Първи стъпки в Python/Django

# Django – въведение

## Инсталиране

1. Отваряме (инсталираме) PyCharm
2. Създаваме нов проект

* Понятие за проект.
* Проектите в Python – папки
* Име на проекта - MyProject

1. Инсталиране на Django

pip install django

1. Създаване на приложение

django-admin startproject blog

N/B При грешка с pip - Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

## Основни файлове

* Да се поясни що е то проект в Django.
* Отваряме папката на проекта и показвам структурата и.
* Обяснявам manage.py.
* Отварям папката с име като името на проекта
* Разглеждаме urls.py
* Разглеждаме settings.py – пояснявам DEBUG, DATABASES, LANGUAGE\_CODE

## Първи тест

* Стартирам тестовия сървър:

python mange.py runserver

* Поправям грешката като преминавам в съответната папка и стартирам отново
* Отварям препратката – отваря се тестовата страница на Django
* Променям езиковата настройка и обновявам страницата

## Създаване на приложение

* Да се поясни що е то приложение в Django.

Python manage.py startapp main

* В settings.py, секция INSTALLED\_APPS регистрираме приложението
* Отваряме папката на приложението и показвам структурата и.
* Разглеждаме файловете
* В папката на приложението създаваме празен файл urls.py
* В главния urls.py пренасочваме проследяване на url-ите към нашето приложение:
* Добавям include в импорта след path: from django.urls import path, include
* Добавям още един път за проследяване

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

* В новия urls.py копирам от стария и коригирам съдържанието:
* from django.urls import path  
  from . import views  
  urlpatterns = [  
   path('', views.index)  
    
  ]
* във vies.py създавам новата функция index:
* from django.shortcuts import render  
  from django.http import HttpResponse  
    
    
  # Create your views here.  
  def index(request):  
   return HttpResponse('<h4> Главна страница </h4>')
* пускам тестовия сървър и обновявам страницата
* добавям още една страница all\_articles (функция във vies.py и път в urls.py)
* def all\_articles(request):  
   return HttpResponse('<h4> Всички публикации </h4>')

## Използване на шаблони

1. В папката main създавам папка templates. Името е важно да е точно такова. В тази папка създавам папка със същото име като това на приложението.
2. В новосъздадената папка създавам шаблон (.html файл) с име index.html и попълвам title и body с някакво съдържание.
3. Във vies.py коригирам функцията index:
4. def index(request):  
    return render(request, 'main/index.html')
5. Обновявам основаната страница
6. Създавам още две страници – all\_articles и article

**Оформяне:**

## Шаблонизатор

1. Разглеждаме трите html файла и коментираме повтаряемостта в тях
2. Създавам нов файл base.html
3. Оформям в блокове променящата се част на съдържанието:

{% block content %}  
<h4>Всички публикации</h4>  
<p>Пълен списък на всички публикации</p>  
{% endblock %}

1. По същия начин в задавам блокове във файловете за отделните страници
2. В началото на всеки шаблон задавам:

{% extends 'main/base.html' %}

1. Например, файлът index.html трябва да изглежда така:

{% extends 'main/base.html' %}  
{% block title %} ГЛАВНА СТРАНИЦА {% endblock %}  
{% block content %}  
<h4>главна страница</h4>  
<p>някакъв текст</p>  
{% endblock %}

1. Предаване на параметри от изгледа от изгледа към шаблона
   1. Във views.py добавям параметър в render чрез който предавам речник с параметрите на страницата

def index(request):  
 params = {'page\_title': 'ГЛАВНА СТРАНИЦА',  
 'page\_header': 'Главна страница',  
 'page\_content': 'Текст на страницата',  
 }  
 return render(request, 'main/index.html', params)  
  
  
def all\_articles(request):  
 params = {'page\_title': 'Всички публикации',  
 'page\_header': 'Всички публикации',  
 'page\_content': 'Пълен списък на всички публикации',  
 }  
 return render(request, 'main/index.html', params)

* 1. Премахвам шаблона all\_articles.html т.к. той вече не е нужен

## Избор на тема:

1. Търсене в google по „free bootstrap templates“
2. Избирам Free Bootstrap Themes & Templates (<https://startbootstrap.com/themes>)
3. От менюто избирам Themes -> Blog->Clean Blog
4. Свалям архива и го разархивирам в някаква папка (например test)
5. В папката на приложението (main) създавам папка static
6. В създадената папка копирам всички общи файлове за темата
7. В началото на шаблона base.html вкарвам директива

{% load static %}

1. Коригирам препратките към всички статични ресурси, например

<link rel="icon" type="image/x-icon" href="{% static 'assets/favicon.ico' %}" />

1. Проверка и корекции
2. В urls.py добавям параметър name:

urlpatterns = [  
 path('', views.index, name='latest'),  
 path('all\_articles', views.all\_articles, name='all'),  
 path('article', views.article, name='single'),

1. В base.html вкарвам директива url на препратките и коригирам менюто:

<li class="nav-item"><a class="nav-link px-lg-3 py-3 py-lg-4" href="{% url 'latest' %}">Последни</a></li>  
<li class="nav-item"><a class="nav-link px-lg-3 py-3 py-lg-4" href="{% url 'all' %}">Всички</a></li>  
<li class="nav-item"><a class="nav-link px-lg-3 py-3 py-lg-4" href="{% url 'single' %}">Единична</a></li>  
<li class="nav-item"><a class="nav-link px-lg-3 py-3 py-lg-4" href="{% url 'latest' %}">Admin</a></li>

# Настройване на Django за работа с БД MySQL

При инсталиране Django установява по подразбиране настройки за работа с база данни sqlite. Така, без необходимост от допълнително конфигуриране, потребителят може с лекотата да съхранява и извлича на данни от БД. Sqlite е подходящa за малки уеб приложения, но за по-големи ще доведе до проблеми с производителността. За да се избегнат това, първата стъпка е да изберете подходяща база данни, способна да обслужва стотици или дори хиляди едновременни посещения. За тази цел може да се използва база данни MySQL.

Ще проследим всички необходими стъпки за свързване на Django с MySQL база данни.

## Настройване на MySQL база данни

### Създаване на нова база данни

Предполагаме, че имаме инсталиран MySQL и достъп до него чрез PhpMyAdmin.

1. Отваряме PhpMyAdmin.
2. Създаваме нова (празна) база данни.
3. Създаваме нов потребител и му даваме права върху създадената база данни.

## Настройване на Django проекта

### Стъпка 1: Инсталиране на MySQL клиент

Можете да инсталирате MySQL клиент директно с командата pip:

*pip install mysqlclient*

Ако инсталирането е успешно преминаваме към следващата стъпка. За съжаление на много системи тази команда дава грешка и инсталацията не минава. В такъв случай може да се използва като алтернативен подход инсталиране от подходящ за конкретната версия на Python mysqlclient wheel файл.

### Стъпка 2: Настройване на връзката към БД

Отваряме settings.py на проекта и настройваме DATABASES по следния начин

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'blog',  
 'USER': 'blog',  
 'PASSWORD': 'admin',  
 'HOST': 'localhost',  
 'PORT': '3306',  
 'OPTIONS':{'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'"}  
 }  
}

### Стъпка 3: подготвяме и изпълняваме миграциите

Отваряме терминала и подаваме команда за подготовка на миграциите:

*python manage.py makemigrations*

Ако няма грешки, то имаме връзка с базата данни и можем да извършим първата миграция:

*python manage.py migrate*

Отваряме PhpMyAdmin и виждаме, че в нашата БД има създадени таблици.

# Модели

## Създаване на модел за статиите

1. Отваряме models.py
2. Създавам клас за статиите:

from django.db import models  
from datetime import date  
  
# Create your models here.  
class article(models.Model):  
 title = models.CharField('Заглавие', max\_length=200)  
 content = models.TextField('Съдържание')  
 author = models.CharField('Автор', max\_length=50)  
 published = models.DateField('Публикувано на', default=date.today)  
   
 def \_\_str\_\_(self):  
 return self.title  
   
 class Meta:  
 verbose\_name = 'Статия'  
 verbose\_name\_plural = 'Статии'

1. Миграция:

* Python manage.py makemigrations
* Python manage.py migrate

1. Можем да видим, че в БД е създадена таблица article

# Панел на администратора

## Създаване на администратор

1. Стартираме тестовия сървър
2. Преминаваме на страница /admin (от адресната лента на браузъра)
3. Променям препратката в base.html да стартира админ панела:

<li class="nav-item"><a class="nav-link px-lg-3 py-3 py-lg-4" href="admin">Admin</a></li>

1. Спираме тестовия сървър и в терминала пишем команда

Python manage.py createsuperuser

1. Въвеждаме име на потребител, парола и потвърждаваме паролата. E-mail не въвеждаме, а на въпроса дали потвърждаваме такава парола отговаряме с “y”.

## Регистриране на таблица

В admin.py импортираме от models създадената таблица и я регистрираме:

from django.contrib import admin  
from .models import article  
  
  
admin.site.register(article)

# Извеждане на данни от БД

1. Във views.py модифицираме изгледа на основната страница

from .models import article  
  
def index(request):  
 articles = article.objects.all()  
 params = {'page\_title': 'ГЛАВНА СТРАНИЦА',  
 'page\_header': 'Главна страница',  
 'page\_content': 'Текст на страницата',  
 'articles': articles,  
 }  
 return render(request, 'main/index.html', params)

1. В index.html добавяме визуализиране на параметъра articles

<p>{{ articles }}</p>

1. Модифицирам извеждането да е по един елемент на ред
2. Примери за различни селекции:
   1. articles = article.objects.filter(title\_\_contains='1')
   2. articles = article.objects.order\_by('-id').filter(title\_\_contains='1')
   3. articles = article.objects.order\_by('-id')[:2]
3. Променям изгледите, така че едната страница да показва новите статии (срез по id), а другата – всички статии

# Коментари (работа със свързана таблица)

## Създаване на свързаната таблица

1. Дефинираме на модела

class Comments(models.Model):  
 article = models.ForeignKey(Article, on\_delete=models.CASCADE)  
 text = models.TextField('Коментар')  
 author = models.CharField('Автор', max\_length=50)  
  
 def \_\_str\_\_(self):  
 return self.author  
  
 class Meta:  
 verbose\_name = 'Коментар'  
 verbose\_name\_plural = 'Коментари'

1. Регистрираме го в администраторския панел

from django.contrib import admin  
from .models import Article, Comments  
  
  
admin.site.register(Article)  
admin.site.register(Comments)

1. Подготвяме и изпълняваме миграциите

(venv) D:\Lessons\MyProject\blog>python manage.py makemigrations

Migrations for 'main':

main\migrations\0002\_comments.py

- Create model Comments

(venv) D:\Lessons\MyProject\blog>python manage.py migrate

Operations to perform:

Apply all migrations: admin, auth, contenttypes, main, sessions

Running migrations:

Applying main.0002\_comments... OK

1. Стартираме локалния сървър
2. Отваряме админ-панела и създаваме няколко коментара

## Извеждане на данни от свързаната таблица

1. В urls.py променям пътя към страницата, показваща единична статия

path('<int:article\_id>/', views.single\_article, name='single'),

1. В views.py променям изгледа за единичната страница (добавям втори параметър id на страницата, така както го зададохме в предишната точка.

def single\_article(request, article\_id):

1. В шаблона base.html премахвам точката от менюто, която сочи към единична страница
2. В шаблона index.html променям тага <a href="post.html">:

<a href="{% url 'single' a.id %}">

1. Обновяваме страницата в уеббрaузера и проверяваме дали правилно работят препратките
2. Редактирам шаблона article.html така, че да извежда предаваните от изгледа параметри:

{% extends 'main/base.html' %}  
{% block title %} {{ page\_title }} {% endblock %}  
{% block content %}  
 <h2 class="section-heading">{{ article.title }}</h2>  
 <em>{{ article.author}}, {{article.published}}</em>  
 </hr>  
 {%for c in comments %}  
 <h6>{{c.author}}</h6>  
 <p>{{c.text}}</p></br>  
 {% endfor %}  
{% endblock %}

Тестваме.

1. Добавяме дизайн към шаблона на страницата. За целта отварям съответната страница (post.html) от свалената базова тема и по неин образец добавям две секции – за самата статия и за коментарите. Добавям подходящите тагове (и стилове) за заглавие, съдържание, автор и т.н.

{% extends 'main/base.html' %}  
{% block title %} {{ page\_title }} {% endblock %}  
{% block content %}  
 <article class="mb-4">  
 <div class="container px-4 px-lg-5">  
 <div class="row gx-4 gx-lg-5 justify-content-center">  
 <div class="col-md-10 col-lg-8 col-xl-7">  
 <h2 class="section-heading">{{ article.title }}</h2>  
 <blockquote class="blockquote">  
 <strong>Автор: </strong>{{ article.author}}</br>  
 {{article.published}}  
 </blockquote>  
 <div style="text-align:justify;">  
 {{ article.content }}  
 </div>  
 </div>  
 </div>  
 </div>  
 </article>  
 </hr>  
 <article class="mb-4">  
 <div class="container px-4 px-lg-5">  
 <div class="row gx-4 gx-lg-5 justify-content-center">  
 <div class="col-md-10 col-lg-8 col-xl-7">  
 <h5>Коментари:</h5>  
 {%for c in comments %}  
 <em>{{c.author}}:</em><br>  
 <blockquote class="blockquote"> {{c.text}}

</blockquote></hr>  
 {% endfor %}  
 </div>  
 </div>  
 </div>  
 </article>  
{% endblock %}

1. Тестваме.
2. Добавяме форма, чрез която читателят да може да въведе коментар. За целта отваряме от базовата тема файла contact.html и копираме цялата секция <main>…</main>. Премахваме излишните коментари и елементите (таговете <div>) за имейл и телефон. Заместваме оригиналните текстове с подходящи за нашето приложение текстове на български език. Тестваме.

<article class="mb-4, border-top">  
 <div class="container px-4 px-lg-5">  
 <div class="row gx-4 gx-lg-5 justify-content-center">  
 <div class="col-md-10 col-lg-8 col-xl-7">  
 <p>Искате да изкажете и Вашето мнение? Попълнете формата за да публикувате Вашия коментар:</p>  
 <div class="my-5">  
 <form action="{% url 'form\_action' article.id %}" method="POST">  
 {% csrf\_token %}  
 <div class="form-floating">  
 <input class="form-control" name="name" type="text" placeholder="Въведете вашето име..." data-sb-validations="required" />  
 <label for="name">Име</label>  
 <div class="invalid-feedback" data-sb-feedback="name:required">Името е задължително.</div>  
 </div>  
 <div class="form-floating">  
 <textarea class="form-control" name="message" placeholder="Въведете вашия коментар тука ..." style="height: 12rem" data-sb-validations="required"></textarea>  
 <label for="message">Коментар</label>  
 <div class="invalid-feedback" data-sb-feedback="message:required">Коментарът е задължителен.</div>  
 </div>  
 <br />  
 <div class="d-none" id="submitSuccessMessage">  
 <div class="text-center mb-3">  
 <div class="fw-bolder">Коментарът Ви е публикуван успешно!</div>  
 </div>  
 </div>  
 <div class="d-none" id="submitErrorMessage"><div class="text-center text-danger mb-3">Грешка при публикуване!</div></div>  
 <!-- Submit Button-->  
 <button class="btn btn-primary" type="submit">Публикувай</button>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
</article>

1. Във файла views.py импортираме функциите HttpResponseRedirect и reverse

from django.http import HttpResponseRedirect  
from django.urls import reverse

и създаваме още един изглед, на който ще предаваме въведените през формата данни и ще ги обработваме (записваме):

def form\_action(request, article\_id):  
 article = Article.objects.get(id=article\_id)  
 article.comment\_set.create(author=request.POST['name'], text=request.POST['message'])  
 return HttpResponseRedirect( reverse('single', args=(article.id, )))

1. Във файла urls.py създаваме още едно проследяване на път

path('<int:article\_id>/form\_action', views.form\_action, name= 'form\_action'),

1. Връщаме се във файла article.html и допълваме тага <form> с указания за действието и метода на предаване на данните:

<form id="contactForm" data-sb-form-api-token="API\_TOKEN" action="{% url 'form\_action' article.id %}" method="POST">

1. тестваме