

---

---

# 19. Django (1)

— 17 декември 2024 —

---

---

# Въпрос

Каква е разликата между конкурентност и паралелизъм?

**Конкурентност:** Когато две изчисления нямат ясно дефинирана последователност на изпълнение.

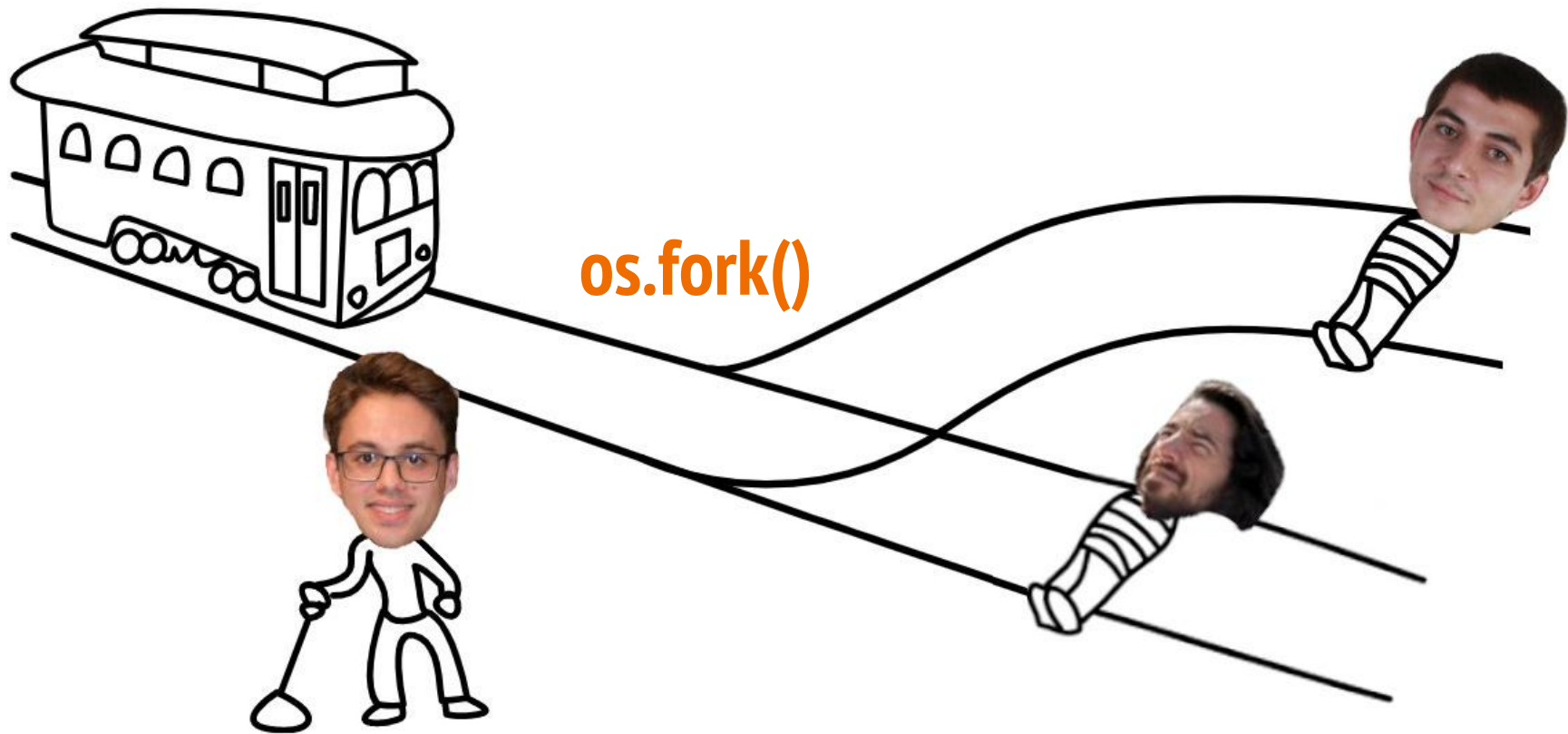
**Паралелизъм:** Две изчисления, които реално се изпълняват едновременно.

# Въпрос+

Каква е разликата между `threading.Semaphore` и `threading.Lock`?

`Lock` позволява само една нишка, а `Semaphore` позволява няколко.

# Въпрос++



Фокусът ще е Django



# Но ще разгледаме и какво става около него

## Servers



dj

## Databases



## Web



## Trinity




# За да визуализираме - списък за пазаруване


<https://github.com/gvkunchev/fmi-shopping-list>

Home Logout

## Списък 1

1	<input type="text" value="New product name"/>	Add
---	---	-----

1 Продукт 1 

1 Продукт 2 

Но първо...

**NETWORKING**



# Примитивно - Socket

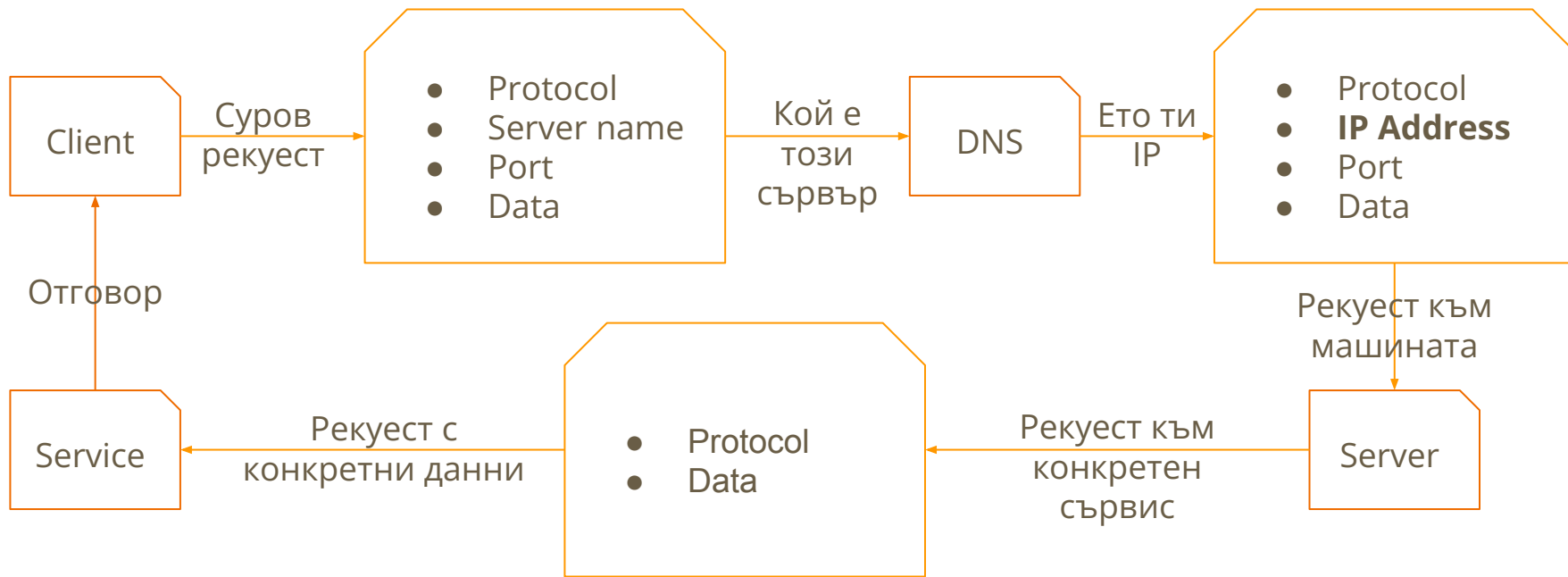
Server

```
1 import socket
2
3 socket_ = socket.socket()
4 host = socket.gethostname()
5 port = 9999
6
7 socket_.bind((host, port))
8 socket_.listen()
9 client_socket, addr = socket_.accept()
10
11 print(f"Got a connection from {addr}")
12 client_socket.send(b"I am listening")
13 client_socket.close()
```

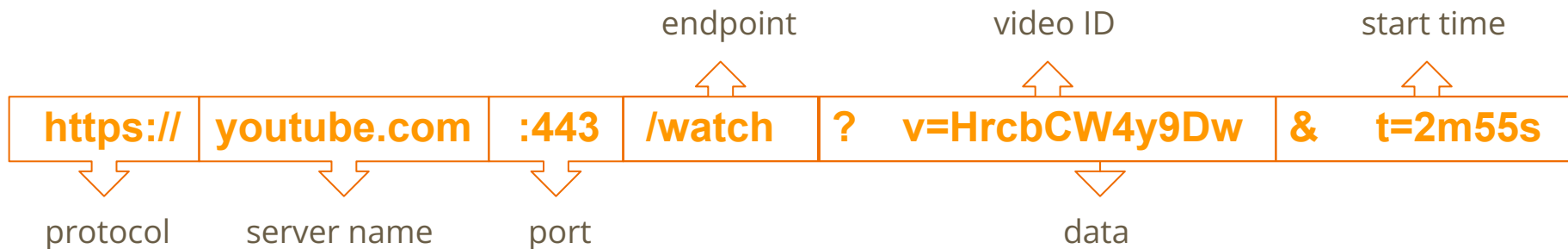
Client

```
1 import socket
2
3 socket_ = socket.socket()
4 host = socket.gethostname()
5 port = 9999
6
7
8
9
10 socket_.connect((host, port))
11
12
13
14 response = socket_.recv(1024)
15 socket_.close()
16 print(response)
```

# “Цялата“ картинка



# Например: Бигъс Дикъс



# Let's unchain Django

```
python -m pip install django
```



# Какво е Django?



# А зацо не Flask?

```
python -m pip install Flask
```



# Не, наистина, защо не Flask?

- Django е много по-лесен за първи web проект
  - Batteries included
- Ако наистина знаете какво правите, опитайте Flask, но...

	django	Flask
Admin Panel	🟢	🟡
Web Framework	🟢	🟢
Database	🟢	🟡
Performance	🟢	🟢
Security	🟢	🟡
Flexibility	🟡	🟢
Usage & Community	🟢	🟡
Template Engine	🟢	🟢
Reusable Components	🟢	🟢



You take the blue pill – the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill – you stay in Wonderland, and I show you how deep the rabbit hole goes. Remember, all I'm offering is the truth – **nothing more.**"

# Единственото преимущество на Flask

- Името
- Но аз ще подготвя петиция за смяна на името на Джанго и ще се справя и с това

ДАМА djangο

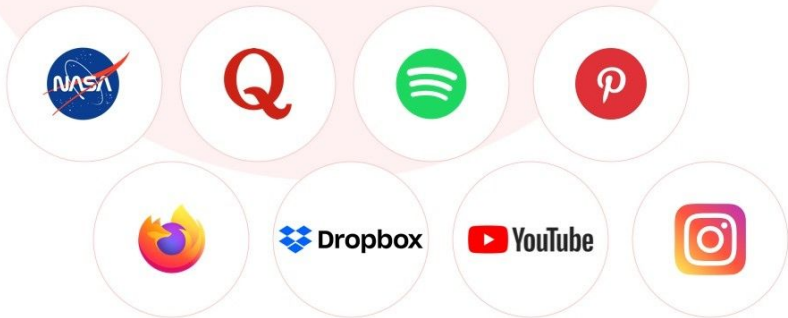




# Да вземем пример



## Popular Django Websites



## Popular Flask Websites



Flask

# Интродукция



- Цялата лекция се върти около един проект, който има за цел да визуализира основните неща, които трябва да знаете, за да направите едно web приложение
- Всеки слайд, в който видите котето горе в дясно, е обвързан с конкретен commit от проекта
- Проектът е на-commit-ен сравнително атомарно, за да можете да проследите всички детайли от процеса възможно най-лесно
- Ще се въртим между
  - слайдовете тук
  - commit-и в GitHub
  - VScode, който ще визуализира всяка стъпка от процеса на разработка
  - тестване на проекта в брауъра
- Ако искате да прегледате/изтествате кода вкъщи, клонирате репото
  - `$ git clone git@github.com:gvkunchev/fmi-shopping-list.git`
- Ако искате да прегледате конкретна версия на проекта, взимате hash-а на съответния commit и го checkout-вате
  - `$ git checkout df81b8098d487cc163d4e6b56cd7e8bd7ce7fbc0`
- Ако сте правили промени преди това, няма да можете преди да ги разкарате:
  - `$ git stash`

# Нов проект



```
$ django-admin startproject shopping_list
```

- + **manage.py** - интерфейс за команди към Django
- + **<project settings dir>** (конфигурации - съвпада с името на проекта)
  - + **\_\_init\_\_.py** - това все пак е пакет, така че...
  - + **wsgi.py** /whiskey/ (Web Server Gateway Interface) - synchronous callable - стандартен интерфейс между сървър и приложение
  - + **asgi.py** - същото като горното, но асинхронно
  - + **settings.py** - настройки за проекта. Виж документацията
  - + **urls.py** - линкове, които Django ще обслужва

```
$ python manage.py runserver # http://localhost:8000/
```

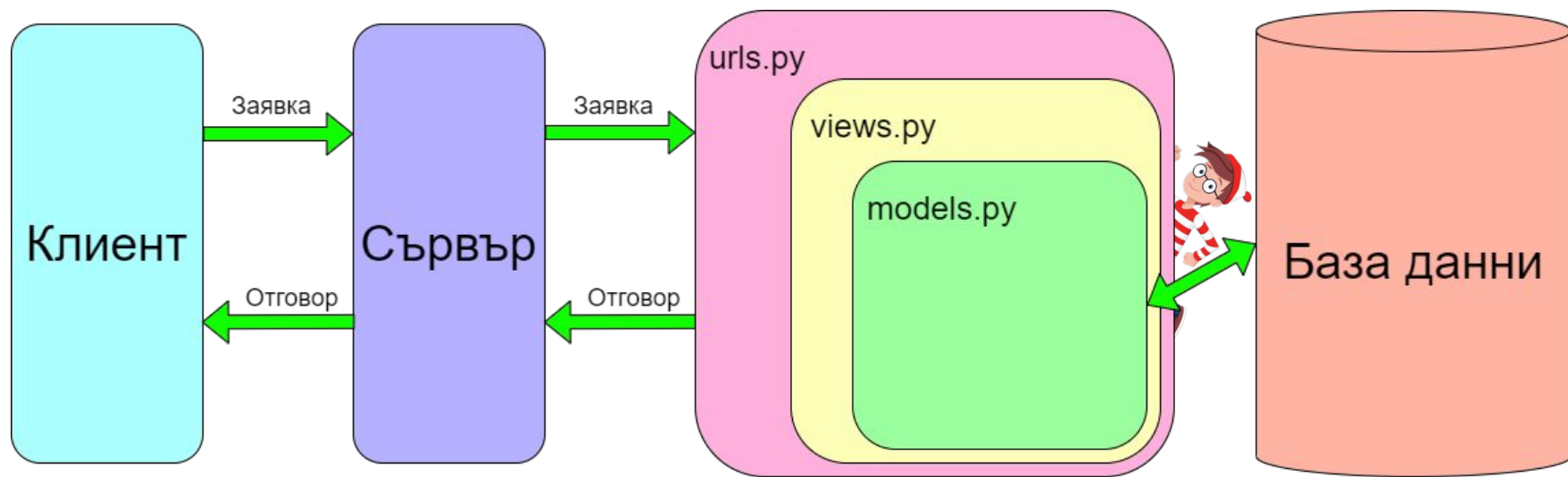


# Ново приложение (app)

```
$ cd shopping_list ; python manage.py startapp gui
```

- + **migrations** - скриптове за миграции на базата данни
- + **\_\_init\_\_.py** - това е пакет, така че...
- + **admin.py** - настройки за админ панела
- + **apps.py** - конфигурация за приложение - мета данни, но не само
- + **models.py** - дефиниции на модели за базата данни (ORM)
- + **tests.py** - то се знае
- + **views.py** - "врати", който обработват заявки и връщат отговори

# Как работи?



# Обратно на проекта



- Създаденото приложение трябва да добавим в `settings.py`, за да го направим отговорност на Django:
  - Django ще следи за промени по базата данни от него
  - Django ще използва моделите му за админ панела
  - Django ще пуска тестовете му автоматично
  - въобще...да се оправя сам, ние само пишем код с инструкции

# Да създадем view



- Връзката между базата данни и потребителя
- Получава заявка, обработва я и връща отговор
- Логиката на приложението
- В един момент ще установите, че е по-добре да не съдържа много логика

# Да създадем url



- Дефиницията на това кой линк към кое view ще води
- Можете да дефинирате статични стойност
- Може и регулярни изрази
- По подразбиране не всеки app има urls.py файл, но практиката показва, че е по-лесно да се направи отделен





# Да кажем на Django за този нов url файл

- Django се интересува само от базовия `urls.py`, така че трябва да го уведомим за ръчно създадения файл

Вече имаме welcome страница: <http://localhost:8000/>



# А потребители? Да започнем с Admin панела

- За да достъпим админ панела, трябва ни superuser
  - Инициализираме база данни
    - `$ python manage.py migrate`
  - Създаваме нов superuser
    - `$ python manage.py createsuperuser`
      - Username: admin
      - Email: admin@admin.admin
      - Password: admin (да, не ме интересува, че е лесна - направи я)

Вече имаме admin панел: <http://localhost:8000/admin>

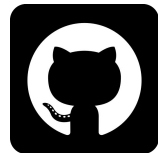
# Да заключим сайта



- Ще се подсигурим, че само логнати потребители виждат съдържание от сайта
- За целта, първо трябва да направим логин страница
- Всичко свързано с потребители може да използвате наготово: `django.contrib.auth`
  - Има си модели в базата данни
  - Има си форми за регистрация, логин, модифициране на потребителска информация...
  - Има си view-та за горепосочените форми
  - Има валидация на всичко в модела - пароли, валиден имейл...

Да видим какво очаква този url: <http://localhost:8000/login>

# Темплейт



- За да сервираме някакъв HTML, удачно е да го дефинираме в темплейт
- Разграничава HTML-а от логиката на приложението
- Позволява ни да дефинираме динамично съдържание
- Позволява ни да преизползваме HTML в различни темплейти

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

# Темплейт (дисекция)

- В конкретния случай използваме готова форма и само изграждаме обвивката ѝ
- `csrf_token` - подписигурява, че формата се събмитва от легитимен контекст
- Формата се събмитва чрез метода POST
  - POST - иска отговор, но добавя повече "скрита" информация в заявката
  - GET - просто иска отговор, въпреки че и той може да добави информация в самия URL, но тя е видима
  - Ако вместо POST, за логин използваме GET, логините ще се видят в адрес бара на брауъра.
  - Ако вместо GET, използваме POST, няма да може да споделим линк с някого

```
{% block content %}  
<h2>Login</h2>  
<form method="post">  
  {% csrf_token %}  
  {{ form.as_p }}  
  <button type="submit">Login</button>  
</form>  
{% endblock %}
```



# Страница след логин

- По подразбиране, потребителите отиват на `/accounts/profile/` след оторизация
- Е...да кажем, че ние не искаме да е така
- Можем просто да сменим глобалната настройка и готово
  - `LOGIN_REDIRECT_URL = '/'`

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

# Готово ли е?



- НЕ
- Да, вече имаме логин страница, но никъде не сме казали, че потребителят трябва да е оторизиран, за да достъпи нашата “Welcome” страница
- Това може лесно да стане с един декоратор

# A logout?



- Досега можехме само да се логваме, но не и да се логанем
- Е, винаги може да използваме админ панела, ако сме админи, но нашите потребители няма да имат достъп до него
- Отново ще използваме `django.contrib.auth`
- Да добавим линк за logout

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





# Къде трябва да отиват хората след logout?

- Логично е да ги подканим да се логнат
- Отново имаме глобална настройка за това: `LOGOUT_REDIRECT_URL = '/login'`

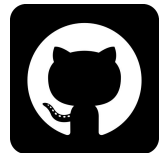
Да видим дали работи: <http://localhost:8000/logout>



# И какво? Сами да си пишат линковете ли?

- Разбира се, че не.
- Простият ни “Welcome” вече не е удачен
- Нека направим темплейт за него, който да съдържа и logout линк
- It ain't much, but it's honest work!

# Темплейтът е готов, но не се използва...



- За да използваме темплейта, просто си преправяме view-то

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

# Резюме дотук

- Създадохме проект
- Създадохме приложение
- Създадохме админ
- Създадохме логин страница
- Създадохме логаут страница
- Създадохме welcome страница
- Заключение 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)
- Нищо сложно
- Въпроси?



# Да го подготвим да расте

- Най-вероятно всички, или почти всички страници, ще се нуждаят от един и същ темплейт за HTML
  - Заглавие на страницата
  - Често използвани CSS/JS
  - Мета информация
  - Навигационни контроли
- Вместо всеки един темплейт да ги дублира, можем да ги отделим в основен HTML
- Можем да използваме block тагове, за да дефинираме къде да мушнем други темплейти



# Да използваме базовия темплейт

- Вече подготовения темплейт можем да преизползваме в останалите темплейти
- В нашия случай - index.html и login.html

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

# За темплейтите с любов

- Джанго използва DTL (Django template language)
- Доста сходен на Jinja2
- Има и други third-party опции, но обикновено не ви интересуват
- Някои базови неща за темплейтите
  - Expressions (variables): `{{ variable }}`
  - Statements (tags): `{% if something %} text {% endif %}`
    - For, if, with, include...and more
  - Filters: `{{ name|lower }}`
  - Inheritance: `{% extends "base.html" %} -> {% block content %}{% endblock %}`

# Да сложим лого



- Всеки сайт си има някакво лого, което седи в таба на браузъра
- И ние можем да сложим, но нека първо го добавим в проекта си
- За целта, създаваме static директория в приложението
- static са всички статични файлове, които сървърът предоставя - обикновено CSS, JS, картинки, файлове за сваляне...



# Да кажем на Django къде са статичните файлове



- Просто трябва да използваме настройката `STATICFILES_DIRS`
- Имайте предвид, че по подразбиране Django не сервира статични файлове в production (т.е. когато `DEBUG=False` в `settings.py`)
- Обикновено това е работа на самия сървър (e.g. Apache)
- Ако държите да сервирате статични файлове с Django, можете да използвате *middleware* - например *whitenoise*
- Така или иначе, преди това трябва да съберете всички статични файлове с:
  - `python manage.py collectstatic`
- Има чудесна статия в документацията на Django с всички детайли по темата

Да видим дали работи: <http://localhost:8000/static/logo.png>



# Да добавим логото в таба на брауъра

- Вече наличното лого можем да добавим в темплейта
- Можем да сложим пълния линк...
- но това ни обвързва с конкретна настройка, така че...
- далеч по-добре да използваме тага на Django за статични файлове, така че да се подсигурим, че настройките на проекта контролират напълно локацията на файловете
  
- Ако пипате static файлове, подсигурете се, че брауърът ви ги рефрешва (Ctrl+F5) - обикновено такива файлове не се нуждаят от презареждане (нали уж са статични) и брауърът ги кешира
- Има опции да форсирате презареждане от сървъра (добавя се dummy променлива в края на линка)

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

# Въпроси?

