
17. Django (2)

— 05 декември 2023 —

Но преди това - домашното

- Да, беше тегаво...
- На тези, които са се справили добре - поздравления!
- На тези, които са се справили - пак поздравления!
- Идеята е да се научите да пишете Python, не да ви тегавеем с оценки накрая на семестъра, така че не взимайте тези 5-6-10 точки навътре.
- Което изисква писане на Python. :)

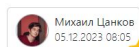
Поука

- Но ако трябва да си извадите една поука от това домашно...
- **Пускайте си примерния код!**

```
>>> from actually_good_solution import Potion
>>>
>>> # Единствената креативност е в тези 3 реда
>>> class Target:
...     def __init__(self):
...         self.size = 5
...
>>> target_1 = Target()
>>>
>>> # Оттук нататък е пример от условието
>>> effect = {'grow': lambda target: setattr(target, 'size', target.size*2)}
>>> grow_potion = Potion(effect, duration=2)
>>> grow_potion.grow(target_1)
Traceback (most recent call last):
  File "<pysHELL#44>", line 1, in <module>
    grow_potion.grow(target_1)
AttributeError: 'Potion' object has no attribute 'grow'
```

Добре де, още няколко поуки

- Не оставяйте всичко за последния момент.
- Възползвайте се от тестовете, които колегите ви публикуват:



Михаил Цанков
05.12.2023 08:05

Ето още малко тестове <https://pastebin.com/HPSDewDa>

Ако някой нещо му гърми 🙄

[Махни звездата](#) [Редактирай](#)



Костадин Русалов
05.12.2023 03:16

Ето малко тестове, като тук-там може да има някой изпуснат кейс или някой недефиниран, или някой грешен (дано няма такива)

```
import unittest
```

```
grow = ('grow': lambda _: setattr(_, 'size', lambda size: size + 2))
heal = ('heal': lambda _: setattr(_, 'health', lambda health: health + 10))
strengthen = ('strengthen': lambda _: setattr(_, 'strength', lambda strength: strength + 20))
fly = ('fly': lambda _: setattr(_, 'fly', True))
invisible = ('invisible': lambda _: setattr(_, 'invisible', True))
glow = ('glow': lambda _: setattr(_, 'glow', lambda glow: glow + 3))
armor = ('armor': lambda _: setattr(_, 'armor', lambda armor: armor + 5))
speed = ('speed': lambda _: setattr(_, 'speed', lambda speed: speed + 2))
power = ('power': lambda _: setattr(_, 'power', 200))
```

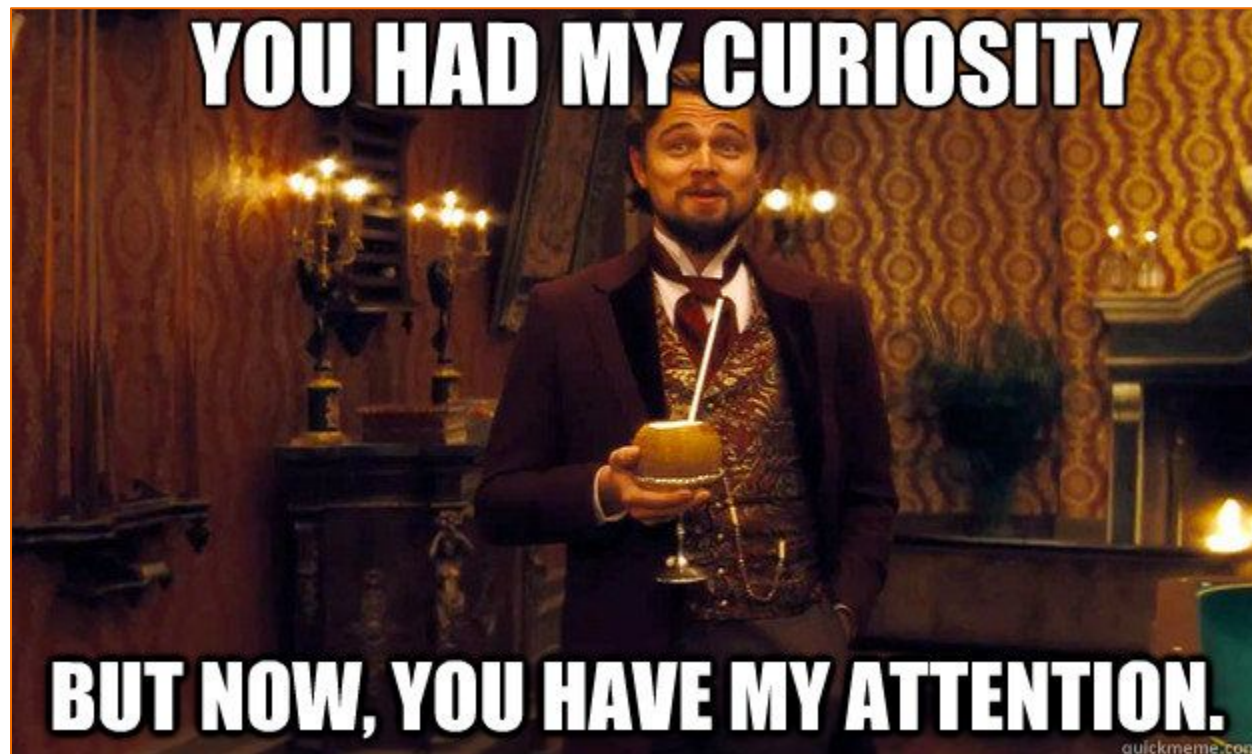
```
TRANSCENDED = 'Potion is now part of something bigger than itself.'
DEPLETED_P = 'Potion is depleted.'
DEPLETED_E = 'Effect is depleted.'
```

- Сумарно имате около 1200 реда тестове.
- Има и такива, които са по-тегави и от нашите!
- Ако техните минат, вероятно и нашите ще минат (изключвайки някакъв мега-corner case)

Резюме дотук

- Създадохме проект
- Създадохме приложение
- Създадохме админ
- Създадохме логин страница
- Създадохме логаут страница
- Създадохме welcome страница
- Заключихме welcome за логнати хора
- Свързахме логин с welcome, welcome с логаут, и логаут с логин
- Визуализирахме основата:
 - client -> server -> url -> view -> template -> HTML (обратно на клиента)
- Добавихме базов HTML темплейт
- Всички останали HTML темплейти го използват
- Добавихме директория за статични файлове
- Домавихме лого и го подадохме на браузъра
- Добавихме CSS файл и го подадохме на браузъра

Бази данни?



ORM (Object–relational mapping)

- Една доста приятна абстракция на базата данни
 - Пишем питонски код, който се конвертира до SQL заявка
 - Не ни интересува каква е точно базата отдолу
 - ORM-а да му мисли
 - Python клас = SQL таблица
 - Атрибут на класа = Колона в таблицата
 - Инстанция на класа = Запис в таблицата (кортеж)
-
- Полетата могат да държат само скаларни типове
 - За сложни типове трябва да създадем нов клас + таблица
 - Всеки обект има уникален номер (id)
 - За да реферират друг обект, трябва да ползвате него

ORM наглядно

```
class Car:
```

```
    def __init__(self, model, year):  
        self.model = model  
        self.year = year
```

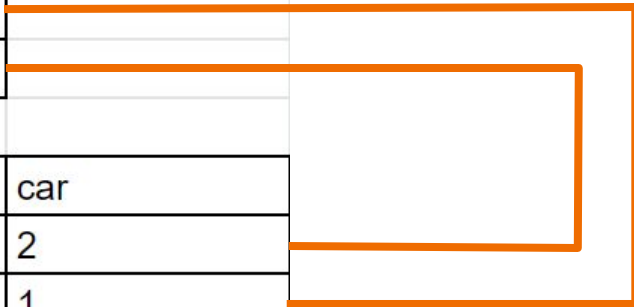
```
car1 = Car('Лада Нива', 1991)  
car2 = Car('Лада петичка ', 1992)
```

```
class Person:
```

```
    def __init__(self, first_name, last_name, car):  
        self.first_name = first_name  
        self.last_name = last_name  
        self.car = car
```

```
person1 = Person('Георги', 'Кунчев', car2)  
person2 = Person('Виктор', 'Бечев', car1)
```

Car	model	year	
1	Лада Нива	1991	
2	Лада петичка	1992	
Person	first_name	last_name	car
1	Георги	Кунчев	2
2	Виктор	Бечев	1





Да се върнем на проекта

- По-късно може да се погрижим за още визуални модификации по проекта
- Както и за регистрацията на нови потребители
- Но ние искаме rapid prototyping, така че веднага се втурваме да дефинираме модели в базата данни
- Вече говорихме за ORM, така че знаем, че за да дефинираме нова таблица в базата данни, просто трябва да дефинираме клас за нея.
- Класът наричаме модел
- Моделите стоят в `models.py`
- Ако ви трябва информация - документацията на Django
- Да направим един модел на продукт за пазаруване



Да добавим модела в admin панела

- Вече видяхме как изглежда admin панела
- Силата му е в това, че може лесно да борави с базата данни
- Стига да му кажете как
- Да представим новия си модел на admin панела

Да видим дали работи: <http://localhost:8000/admin>



Опааа - миграции

- Променяйки няколко реда код, не можем да очакваме базата данни да се промени сама
- А и не би трябвало да е така
- Обикновено промени по базата данни са трудоемки, но...
- Django is unchained
- Всяка промяна по моделите, която изисква промени по базата данни, изисква миграция:
 - `$ python manage.py makemigrations`
 - `$ python manage.py migrate`
- Фактически, първата команда създава един файл, който описва какво да се случи по време на самата миграция
- Файлът седи в директорията *migrations* на приложението
- Ако трябва - променят го
- Обикновено никога не трябва и просто пускате *migrate* за да изпълните файла

* Аз съм добавил самата база данни в репо-то си, за да мога да ви споделя всичко.
Обикновено това не се прави.

Да видим дали работи: <http://localhost:8000/admin>



Да добавим и модел за списъка за пазаруване

- Имаме модел за продукт, но той не е много полезен, освен ако не е в списък за пазаруване
- Да добавим и един такъв модел
- Разбира се, добавяме го в admin панела
- Разбира се, мигрираме базата данни

Да видим дали работи: <http://localhost:8000/admin>

А как списъкът ще знае кои продукти съдържа?



- За да свържем двата модела, трябва да създадем поле (колона) в единия от тях, който сочи към другия
- Базата данни по подразбиране (SQLite) не поддържа колони с променлива дължина
- Това значи, че не можем да дефинираме колона в списъка за пазар, която да съдържа много продукти
- Но можем да дефинираме колона в продукта, която посочва списъкът, от който е част този продукт
- Така или иначе Django може лесно да събере всички продукти от даден списък, така че връзката работи двустранно
- Дори да направите обратното, пак работи, но само защото Django прави едни магии с таблица посредник

- Разбира се, после мигрираме
- Забележете `null=True` - без него не можем да мигрираме, защото Django не знае какво да сложи в новодобавената колона

Да видим дали работи: <http://localhost:8000/admin>



А това `null=True` не е ли проблем?

- Да, реално погледнато, не искаме продукти без списък, но нямаше как да мигрираме
- След като вече сме мигрирали, можем да променим базата данни, така че да няма продукт без списък, след което да махнем `null=True`
- Това е просто да за визуализираме как става. По време на development, далеч по-лесно е да се изтрие базата данни, защото тя съдържа просто тестова информация

```
$ python manage.py shell
>>> from gui.models import ShoppingList
>>> some_list = ShoppingList.objects.all()[0]
>>> from gui.models import ShoppingItem
>>> all_items = ShoppingItem.objects.all()
>>> for item in all_items: item.shopping_list = some_list
>>> for item in all_items: item.save()
>>> quit()
```

Да видим дали работи: <http://localhost:8000/admin>



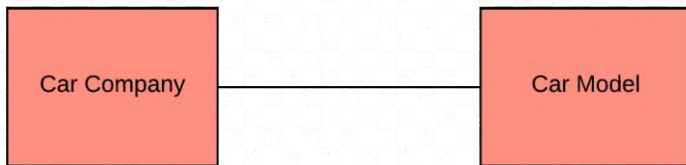
Една бърза вметка (модификация)

- Моделите ни в admin панела излизат с пореден номер на реда от базата данни
- Лесно можем да го контролираме с `__str__` на модела
- Да го направим

Да видим дали работи: <http://localhost:8000/admin>

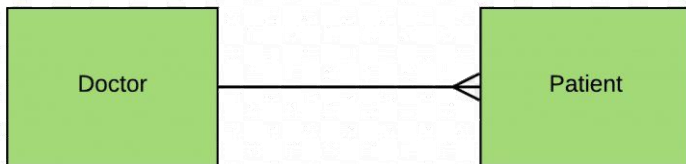
Връзка между моделите от базата данни

One to One



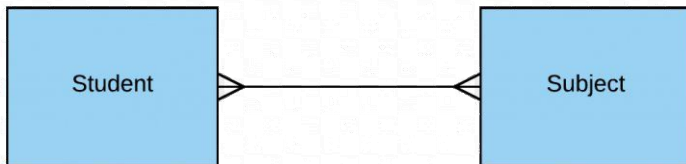
`models.OneToOneField`

One to Many



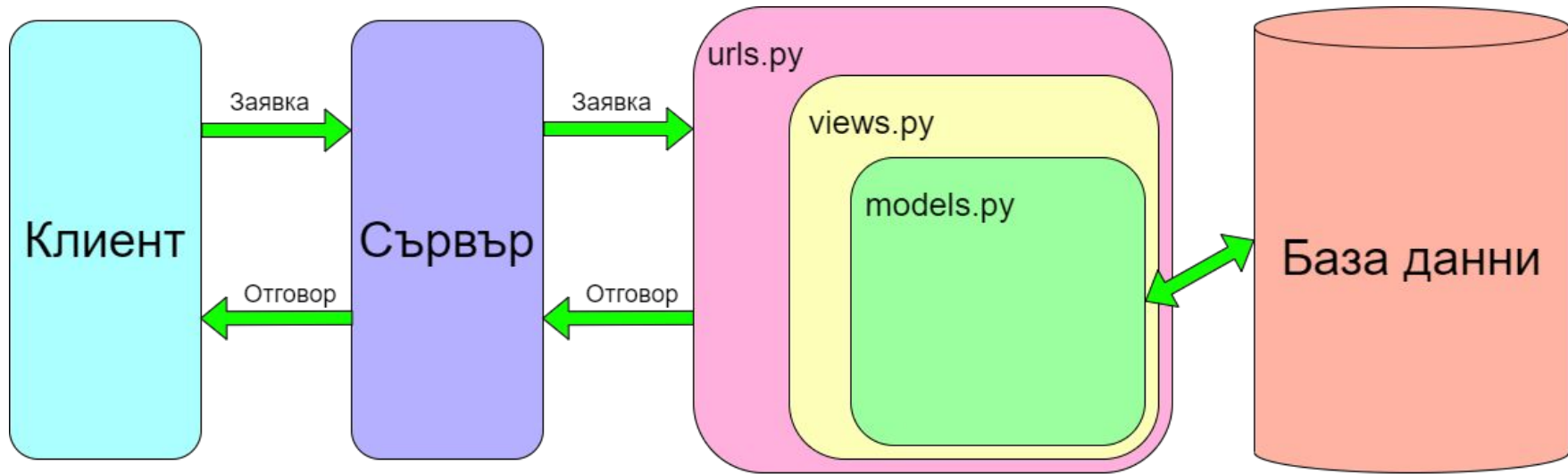
`models.ForeignKey`

Many to Many



`models.ManyToManyField` (Django прави таблица задкулисно, за да се справи с това)

Да си припомним една стара картинка





Да направим последната връзка

- Ако сега view-то бръкне в базата данни, имаме пълната картинка
- Ще съберем всички списъци, собственост на текущия потребител
- Ще ги подадем на темплейта
- А той ще ги визуализира в HTML

- Казах ли че темплейтите поддържат цикли и условия?

Да видим дали работи: <http://localhost:8000>



И какво? Как да видя какво има в списъка?

- Ще направим всеки списък - линк
- Линкът ще води към нова страница, която ще показва продуктите на списъка
- Линкът е базиран на "pk" (primary key) на обекта
- Това е уникален идентификационен номер за обекта в базата данни (пореден номер)

Да видим дали работи: <http://localhost:8000>

Да направим отделна страница за всеки списък



- Добре де, не отделна за всеки списък
- One page to rule them all!
- Добавяме `related_name` в модела, за да можем да вземем всички продукти от даден списък
- Добавяме нов `url`, спрямо линковете, които вече създадохме
- Добавяме ново `view`, което отговаря за този `url`
- Добавяме нов темплейт, който това `view` ще използва

Да видим дали работи: <http://localhost:8000>



Няма линк към началната страница

- Така е добре, но ако съм в даден списък, не мога да се върна в началната страница
- Освен това има някои неща, които изглеждат кофти
- Да го пипнем тук-там

Да видим дали работи: <http://localhost:8000>



Хей, да не забравяме мобилните устройства

- Има някои стандартни неща, които мобилните устройства изискват
- Най-вече, защото са в портрет, а не в лендскейп
- Тествайте с dev менютата на брауъра си
- Да го направим една идея по-добре

Да видим дали работи: <http://localhost:8000>



Да добавим останалите полета на продукта

- Имаме име на продукт, но нямаме количество и информация дали вече е закупен
- Цялата информация е вече в темплейта
- Само трябва да я използваме
- За (не)закупените продукти ще използваме input, защото той ще е полезен съвсем скоро

Да видим дали работи: <http://localhost:8000>

Едно лирическо отклонение

- Вече видяхме какво са
 - View-та
 - Темплейти
 - Модели
- Сигурно сте чували за шаблона MVC (Model-View-Controller)
- Реално тук правим същото нещо:
 - Django view - controller
 - Django template - view
 - Django model - model

Резюме дотук

- Дефинирахме модели
- Добавихме моделите в admin панела
- Приложихме нужните миграции по базата данни
- Свързахме двата модела
- Направихме така, че view да използва базата данни, за да визуализира списъци и продукти
- Добавихме отделни страници за всеки списък
- Погрижихме се за навигация между страниците и за мобилните устройства

Front-end функционалност



- За да позволим на потребителите да модифицират базата данни, трябва да им дадем контрол
- Можем да използваме форми като тази, която използваме за логин
- Но това е дървено, защото презарежда цялата страница
- Далеч по-удачно е да изпратим заявката задкулисно и да променим само това, което е нужно
- За целта ни трябва JavaScript
- Нека като начало направим продуктите clickable

- Аз използвам vanilla JavaScript, но ви съветвам да използвате някакъв фреймуърк - jQuery, ReactJS, AngularJS...
- Закачвам click ивенти за всеки продукт
- При всеки click (от)цъквам продукта чрез вече скрития checkbox и добавям клас на продукта, за да го визуализирам различно посредством CSS
- Засега това няма отношение към back-end-а, т.е. Django
- Wait for it...

Да видим дали работи: <http://localhost:8000>

AJAX



- За да изпратим заявка към сървъра задкулисно, използваме асинхронни заявки
- AJAX - Asynchronous JavaScript and XML
- Първо правим нов url, който ще се използва за заявката
- Правим ново view, което да я обработва и да връща отговор
- Правим AJAX заявка през JS към Django и обработваме отговора

Да видим дали работи: <http://localhost:8000>



Купуването - добре. А премахването?

- Да добавим и функционалност за премахване на продукт
- Вече знаем как:
 - Нов url
 - Ново view
 - Нов бутон
 - Нов JS, който слуша бутона и праща заявка на новия url

Да видим дали работи: <http://localhost:8000>



Въх - магнах нещо без да искам

- Да направим така, че хората да могат да добавят нови продукти
- Нов url
- Ново view
- НО, този път, няма да използваме AJAX, за да визуализираме и другия подход
- Правим форма във вече съществуващия темплейт
- Събмитвайки формата, отиваме на новия url и съответно на новото view, а то ще ни връща на оригиналното view след като модифицира базата данни
- Да, така презареждаме цялата страница, но пък не се притесняваме как ще модифицираме HTML-а при добавяне на нов продукт (което не е драма, ако парснем темплейта от view-то)
- За да не валидираме като алтави, в Django имаме възможността да дефинираме форма, която сама да валидира данните автоматично
- Ясно ли е? Разбира се, че не. Да видим кода...

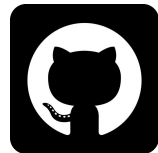
Да видим дали работи: <http://localhost:8000>

А списъците?



- Ок, да преговорим процеса като направим така, че да мога да добавям и списъци
- Нов url
- Ново view
- Нова форма в темплейта
- Нова форма във form.py

Да видим дали работи: <http://localhost:8000>



Въх - добавих нещо без да искам

- Да направим така, че да можеш да махаш списъци
- Рецептата вече приложихме за продуктите, така че просто преговаряме
- Нов url
- Ново view
- Нов бутон
- Нов JS, който праща AJAX към новия url

Да видим дали работи: <http://localhost:8000>

Резюме дотук

- Разгледахме разликите между събмитване на форма и AJAX заявки
- Позволихме купуването на продукти чрез AJAX
- Позволихме добавяне на нови списъци и продукти чрез презареждане на цялата страница
- Позволихме премахването на списъци и продукти чрез AJAX
- Видяхме как можем да валидираме данни чрез форма
- Видяхме как можем да редактираме базата данни през view
 - Добавяне
 - Премахване
 - Модифициране



Да се върнем малко на потребителите

- Крайно време е да позволим регистрация
- За жалост, `django.contrib.auth` няма готово view за това
- Не знам защо
- Но ще си направим
- Правим url
- Правим view
- Правим форма
- Правим темплейт

Да видим дали работи: <http://localhost:8000/register>

Потребителите са в admin панела: <http://localhost:8000/admin/auth/user/>



Да го довършим

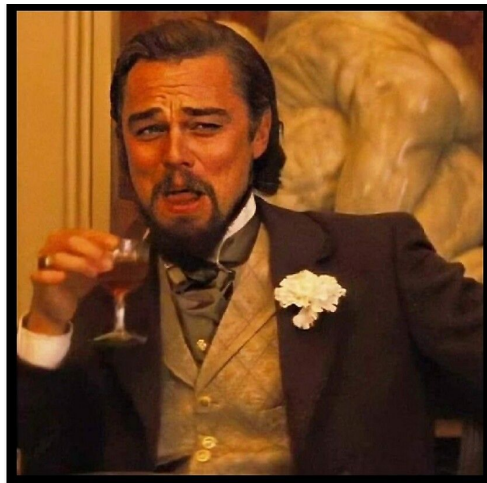
- Да добавим линк за регистрация в логин темплейта
- Да добавим линк за логин в регистрационния темплейт
- Да разкараме грозните инструкции за регистрация

Да видим дали работи: <http://localhost:8000>



Грандиозната промяна

- Е...има един самотен commit, който оцветява грешките в червено
- Не заслужава слайд, но все пак го има като commit, така че нямам избор
- За да не стои празен слайда...

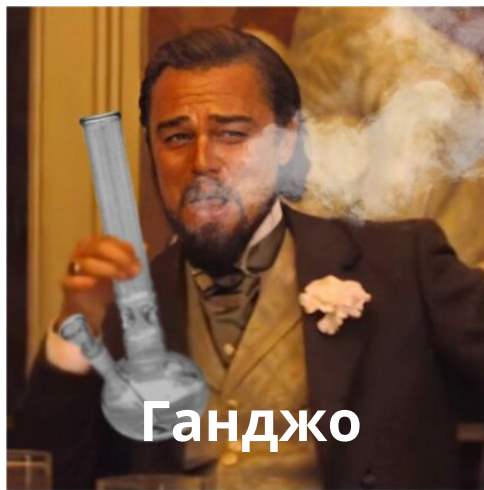


Да видим дали работи: <http://localhost:8000>



Грандиозната промяна се завръща

- Не искаме да показваме просто празен контейнер при празен списък или липса на списъци
- Просто добавяме едно малко if-че за да се справим с това
- Междувременно, най-готината анаграма на Django е...

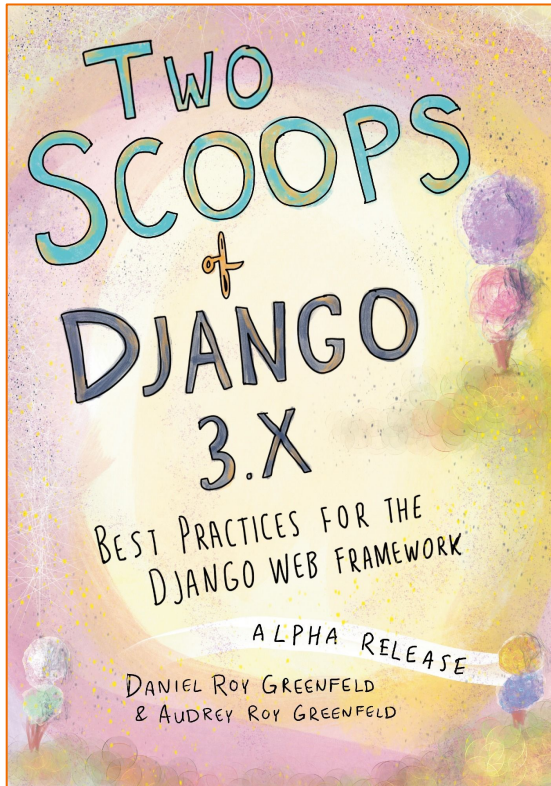


Да видим дали работи: <http://localhost:8000>

Грандиозно резюме

- Направихме “пълна” потребителска функционалност
 - Регистрация
 - Логин
 - Логаут
 - Модифициране (примерно смяна на парола) ще направим, ама друг път
- Направихме модели за базата данни и ги свързахме помежду им
- Направихме URL-и
- Направихме VIEW-та
- Направихме темплейти
- Направихме пълно презареждане на страницата при заявка, както и задкулисни заявки с AJAX
- Направихме сервиране на статични файлове
- Направихме валидации с форми
- Запознахме се с admin панел
- Модифицирахме базата данни чрез shell-a
- Разгледахме основни понятия и техники в HTML/CSS/JS
- Разгледахме основите на бази данни и сървъри

Добра основа, която да надградите



django The web framework for perfectionists with deadlines.

OVERVIEW DOWNLOAD DOCUMENTATION NEWS COMMUNITY CODE ABOUT DONATE

Documentation

Django documentation

Everything you need to know about Django.

How the documentation is organized

Django has a lot of documentation. A high-level overview of how it's organized will help you know where to look for certain things:

- **Tutorials** take you by the hand through a series of steps to create a Web application. Start here if you're new to Django or Web application development. Also look at the "First steps" below.
- **Topic guides** discuss key topics and concepts at a fairly high level and provide useful background information and explanation.
- **Reference guides** contain technical reference for APIs and other aspects of Django's machinery. They describe how it works and how to use it but assume that you have a basic understanding of key concepts.
- **How-to guides** are recipes. They guide you through the steps involved in addressing key problems and use-cases. They are more advanced than tutorials and assume some knowledge of how Django works.


First steps

Are you new to Django or to programming? This is the place to start!

- **From scratch:** [Overview | Installation](#)

Search 2.0 documentation

Support Django!

 [python](#) donated to the Django Software Foundation to support Django development. Donate today!

Browse

- Prev: [Django documentation contents](#)
- Next: [Getting started](#)
- [Table of contents](#)
- [General index](#)
- [Python Module Index](#)

You are here:





Бонус - API приложение

- Да боравим с данните през брауъра е добре
- Но понякога е нужно някаква програма да го прави
- Например, толкова много си харесвате списъка за пазаруване, че искате да направите мобилно приложение, което го използва - без брауър
- Готово - с Django можете да направите API, с което вашето приложение да общува
- За целта, инсталираме няколко фреймуърка
 - `$ pip install djangorestframework`
 - Подготвя Django да изпълнява REST заявки
 - `$ pip install djangorestframework-api-key`
 - Аутентикира проекта ви чрез token
- Другото го знаем
 - `$ python manage.py startapp api`
 - Добавяме двата фреймуърка и api в `setting.py`

Какво е REST

- REST - REpresentational State Transfer
- Това не е протокол, а архитектурен дизайн
- Изисква
 - Обмен на данни между сървър и клиент
 - Stateless - сървърът не пази информация за клиента
 - Сервираната информация е самодостатъчна за интерпретиране
 - Кеширане - сървърът изрично казва какво може и какво не може да се кешира с цел спестяване на последващи заявки
 - Възможност за прилагане на многослойна система (сървъри посредници)



Да добавим ново view

- Нека просто имаме view, което връща списъка с всички списъци за пазаруване
- Правим urls.py
- Добавяме референция за този файл към глобалното urls.py
- Дефинираме нов url
- Дефинираме ново view



Да си добавим един файл, с който да тестваме

- За по-лесно, можем да си направим един файл, с който да пращаме заявки.
- Той просто праща заявка и принтира отговора

Може и тук, но само засега: http://localhost:8000/api/get_shopping_lists

Да добавим още едно view



- Нека да опитаме да аутентикираме потребител посредством заявка
- Най-вероятно доста тъпа идея за production, но тук учим Django, така че става
- Ако искате да аутентикирате отделни потребители в Django Rest, най-добре използвайте `rest_framework.authtoken`, което може динамично да създаде token, за цялата “сесия”
- В конкретния случай по-скоро правим дизайн, който да позволи друг наш сървър да използва shopping list-a, защото имаме глобален token, който има достъп до всички API view-та.

Да се защитим



- Всичко до този момент можеше да направим и без Django Rest API
- Нека кажем на Django, че за да използваш това API, трябва да имаш ключ
- Това е една от големите ползи
- Това изисква миграция, защото ключовете стоят в базата данни

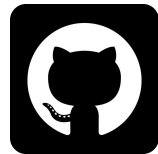


Да генерираме ключ

- ```
$ python manage.py shell
>>from rest_framework_api_key.models import APIKey
>>api_key, key = APIKey.objects.create_key(name='mobile app')
>>key # 7LFZ9VrM.Ec2d76wKcMzN5DNJcU4YMw40CxUwgbat
```
- И добавяме токена в хедъра на заявките си

Контролирайте ключовете през admin панела: <http://localhost:8000/admin>

# Сериализиране



- Истинската сила на Django Rest е, че може да сериализира обекти сам
- Това ни позволява лесно да добавяме и махаме обекти от базата данни, базирано на заявки
- Да го направим

# Рефакторирайте

- Погледнете проекта си от птичи поглед
- И не само - влезте в детайлите
- Ако трябва - сменете дизайна
- Преименувайте променливи
- Преместете код
- Сменете технология
- Кодът трябва да е ЖИВ
- Е...аз поглендах проекта си и...



# Финални промени

- Не ми хареса API да бърка в GUI и добавих нов app - base
- Добавих списък с продукти за всеки списък за пазаруване в админ панела
- Добавих декоратор за валидиране на права върху даден списък, за да не дублирам код
- Мигрирах към Bootstrap и jQuery
- Направих всичко красиво

По-добре е, нали: <http://localhost:8000>



# Въпроси?

