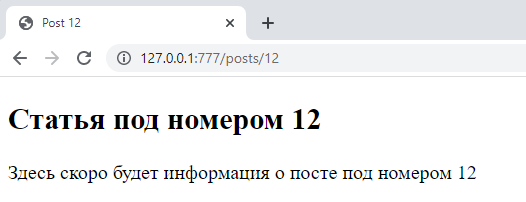
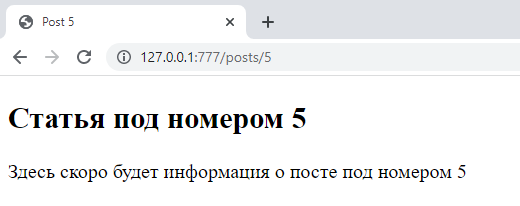
1. detail\_by\_name.html
2. <!DOCTYPE html>
3. <html lang="en">
4. <head>
5. <meta charset="UTF-8">
6. <title> Post «имя» </title>
7. </head>
8. <body>
9. <h2>Статья о «имя»</h2>
10. <p>Здесь скоро будет информация о «имя»</p>
11. </body>

</html>

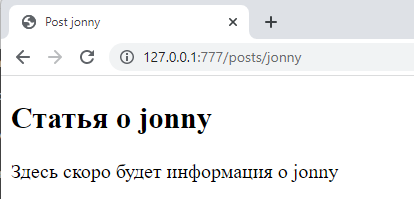
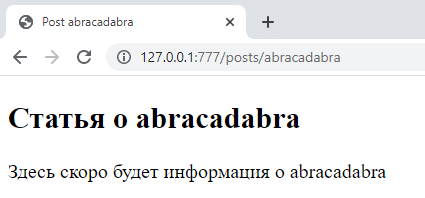
1. detail\_by\_number.html
2. <!DOCTYPE html>
3. <html lang="en">
4. <head>
5. <meta charset="UTF-8">
6. <title> Post «номер» </title>
7. </head>
8. <body>
9. <h2>Статья под номером «номер»</h2>
10. <p>Здесь скоро будет информация о посте под номером «номер»</p>
11. </body>

</html>

Теперь если мы обращаемся по роуту posts/<номер> должно выводится следующее

Но если обращаемся по роуту posts/<строка> должно быть следующее:

### Решение

def about\_info(request, info):  
 data = {  
 'info': info,  
 }  
 return render(request, 'blog/detail\_by\_name.html', context=data)  
  
def about\_info\_int(request, info):  
 data = {  
 'info': info,  
 }  
 return render(request, 'blog/detail\_by\_number.html', context=data)

**3.4 Фильтры в шаблонах Django**

**Основные встроенные фильтры**

**length**

Возвращает размер значения. Работает для строк и списков.

{{ value|length }}

**lower**

Конвертирует строку в нижний регистр.

{{ value|lower }}

**title**

Преобразует первый символ слов в верхний регистр, остальные в нижний.

{{ value|title }}

**upper**

Конвертирует строку в верхний регистр.

{{ value|upper }}

**wordcount**

Возвращает количество слов

{{ value|wordcount }}

Если value равно "Joel is a slug", вернет 4.

Более подробно о фильтрах можно почитать в [документации](https://docs.djangoproject.com/en/4.0/ref/templates/builtins/#built-in-filter-reference)([на русском версия 3.0](https://djbook.ru/rel3.0/ref/templates/builtins.html#built-in-filter-reference))

Верная ссылка <https://docs.djangoproject.com/en/4.2/ref/templates/builtins/>

Рассмотрены

### capfirst add upper center cut date default default\_if\_none divisibleby first last get\_digit join length length\_is ljust rjust lower random make\_list safe slice slugify title

<body>  
 <h1>Приложение Гороскоп</h1>  
 <h2>{{description|title|safe}}</h2>  
 <h2>Число {{intt|add:5}}</h2>  
 <h2>Список {{listt|first}}</h2>  
 <h2>Кортеж {{kort|join:'//'}}</h2>  
 <h2>Словарь {{dictt}}, ключ {{dictt.name}}</h2>  
 <h2>Экземпляр класса {{my\_class}}, {{my\_class.age}} лет</h2>  
 <h2>default {{value|default:"qwerty"}}</h2>  
</body>

**3.5 Теги в Django Template Language. Тег if**

### if[¶](https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#if)

The **{% if %}** tag evaluates a variable, and if that variable is “true” (i.e. exists, is not empty, and is not a false boolean value) the contents of the block are output:

{% **if** athlete\_list %}

Number of athletes: {{ athlete\_list|length }}

{% **elif** athlete\_in\_locker\_room\_list %}

Athletes should be out of the locker room soon!

{% **else** %}

No athletes.

{% **endif** %}

In the above, if **athlete\_list** is not empty, the number of athletes will be displayed by the **{{athlete\_list|length }}** variable.

As you can see, the **if** tag may take one or several **{% elif %}** clauses, as well as an **{% else %}** clause that will be displayed if all previous conditions fail. These clauses are optional.

Работает также вместе с операторами сравнения и другими, и фильтрами

**3.6 Тег for**

### for[¶](https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#for)

Называет каждый элемент в массиве, делая элемент доступным в переменной контекста. Например, чтобы отобразить список спортсменов, представленный в **athlete\_list**:

<**ul**>

{% **for** athlete **in** athlete\_list %}

<**li**>{{ athlete.name }}</**li**>

{% **endfor** %}

</**ul**>

You can loop over a list in reverse by using **{% for obj in list reversed %}**.

Если вам нужно просмотреть список списков, вы можете распаковать значения в каждом подсписке в отдельные переменные. Например, если ваш контекст содержит список (x,y) координат, называемых **points**, вы можете использовать следующее для вывода списка точек:

{% **for** x, y **in** points %}

There is a point at {{ x }},{{ y }}

{% **endfor** %}

This can also be useful if you need to access the items in a dictionary. For example, if your context contained a dictionary **data**, the following would display the keys and values of the dictionary:

{% **for** key, value **in** data.items %}

{{ key }}: {{ value }}

{% **endfor** %}

Keep in mind that for the dot operator, dictionary key lookup takes precedence over method lookup. Therefore if the **data** dictionary contains a key named **'items'**, **data.items** will return **data['items']** instead of**data.items()**. Avoid adding keys that are named like dictionary methods if you want to use those methods in a template (**items**, **values**, **keys**, etc.). Read more about the lookup order of the dot operator in the[documentation of template variables](https://docs.djangoproject.com/en/4.2/ref/templates/language/#template-variables).

Цикл for устанавливает ряд переменных, доступных в цикле:

| **Переменная** | **описание** |
| --- | --- |
| **forloop.counter** | Текущая итерация цикла (1-индексированная) |
| **forloop.counter0** | Текущая итерация цикла (0-индексированная) |
| **forloop.revcounter** | Количество итераций от конца цикла (1-индексированное) |
| **forloop.revcounter0** | Количество итераций от конца цикла (0-индексировано) |
| **forloop.first** | Правда, если это первый раз через цикл |
| **forloop.last** | Правда, если это последний раз через цикл |
| **forloop.parentloop** | Для вложенных циклов это контур, окружающий текущий |

def index(request):  
</li>"  
 context = {  
 'zodiacs': zodiacs  
 }  
 return render(request, 'horoscope/index.html', context=context)

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Меню</title>  
</head>  
<body>  
 <ul>  
 {% for zodiac in zodiacs %}  
 <li>{{forloop.counter}} {{ zodiac|title }}</li>  
 {% endfor %}  
 </ul>  
</body>  
</html>

### Задание 3.6.1/3.6.2

Давайте усложним предыдущее задание. Потому что скорее всего о человеке будет хранится информация не только о имени, но и о других атрибутах, таких как например возраст, паспорт, телефон и т.д.

Взгляните на список словарей ниже

people = [

{'name': 'Жанна Ивановна Бобылева', 'age': 28, 'phone': '+72609577301'},

{'name': 'Спиридон Феликсович Алексеев', 'age': 48, 'phone': '8 445 133 42 50'},

{'name': 'Лыткина Зоя Рубеновна', 'age': 34, 'phone': '84061070300'},

{'name': 'Олимпиада Святославовна Петухова', 'age': 70, 'phone': '8 740 992 96 95'},

{'name': 'Лазарева Нина Кирилловна', 'age': 67, 'phone': '89040731989'},

{'name': 'Каллистрат Ильич Ширяев', 'age': 63, 'phone': '+7 418 298 8976'},

{'name': 'Евсеев Любосмысл Чеславович', 'age': 47, 'phone': '83111461302'},

{'name': 'Прохор Харламович Артемьев', 'age': 47, 'phone': '+77827445919'},

{'name': 'Кондрат Игнатьевич Ершов', 'age': 35, 'phone': '+7 419 594 39 00'},

{'name': 'Ипат Власович Ильин', 'age': 47, 'phone': '88004779773'}

]

В каждом элементе списка хранится словарь, содержащий ключи name, age, phone. Для вывода информации об одном человеке мы будем создавать в html блок div, состоящий из

1. элемента h5, где выводится информация о имени
2. элемент p, где выводится информация о возрасте
3. элемент p, где выводится информация о телефоне

### Решение

path("people", views.people),  
path("people2", views.people\_detail),

context = {  
 'peoples': people  
}  
return render(request, 'blog/people\_detail.html', context=context)

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>People</title>  
</head>  
<body>  
 <h2>Информации о людях</h2>  
 {% for people in peoples %}  
 <div>  
 <h5>ФИО - {{people.name}} </h5>  
 <p>Возраст - {{people.age}} </p>  
 <p>Телефон - {{people.phone}} </p>  
 </div>  
 {% endfor %}  
</body>  
</html>

**3.7 Тег url**

1 Способ возможный

<body>  
 <ul>  
 {% for zodiac in zodiacs %}  
 <li><a href="/horoscope/{{zodiac}}">{{ zodiac|title }}</a></li>  
 {% endfor %}  
 </ul>  
</body>

Важно про первый слэш в /horoscope/:

Слэш слева сразу задает путь путь после доменного имени!!!

Если его удалить, то будет указан путь после текущего роута

То есть horoscope/ даст ошибку /horoscope/ horoscope/..

2 Способ правильный уходим от ручного вбивания /horoscope/

Ранее пользовались функцией reverse теперь аналог тег url

***{%*** **url** 'some-url-name' v1 v2 ***%}***

***или***

***{%*** **url** 'some-url-name' arg1**=**v1 arg2**=**v2 ***%}***

<body>  
 <ul>  
 {% for zodiac in zodiacs %}  
 <li><a href="{% url 'horoscope-name' sign\_zodiac=zodiac %}">{{ zodiac|title }}</a></li>  
 {% endfor %}  
 </ul>  
</body>

**3.8 Наследование шаблонов Django. Теги block и extends**

Удаляем образовательные блоки

<h2>Число {{intt|add:5}}</h2>  
  
{% if listt|length >= 3 %}  
<h2>Список из 3 и более элементов {{listt}}</h2>  
{% elif listt|length < 3 %}  
<h2>Список из менее 3 элементов {{listt}}</h2>  
{% endif %}  
  
<h2>Кортеж {{kort|join:'//'}}</h2>  
<h2>Словарь {{dictt}}, ключ {{dictt.name}}</h2>  
<h2>Экземпляр класса {{my\_class}}, {{my\_class.age}} лет</h2>  
<h2>default {{value|default:"qwerty"}}</h2>

Чистим базовый html

Наследование шаблонов избавление от дубля кодов

В базовом шаблоне оставляем все неизменяемые параметры дочерних

В папке my\_page создаем папку templates

В папку templates копируем файл index.html, называем его base.html

В файле settings в списке TEMPLATES прописываем новый путь к base.html

BASE\_DIR / 'templates'

TEMPLATES = [  
 {  
 "BACKEND": "django.template.backends.django.DjangoTemplates",  
 "DIRS": [  
 BASE\_DIR / 'templates'  
 ],  
 "APP\_DIRS": True,

Файл base.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title> {% block title %} {% endblock %} </title>  
</head>  
<body>  
 <h1> Base HTML </h1>  
 {% block content %} {% endblock %}  
  
</body>  
</html>

файл index.html

{% extends 'base.html' %}

{% block title %}  
 Меню  
{% endblock %}  
  
{% block content %}  
 <h2>Content from index.html</h2>  
 <ul>  
 {% for zodiac in zodiacs %}  
 <li><a href="{% url 'horoscope-name' sign\_zodiac=zodiac %}">{{ zodiac|title }}</a></li>  
 {% endfor %}  
 </ul>  
{% endblock %}

файл info-zodiac.html

{% extends 'base.html' %}

{% block title %}  
 Гороскоп для {{sign|upper}}  
{% endblock %}  
  
{% block content %}  
 <h2>Content from info\_zodiac.html</h2>  
 {% if description is None %}  
 <h2>Неправильный знак Зодиака {{sign}}</h2>  
 {% else %}  
 <h2>{{description|capfirst|safe}}</h2>  
 {% endif %}  
{% endblock %}

Задания 3.8…

"DIRS": [  
 BASE\_DIR / 'templates'  
],

{% block content %}  
 <div class="leftcolumn">  
 {% for pst in posts\_ls %}  
 <div class="card">  
 <h2>{{pst.title}}</h2>  
 <h5>{{pst.description}} {{pst.date}}</h5>  
 <div class="fakeimg" style="height:200px;">Image</div>  
 <p>{{pst.content}}</p>  
 </div>  
 {% endfor %}  
 </div>  
{% endblock %}

def empty(request):  
 # response = render\_to\_string('base.html')  
 context = {  
 'posts\_ls': sample(posts\_ls, 3)  
 }  
 return render(request, 'blog/index.html', context=context)

**3.9 Тег include**

В разных шаблонах одинаковые куски

Куски лучше разместить в отдельном шаблоне, и брать их оттуда с помощью тега include

В папке templates проекта создаем папку includes

В папке includes создаем файл navbar.html

В этот файл переносим нужный кусок кода

Добавляем во всех шаблонах этот кусок с помощью include

{% include 'includes/navbar.html' %}

Include принимает название пути шаблона

Имеет доступ ко всем переменным в этом шаблоне

Если указать любые другие переменные, выведет только те что в шаблоне заказчике, остальные выведутся именем переменной без значения (ошибки не будет)

<nav>  
 <a href="/horoscope">Главная</a> |  
 <a href="https://www.homeenglish.ru/OtherGoroscop.htm">Источник</a>  
</nav>  
  
<p>{{sign}}</p>  
<p>{{description}}</p>  
<p>{{zodiacs}}</p>

Можно запретить доступ к переменным шаблона с помощью only

Все переменные недоступны

или only with only with a = 100, b =. 200 доступны только a, b

Задания 3.9.2

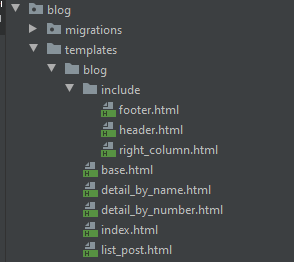
## Проект «Мой блог»

Давайте теперь и мы вынесем блоки нашего кода в отдельные файлики и подключим их затем к базовому шаблону

Для этого в каталоге blog/template/blog создайте папку include и внутри нее три файла:

* header.html
* right\_column.html
* footer.html

В итоге должна получится такая структура



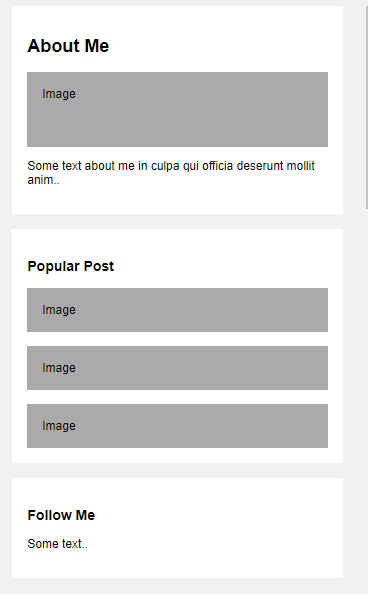
### 

### Перемещаем html код

В файл header.html вынесите из base.html шапку сайта



В файл right\_column.html вынесите из base.html правую колонку



В файл footer.html вынесите из base.html нижнюю часть сайта



После этого подключите указанные файлы к базовому шаблону base.html при помощи тега include

Решение

**3.10 Подключаем статические файлы**

В папке Гороскоп Статик Гороскоп CSS создаем файл index.css

ul {

border-left: 5px solid red;

background-color: #f1f1f1

}

В Сеттинге в INSTALLED\_APPS дб строка

"django.contrib.staticfiles",

В файле index.html

{% load static %}

<link rel="stylesheet" href="{% static ‘horoscope/css/index.css’ %}">

В файле base.html

{% block css\_additional %} {% endblock %}

Копируем этот блок и вставляем в файле index.html

{% block css\_additional %}   
<link rel="stylesheet" href="{% static ‘horoscope/css/index.css’ %}">  
{% endblock %}

Обновляем стр ОК

В папке Гороскоп Статик Гороскоп CSS создаем файл info.css

Подключим его к файлу инфо\_зодиак для <h2> создадим стиль

h2 {

color: blue

}

Повторяем все верхние шаги ОК

Подключаем 2 стиль к index.html

{% block css\_additional %}  
 <link rel="stylesheet" href="{% static 'horoscope/css/index.css' %}">  
 <link rel="stylesheet" href="{% static 'horoscope/css/info.css' %}">  
{% endblock %}

ОК но так делать не нужно, к каждому css свой файл возвращаем == удаляем из индекса второй стиль

Статика к проекту в целом

Создаем папку static в папке проекта my\_page

my\_page CSS создаем файл base.css

html {

background: red;

}

В файле base.html

<meta charset="UTF-8">  
<title> {% block title %} {% endblock %} </title>  
<link rel="stylesheet" href="{% static 'css/base.css' %}">  
{% block css\_additional %} {% endblock %}

Идем в Сеттинг Находим STATIC\_URL = "/static/" и рядом пишем

STATICFILES\_DIRS = [  
 BASE\_DIR / 'static'  
]

**3.11 Изменяем стиль проекта**

Задаем новые css стили во всех трех файлах

Убираем лишние инфостроки в файлах html и вводим новые инфостроки

Делаем правки во VIEWS

**3.12 Бонус🚀 Создаем свой собственный фильтр**

Как создать собственный фильтр

Внутри папки гороскоп создаем папку==питоновский пакет с названием

templatetags

В этой папке создаем питон файл my\_filter

from django import template  
  
register = template.library()  
  
@register.filter(name='split')  
def split(value, key= ' '):  
 return value.split(key)

В файле инфо-зодииак

{% load my\_filter %}

<h1>{{description|split|first}}</h1>

Меняем латинские названия в браузере на русские

Во VIEWS

'zodiacs': signs

В Навбар

{% for zodiac, descr\_zodiak in zodiacs.items %}  
<a href="{% url 'horoscope-name' sign\_zodiac=zodiac %}">{{ descr\_zodiak|split|first }}</a>

Ссылка на док

<https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/>

### Задание попроще

Создать фильтр times, который принимает один аргумент - количество повторений.

{% load my\_filters %}

{% for i in 10|times %}

<p>Hello</p>

{% endfor %}

В результате код выше должен создавать 10 параграфов со строкой Hello

### Задание посложнее

Создать фильтр filter\_range, который принимает два аргумент - начало последовательности и ее конец.  Фильтр должен быть аналогом функции range: возвращать коллекцию чисел от начала до конца не включительно

{% load my\_filters %}

{% for c in 3|filter\_range:8 %}

<p>{{ c }}</p>

{% endfor %}

В результате код выше должен создавать 5 параграфов с числами от 3 до 7.

### Решение

def filtered(request):  
 return render(request, 'table/filtered.html')

path("filtered", views.filtered),

from django import template  
from django.template.defaultfilters import stringfilter  
register = template.Library()  
  
  
@register.filter(name='times')  
def times(n):  
 return range(n)  
  
  
@register.filter(name='filter\_range')  
def filter\_range(n, m):  
 return range(n, m)

<!DOCTYPE html>  
  
{% load static %}  
<link rel="stylesheet" href="{% static 'css/tbl.css' %}">  
  
  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Filter</title>  
  
</head>  
<body>  
 <table class="styled-table">  
 <thead>  
 <tr>  
 <th>name</th>  
 <th>entrance</th>  
 <th>exit</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr>  
 <td>times</td>  
 <td>5 повторов</td>  
 <td>  
 {% load my\_filters %}  
 {% for i in 5|times %}  
 <p>Hello</p>  
 {% endfor %}  
 </td>  
 </tr>  
 <tr class="active-row">  
 <td>filter\_range</td>  
 <td>коллекция от 2 до 4</td>  
 <td>  
 {% load my\_filters %}  
 {% for c in 2|filter\_range:5 %}  
 <p>{{ c }}</p>  
 {% endfor %}  
 </td>  
 </tr>  
 </tbody>  
 </table>  
</body>  
</html>

**4  База данных, ORM, Модели**

Создаем новый проект переходим в терминале в папку МайДжангоПроект

django-admin startproject movie\_proj

Открываем проект ОПЕН и тд

См Сеттинг там есть

DATABASES = {  
 "default": {  
 "ENGINE": "django.db.backends.sqlite3",  
 "NAME": BASE\_DIR / "db.sqlite3",  
 }  
}

БД будет называться db.sqlite3

Проверяем, что настроен интерпретатор

ПАЙЧАРМ Преференс Project movie\_proj Пайтон интерпретатор видим что нет Джанго устанавливаем + инсталл Django

В терминале создаем приложение

python3 manage.py startapp movie\_app

Вся работа с бд в файле models.py

Создаем в нем таблицу

from django.db import models  
  
# Create your models here.  
  
class Movie(models.Model):  
 name = models.CharField(max\_length=40)  
 rating = models.IntegerField()

Задание

Создайте новый проект bookshop(книжный магазин), затем внутри него приложение book\_app

В приложении book\_app в файле models.py создайте модель Book со следующими полями:

* title - поле типа CharField с максимальной длиной значения 70 символов
* rating - рейтинг книги, поле типа IntegerField

Решение

django-admin startproject bookshop

cd bookshop

Инсталлируем джанго

python3 manage.py startapp book\_app

class Book(models.Model):  
 title = models.CharField(max\_length=70)  
 rating = models.IntegerField()

------- ОК

**4.4 Миграции: создание и выполнение**

Переходим в проект movie\_proj

(base) lemon@MacBook-Air-Lemon movie\_proj %

Запускаем

python3 manage.py runserver

Появляется файл .db

Открываем и устнавливаем программу DB BROUSER for SQLite

В таблицах и везде 0 МИГРАЦИИ

Создаем первую миграцию

python3 manage.py makemigrations

Видим No changes detected

Добавляем приложение в Сеттинге

INSTALLED\_APPS = [  
 "django.contrib.admin",  
 "django.contrib.auth",  
 "django.contrib.contenttypes",  
 "django.contrib.sessions",  
 "django.contrib.messages",  
 "django.contrib.staticfiles",  
 "movie\_app"  
]

Повторяем

python3 manage.py makemigrations

Видим

Migrations for 'movie\_app':

movie\_app/migrations/0001\_initial.py

- Create model Movie

(base) lemon@MacBook-Air-Lemon movie\_proj %

Чтобы применить созданную миграцию к БД применяем команду

python3 manage.py migrate

Переоткрываем нашу бд в приложении и видим, что появилась куча всего

Команды

python3 manage.py makemigrations

python3 manage.py migrate

надо каждый раз запускать после изменений в class Movie(models.Model) или просто в модели

## **4.4.3 Подробнее о миграциях Атрибут dependencies Атрибут**operations **Команда migrate**

## **4.4.4 Где хранятся миграции**

Команда

python3 manage.py showmigrations

инфа о созданных и накатанных миграциях

## **4.4.5 Управление миграциями**

Для того, чтобы откатиться назад нужно воспользоваться командой

python3 manage.py migrate app номер\_миграции

## **Накатит часть миграций**

Если мы выполним команду

python3 manage.py migrate posts

то у накатятся все имеющиеся миграции сразу. Но мы можем выполнить только часть миграций, для этого нужно вновь указать порядковый номер миграции, до которой нужно накатить, например четвертая

python3 manage.py migrate posts 0004

## **4.5 Добавление записи в таблицу**

Терминал

python3 manage.py shell

from movie\_app.models import Movie

a = Movie(name='Matrix', rating=85)

In [3]: a

Out[3]: <Movie: Movie object (None)>

a.save()

from django.db import connection

connection.queries

b = Movie(name='Avatar', rating=80)

b.save()

connection.queries

копируем, меняем

INSERT INTO movie\_app\_movie ("name", "rating") VALUES('Django', 83);

Переходим в таблицу в SQL вставляем run ЗАПИСАТЬ ИЗМЕНЕИЯ

Улучшим подсказками терминал

Пайчарм преференсе + ipython

Выходим из терминала Ctrl + Z и заходим обратно

python3 manage.py shell

from movie\_app.models import Movie

Movie(name='XXX', rating=75).save()

--------------------------ОК

**Модуль django-extensions**

Для просмотра sql запросов при работе с ORM воспользуемся модулем [django-extensions](https://django-extensions.readthedocs.io/en/latest/)

1. устанавливаем

pip install django-extensions

1. добавляем в наши приложения
2. INSTALLED\_APPS = (
3. ...
4. 'django\_extensions',

)

1. пользуемся в терминале командой

python3 manage.py shell\_plus --print-sql

Стрелками верх вниз

In [1]: from movie\_app.models import Movie

In [2]: Movie(name='XXX 2', rating=75).save()

Задание

Теперь давайте наполним таблицу Book при помощи консоли как минимум пятью записями.

Вы можете добавить любые книги, которые хотите. Если лень придумывать названия, то воспользуйтесь данными из списка ниже (имя книги, рейтинг)

1. Зелёная миля, 89
2. Гарри Поттер и узник Азкабана, 85
3. Буря мечей, 80
4. Вторая жизнь Уве, 84
5. Малыш и Карлсон, 86

РЕШЕНИЕ

pip install django-extensions

НЕТ уже установлено

python3 manage.py shell

from book\_app.models import Book

Book(title="Зелёная миля", rating=89).save()

Book(title=" Гарри Поттер и узник Азкабана ", rating=85).save()

Book(title="Буря мечей", rating=85).save()

Book(title="Вторая жизнь Уве", rating=85).save()

Book(title="Малыш и Карлсон", rating=85).save()

\_---------------OK

## **4.6 Выборка записей из таблицы**

Терминал

python3 manage.py shell\_plus --print-sql

from movie\_app.models import Movie

Movie.objects

Movie.objects.all()

Видим

<QuerySet [<Movie: Movie object (1)>, <Movie: Movie object (2)>, <Movie: Movie object (3)>, <Movie: Movie object (4)>, <Movie: Movie object (5)>]>

Чтобы изменить форму вывода маг метод str в классе МУВЕ

def \_\_str\_\_(self):  
 return f'{self.name} - {self.rating}%'

В этом случае / маг метод str / миграций делать не надо

python3 manage.py makemigrations изменений нет

python3 manage.py shell\_plus --print-sql

from movie\_app.models import Movie

Видим

<QuerySet [<Movie: Matrix - 85%>, <Movie: Avatar - 80%>, <Movie: Django - 83%>, <Movie: XXX - 75%>, <Movie: XXX 2 - 75%>]>

Все понятно на выводе

Копируем SQL запрос

SELECT "movie\_app\_movie"."id",

"movie\_app\_movie"."name",

"movie\_app\_movie"."rating"

FROM "movie\_app\_movie"

LIMIT 21

И выполняем его в ТАБЛИЦАХ ОК

Можно выполнять по срезам

Movie.objects.all()[:2]

<QuerySet [<Movie: Matrix - 85%>, <Movie: Avatar - 80%>]>

Можно обращаться и по индексам, но только неотрицательным

a = Movie.objects.all()[1]

In [7]: a.name

Out[7]: 'Avatar'

In [8]: a.rating

Out[8]: 80

Задание

Переопределите для модели Book магический метод \_\_str\_\_

Затем выберите все записи из Book модели и сохраните их в переменную books.

После этого обратитесь ко второй книге и узнайте какой у нее рейтинг

РЕШЕНИЕ

python manage.py shell\_plus --print-sql

from book\_app.models import Book

a = Book.objects.all()

a[1].rating

Book.objects.all()[1].rating

--------------OK

## **4.7 Изменение и удаление записей**

class Movie(models.Model):  
 name = models.CharField(max\_length=40)  
 rating = models.IntegerField()  
 year = models.IntegerField()  
 budget = models.IntegerField()

python3 manage.py makemigrations

Видим

It is impossible to add a non-nullable field 'budget' to movie without specifying a default. This is because the database needs something to populate existing rows.

Please select a fix:

1) Provide a one-off default now (will be set on all existing rows with a null value for this column)

2) Quit and manually define a default value in models.py.

Select an option:

Выбираем 2

Пишем в коде

year = models.IntegerField(null=True)  
budget = models.IntegerField(default=1000000)

python3 manage.py makemigrations

Видим

Migrations for 'movie\_app':

movie\_app/migrations/0002\_movie\_budget\_movie\_year.py

- Add field budget to movie

- Add field year to movie

Создался файл второй миграции

class Migration(migrations.Migration):  
  
 dependencies = [  
 ("movie\_app", "0001\_initial"),  
 ]  
  
 operations = [  
 migrations.AddField(  
 model\_name="movie",  
 name="budget",  
 field=models.IntegerField(default=1000000),  
 ),  
 migrations.AddField(  
 model\_name="movie", name="year", field=models.IntegerField(null=True),  
 ),  
 ]

python3 manage.py migrate

Идем в таблицы обновляем ОК

Значения можно менять сразу в таблице вручную

Меняем и нажимаем записать данные

Но так долго и неудобно

Терминал

python manage.py shell\_plus --print-sql

from movie\_app.models import Movie

Movie.objects.all()

dj = Movie.objects.all()[2]

In [4]: dj

Out[4]: <Movie: Dj - 83%>

In [5]: dj.name

Out[5]: 'Dj'

Здесь в терминале перезапишем ИМЯ

In [6]: dj.name = 'Django'

И сохраним

dj.save()

UPDATE "movie\_app\_movie"

SET "name" = 'Django',

"rating" = 83,

"year" = NULL,

"budget" = 1000000

WHERE "movie\_app\_movie"."id" = 3

Обновляем ТАБЛИЦУ ОК

In [8]: dj.year = 2014

In [9]: dj.budget = 999999999

In [10]: dj.save()

ОК

МОЖНО РАБОТАТЬ ТОЛЬКО С 1 записью

dj = Movie.objects.all()[4]

dj.rating = 99

dj.save()

УДАЛЕНИЕ ЗАПИСЕЙ

Movie.objects.all()[3].delete()

Видим

DELETE

FROM "movie\_app\_movie"

WHERE "movie\_app\_movie"."id" IN (4)

Можно через переменную

A = Movie.objects.all()[0]

A.delete()

--------------------OK

Задание

Давайте добавим для модели Book новое поле:

* is\_best\_selling - логическое поле типа BooleanField. В этой поле будет хранится информация о популярности книги: True - книга популярна, False - непопулярна

В итоге код должен получится таким

class Book(models.Model):

title = models.CharField(max\_length=70)

rating = models.IntegerField()

is\_best\_selling = models.BooleanField()

После этого создайте свою вторую миграцию, в процессе создания миграции будет следующее сообщение ……..

Выбирайте первый вариант и вводите значение True или False для того, чтобы существующим записям проставились значения

После создания миграции не забудьте ее применить

Посмотри на код новой миграции и обратите внимание на атрибут dependencies, в нем должна быть ссылка на первую миграцию

----------------------- OK

Для любых трех записей модели Book поменяйте значение, хранящееся в поле is\_best\_selling, на противоположное

РЕШЕНИЕ

a = Book.objects.all()[2]

a.is\_best\_selling = False

a.save()

----------------------- OK

Задание

Давайте добавим для модели Book новое поле:

* author- строковое поле типа CharField, в котором будет хранится информация об авторе книги (максимум 100 символов). Проставим атрибут null=True для того, чтобы в существующие строки нашей таблицы в это поле проставилось значение null

В итоге код должен получится таким

class Book(models.Model):

title = models.CharField(max\_length=70)

rating = models.IntegerField()

is\_best\_selling = models.BooleanField()

author = models.CharField(max\_length=100, null=True)

Выберите  любые три записи в БД и в терминале проставьте для них вручную атрибут author

РЕШЕНИЕ

# python manage.py shell\_plus --print-sql  
# from book\_app.models import Book  
# Book.objects.all()  
# a = Book.objects.all()[0]  
# a.save()  
  
# python3 manage.py makemigrations  
# python3 manage.py migrate

a = Book.objects.all()[4]

a

a.author = 'Астрид Линдгрен'

a.save()

----------------------- OK

## **4.8 Фильтрация данных при помощи filter и get**

manage.py shell\_plus --print-sql

from movie\_app.models import Movie

Movie.objects.all()[0]

Обращение по индексу неудобно, трудно искать, может меняться при удалении или записи новой строки

Получение по ID

Movie.objects.get(id=5)

Get нельзя обратиться к несуществующей записи

Get работает только с одним обьектом

Если нужно =

Movie.objects.filter(budget=1000000) 2 обьекта

Можно обращаться по индексу

Movie.objects.filter(budget=1000000)[0]

Если нужно !=

Movie.objects.exclude(budget=1000000)

Если нужно больше

Movie.objects.filter(budget\_\_gt=1000000)

Если нужно меньше

Movie.objects.filter(budget\_\_lt=1000000)

Если нужно больше или =

Movie.objects.filter(budget\_\_gte=1000000)

Если нужно меньше или =

Movie.objects.filter(budget\_\_lte=1000000)

Там где null

Movie.objects.filter(year\_\_isnull=True)

Несколько полей через and

Movie.objects.filter(year\_\_isnull=True, name='Avatar')

Комбинирование filter exclude

Movie.objects.exclude(year\_\_isnull=False).filter(name='Avatar')

Создадим строку без save()

Movie.objects.create(name='Avatar 2', rating=83)

Все строки содержащие АВАТАР

Movie.objects.filter(name\_\_contains='Avatar')

Чувствительно к регистру

Все строки содержащие АВАТАР

Movie.objects.filter(name\_\_icontains='Avatar')

нечувствительно к регистру

Все строки содержащие бккву a

Movie.objects.filter(name\_\_icontains='a')

все с a первый символ

Movie.objects.filter(name\_\_startswith='a')

Реализация оператора in

Movie.objects.filter(id\_\_in=[3, 5, 6])

Ссылка на док

<https://docs.djangoproject.com/en/3.2/ref/models/querysets/>

Комментарии

Фильтрация данных

1. Метод get()- используется когда хотим получить только одну запись  
   name\_class.objects.get(аргумент=100 или 'Фиеста') - Movie.objects.get(rating=80) - найдет первую подходящую запись в БД по переданным   
   параметрам

2. Метод filter()- используется когда хотим получить все подходящие записи  
   name\_class.objects.filter(аргумент=10000 или 'Фиеста') - Movie.objects.filter(budget=10000) - найдет все подходящие записи,   
   которые подходят под условие аргумента

3. Метод \_\_gt - если хотим найти значение >  
   name\_class.objects.filter(аргумент\_\_gt=10000) - Movie.objects.filter(budget\_\_gt=10000) - найдет все подходящие записи,   
   которые подходят под условие аргумента

4. Метод \_\_lt- если хотим найти значение <  
   name\_class.objects.filter(аргумент\_\_lt=10000) - Movie.objects.filter(budget\_\_lt=10000) - найдет все подходящие записи,   
   которые подходят под условие аргумента

5. Метод \_\_gte - если хотим найти значение >=  
   name\_class.objects.filter(аргумент\_\_gte=10000) - Movie.objects.filter(budget\_\_gte=10000) - найдет все подходящие записи,   
   которые подходят под условие аргумента

6. Метод exclude - если мы хотим исключить запись (типо если не равно значению !=)  
   name\_class.objects.exclude(аргумент=10000) - Movie.objects.exclude(budget=10000) - найдет все подходящие записи,   
   которые подходят под условие аргумента, если не подходит под условие - исключаем

7. Метод \_\_isnull - если хотим поля где есть null или нет null  
   name\_class.objects.filter(аргумент\_\_isnull=True или False) - Movie.objects.filter(budget\_\_isnull=True) - выдаст нам запись,  
   где есть null(если True) или его нет(если False)

8. Метод \_\_isnull - с двумя параметрами  
   name\_class.objects.filter(аргумент\_\_isnull=True или False,name='Аватар') - Movie.objects.filter(budget\_\_isnull=True,name = 'Аватар') -   
   условие будет таким: откроет запись, если подходит под два условия

9. Метод create - можно создавать новые табличные значения легче  
   name\_class.objects.create(аргумент=...,аргумент=...) - Movie.objects.create (name = 'Аватар',rating=88) -   
   создаст новую запись в БД

10. Метод \_\_contains(содержит)(чувствителен к регистру) и \_\_icontains(содержит)(не чувствителен к регистру) - выдаст               нам все записи,  
    где есть содержимое переданного аргумента

    1. name\_class.objects.filter(аргумент\_\_icontains=...) - Movie.objects.filter(name\_\_iconteins='Аватар') - найдет все записи,   
       где есть начало 'Аватар'

    2. name\_class.objects.filter(аргумент\_\_icontains=...) - Movie.objects.filter(name\_\_iconteins='а') - найдет все записи,   
       где есть буква 'а'  
       
11. Методы \_\_startswith и \_\_endswith - (\_\_startswith) фильтер будет искать по первым буквал или цифрам,  
    (\_\_endswith) по крайним символам

    1. name\_class.objects.filter(аргумент\_\_startswith=...) - Movie.objects.filter(name\_\_startswith='ава') - если есть записи в таблице,   
       которые начинаются на 'ава', нам их выдаст  
      
    2. name\_class.objects.filter(аргумент\_\_endswith=...) - Movie.objects.filter(name\_\_endswith='ава') - если есть записи в таблице,   
       которые заканчиваются на 'ава', нам их выдаст

12. Метод \_\_in - выдаст все передаваемые аргументы, которые мы хотим видеть  
    name\_class.objects.filter(аргумент\_\_in=[.,.,.]) - Movie.objects.filter(id\_\_in=[2,3,4]) - выдаст нам всю информацию   
    об этом id(name,rating,year и т.д.)

Задание

Давайте сделаем несколько запросов для модели Book

1. Выберите книгу с id=3
2. Выберите все книги, которые являются бестселлерами (is\_best\_selling=True)
3. Выберите все книги, у которых не проставлен автор
4. Выберите все книги, у которых рейтинг выше 80, но при этом плохо продаются

РЕШЕНИЕ

Book.objects.get(id=3)

Book.objects.filter(is\_best\_selling=True)

Book.objects.filter(author\_\_isnull=True)

Book.objects.filter(is\_best\_selling=False, rating\_\_gt=80)

## **4.9 Объект Q. Союзы AND и OR в запросе**

from django.db.models import Q

Movie.objects.filter(year\_\_isnull=True, rating=80)

Оператор and

Movie.objects.filter(Q(year\_\_isnull=True), Q(rating=80))

Оператор and

Movie.objects.filter(Q(year\_\_isnull=True) & Q(rating=80))

Оператор or

Movie.objects.filter(Q(year\_\_isnull=True) | Q(rating=80))

Длинные обьединения

Movie.objects.filter(Q(year\_\_isnull=True) | Q(rating=80) | Q(rating=87))

Оператор или не | ~

Movie.objects.filter(Q(year\_\_isnull=True) | Q(rating=80) | ~ Q(rating=87))

Оператор не ~

Movie.objects.filter(~ Q(year\_\_isnull=True))

Запятая == and

Movie.objects.filter(Q(year\_\_isnull=True) | Q(rating=80), ~ Q(rating=87))

Допустимо, без Q в конце

Movie.objects.filter(Q(year\_\_isnull=True) | Q(rating=80), rating=87)

Недопустимо, без Q в начале

Movie.objects.filter(rating=87 | Q(year\_\_isnull=True) | Q(rating=80))

## **4.10 Выводим QuerySet в шаблон**

Создаем в приложении МОВИ АПП папку templates

В templates создаем папку МОВИ АПП

В папке МОВИ АПП создаем файл base.html

<title>{% block title %} {% endblock %}</title>

{% block content %} {% endblock %}

В папке МОВИ АПП создаем файл all\_movies.html

{% extend 'movie\_app/base.html' %}

{% block title %} Все фильмы {% endblock %}

{% block content %}  
 <h2>Список всех фильмов</h2>  
{% endblock %}

В папке МОВИ АПП создаем файл urls.py

Копируем аналог из базового urls

from django.contrib import admin  
from django.urls import path  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
]

Добавляем в

from django.urls import path, include

Дочерний

from django.urls import path  
from . import views  
  
urlpatterns = [  
 path("", views.show\_all\_movie),  
]

Views

def show\_all\_movie(request):  
 return render(request, 'movie\_app/all\_movies.html')

Отладчик movie запускаем ОК Список всех фильмов

Views

{% block content %}  
 <h2>Список всех фильмов</h2>  
 <ul>  
 {% for movie in movies %}  
 <li>{{movie.name}}, рейтинг - {{movie.rating}}, год - {{movie.year|default:'Неизвестен'}}</li>  
 {% endfor %}  
 </ul>  
{% endblock %}

Шаблон

{% block content %}  
 <h2>Список всех фильмов</h2>  
 <ul>  
 {% for movie in movies %}  
 <li>{{movie}}</li>  
 {% endfor %}  
 </ul>  
{% endblock %}

**4.11 Методы в модели**

Копипастом создаем шаблон УАНМУВИ

шаблон УАНМУВИ

{% block content %}  
 <h2>Информация о фильме</h2>  
 <h3> {{movie}} </h3>  
  
{% endblock %}

Views

def show\_one\_movie(request, id\_movie: int):  
 movie = Movie.objects.get(id=id\_movie)  
 return render(request, 'movie\_app/one\_movie.html', {  
 'movie': movie  
 })

urls

path("movie/<int:id\_movie>", views.show\_one\_movie),

Избавляемся от вызова по несуществ ID

urls

path("movie/<int:id\_movie>", views.show\_one\_movie, name='movie-detail'),

Views

def show\_one\_movie(request, id\_movie: int):  
 movie = get\_object\_or\_404(Movie, id=id\_movie)  
 return render(request, 'movie\_app/one\_movie.html', {  
 'movie': movie  
 })

шаблон УАНМУВИ

{% block content %}  
 <h2>Информация о фильме - {{movie.name}}</h2>  
 <h3> Рейтинг {{movie.rating}} </h3>  
 <h3> Бюджет {{movie.budget}} </h3>  
 <h3> Год выхода {{movie.year}} </h3>  
  
{% endblock %}

Браузер с ссылками

**Список всех фильмов**

* [Avatar](http://127.0.0.1:8000/movie/2), рейтинг - 80, год - Неизвестен
* [Django](http://127.0.0.1:8000/movie/3), рейтинг - 83, год - 2014
* [XXX 2](http://127.0.0.1:8000/movie/5), рейтинг - 80, год - Неизвестен
* [Avatar 2](http://127.0.0.1:8000/movie/6), рейтинг - 83, год - Неизвестен

Выносим логику ссылок из шаблона в модель

urls

path("movie/<int:id\_movie>", views.show\_one\_movie, name='movie-detail'),

Views

from django.shortcuts import render, get\_object\_or\_404  
from . models import Movie

from django.shortcuts import render, get\_object\_or\_404  
from . models import Movie

шаблон УАНМУВИ

{% block content %}  
 <h2>Информация о фильме - {{movie.name}}</h2>  
 <h3> Рейтинг {{movie.rating}} </h3>  
 <h3> Бюджет {{movie.budget}} </h3>  
 <h3> Год выхода {{movie.year}} </h3>  
  
{% endblock %}

Models

from django.urls import reverse

def get\_url(self):  
 return reverse('movie-detail', args=[self.id])

**4.12 Slug поле**

slug= models.SlugField(default='', null=False)

python3 manage.py makemigrations

python3 manage.py migrate

from django.utils.text import slugify

def save(self, \*args, \*\*kwargs):  
 self.slag = slugify(self.name)  
 super(Movie, self).save(\*args, \*\*kwargs)

# python manage.py shell\_plus --print-sql  
# from movie\_app.models import Movie

Movie.objects.get(id=5).save()

def show\_all\_movie(request):  
 movies = Movie.objects.all()  
 for movie in movies:  
 movie.save()  
 return render(request, 'movie\_app/all\_movies.html', {  
 'movies': movies

Обновляем стр и убираем из кода

for movie in movies:  
 movie.save()

path("movie/< slug: slug\_movie>", views.show\_one\_movie, name='movie-detail'),

def show\_one\_movie(request, slug\_movie: str):  
 movie = get\_object\_or\_404(Movie, slug= slug\_movie)  
 return render(request, 'movie\_app/one\_movie.html', {  
 'movie': movie  
 })

def get\_url(self):  
 return reverse('movie-detail', args=[self. slug])

**4.13 Сортировка QuerySet**

movies = Movie.objects.order\_by('name')

movies = Movie.objects.order\_by('-name')

movies = Movie.objects.order\_by('-name')[0:2]

movies = Movie.objects.order\_by('name', 'budget')

movies = Movie.objects.order\_by('year')

movies = Movie.objects.order\_by('-year')

from django.db.models import F

movies = Movie.objects.order\_by(F('year').desc(nulls\_last=True))

movies = Movie.objects.order\_by(F('year').desc(nulls\_first=True))

movies = Movie.objects.order\_by(F('year').asc(nulls\_first=True))

movies = Movie.objects.order\_by(F('year').asc(nulls\_first=True), 'rating')

**4.14 Функции агрегации**

from django.db.models import F, Sum, Max, Min, Count, Avg

movies.aggregate(Sum('budget'))

movies.aggregate(Sum('budget'), Avg('rating'), Max('year'), Min('year'))

def show\_all\_movie(request):  
 movies = Movie.objects.order\_by(F('year').asc(nulls\_first=True), 'rating')  
 agg = movies.aggregate(Avg('budget'), Max('rating'), Min('rating'))  
 # for movie in movies:  
 # movie.save()  
 return render(request, 'movie\_app/all\_movies.html', {  
 'movies': movies,  
 'agg': agg  
 })

<h3>Средний бюджет - {{agg.budget\_\_avg}}</h3>  
<h3>Минимальный рейтинг - {{agg.rating\_\_min}}</h3>  
<h3>Максимальный рейтинг - {{agg.rating\_\_max}}</h3>

<h3>Всего фильмов - {{agg.id\_\_count}}</h3>

**4.15 Django Debug Toolbar**

<https://django-debug-toolbar.readthedocs.io/en/latest/>

Установка и в терминале и в прференсе

**4.16 Вычисляемые поля с annotate**

from django.db.models import F, Sum, Max, Min, Count, Avg, Value

movies = Movie.objects.annotate(new\_field\_bool=Value(True))

def show\_all\_movie(request):  
 # movies = Movie.objects.order\_by(F('year').asc(nulls\_first=True), 'rating')  
 movies = Movie.objects.annotate(  
 true\_bool=Value(True),  
 false\_bool=Value(False),  
 str\_field=Value('Hello'),  
 int\_field=Value(123),

new\_budget=F('budget') + 100

)

**4.17 Итоги модуля. Паттерн MVC**

**5.1 Admin панель**

Терминал

python3 manage.py createsuperuser

Username (leave blank to use 'lemon'): lemon

Email address: lemon@lemon.com

Password: 12345

This password is too short. It must contain at least 8 characters.

This password is too common.

This password is entirely numeric.

Bypass password validation and create user anyway? [y/N]: y

http://127.0.0.1:8000/admin

Файл admin.py

admin.site.register(Movie)

**5.2 Настройка главной панели**

Меняем язык на русский

Сеттинг LANGUAGE\_CODE = "ru-RU"

Cменим заголовки

Главный УРЛ

admin.**site**.site\_header = 'Наша админка'

**5.3 Настройка формы списка**

Файл admin.py

class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'year']

admin.site.register(Movie, MovieAdmin)

2 способ

@admin.register(Movie)  
class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'year', 'budget']

Для редактирования полей

@admin.register(Movie)  
class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'year', 'budget']  
 list\_editable = ['rating', 'year', 'budget']

class Movie(models.Model):  
 name = models.CharField(max\_length=40)  
 rating = models.IntegerField()  
 year = models.IntegerField(null=True, blank=True)

# python3 manage.py makemigrations  
# python3 manage.py migrate

Хотим чтобы при открытии стр была сортировка по рейтингу, потом по имени

class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'year', 'budget']  
 list\_editable = ['rating', 'year', 'budget']  
 ordering = ['rating', 'name']

Задаем кол фильмов на стр

class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'year', 'budget']  
 list\_editable = ['rating', 'year', 'budget']  
 ordering = ['rating', 'name']  
 list\_per\_page = 3

**5.4 Вычисляемое поле в админке**

Создадим новую колонку с привязкой к колонке рейтинга

@admin.display(ordering='rating', description='Статус')  
def rating\_stat(self, mov: Movie):  
 if mov.rating < 50:  
 return f'Не смотрим это'  
 if 50 <= mov.rating < 85:  
 return f'Смотрим фильм'  
 else:  
 return f'Отличный фильм'

list\_display = ['name', 'rating', 'year', 'budget', 'rating\_stat']

**5.5 Поле choices. Choices field**

class Movie(models.Model):  
  
 CURRENCY\_CHOICE = [  
 ('E', 'Euro'),  
 ('D', 'Dollars'),  
 ('R', 'Rubles')  
 ]  
  
 name = models.CharField(max\_length=40)  
 rating = models.IntegerField()  
 year = models.IntegerField(null=True, blank=True)  
 budget = models.IntegerField(default=1000000)  
 currency = models.CharField(max\_length=1, choices=CURRENCY\_CHOICE, default='R')  
 slug = models.SlugField(default='', null=False, db\_index=True)

# python3 manage.py makemigrations  
# python3 manage.py migrate

@admin.register(Movie)  
class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']  
 list\_editable = ['rating', 'currency', 'budget']  
 ordering = ['-rating', 'name']  
 list\_per\_page = 10

Можно по другому лучше

class Movie(models.Model):  
 EUR = 'EUR'  
 USD = 'USD'  
 RUB = 'RUB'  
 CURRENCY\_CHOICE = [  
 (EUR, 'Euro'),  
 (USD, 'Dollars'),  
 (RUB, 'Rubles')  
 ]  
 name = models.CharField(max\_length=40)  
 rating = models.IntegerField()  
 year = models.IntegerField(null=True, blank=True)  
 budget = models.IntegerField(default=1000000)  
 currency = models.CharField(max\_length=3, choices=CURRENCY\_CHOICE, default=RUB)  
 slug = models.SlugField(default='', null=False, db\_index=True)

Choices надо использовать для неизменяемых величин – пол, времена года и тд, но не для валют

**5.6 Создание действия в админке**

from django.db.models import QuerySet

@admin.register(Movie)

class MovieAdmin(admin.ModelAdmin):

list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']

list\_editable = ['rating', 'currency', 'budget']

ordering = ['-rating', 'name']

list\_per\_page = 10

actions = ['set\_dollars']

@admin.action(description='Установить валюту в доллар')

def set\_dollars(self, request, qs: QuerySet):

qs.update(currency=Movie.USD)

Начинает работать ДЕЙСТВИЕ и можем ставить галки и выбирать

Далее

from django.contrib import admin, messages

class MovieAdmin(admin.ModelAdmin):

list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']

list\_editable = ['rating', 'currency', 'budget']

ordering = ['-rating', 'name']

list\_per\_page = 10

actions = ['set\_dollars', 'set\_euro']

@admin.action(description='Установить валюту в евро')

def set\_euro(self, request, qs: QuerySet):

count\_updated = qs.update(currency=Movie.EUR)

self.message\_user(

request,

f'Было обновлено {count\_updated} записей',

messages.Error

)

**5.7 Создание поисковой панели в админке**

ПОИСК во вкладке Пользователи

Сделаем поиск в приложении проекта

class MovieAdmin(admin.ModelAdmin):

list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']

list\_editable = ['rating', 'currency', 'budget']

ordering = ['-rating', 'name']

list\_per\_page = 10

actions = ['set\_dollars', 'set\_euro']

search\_fields = ['name', 'rating']

# search\_fields = ['name\_\_startswidth', 'rating']

# search\_fields = ['name\_\_istartswidth', 'rating']

# search\_fields = ['name\_\_endswidth', 'rating']

**5.8 Создание фильтра в Django Admin**

class MovieAdmin(admin.ModelAdmin):  
 list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']  
 list\_editable = ['rating', 'currency', 'budget']  
 ordering = ['-rating', 'name']  
 list\_per\_page = 10  
 actions = ['set\_dollars', 'set\_euro']  
 search\_fields = ['name', 'rating']  
 list\_filter = ['name', 'rating', 'currency']

Сделаем фильтр по значениям рейтинга

class RatingFilter(admin.SimpleListFilter):

title = 'Фильтр по рейтингу'

parameter\_name = 'rating'

def lookups(self, request, model\_admin):

return [

('<40', 'Низкий'),

('от 40 до 80', 'Средний'),

('>80', 'Высокий'),

]

def queryset(self, request, queryset: QuerySet):

if self.value() == '<40':

return queryset.filter(rating\_\_lt=40)

if self.value() == 'от 40 до 80':

return queryset.filter(rating\_\_gte=40).filter(rating\_\_lt=60)

if self.value() == '>80':

return queryset.filter(rating\_\_gte=80)

list\_filter = ['name', 'currency', RatingFilter]

**5.9 Настройка формы элемента**

class MovieAdmin(admin.ModelAdmin):

fields = ['name', 'rating'] # хотим видеть

exclude = ['slug', 'budget'] # за исключением

readonly\_fields = ['year'] # нельзя менять в админке

Джанго автоматом проверяет валидацию при СОХРАНИТЬ

Исправим нашу валидацию на верную

year = models.IntegerField(null=True, blank=True)

blank=True # делает поле year необязательным

from django.core.validators import MaxValueValidator, MinValueValidator

rating = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(100)])

budget = models.IntegerField(default=1000000, validators=[MinValueValidator(1)])

# slug

1 способ

Исключим

exclude = ['slug']

Обновляем слаг ставится по умолчанию

slug = models.SlugField(default='', null=False, db\_index=True)

Все сломалось не работает 1 способ как в уроке

2 способ

class MovieAdmin(admin.ModelAdmin):

# fields = ['name', 'rating']

# exclude = ['slug']

# readonly\_fields = ['year']

prepopulated\_fields = {'slug': ('name', )}

#list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']

Метод save() убираем

# def save(self, \*args, \*\*kwargs):

# self.slag = slugify(self.name)

# super(Movie, self).save(\*args, \*\*kwargs)

**6.1 Связи между таблицами**

# currency = models.CharField(max\_length=3, choices=CURRENCY\_CHOICE, default=RUB)

# slug = models.SlugField(default='', null=False, db\_index=True)

director = models.CharField(max\_length=100, default='Тарантино')

# python3 manage.py makemigrations

# python3 manage.py migrate

list\_display = ['name', 'rating', 'director', 'budget', 'rating\_stat']

list\_editable = ['rating', 'director', 'budget']

Добавим почту директора

director\_email = models.EmailField(default='qqq@qq.com')

# python3 manage.py makemigrations

# python3 manage.py migrate

А если нужен тел директора и прочее о нем --> нужен новый обьект Директор

вместо доп колонок в таблице

НОРМАЛИЗАЦИЯ - создание связей между таблицами БД, у нас Фильмы и Директора

У нас директор - отдельная сущность или таблица, создадим ее

Создаем

class Director(models.Model):

f\_name = models.CharField(max\_length=100)

l\_name = models.CharField(max\_length=100)

director\_email = models.EmailField()

Удаляем

director = models.CharField(max\_length=100, default='Тарантино')

director\_email = models.EmailField(default='qqq@qq.com')

В Админ удаляем их тоже

prepopulated\_fields = {'slug': ('name', )}

list\_display = ['name', 'rating', 'currency', 'budget', 'rating\_stat']

list\_editable = ['rating', 'currency', 'budget']

ordering = ['-rating', 'name']

# python3 manage.py makemigrations

# python3 manage.py migrate

from django.contrib import admin, messages

from .models import Movie, Director

from django.db.models import QuerySet

admin.site.register(Director)

Обновляем стр Появились Director перходим добавляем

Кв

Тарантино

qv@qv.com

Сохраняем

Переопределяем str

def \_\_str\_\_(self):

return f'{self.f\_name} {self.l\_name}'

## **Нормализация**

Зачем вообще нужно создавать несколько таблиц в БД? Почему бы нам не хранить все данные в одной большой таблице? Не посещал ли вас такой вопрос? Давайте разбираться

Если мы будем все данные хранить в одной большой таблице, это может привести к дублированию данных. А такое повторение данных может привести к:

* созданию очень больших отношений;
* поддерживать и обновлять данные станет непросто, так как это потребует поиска множества связанных записей;
* потери и плохое использование дискового пространства и ресурсов;
* увеличивается вероятность ошибок и несоответствий.

Таким образом, чтобы справиться с этими проблемами, мы должны проанализировать и разложить отношения с избыточными данными на более мелкие, простые и хорошо структурированные отношения, которые удовлетворяют желаемым свойствам. Нормализация — это процесс разложения отношений на отношения с меньшим количеством атрибутов.

### Что такое нормализация?

* Нормализация — это процесс организации данных в базе данных.
* Нормализация используется для минимизации избыточности отношения или набора отношений. Она также используется для устранения нежелательных характеристик, таких как аномалии вставки, обновления и удаления.
* Нормализация делит большую таблицу на меньшие и связывает их с помощью отношений.
* Нормальная форма используется для уменьшения избыточности таблицы базы данных.

### Зачем нужна нормализация?

Основной причиной нормализации отношений является устранение этих аномалий. Если не устранить аномалии, это приведет к избыточности данных и может привести к нарушению целостности данных и другим проблемам по мере роста базы данных. Нормализация состоит из ряда рекомендаций, которые помогут вам создать хорошую структуру базы данных.

**Аномалией** называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов. Аномалии модификации данных можно разделить на три типа:

1. **Аномалии-модификации** проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.
2. **Аномалии-удаления** — при удалении какого либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.
3. **Аномалии-добавления** возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

### Как выполнить нормализацию?

Чтобы привести БД к нормальной форме, необходимо:

1. Объединить имеющиеся данные в группы.
2. Выяснить логические связи между группами. Чтобы обеспечить правильность связей, связываемые поля должны иметь один тип.

Если таблица не нормализована, она может хранить информацию о нескольких сущностях и включать в себя повторяющиеся столбцы, а они, в свою очередь, могут хранить дублируемые значения. Если же нормализована, то каждая таблица хранит информацию лишь об одной сущности.

При нормализации предполагается использование нормальных форм по отношению к структуре имеющихся данных. Есть несколько правил нормализации. Каждое из них носит название «нормальная форма» (НФ). Каждая такая форма, кроме первой, предполагает, что к данным уже применили предыдущую нормальную форму. При выполнении первого правила БД представлено в первой нормальной форме (1НФ), при выполнении трех правил — в третьей нормальной форме (3НФ).

Таких форм (уровней) — семь, однако на практике для большей части приложений вполне достаточно нормализовать БД до третьей нормальной формы (строго говоря, БД и будет считаться нормализованной, когда к ней применяется 3НФ и выше).

Да, обеспечить полное соответствие правилам и спецификациям — задача не всегда выполнимая, ведь для нормализации придется создавать дополнительные таблицы, а это не всегда приемлемо или не находит отклика у клиентов. Но если правила приходится нарушать, надо понимать, что все, связанные с этим проблемы, включая несогласованные зависимости и избыточность, будут учтены, и что это допустимо для приложения, не нарушит его работоспособность.

Мы с вами разберем три первых НФ

## **Типы нормальных форм**

Нормализация проходит через ряд этапов, называемых нормальными формами. Нормальные формы применяются к индивидуальным отношениям. Говорят, что отношение находится в определенной нормальной форме, если оно удовлетворяет ограничениям. **Нормальная форма** — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

## Требования первой нормальной формы (1NF)

Требование первой нормальной формы (1NF) очень простое и оно заключается в том, чтобы таблицы соответствовали реляционной модели данных и соблюдали следующие реляционные принципы:

* В таблице не должно быть дублирующих строк
* В каждой ячейке таблицы хранится атомарное значение (одно не составное значение)
* В столбце хранятся данные одного типа

### Пример приведения таблицы к первой нормальной форме

Следующая таблица не находится даже в первой нормальной форме, так как  в некоторых ячейках хранятся списки значений (каждый номер телефона — это одно значение) и есть повторяющиеся строки.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |
| 20 | Harry | 8574783832 | Bihar |

Чтобы привести эту таблицу к первой нормальной форме, необходимо  в ячейках хранить один номер телефона, а не список, и убрать одинаковые строки.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

## **Требования второй нормальной формы (2NF)**

Чтобы база данных находилась во второй нормальной форме (2NF), необходимо чтобы ее таблицы удовлетворяли следующим требованиям:

* Таблица должна находиться в первой нормальной форме
* Таблица должна иметь первичный ключ
* Все неключевые столбцы таблицы должны зависеть от первичного ключа (в случае если он составной)

*Первичный ключ – это столбец или набор столбцов, по которым гарантировано можно отличить строки друг от друга, т.е. ключ идентифицирует каждую строку таблицы. По ключу мы можем обратиться к конкретной строке данных в таблице..*

Если ключ составной, т.е. состоит из нескольких столбцов, то все остальные неключевые столбцы должны зависеть от всего ключа, т.е. от всех столбцов в этом ключе. Если какой-то атрибут (столбец) зависит только от одного столбца в ключе, значит, база данных не находится во второй нормальной форме.

Иными словами, в таблице не должно быть данных, которые можно получить, зная только половину ключа, т.е. только один столбец из составного ключа.

Главное правило второй нормальной формы (2NF) звучит следующим образом:

*Таблица должна иметь правильный ключ, по которому можно идентифицировать каждую строку*

### Пример приведения таблицы ко второй нормальной форме

У нас имеется таблица сотрудников в1NF.

|  |  |  |  |
| --- | --- | --- | --- |
| **ФИО** | **Должность** | **Подразделение** | **Описание подразделения** |
| Булыкин И.И. | Программист | Отдел разработки | Разработка и сопровождение приложений и сайтов |
| Григорьев С.С. | Бухгалтер | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |
| Анджелина Джоли | Продавец | Отдел реализации | Организация сбыта продукции |
| Григорьев С.С. | Программист | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |

Мы видим, что она удовлетворяет условиям первой нормальной формы, т.е. в ней нет дублирующих строк и все значения атомарны.

Но мы не можем однозначно идентифицировать одну строку от другой. У нас есть сотрудники с одинаковыми именами, но при этом имеющие разные должности.

Для решения этой проблемы мы можем присвоить каждому сотруднику уникальный табельный номер, который никогда не будет изменен. Табельный номер, в нашем случае и будет являться первичным ключом, зная который мы можем четко идентифицировать каждого сотрудника, т.е. каждую строку нашей таблицы.

Таким образом, чтобы привести эту таблицу ко второй нормальной форме, мы должны добавить в нее еще один атрибут, т.е. столбец с табельным номером.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Табельный номер** | **ФИО** | **Должность** | **Подразделение** | **Описание подразделения** |
| 1 | Булыкин И.И. | Программист | Отдел разработки | Разработка и сопровождение приложений и сайтов |
| 2 | Григорьев С.С. | Бухгалтер | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |
| 3 | Анджелина Джоли | Продавец | Отдел реализации | Организация сбыта продукции |
| 4 | Григорьев С.С. | Программист | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |

В данном случае используется целочисленный тип данных, который автоматически увеличивался в случае добавления новый записи в таблицу. Тем самым мы бы точно также четко идентифицировали каждую строку в таблице

## **Требования третьей нормальной формы (3NF)**

Чтобы база данных находилась во третьей нормальной форме (3NF), необходимо чтобы ее таблицы удовлетворяли следующим требованиям:

* Таблица должна находиться во второй нормальной форме
* Каждый не ключевой атрибут нетранзитивно зависит от первичного ключа

Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

### Пример приведения таблицы к третьей нормальной форме

У нас имеется таблица сотрудников в2NF.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Табельный номер** | **ФИО** | **Должность** | **Подразделение** | **Описание подразделения** |
| 1 | Булыкин И.И. | Программист | Отдел разработки | Разработка и сопровождение приложений и сайтов |
| 2 | Григорьев С.С. | Бухгалтер | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |
| 3 | Анджелина Джоли | Продавец | Отдел реализации | Организация сбыта продукции |
| 4 | Григорьев С.С. | Программист | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |

Чтобы определить, находится ли эта таблица в 3NF, мы должны проверить все неключевые столбцы. Каждый из них должен зависеть только от первичного ключа, и никаким образом к другим неключевым столбцам он не должен относиться.

В результате проверки мы выясняем, что столбец «Описание подразделения» не зависит напрямую от первичного ключа. Мы это выяснили, когда задали себе один вопрос «Каким образом описание подразделения связано с сотрудником?». И наш ответ звучит следующим образом: «Атрибут описание подразделения содержит детальные сведения того подразделения, в котором работает сотрудник».

Отсюда следует, что столбец «Описание подразделения» не связан на прямую с сотрудником, он связан напрямую со столбцом «Подразделение», который напрямую связан с сотрудником, ведь сотрудник работает в каком-то конкретном подразделении. Это и есть транзитивная зависимость, когда один неключевой столбец связан с первичным ключом через другой неключевой столбец.

Для решения этой проблемы мы должны выполнить декомпозицию - разбить нашу таблицу на две. В одной таблице будем хранить сотрудников, а во второй - подразделения. В подразделениях мы обязаны создать первичный ключ, в нашем случае это будет колонка «**Идентификатор подразделения**».

Таблица подразделений в третьей нормальной форме.

|  |  |  |
| --- | --- | --- |
| **Идентификатор подразделения** | **Подразделение** | **Описание подразделения** |
| 1 | Отдел разработки | Разработка и сопровождение приложений и сайтов |
| 2 | Бухгалтерия | Ведение бухгалтерского и налогового учета финансово-хозяйственной деятельности |
| 3 | Отдел реализации | Организация сбыта продукции |

И затем для реализации связи в таблице сотрудников создать ссылку на таблицу подразделений, т.е. добавить внешний ключ.

Таблица сотрудников в третьей нормальной форме.

|  |  |  |  |
| --- | --- | --- | --- |
| **Табельный номер** | **ФИО** | **Должность** | **Подразделение** |
| 1 | Булыкин И.И. | Программист | 1 |
| 2 | Григорьев С.С. | Бухгалтер | 2 |
| 3 | Анджелина Джоли | Продавец | 3 |
| 4 | Григорьев С.С. | Программист | 2 |

Внешний ключ связывает значения столбца «**Идентификатор подразделения**» таблицы подразделений со столбцом «**Подразделение**» таблицы сотрудников

**6.2 Типы связей**

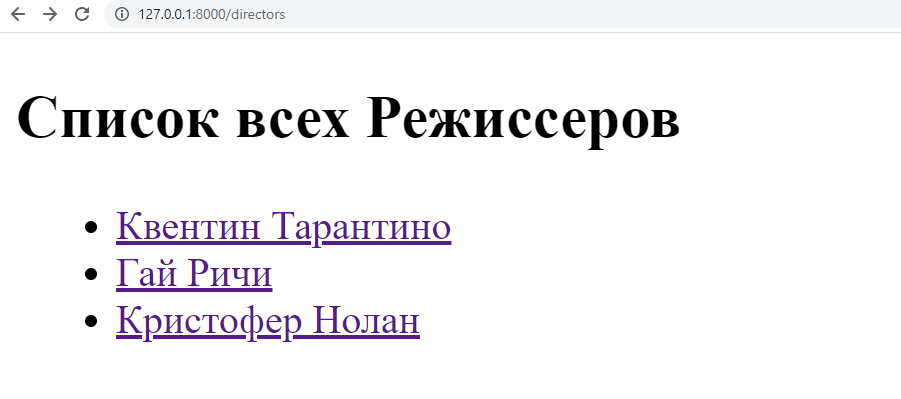
Один к одному

Один ко многим

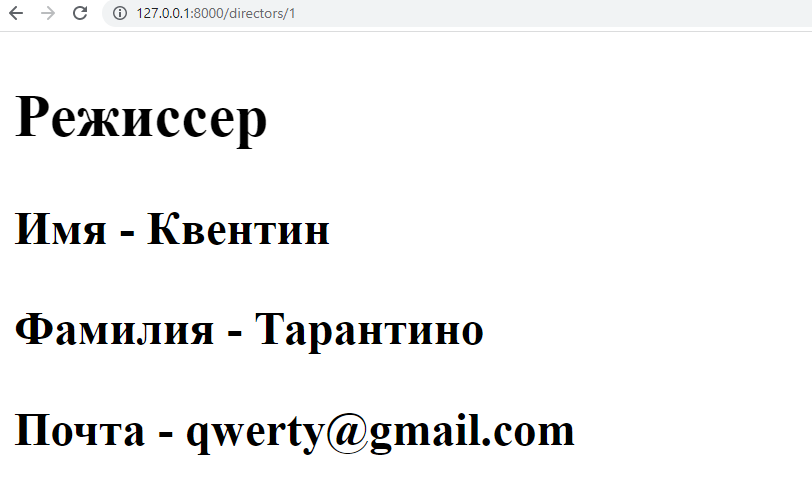
Многие к многим

**6.3 Связь "Один ко многим"**

Создайте роут, функцию-представление и шаблоны для отображения информации о всех режиссерах.



И необходимо также сделать страницу для отображения детальной информации о режиссере. Поиск вы можете осуществлять по id как в примере ниже



urlpatterns = [  
 path("", views.show\_all\_movie),  
 path("movie/<slug:slug\_movie>", views.show\_one\_movie, name='movie-detail'),  
 path('\_\_debug\_\_/', include('debug\_toolbar.urls')),  
 path("directors", views.show\_all\_dirs, name='all\_dirs'),  
 path("directors/<int:ind>", views.show\_one\_dirs, name='one\_dir'),  
]

def show\_all\_dirs(request):  
 directors = Director.objects.all()  
 return render(request, 'movie\_app/all\_directors.html', {  
 'directors': directors  
 })  
  
  
def show\_one\_dirs(request, ind):  
 # dir = Director.objects.all()[ind]  
 dir = get\_object\_or\_404(Director, id=ind)  
 return render(request, 'movie\_app/one\_dir.html', {  
 'dir': dir  
 })

class Director(models.Model):  
 f\_name = models.CharField(max\_length=100)  
 l\_name = models.CharField(max\_length=100)  
 director\_email = models.EmailField()  
  
 def \_\_str\_\_(self):  
 return f'{self.f\_name} {self.l\_name}'  
  
 def get\_url(self):  
 return reverse('one\_dir', args=[self.id])

{% extends 'movie\_app/base.html' %}  
  
{% block title %} Информация о режиссере - {{movie.name}} {% endblock %}  
  
{% block content %}  
 <h2>Режиссер</h2>  
 <h3> Имя {{dir.f\_name}} </h3>  
 <h3> Фамилия {{dir.l\_name}} </h3>  
 <h3> Почта {{dir.director\_email}} </h3>  
  
{% endblock %}

{% extends 'movie\_app/base.html' %}  
  
{% block title %} Режиссеры {% endblock %}  
  
{% block content %}  
 <h2>Список всех режиссеров</h2>  
 <ul>  
 {% for dir in directors %}  
 <li>  
 <a href="{{dir.get\_url}}">{{dir.f\_name}} {{dir.l\_name}}</a>  
 </li>  
 {% endfor %}  
 </ul>  
{% endblock %}

**Связь "Многие ко многим"**

Создаем актеров

class Actors(models.Model):

f\_name = models.CharField(max\_length=100)

l\_name = models.CharField(max\_length=100)

MALE = 'M'

FEMALE = 'F'

GENDERS = [

(MALE, 'Мужчина'),

(FEMALE, 'Женщина'),

]

gender = models.CharField(max\_length=1, choices=GENDERS, default=MALE)

def \_\_str\_\_(self):

if self.gender == self.MALE:

return f'Актер {self.f\_name} {self.l\_name}'

else:

return f'Актриса {self.f\_name} {self.l\_name}'

actors = models.ManyToManyField(Actor)

# python3 manage.py makemigrations

# python3 manage.py migrate

from .models import Movie, Director, Actor

admin.site.register(Actor)

Обновляем браузер админки

Добавляем актеров в фильм

Улучшим добавление

list\_editable = ['rating', 'director', 'budget']

filter\_horizontal = ['actors']

ordering = ['-rating', 'name']

list\_display = ['name', 'rating', 'director', 'actors', 'rating\_stat']

**6.5 Получение связанных объектов. Related objects**

python manage.py shell\_plus --print-sql

from movie\_app.models import Movie

f = Movie.objects.get(name='Django')

f.director

f.director.f\_name

{% block content %}

<h2>Информация о фильме - {{movie.name}}</h2>

<h3> Рейтинг {{movie.rating}} </h3>

<h3> Бюджет {{movie.budget}} </h3>

<h3> Год выхода {{movie.year}} </h3>

<hr>

<h3> Информация о режиссере </h3>

<h3> Имя {{movie.director.f\_name}}</h3>

<h3> Фамилия {{movie.director.l\_name}}</h3>

<h3> Почта {{movie.director.director\_email}}</h3>

{% endblock %}

f.actors

f.actors.all()

<hr>

<h3> Информация об актерах </h3>

<ul>

{% for mov in movie.actors.all %}

<li> {{mov}} </li>

{% endfor %}

</ul>

Обратная задача У режиссера увидеть все его фильмы

from movie\_app.models import Director

Director.objects.all()

g = Director.objects.all()[1]

g.movie\_set

g.movie\_set.all()

# <QuerySet [<Movie: Avatar - 80%>, <Movie: Avatar 2 - 20%>]>

{% block content %}

<h2>Режиссер</h2>

<h3> Имя {{dir.f\_name}} </h3>

<h3> Фамилия {{dir.l\_name}} </h3>

<h3> Почта {{dir.director\_email}} </h3>

<hr>

<h3> Информация об фильмах </h3>

<ul>

{% for mov in dir.movie\_set.all %}

<li> {{mov}} </li>

{% endfor %}

</ul>

director = models.ForeignKey(Director, on\_delete=models.CASCADE, null=True, related\_name='movies')

<hr>

<h3> Информация об фильмах </h3>

<ul>

{% for mov in dir.movies.all %}

<li> {{mov}} </li>

{% endfor %}

</ul>

Актеры

from movie\_app.models import Actor

d = Actor.objects.all()[1]

d.movie\_set

d.movie\_set.all()

<hr>

<h3> Информация об фильмах </h3>

<ul>

{% for mov in act.movie\_set.all %}

<li> {{mov}} </li>

{% endfor %}

</ul>

**6.6 Создание записей в коде с полями ForeignKey и Manytomany**

Переход в терминал и добавление там инфы к фильмам

Из комментов перестала работать или испортилась после действий в терминала админка.

Поэтому только конспект действий, без правок в проекте

python manage.py shell\_plus

Movie.objects.all()

dj = Movie.objects.get(name='Django')

dj.director

Дальше только конспект, в терминале ничего не изменяем

dj.director = None

dj.director.save()

Director.objects.all()

kv = Director.objects.all()[0]

dj.director = kv

dj.director.save()

Новый режиссер

Director(f\_name='Кристофер', l\_name='Нолан')

kr = Director(f\_name='Кристофер', l\_name='Нолан')

kr.save()

Новый фильм

Movie(name='story 2', rating=75, director=kr)

st\_3 = Movie(name='story 2', rating=75, director=kr)

st\_3.save()

\_\_\_\_\_\_ Все сделал в админке, а не в терминале

Вспоминаем, что можно создавать без save() через create

kr = Director.objects.create(f\_name='Кристофер', l\_name='Нолан')

st\_3 = Movie(name='story 2', rating=75, director=kr)

st\_3.save()

Добавление актеров

st\_2 = Movie.objects.get(name='story 2')

st\_2.actors

st\_2.actors.all()

Добавление актеров в терминале

Actor.objects.all()

x = Actor.objects.all()[0]

y = Actor.objects.all()[1]

st\_3.actors.add(x)

st\_3.actors.add(y)

Удаление актеров в терминале

st\_3.actors.remove(x)

st\_3.actors.remove(y)

**6.7 Связь "Один к одному"**

class DressingRoom(models.Model):

floor = models.IntegerField

number = models.IntegerField

def \_\_str\_\_(self):

return f'{self.floor} {self.number}'

from .models import Movie, Director, Actor, DressingRoom

admin.site.register(DressingRoom)

# python3 manage.py makemigrations

# python3 manage.py migrate

в Актор

dressing = models.OneToOneField(DressingRoom, on\_delete=models.SET\_NULL, null=True, blank=True)

# python3 manage.py makemigrations

# python3 manage.py migrate

В админке раздаем актерам гримерки

Дальше терминал

python manage.py shell\_plus

DressingRoom.objects.all()

d = DressingRoom.objects.all()[0]

d.actor

d.actor.f\_name

d = Actor.objects.all()[0]

-----------

Выводим класс DressingRoom в админку

@admin.register(DressingRoom)

class DressingRoomAdmin(admin.ModelAdmin):

list\_display = ['floor', 'number', 'actor']

Строку в коммент

# admin.site.register(DressingRoom)

Обновляем админку в браузеру DressingRoom Ок

**7.1 Введение в модуль "Формы"**

Создаем проект form\_project

django-admin startproject form\_project

python manage.py startapp feedback

INSTALLED\_APPS = [

"django.contrib.admin",

"django.contrib.auth",

"django.contrib.contenttypes",

"django.contrib.sessions",

"django.contrib.messages",

"django.contrib.staticfiles",

"feedback"

]

создаем urls в приложении

копируем из родительского

from django.contrib import admin

from django.urls import path

urlpatterns = [

path("admin/", admin.site.urls),

]

меняем

from django.contrib import admin

from django.urls import path

from .views import index

urlpatterns = [

path("", index),

]

VIEWS

def index(request):

return render(request, 'feedback/feedback.html')

создаем в приложении папку templates, в ней папку feedback, в ней файл feedback.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Оставьте отзыв</title>

</head>

<body>

</body>

</html>

подключаем дочерний urls к родительскому

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path("admin/", admin.site.urls),

path("", include('feedback.urls')),

]

создаем конфигурацию run server запускаем ссылка браузер ок/оставьте отзыв

**7.2 Создание формы**

Способы создания разные, начнем с создания в html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<h2>Оставьте отзыв</h2>

<form>

<label for="name\_id">Имя:</label>

<input type="text" id="name\_id" name="name">

<br>

<br>

<label for="surname\_id">Фамилия:</label>

<input type="text" id="surname\_id" name="surname">

<br>

<br>

<label for="feedback\_id">Отзыв:</label>

<textarea name="feedback" id="feedback\_id" cols="30" rows="5"></textarea>

<br><br>

<button>Отправить</button>

</form>

</body>

</html>

для каждого отзыва свой роут, за это отвечает name=

по умолчанию

<button type="submit">Отправить</button> отправка

<button type="button">Отправить</button> ничего не будет

<button type="reset">Отправить</button> очистка полей без отправки

Пример кода из комментариев

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>feedback</title>

</head>

<body>

<h2>it works</h2>

<form>

<label for="fname">First name:</label><br>

<input type="text" id="fname" name="fname" value="Dart"><br>

<label for="lname">Last name:</label><br>

<input type="text" id="lname" name="lname" value="Vaider"><br>

<label for="feedback\_id">Feedback:</label><br>

<textarea name="feedback" id="feedback\_id" cols="30" rows="10"></textarea>

<br><br>

<input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">

<label for="vehicle1"> I have a bike</label><br>

<input type="checkbox" id="vehicle2" name="vehicle2" value="Car">

<label for="vehicle2"> I have a car</label><br>

<input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">

<label for="vehicle3"> I have a boat</label>

<br><br>

<input type="radio" id="html" name="fav\_language" value="HTML">

<label for="html">HTML</label><br>

<input type="radio" id="css" name="fav\_language" value="CSS">

<label for="css">CSS</label><br>

<input type="radio" id="javascript" name="fav\_language" value="JavaScript">

<label for="javascript">JavaScript</label><br><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

**7.3 Формы. Get и Post запросы**

Get запросы

name=&surname=&feedback=

?name=ккк&surname=ооо&feedback=тбьтити

<form action="/hello">

urlpatterns = [

path("", index),

path("hello", hello),

]

from django.http import HttpResponseRedirect

def hello(request):

print(request.GET['name'])

print(request.GET['feedback'])

return HttpResponseRedirect('/')

По умолчанию в форме

<form action="/" method="get">

method="get" небезопасная форма, видна в браузере, схраняется им, доступна любому

method="post" безопасная форма, используется с токеном

<form action="/" method="post">

{% csrf\_token %}

HTTP-методы GET и POST самые распространённые способы отправить или получить данные с сервера. Но в разных случаях оба метода могут быть небезопасными или неудобными в использовании

### GET

GET — метод для чтения данных с сайта. Например, для доступа к указанной странице. Он говорит серверу, что клиент хочет прочитать данные из указанного ресурса. На практике этот метод используется чаще всего, например, в интернет-магазинах на странице каталога. Фильтры, которые выбирает пользователь, передаются через метод.

**Обратите внимание, что строка запроса (пары «имя-значение») отправляется в URL-адрес запроса GET**:

http://127.0.0.1:8000/?name=Barak&surname=Obama&feedback=Very+cool+hostel

**Некоторые другие заметки о запросах GET:**

* GET запросы могут кэшироваться
* GET запросы остаются в истории браузера
* GET запросы могут быть закладками
* GET запросы никогда не должны использоваться при работе с конфиденциальными данными
* GET запросы имеют ограничения по длине
* GET запросы должны использоваться только для извлечения данных

### POST

POST — метод для отправки данных на указанный ресурс. Чаще всего с помощью метода POST передаются формы.

**Обратите внимание, что строка запроса (пары «имя-значение») отправляется в теле HTTP-сообщения запроса POST:**

POST / HTTP/1.1

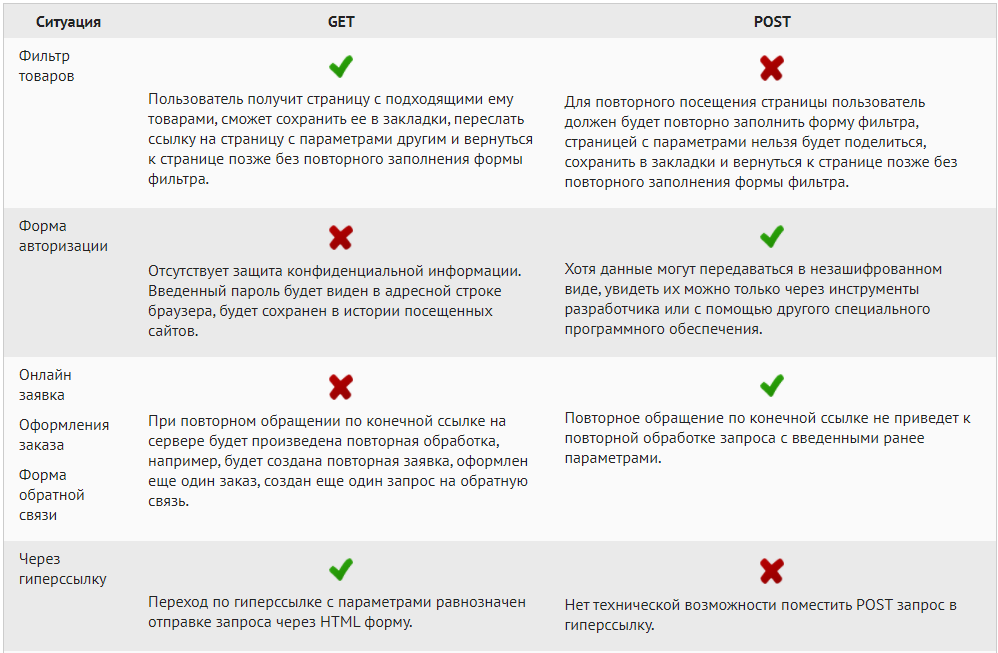
Host: http://127.0.0.1:8000

name=Barak&surname=Obama&feedback=Very+cool+hostel

**Некоторые другие заметки о запросах POST:**

* POST запросы никогда не кэшируются
* Запросы POST не сохраняются в журнале обозревателя
* Запросы POST не могут быть закладками
* Запросы POST не имеют ограничений по длине данных

### Варианты использования



## Что такое CSRF

**CSRF**(англ. **c**ross-**s**ite **r**equest **f**orgery перевод межсайтовая подделка запроса) —  вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Это атака, которой может подвергаться любой веб-ресурс или веб-приложение. В первую очередь это касается сайтов, которые используют cookies, сертификаты авторизации и браузерную аутентификацию. В результате атаки страдают клиенты и репутация ресурса.

Причина CSRF кроется в том, что браузеры не понимают, как различить, было ли действие явно совершено пользователем (как, скажем, нажатие кнопки на форме или переход по ссылке) или пользователь неумышленно выполнил это действие (например, при посещении *bad.com*, ресурсом был отправлен запрос на *good.com/some\_action*, в то время как пользователь уже был залогинен на *good.com*).

### Пример атаки

Атака осуществляется путём размещения на веб-странице ссылки или скрипта, пытающегося получить доступ к сайту, на котором атакуемый пользователь заведомо (или предположительно) уже аутентифицирован. Например, пользователь Алиса может просматривать форум, где другой пользователь, Боб, разместил сообщение. Пусть Боб создал тег <img>, в котором в качестве источника картинки указал URL, при переходе по которому выполняется действие на сайте банка Алисы, например:

Боб: Привет, Алиса! Посмотри, какой милый котик:

​​​​​​​<img src="http://bank.example.com/?account=Alice&amount=1000000&for=Bob">

Если банк Алисы хранит информацию об аутентификации Алисы в куки, и если куки ещё не истекли, при попытке загрузить картинку браузер Алисы отправит куки в запросе на перевод денег на счёт Боба, чем подтвердит аутентификацию Алисы. Таким образом, транзакция будет успешно завершена, хотя её подтверждение произойдет без ведома Алисы.

### Как от нее защититься?

Эффективным и общепринятым на сегодня способом защиты от CSRF-Атаки является **токен**. Под токеном имеется в виду случайный набор байт, который сервер передает клиенту, а клиент возвращает серверу.

Защита сводится к проверке токена, который сгенерировал сервер, и токена, который прислал пользователь.

### Что такое CSRF-token и как он работает

В общем понимании **токен** — это механизм, который позволяет идентифицировать пользователя или конкретную сессию для безопасного обмена информацией и доступа к информационным ресурсам. Токены помогают проверить личность пользователя (например, клиента, который онлайн получает доступ к банковскому счёту). Их используют как вместо пароля, так и вместе с ним. Токен — это в каком-то смысле электронный ключ.

**CSRF-token** — это максимально простой и результативный способ защиты сайта от CSRF-мошенников. Он работает так: сервер создаёт случайный ключ (он же токен) и отправляет его браузеру клиента. Когда браузер запрашивает у сервера информацию, сервер, прежде чем дать ответ, требует показать ключ и проверяет его достоверность. Если токен совпадает, сессия продолжается, а если нет — прерывается. Токен действителен только одну сессию — с новой сессией он обновляется.

Чтобы получить ответ от сервера, используются разные методы запроса. Условно они делятся на две категории: те, которые не изменяют состояние сервера (GET, TRACE, HEAD), и те, которые изменяют (PUT, PATCH, POST и DELETE). Последние имеют большую CSRF-уязвимость и поэтому должны быть защищены в первую очередь.

При создании и использовании токена должны соблюдаться следующие условия:

* нахождение в скрытом параметре;
* генерация с помощью генератора псевдослучайных чисел;
* ограниченное время жизни (одна сессия);
* уникальность для каждой транзакции;
* устойчивый к подбору размер (в битах);
* невозможно переиспользовать.

### Источники

1. [Хабр](https://habr.com/ru/post/318748/)
2. [Что означает ошибка «CSRF токен истек»](https://help.reg.ru/hc/ru/articles/4417799125777-%D0%A7%D1%82%D0%BE-%D0%BE%D0%B7%D0%BD%D0%B0%D1%87%D0%B0%D0%B5%D1%82-%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B0-CSRF-%D1%82%D0%BE%D0%BA%D0%B5%D0%BD-%D0%B8%D1%81%D1%82%D0%B5%D0%BA-)
3. [Вика](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D1%81%D0%B0%D0%B9%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%B4%D0%B5%D0%BB%D0%BA%D0%B0_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0)

**7.4 Ручная валидация формы**

urlpatterns = [

path("", index),

path("done", done),

]

def done(request):

return render(request, 'feedback/done.html')

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<h2>Ваш отзыв успешно обработан</h2>

</body>

</html>

def index(request):

if request.method == 'POST':

name = request.POST['name']

print(name)

return HttpResponseRedirect('/done')

return render(request, 'feedback/feedback.html')

Валидация полей - проверка что поля не пусты (поле отзыва)

def index(request):

if request.method == 'POST':

name = request.POST['name']

if not len(name):

return render(request, 'feedback/feedback.html', context={'got\_error': True})

print(name)

return HttpResponseRedirect('/done')

return render(request, 'feedback/feedback.html', context={'got\_error': False})

<h2>Оставьте отзыв</h2>

<form action="/" method="post">

{% csrf\_token %}

{% if got\_error %}

<p>Нельзя вводить пустое имя</p>

{% endif %}

<label for="name\_id">Имя:</label>

<input type="text" id="name\_id" name="name">

**7.5 Класс Django Form**

создаем внутри приложения файл forms

from django import forms

class FeedbackForm(forms.Form):

name = forms.CharField()

surname = forms.CharField()

feedback = forms.CharField()

def index(request):

form = FeedbackForm()

if request.method == 'POST':

name = request.POST['name']

if not len(name):

return render(request, 'feedback/feedback.html', context={'form': form})

print(name)

return HttpResponseRedirect('/done')

return render(request, 'feedback/feedback.html', context={'got\_error': False})

<h2>Оставьте отзыв</h2>

<form action="/" method="post">

{% csrf\_token %}

{{ form }}

<button>Отправить</button>

</form>

Совершенствуем

feedback = forms.CharField(widget=forms.Textarea(attrs={'rows': 5, 'cols': 20}))

def index(request):

if request.method == 'POST':

form = FeedbackForm(request.POST)

if form.is\_valid():

print(form.cleaned\_data)

return HttpResponseRedirect('/done')

form = FeedbackForm()

return render(request, 'feedback/feedback.html', context={'form': form})

**7.6 Валидация формы**

class FeedbackForm(forms.Form):

name = forms.CharField(label='Имя', max\_length=10, min\_length=2, error\_messages={

'max\_length': 'Слишком много символов',

'min\_length': 'Слишком мало символов',

'required': 'Должен быть хотя бы 1 символ'

})

rating = forms.IntegerField(label='Рейтинг', min\_value=1, max\_value=5)

def index(request):

if request.method == 'POST':

form = FeedbackForm(request.POST)

if form.is\_valid():

print(form.cleaned\_data)

return HttpResponseRedirect('/done')

else:

form = FeedbackForm()

return render(request, 'feedback/feedback.html', context={'form': form})

**7.7 Ручная настройка полей и стилей**

class FeedbackForm(forms.Form):

name = forms.CharField(label='Имя', max\_length=10, min\_length=2, error\_messages={

'max\_length': 'Слишком много символов',

'min\_length': 'Слишком мало символов',

'required': 'Должен быть хотя бы 1 символ'

})

surname = forms.CharField(label='Фамилия', max\_length=15, min\_length=2)

feedback = forms.CharField(label='Отзыв', widget=forms.Textarea(attrs={'rows': 5, 'cols': 20}))

rating = forms.IntegerField(label='Рейтинг', min\_value=1, max\_value=5)

<form action="/" method="post">

{% csrf\_token %}

<div>

{{ form.name.label\_tag }}

{{ form.name.errors }}

{{ form.name }}

</div>

<br>

<div>

{{ form.surname.label\_tag }}

{{ form.surname.errors }}

{{ form.surname }}

</div>

<br>

<div>

{{ form.feedback.label\_tag }}

{{ form.feedback.errors }}

{{ form.feedback }}

</div>

<br>

<div>

{{ form.rating.label\_tag }}

{{ form.rating.errors }}

{{ form.rating }}

</div>

<br>

<button>Отправить</button>

Совершенствуем

<form action="/" method="post">

{% csrf\_token %}

{% for field in form %}

<div>

<br>

{{ field.label\_tag }}

{{ field.errors }}

{{ field }}

<br>

</div>

{% endfor %}

<br>

<button>Отправить</button>

</form>

Подключаем стили

Создаем папку static feedback файл forms.css

body {

margin: 0;

font-family: Sans-Serif;

}

h2{

width:50%;

margin: 20px auto 5px;

text-align: center;

padding: 5px;

border-radius: 5px;

box-shadow: 0 2px 8px #3ca3c1;

}

form {

margin: 1.5rem auto 5px;

width: 90%;

max-width: 40rem;

padding: 1rem;

border-radius: 10px;

font-size: 16px;

background-color: white;

box-shadow: 0 2px 8px #3ca3c1;

}

button {

cursor: pointer;

background-color: #bf0d55;

color: white;

border: 10px solid #bf0d55;

}

button:hover,

button:active {

background-color: #320777;

border-color: #320777;

}

.form-style {

margin-bottom: 1rem;

}

.form-style input {

font: inherit;

padding: 0.25rem;

border-radius: 4px;

border: 1px solid #ccc;

display: block;

width: 10rem;

}

.form-style input:focus {

border-color: #1f0050;

background-color: #f4eeff;

outline: none;

}

.form-style label {

font-weight: bold;

display: block;

margin-bottom: 0.5rem;

}

.errorlist {

list-style: none;

margin: 0;

padding: 0;

color: #ff2650

}

.errors input {

border-color: #aa1754;

background-color: #f4eeff;

}

.errors label{

color: #aaff54;

}

<!DOCTYPE html>

{% load static %}

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Заголовок</title>

<link rel="stylesheet" href="{% static 'feedback/forms.css' %}">

</head>

<body>

<h2>Оставьте отзыв</h2>

<form action="/" method="post">

{% csrf\_token %}

{% for field in form %}

<div class="form-style {% if field.errors %} errors {% endif %} ">

{{ field.label\_tag }}

{{ field.errors }}

{{ field }}

</div>

{% endfor %}

**7.8 Сохранение данных с формы в БД**

В файле моделс

from django.db import models

class Feedback(models.Model):

name = models.CharField(max\_length=40)

surname = models.CharField(max\_length=60)

feedback = models.TextField()

rating = models.PositiveIntegerField()

# python3 manage.py makemigrations

# python3 manage.py migrate

views

def index(request):

if request.method == 'POST':

form = FeedbackForm(request.POST)

if form.is\_valid():

print(form.cleaned\_data)

feed = Feedback(

name=form.cleaned\_data['name'],

surname=form.cleaned\_data['surname'],

feedback=form.cleaned\_data['feedback'],

rating=form.cleaned\_data['rating'],

)

feed.save()

return HttpResponseRedirect('/done')

else:

form = FeedbackForm()

return render(request, 'feedback/feedback.html', context={'form': form})

в бд sqlite видим появление новых записей после каждой отправки

**7.9 ModelForm**

В файле forms

from .models import Feedback

class FeedbackForm(forms.ModelForm):

class Meta:

model = Feedback

# fields = ['name', 'surname', 'feedback', 'rating']

# exclude = ['rating']

fields = '\_\_all\_\_'

labels = {

'name': 'Имя',

'surname': 'Фамилия',

'feedback': 'Отзыв',

'rating': 'Рейтинг',

}

Валидация

В файле models

from django.core.validators import MinLengthValidator

class Feedback(models.Model):

name = models.CharField(max\_length=10, validators=[MinLengthValidator(2)])

surname = models.CharField(max\_length=60, validators=[MinLengthValidator(3)])

feedback = models.TextField()

rating = models.PositiveIntegerField()

class FeedbackForm(forms.ModelForm):

class Meta:

model = Feedback

fields = '\_\_all\_\_'

labels = {

'name': 'Имя',

'surname': 'Фамилия',

'feedback': 'Отзыв',

'rating': 'Рейтинг',

}

error\_messages = {

'name': {

'max\_length': 'Слишком много символов',

'min\_length': 'Слишком мало символов',

'required': 'Должен быть хотя бы 1 символ'

},

'surname': {

'max\_length': 'Слишком много символов',

'min\_length': 'Слишком мало символов',

'required': 'Должен быть хотя бы 1 символ'

},

}

# python3 manage.py makemigrations

# python3 manage.py migrate

def index(request):

if request.method == 'POST':

form = FeedbackForm(request.POST)

if form.is\_valid():

print(form.cleaned\_data)

form.save()

return HttpResponseRedirect('/done')

else:

form = FeedbackForm()

return render(request, 'feedback/feedback.html', context={'form': form})

**7.10 Изменение данных через форму**

urlpatterns = [

path("", index),

path("done", done),

path("<int:id\_feedback>", update\_feedback),

]

def update\_feedback(request, id\_feedback):

feed = Feedback.objects.get(id=id\_feedback)

form = FeedbackForm(instance=feed)

return render(request, 'feedback/feedback.html', context={'form': form})

Делаем редактирование записей в бд

<h2>Оставьте отзыв</h2>

<form action="/" method="post">

Удаляем action="/"

<h2>Оставьте отзыв</h2>

<form method="post">

def update\_feedback(request, id\_feedback):

feed = Feedback.objects.get(id=id\_feedback)

if request.method == 'POST':

form = FeedbackForm(request.POST, instance=feed)

if form.is\_valid():

print(form.cleaned\_data)

form.save()

return HttpResponseRedirect('/done')

else:

form = FeedbackForm(instance=feed)

return render(request, 'feedback/feedback.html', context={'form': form})

**7.11 Class Based Views**

from django.views import View  
  
# Create your views here.  
  
class FeedbackView(View):  
 def get(self, request):  
 form = FeedbackForm()  
 return render(request, 'feedback/feedback.html', context={'form': form})  
  
 def post(self, request):  
 form = FeedbackForm(request.POST)  
 if form.is\_valid():  
 form.save()  
 return HttpResponseRedirect('/done')  
 return render(request, 'feedback/feedback.html', context={'form': form})  
  
  
class Feedback\_update\_View(View):  
  
 def get(self, request, id\_feedback):  
 feed = Feedback.objects.get(id=id\_feedback)  
 form = FeedbackForm(instance=feed)  
 return render(request, 'feedback/feedback.html', context={'form': form})  
  
 def post(self, request, id\_feedback):  
 feed = Feedback.objects.get(id=id\_feedback)  
 form = FeedbackForm(request.POST, instance=feed)  
 if form.is\_valid():  
 print(form.cleaned\_data)  
 form.save()  
 return HttpResponseRedirect('/done')  
 return render(request, 'feedback/feedback.html', context={'form': form})

У нас реализована функция done в файле views.py

def done(request):

return render(request, 'feedback/done.html')

Ваша задача создать класс-представление DoneView, который должен заменить эту функцию.

Затем замените в файле urls.py функцию done на вновь созданный класс DoneView

urlpatterns = [

path("", FeedbackView.as\_view()),

path("done", DoneView.as\_view()),

path("<int:id\_feedback>", Feedback\_update\_View.as\_view()),

]

class DoneView(View):

def get(self, request):

return render(request, 'feedback/done.html')

def post(self, request):

return render(request, 'feedback/done.html')

**8.1 Введение в CBV**

Давайте вспомним тему «Наследование шаблонов Django. Теги block и extends» и сформируем иерархию наследования для наших шаблонов.

Для этого создайте базовый шаблон base.html со следующим содержимым

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title> {% block title %} {% endblock %} </title>

<link rel="stylesheet" href=" {% static 'feedback/forms.css' %}">

</head>

<body>

{% block content %} {% endblock %}

</body>

</html>

Теперь при помощи тега extends расширьте шаблон feedback.html за счет base.html

в блок title запишите строку «Страница отзывов»

в блоке content разместите отображении формы как было ранее в шаблоне feedback.html

Также при помощи тега extends расширьте шаблон done.html за счет шаблона base.html

в блок title запишите строку «Отзыв готов»

в блоке content разместите заголовок Ваш отзыв успешно обработан

<!DOCTYPE html>

{% load static %}

<html lang="en">

<head>

<meta charset="UTF-8">

<title> {% block title %} {% endblock %} </title>

<link rel="stylesheet" href=" {% static 'feedback/forms.css' %}">

</head>

<body>

{% block content %} {% endblock %}

</body>

</html>

# -------

{% extends 'feedback/base.html' %}

{% block title %} Страница отзывов {% endblock %}

{% block content %}

<h2>Оставьте отзыв</h2>

<form method="post">

{% csrf\_token %}

{% for field in form %}

<div class="form-style {% if field.errors %} errors {% endif %} ">

{{ field.label\_tag }}

{{ field.errors }}

{{ field }}

</div>

{% endfor %}

<br>

<button>Отправить</button>

</form>

{% endblock %}

# -------

{% extends 'feedback/base.html' %}

{% block title %} Отзыв готов {% endblock %}

{% block content %}

<h2>Ваш отзыв успешно обработан</h2>

{% endblock %}

**8.2 TemplateView**

from django.views.generic.base import TemplateView

class DoneView(TemplateView):

# def get(self, request):

# return render(request, 'feedback/done.html')

template\_name = 'feedback/done.html'

В этом задании необходимо:

создать шаблон list\_feedback.html и расширить его за счет base.html. В шаблоне должен выводится список всех отзывов, хранящихся в БД

создать класс-представление ListFeedBack, который унаследован от класса TemplateView. Данный класс должен использовать шаблон list\_feedback.html и в контекст передать все записи из модели Feedback

создать роут /list и привязать к нему класс-представление ListFeedBack

path("list", ListFeedBack.as\_view()),

class ListFeedBack(TemplateView):

template\_name = 'feedback/list\_feedback.html'

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['feedback'] = Feedback.objects.all()

return context

{% extends 'feedback/base.html' %}  
  
{% block title %} Отзывы {% endblock %}  
  
{% block content %}  
<h2>Список всех отзывов</h2>  
  
<form>  
 <table>  
 <tr>  
 <th><h2>Имя</h2></th>  
 <th><h2>Фамилия</h2></th>  
 <th><h2>Отзыв</h2></th>  
 <th><h2>Рейтинг</h2></th>  
 </tr>  
 {% for field in feedback %}  
 <tr>  
 <td><h2>{{ field.name }}</h2></td>  
 <td><h2>{{ field.surname }}</h2></td>  
 <td><h2>{{ field.feedback }}</h2></td>  
 <td><h2>{{ field.rating }}</h2></td>  
 </tr>  
 {% endfor %}  
 </table>  
</form>  
{% endblock %}

В этом задании необходимо:

создать шаблон detail\_feedback.html и расширить его за счет base.html. В шаблоне должна выводится детальная информация об одном отзыве, хранящемся в БД. Вид отображения в html выберите сами

создать класс-представление DetailFeedBack, который унаследован от класса TemplateView. Данный класс должен использовать шаблон detail\_feedback.html и в контекст передать одну запись из модели Feedback, полученную по идентификатору отзыва

создать роут /detail/<id\_feedback>и привязать к нему класс-представление DetailFeedBack. Обратите внимание на то, что в роут передается идентификатор отзыва. В методе get\_context\_data вы можете найти этот идентификатор в kwargs

urlpatterns = [

path("", FeedbackView.as\_view()),

path("done", DoneView.as\_view()),

path("<int:id\_feedback>", FeedBackUpdateView.as\_view()),

path("list", ListFeedBack.as\_view()),

path("detail/<int:id\_feedback>", DetailFeedBack.as\_view()),

]

class DetailFeedBack(TemplateView):

template\_name = 'feedback/detail\_feedback.html'

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['feedback'] = Feedback.objects.get(id=kwargs['id\_feedback'])

return context

{% extends 'feedback/base.html' %}

{% block title %} Отзыв {{ feedback.id }} {% endblock %}

{% block content %}

<h2> Уважаемый {{ feedback.name }}! </h2>

<h2> Ваш отзыв {{ feedback.feedback }} </h2>

<h2> успешно обработан</h2>

{% endblock %}

**8.3 ListView**

class ListFeedBack(ListView):

template\_name = 'feedback/list\_feedback.html'

model = Feedback

context\_object\_name = 'feedback'

# по умолчанию context\_object\_name = 'object\_list'

{% for field in feedback %}

<tr>

<td><h2>{{ field.name }}</h2></td>

<td><h2>{{ field.surname }}</h2></td>

<td><h2>{{ field.feedback }}</h2></td>

<td><h2>{{ field.rating }}</h2></td>

</tr>

{% endfor %}

Создайте отображение списков актеров и режиссеров в предыдущем проекте на основании класса ListView

path("directors", views.Show\_all\_dirs.as\_view(), name='all\_dirs'),

path("actors", views.Show\_all\_actors.as\_view(), name='all\_actors'),

class Show\_all\_dirs(ListView):

template\_name = 'movie\_app/all\_directors.html'

model = Director

context\_object\_name = 'directors'

class Show\_all\_actors(ListView):

template\_name = 'movie\_app/all\_actors.html'

model = Actor

context\_object\_name = 'actors'

**8.4 DetailView**

path("detail/<int:pk>", DetailFeedBack.as\_view()),

from django.views.generic import ListView, DetailView

class DetailFeedBack(DetailView):

template\_name = 'feedback/detail\_feedback.html'

model = Feedback

#-------

feedback == Feedback в нижнем регистре, поэтому все получилось

если имя будет не feedback, то:

class DetailFeedBack(DetailView):

template\_name = 'feedback/detail\_feedback.html'

model = Feedback

context\_object\_name = 'feed'

{% block content %}

<h2> Информация об отзыве </h2>

<h2> Уважаемый {{ feed.name }}! </h2>

<h2> Ваш отзыв {{ feed.feedback }} </h2>

<h2> успешно обработан</h2>

{% endblock %}

вместо

<h2> Уважаемый {{ feed.name }}! </h2>

можно писать

<h2> Уважаемый {{ object.name }}! </h2>

Создайте отображение детальной информации об одном актере и режиссере в предыдущем проекте на основании класса DetailView

path("directors/<int:pk>", views.DetailDirector.as\_view(), name='one\_dir'),

path("actors/<int:pk>", views.DetailActor.as\_view(), name='one\_actors'),

class DetailDirector(DetailView):

template\_name = 'movie\_app/one\_dir.html'

model = Director

context\_object\_name = 'dir'

class DetailActor(DetailView):

template\_name = 'movie\_app/one\_actor.html'

model = Actor

context\_object\_name = 'act'

**8.5 FormView**

from django.views.generic.edit import FormView

class FeedbackView(FormView):

# аналог функции get

form\_class = FeedbackForm

template\_name = 'feedback/feedback.html'

# аналог функции post

success\_url = '/done'

# Работает, но новые записи в бд не попадают, создаем функцию и все ок

def form\_valid(self, form):

form.save()

return super(FeedbackView, self).form\_valid(form)

# внутри шаблона нужно обращаться к переменной под тем именем form!

**8.6 CreateView**

from django.views.generic.edit import FormView, CreateView

class FeedbackView(CreateView):

model = Feedback

form\_class = FeedbackForm

template\_name = 'feedback/feedback.html'

success\_url = '/done'

# функция form\_valid теперь не нужна

# Можно и так

class FeedbackView(CreateView):

model = Feedback

fields = "\_\_all\_\_"

# form\_class = FeedbackForm

template\_name = 'feedback/feedback.html'

success\_url = '/done'

**8.6 UpdateView**

from .views import index, done, update\_feedback, FeedbackView, FeedBackUpdateView, DoneView, ListFeedBack, DetailFeedBack, FeedbackViewUpdate

path("update/<int:pk>", FeedbackViewUpdate.as\_view()),

from django.views.generic.edit import FormView, CreateView, UpdateView

class FeedbackViewUpdate(UpdateView):

model = Feedback

form\_class = FeedbackForm

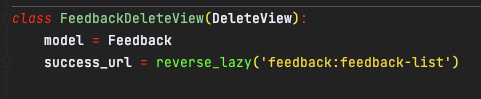
template\_name = 'feedback/feedback.html'

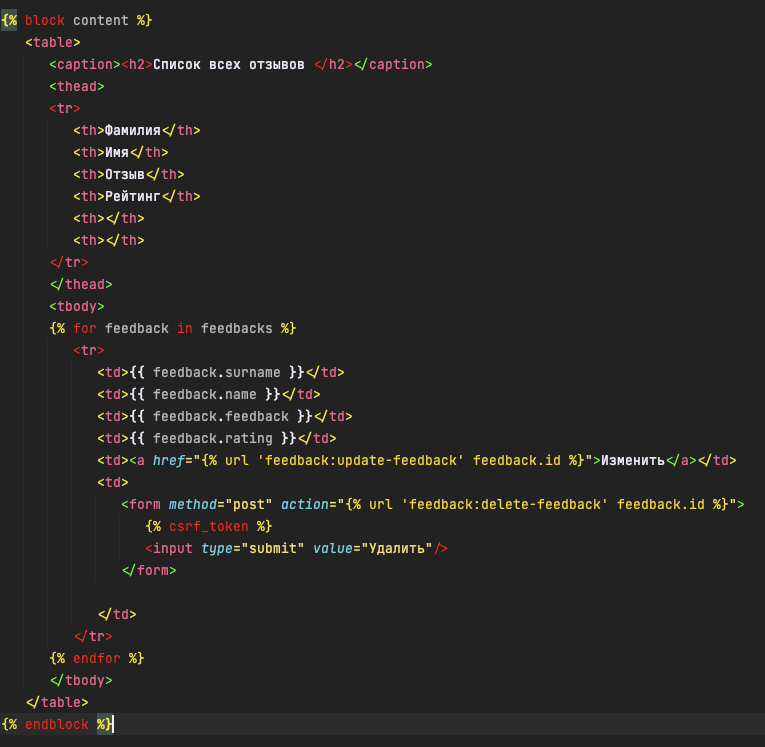
success\_url = '/done'

# Теперь можем редактировать записи в бд прямо в браузере

Из комментариев

Если захотите сделать класс для удаления отзыва





**9.1 Загрузка файла с формы**

python3 manage.py startapp gallery

INSTALLED\_APPS = [

"django.contrib.admin",

"django.contrib.auth",

"django.contrib.contenttypes",

"django.contrib.sessions",

"django.contrib.messages",

"django.contrib.staticfiles",

"feedback",

"gallery"

]

gallery templates gallery load\_file.html

{% extends 'feedback/base.html' %}

{% block title %} Добавление файла в галлерею {% endblock %}

{% block content %}

<h2>Загрузите файл</h2>

<form method="">

<input type="file">

<button type="submit"> Загрузить </button>

</form>

{% endblock %}

urlpatterns = [

path("admin/", admin.site.urls),

path("", include('feedback.urls')),

path("", include('gallery.urls')),

]

from django.urls import path, include

from . import views

urlpatterns = [

path("load\_image", views.GalleryViev.as\_view),

]

from django.shortcuts import render

from django.views import View

# Create your views here.

class GalleryViev(View):

def get(self, request):

return render(request, 'gallery/load\_file.html')

def post(self, request):

pass

проходим по http://127.0.0.1:8000/load\_image

можно выбрать файл для загрузки из проводника

после загрузить появляется

http://127.0.0.1:8000/load\_image?

? это гет запрос

Но ничего не загружается пока

<h2>Загрузите файл</h2>

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

<input type="file" name="image">

<button type="submit"> Загрузить </button>

</form>

def storage\_file(file):

with open(f'gallery\_tmp/{file.name}', 'wb+') as new\_file:

for chunk in file.chunks():

new\_file.write(chunk)

class GalleryViev(View):

def get(self, request):

return render(request, 'gallery/load\_file.html')

def post(self, request):

temp = request.FILES['image']

storage\_file(temp)

return render(request, 'gallery/load\_file.html')

В папке проекта создаем папку gallery\_tmp

проходим по http://127.0.0.1:8000/load\_image

можно выбрать файл для загрузки из проводника загружаем и видим файл new\_image.jpeg

в папке gallery\_tmp

**9.2 Forms FileField**

Если нажать загрузить не выбрав файл получим ошибку исправляем

можно так

<input type="file" name="image" required>

но правильнее через создаие формы

в папке galery создаем файл forms.py

from django.forms import forms

class Gallery\_Upload\_Forms(forms.Form):

image = forms.FileField() # image берем из html

from django.http import HttpResponseRedirect

class GalleryViev(View):

def get(self, request):

form = Gallery\_Upload\_Forms()

return render(request, 'gallery/load\_file.html', {'form': form})

def post(self, request):

form = Gallery\_Upload\_Forms(request.POST, request.FILES)

if form.is\_valid():

# temp = request.FILES['image'] # можно и так оставить

temp = form.cleaned\_data['image']

storage\_file(temp)

return HttpResponseRedirect('load\_9.3 Model FileFieldimage')

return render(request, 'gallery/load\_file.html', {'form': form})

**9.3 Model FileField**

Делегирование джанге процесса сохрарнения файла без функции storage\_file

Создаем модель

Файлы сохраняем не в бд, а в спецхране, в бд только ссылки

from django.db import models

class Gallery(models.Model):

image = models.FileField(upload\_to='my\_gallery')

# python3 manage.py makemigrations

# python3 manage.py migrate

from .models import Gallery

class GalleryViev(View):

def get(self, request):

form = Gallery\_Upload\_Forms()

return render(request, 'gallery/load\_file.html', {'form': form})

def post(self, request):

form = Gallery\_Upload\_Forms(request.POST, request.FILES)

if form.is\_valid():

temp = form.cleaned\_data['image']

new\_image = Gallery(image=temp)

new\_image.save()

return HttpResponseRedirect('load\_image')

return render(request, 'gallery/load\_file.html', {'form': form})

Чтобы не плодить папки 'my\_gallery' в каждом приложении,

делаем одну папку 'uploads' в проекте

В сеттинге

MEDIA\_ROOT = BASE\_DIR / 'uploads'

теперь папка 'my\_gallery' внутри папки 'uploads'

все ок

Идем в бд, находим таблицу gallery\_gallery, удаляем лишние предыдущие записи,

записываем изменния

В видео мы создали с вами класс-представление GalleryView , унаследованный от класса View

Ваша задача создать класс CreateGalleryView . Он должен выполнять все тоже самое, что и класс GalleryView , только наследоваться нужно от класса CreateView

from django.views.generic.edit import CreateView

class GalleryViev(CreateView):

model = Gallery

fields = "\_\_all\_\_"

template\_name = 'gallery/load\_file.html'

success\_url = '/load\_image'

**9.4 Отображение файлов в шаблоне**

urlpatterns = [

path("load\_image", views.GalleryViev.as\_view()),

path("list\_image", views.ListGallery.as\_view()),

]

создаем шаблон

{% extends 'feedback/base.html' %}

{% block title %} Добавление файла в галлерею {% endblock %}

{% block content %}

<h2>Список файлов</h2>

{% endblock %}

from django.views.generic import ListView

class ListGallery(ListView):

model = Gallery

template\_name = 'gallery/list\_file.html'

идем в бразер путь ОК, но пока файлы не отображаются

В сеттинге

MEDIA\_URL = '/my\_gallery/'

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path("admin/", admin.site.urls),

path("", include('feedback.urls')),

path("", include('gallery.urls')),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

{% block content %}

<h2>Список файлов</h2>

<ol>

{% for record in records %}

<li>

<img src="{{ record.image.url }}" alt="" height="400">

</li>

{% endfor %}

</ol>

{% endblock %}

все файлы отобразились в браузере --> OK