**Съдържание**

[1 Описание на приложението „Електронна библиотека” към сайта на ПГЕЕ: 3](#_Toc133577188)

[2 Създаване на „Електронна библиотека”: 3](#_Toc133577189)

[2.1 Създаваме проект: 3](#_Toc133577190)

[2.2 Създаване на приложение: 3](#_Toc133577191)

[2.3 Използване на шаблони: 4](#_Toc133577192)

[2.4 Оформяне: 4](#_Toc133577193)

[2.5 Миграции: 6](#_Toc133577194)

[2.6 Извеждане на данни от БД: 7](#_Toc133577195)

[3 Django: 11](#_Toc133577196)

[3.1 Изработка на сайт с Django: 11](#_Toc133577197)

[3.1.1 Някои от ключовите елементи за уеб приложенията на Django: 11](#_Toc133577198)

[3.1.2 Поддръжка на Django приложение: 12](#_Toc133577199)

[3.2 ORM: 12](#_Toc133577200)

[3.2.1 Основни понятия в ORM: 12](#_Toc133577201)

[3.2.2 Средства за изграждане на ORM модели: 13](#_Toc133577202)

[3.3 Хостинг за Django: 14](#_Toc133577203)

[3.4 Предимства на Django: 14](#_Toc133577204)

[3.5 Версии: 14](#_Toc133577205)

[3.6 Django-admin и manage.py: 15](#_Toc133577206)

[3.7 URL адреси: 15](#_Toc133577207)

[3.8 Преобразуватели на пътища: 16](#_Toc133577208)

[4 Debugger: 16](#_Toc133577209)

[4.1 Дебъгване: 17](#_Toc133577210)

[4.2 Обратно дебъгване: 17](#_Toc133577211)

[5 База данни: 18](#_Toc133577212)

[5.1 Преглед на база данни: 18](#_Toc133577213)

[5.2 Приложение на база данни: 19](#_Toc133577214)

[5.3 СУБД с общо и специално предназначение: 20](#_Toc133577215)

[6 Шаблонизатор Jinja. 20](#_Toc133577216)

[6.1 Функционални възможности. 20](#_Toc133577217)

[6.2 Функции на Jinja: 21](#_Toc133577218)

[6.3 Инсталиране на Jinja: 21](#_Toc133577219)

[7 HTML 22](#_Toc133577220)

[7.1 обща информация: 22](#_Toc133577221)

[7.2 Общо представяне: 23](#_Toc133577222)

[7.3 Версии на HTML: 25](#_Toc133577223)

[7.4 Алтернативни версии на HTML: 25](#_Toc133577224)

[7.5 Създаване на HTML страници: 26](#_Toc133577225)

[8 GitHub: 35](#_Toc133577226)

[8.1 Отдалечено хранилище на Git: 35](#_Toc133577227)

[8.2 Репозитори в GitHub: 36](#_Toc133577228)

[8.3 Git status: 37](#_Toc133577229)

[8.3.1 Git pull: 38](#_Toc133577230)

[8.3.2 Git clone: 39](#_Toc133577231)

[8.3.3 Git fetch: 39](#_Toc133577232)

[9 Административен панел: 39](#_Toc133577233)

[10 Списък с използвана литература: 41](#_Toc133577234)

# Описание на приложението „Електронна библиотека” към сайта на ПГЕЕ:

В 21 век има изключително много технологии за създаване на онлайн среди за обучение. Именно поради тази причина, при липсата на учебни материали на хартиен носител, нашата задача е да създадем т. нар. „Електронна библиотека”, прикачена към официалния сайт на нашето училище – Професионална гимназия по електроника и енергетика, град Банско, която да бъде достъпна за всички и по този начин всеки ученик да има възможност да се сдобие с необходими уроци и материали за съответния специален предмет.

# Създаване на „Електронна библиотека”:

* Първо, инсталираме и отваряме PyCharm;
* Създаваме нов проект, под името – MyProject;
* Всеки проект в Python трябва да бъде в папка;
* Инсталираме Django:

pip install django

## Създаваме проект:

django-admin startproject blog

## Създаване на приложение:

* 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:

* 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.
* В новосъздадената папка създавам шаблон (.html файл) с име index.html и попълвам title и body с някакво съдържание.
* Във vies.py коригирам функцията index:

def index(request):  
 return render(request, 'main/index.html')

* Обновявам основаната страница
* Създавам още две страници – all\_articles и article

## Оформяне:

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

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

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

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

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 %}

* Предаване на параметри от изгледа от изгледа към шаблона
  + 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)

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

Избираме тема:

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

{% load static %}

* Коригирам препратките към всички статични ресурси, например

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

Проверка и корекции

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'),

* 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 връзката към База данни:

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'"} }}

## Миграции:

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.

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

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”.

## 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:

articles = article.objects.filter(title\_\_contains='1')

articles = article.objects.order\_by('-id').filter(title\_\_contains='1')

articles = article.objects.order\_by('-id')[:2]

1. Променям изгледите, така че едната страница да показва новите статии (срез по id), а другата – всички статии

Създаване на свързаната таблица:

1. Дефиниране 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

Обща информация:

# Django:

Django e известен framework за създаване на динамични уеб приложения и сайтове. Базира се на MVC шаблона и е напълно безплатен за ползване и инсталиране. Използваме Django за проекти на наши клиенти за създаването на сложни уеб системи, които трябва да включват разнообразни функционалности. Разработен е в САЩ от Django Software Foundation и се поддържа активно от community. Системата се базира изцяло на езика за програмиране Python. Това гарантира лесното четене и разбиране на програмния код, както и възможността за надграждане и поддръжка на готовото уеб приложение.

## Изработка на сайт с Django:

Django се използва за разработка и създаване както на статични сайтове, които имат административен панел и възможност за редакция на съдържанието, така и за сложни и големи портали, които трябва да се справят със сериозен трафик и натоварване.

### Някои от ключовите елементи за уеб приложенията на Django:

* сайтовете се зареждат много бързо, понеже не се ползват излишни кодови фрагменти. За разлика от често използвани CMS платформи за изграждане на малки статични сайтове, Django предоставя възможност използване само на най-необходимото, за да се зареждат страниците светкавично, без излишен товар;
* администрацията позволява промяна на елементи от дизайна и съдържанието на уеб приложението. Това изцяло зависи от разработчиците, но платформата го поддържа напълно;
* има възможност за свързване с различни външни системи;
* на разположение са библиотеки за почти всичко, необходимо е единствено да се включат;
* приложенията може да се надграждат съвсем лесно с времето. Практически няма ограничения за допълнителните функционалности, които може да има Django сайт.

### Поддръжка на Django приложение:

Поддръжката на различни уеб базирани приложения, написани на Python и на базата на Django често се състои в:

* инсталиране на допълнителни апликации (apps);
* конфигуриране на основния settings.py файл;
* настройка на URL адресите на апликация, чрез urls.py;
* дефиниране на различни модели за обмен на информация с базата данни;
* генериране и настройка на форми през forms.py;
* дизайн и промяна на изгледи и шаблони.

## ORM:

“Object Role Modeling” е мощен метод за изграждане на модели на бази данни на концептуално ниво. Моделите са лесно разбираеми за потребители с нетехнически познания.

www.orm.net “Object Role Modeling” е базиран на факти метод, който се използва за извършване на системен анализ на концептуално ниво. За да се осигури конкретност, яснота, приспособимост и адаптивност, информационните системи се специфицират най-добре първо на концептуално ниво, използвайки понятия и език, достъпни за непрофесионалисти. ORM опростява процеса на дизайн, използвайки естествен език, както и интуитивни диаграми, които могат да бъдат подкрепени с примери и с езика на прости или елементарни факти.

### Основни понятия в ORM:

* Тип факт – шаблон за представяне структурата на изречения;
* Факт – твърдение за част от предметната област;
* Елементи на типа факт:
* Тип обект – съществителното име в изречението. Представя се уникално чрез идентификатор. Типовете обекти могат да бъдат:
* Тип същност – скаларен (с един уникален идентификатор) и съставен (с няколко уникални идентификатора);
* Тип етикет – символен низ / число. Няма ID;
* Вложен тип обект – обект, формиран от връзката между други два обекта;

### Средства за изграждане на ORM модели:

* InfoModeler
* VisioModeler MS Visio for Enterprise Architects (VEA)
* NORMA – свободен plug-in с отворен код за MS Visual Studio 2005 / 2008 / 2010

Създаване на нов проект:

* File -> New -> Database -> ORM Source Model
* бутон New -> Database -> ORM Source Model
* Choose Drawing Type Pane -> Database -> ORM Source Model

Работи се основно с меню Database:

* View -> Output
* View -> Verbalizer
* View -> Fact Editor
* View -> Database Properties
* View -> Business rules

Техниката за Object-relational mapping е много добре имплементирана в Django. Предоставя се възможност за боравене с база данни по един много лесен и интуитивен начин.

Ето един семпъл пример за извличане на данни за страница, чрез използването на ID:

try:

current\_page = Page.objects.get(pk=page\_id)

except Page.DoesNotExist:

raise Http404("Page does not exist")

Част от типовете бази данни, които може да се свържат с Django приложение са:

* SQLite;
* PostgreSQL;
* MySQL;
* Oracle.

В зависимост от конкретната ситуация – изборът на тип на базата данни е от изключително значение за производителността на платформата.

## Хостинг за Django:

Възможно е използването на споделен хостинг на уеб приложения на Python. В по-голямата част от случаите се ползва модулът flup за Python.

Инсталацията на flup става чрез следната команда:

pip install flup

Като за целта е добра практика да се използва virtualenv (Виртуална среда).

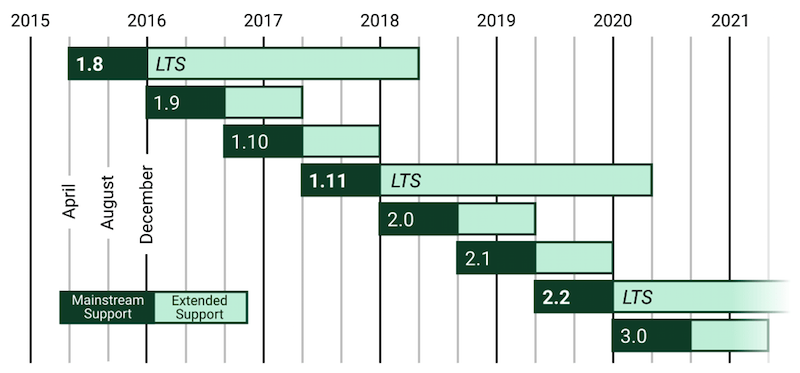
## Предимства на Django:

Има много предимства, които правят Django предпочитан framework за изработка на скалируеми уеб приложения на Python. Част от тях са:

* възможност за неограничено развитие на проекта;
* четим и лесен за разбиране Python код;
* сигурност и стабилност на системата;
* голяма общност и бързо намиране на решение на всеки проблем;
* бързина на зареждане при правилна оптимизация на модулите;

## Версии:

В зависимост от версията на Python, която ползваме, както и от хостинга е необходимо да изберете подходяща версия на Django. Някои от версиите вече не се поддържат официално от общността.



Нови версии излизат приблизително на всеки 8 месеца, като се предоставят security update-и няколко години след това. Този факт гарантира сериозната поддръжка и положителен тренд на развитие на системата. За някои уеб проекти е необходимо използването на последната версия на framework-a, но в общия случай е достатъчна и по-стара версия. Например версия 1.6.5 се използва активно, макар вече да не се поддържа официално. Въпреки липсата на официална поддръжка, системите работят без забележими дефекти и са напълно функционални.

## Django-admin и manage.py:

django-adminе помощната програма за команден ред на Django за административни задачи. Този документ очертава всичко, което може да направи.

Освен това manage.py се създава автоматично във всеки Django проект, Прави същото като django-admin, но също така задава DJANGO SETTINGS MODULE променлива на средата, така че да сочи към settings.py файла на нашия проект.

Скриптът django-admin трябва да е на нашия системен път, ако сме инсталирали Django чрез pip. Ако не е на пътя ни, трябва да се уверим, че нашата виртуална среда е активирана.

Като цяло, когато работим върху един проект на Django, той е по-лесен за използване manage.py от django-admin. Ако трябва да превключваме между множество файлове с настройки на Django, използваме django-admin with DJANGO SETTINGS MODULE или --settings опцията на командния ред.

Примерите за командния ред в този документ трябва django-admin да бъдат последователни, но всеки пример може да използва manage.py или също толкова добре .python –m django.

**$** django-admin <command> **[**options**]**

**$** manage.py <command> **[**options**]**

**$** python -m django <command> **[**options**]**

## URL адреси:

Когато потребител поиска страница от вашия базиран на Django сайт, това е алгоритъмът, който системата следва, за да определи кой код на Python да изпълни:

Django определя основния URLconf модул, който да се използва. Обикновено това е стойността на настройката ROOT\_URLCONF, но ако входящият HttpRequest обект има urlconf атрибут, неговата стойност ще се използва вместо настройката ROOT\_URLCONF.

Django зарежда този Python модул и търси променливата urlpatterns. Това трябва да е последователност от django.urls.path() и/или django.urls.re\_path() случаи.

Django преминава през всеки URL шаблон по ред и спира на първия, който съответства на искания URL, съпоставяйки се с path\_info.

След като един от URL шаблоните съвпадне, Django импортира и извиква дадения изглед, който е функция на Python (или изглед, базиран на клас). Изгледът получава следните аргументи:

* екземпляр на HttpRequest.

Ако съответстващият URL образец не съдържа наименувани групи, тогава съвпаденията от регулярния израз се предоставят като позиционни аргументи.

Аргументите на ключовата дума са съставени от всякакви наименувани части, съответстващи на израза на пътя, които са предоставени, заменени от всички аргументи, посочени в незадължителния kwargs аргумент към django.urls.path() или django.urls.re\_path(). Ако нито един шаблон на URL не съвпадне или ако се появи изключение, по време на който и да е момент от този процес, Django извиква подходящ изглед за обработка на грешки.

Пример URLconf:

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns **=** [

path("articles/2003/", views**.**special\_case\_2003),

path("articles/<int:year>/", views**.**year\_archive),

path("articles/<int:year>/<int:month>/", views**.**month\_archive),

path("articles/<int:year>/<int:month>/<slug:slug>/", views**.**article\_detail),

]

## Преобразуватели на пътища:

* str - съвпада с всеки непразен низ, с изключение на разделителя на пътя, '/ '. Това е по подразбиране, ако в израза не е включен конвертор.
* int- съвпада с нула или всяко положително цяло число. Връща int.
* slug- съвпада с всеки slug низ, състоящ се от ASCII букви или цифри, плюс тире и долна черта. Например building-your-1st-django-site.
* uuid- съвпада с форматиран UUID. За да предотвратите съпоставяне на няколко URL адреса към една и съща страница, трябва да бъдат включени тирета и малки букви. Например 075194d3-6885-417e-a8a8-6c931e272f00, връща UUID екземпляр.
* path- съвпада с всеки непразен низ, включително разделителя на пътя, '/ '. Това ви позволява да съвпадате с пълен URL път, а не сегмент от URL път, както при str.

# Debugger:

**Дебъгер** е компютърна програма за проследяване на процеса на изпълнение на компютърни програми (дебъгване). Обикновено целта на такова проследяване е намиране на грешки, които се наричат „бъгове“.

Този тип програми се използва от програмисти за тестване на новосъздадени продукти, за да се предотвратят възможни неизправности в процеса на работа на дадения продукт. Когато програмата се изпълнява в дебъг режим, тя е значително забавена, за да може да се открие моментът на възникването на дадения „бъг“.

Програмна грешка или „бъг“ възниква, когато дадена програма не може да продължи с изпълнението си по нормален начин. Например, програмата може да се опитва да използва ресурси, които са забранени или вече не съществуват. Също така е възможно в написания код на дадената програма да има грешка в някой алгоритъм или функция, което ще доведе до „crash“, което е еквивалентно и на „бъг“.

## Дебъгване:

Дебъгване е самият процес на откриване на грешки с една програма в друга. Чрез специални програми, които могат да изпълняват и проверяват в себе си продуктите на софтуерните инженери. Някои дебъгери могат да правят и по-сложни неща, например да изпълняват програмата стъпка по стъпка (Stepping) или да я спрат в даден момент чрез Break Point (брейк пойнт е точка, която се поставя на даден ред в програмния код с цел проследяване на стойностите на дадени променливи).

Други дебъгери имат способността да променят състоянието на програмата, докато е стартирана, което от своя страна се изразява в промяна на някоя от стойностите на участващи променливи в нея и дори на някои функции. По този начин е възможно по-нататъчното изпълнение на програмата и изследване на поведението ѝ с новозададените стойности на променените параметри. Осигурява се и възможност да бъде стартирана програмата на различни места от програмния код, с цел да се избегне срив или логическа грешка, породена от предхождащ код по време на изпълнението ѝ.

## Обратно дебъгване:

Някои дебъгери се характеризират с особеност наречена „обратно дебъгване“, което се отличава от нормалния режим на дебъгване по това, че се извършва в обратен ред на хронологията на изпълнение на дадена програма (отзад-напред). Няколко дебъгера включват тази функционалност за езици като C#, Visual Basic .NET, както и някои други езици с изключение на C++. Такива са Visual Studio Ultimate Edition, а за JAVA – [gdb](https://bg.wikipedia.org/w/index.php?title=Gdb&action=edit&redlink=1) ver. 7.0+, както и „Omniscient Debugger“. Дебъгването в обратен хронологичен ред е много удобно за откриването на някои специфични проблеми, но все още не се използва толкова често.

# База данни:

База данни представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. В първоначалния смисъл на понятието, използван в компютърната индустрия, базата от данни се състои от записи, подредени систематично, така че компютърна програма да може да извлича информация по зададени критерии. Например БД може да се използват в моделирането на хотелските системи, за да се проверява дали има налични свободни стаи в даден хотел. Поддръжката на база от данни се осъществява от т.нар. система за управление на бази от данни (СУБД).

Система за управление на бази данни е компютърно приложение (софтуер), създадено за комуникация между потребителя, други приложения, както и други БД, с цел да се сравнят и анализират данни. Общото специфично предназначение на СУБД е да позволи определянето, създаването, заявки, актуализацията и администрирането на бази данни. Добре известни СУБД включват MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, SAP HANA, и IBM DB2. Бази данни не са съвместими с различните СУБД, за това различните СУБД работят със стандартни като SQL и ODBC или JDBC, за да позволи на всяко приложение да работи с различни СУБД, а така и с различни БД. Управлението на БД често се избира от модела им, които те подкрепят. Най-използвани системи от бази данни от 1980 г. насам са всички поддържани релационни модели на езика SQL. Често срещано е СУБД да се нарича само „база данни“.

## Преглед на база данни:

„База данни“ дефинира множество свързани данни и начинът, по който са организирани. Достъпът до тези данни обикновено се осигурява чрез „система за управление на база данни“ (СУБД), състояща се от интегриран набор от компютърен софтуер, който позволява на потребителите да взаимодействат с една или повече бази данни и осигурява достъп до всички данни, съдържащи се в базата данни (въпреки че може да има ограничения спрямо достъпа до точно определени данни). СУБД предоставя различни функции, които позволяват влизане, съхранение и извличане на огромни количества информация и осигурява начини за управление как точно да бъде организирана тази информация.Поради тясната връзка между тях, терминът „база данни“ често се използва за наименование и на двете – „база данни“ и „система за управление на база данни“, използвана за управлението ѝ.

Извън света на професионалните информационни технологии, терминът база данни често се използва за обозначаване на всяка колекция от свързани данни (например електронна таблица или картотека). Тази статия се отнася само до бази данни, в които размерът на данните и изискванията за използването им се нуждаят от система за управление на база данни.

Съществуващите „Системи за управление на бази данни” осигуряват различни функции, които позволяват управлението на базата данни и самите данни, които могат да бъдат класифицирани в четири основни функционални групи:

* Дефиниране на данни – Създаване, модифициране и премахване на дефинициите, които определят организацията на данните.
* Промяна – Вмъкване, модификация и заличаване на актуалните данни.
* Извличане – Предоставяне на информация във формата на пряко използваем или за по-нататъшна обработка от други приложения. Извлечените данни могат да се предоставят направо в същата форма, в която са били съхранени в базата данни или в нова форма, получена чрез промяна или комбиниране на съществуващи данни от базата данни.
* Администриране – Регистриране и наблюдение на потребителите, налагане на сигурността на данните, наблюдение на изпълнението, запазвайки целостта на данните, които се занимават с едновременния контрол и възстановяване на информацията, която е била повредена от някакво събитие, като например неочакван срив на системата.

И двете – базата данни и нейната система за управление съответстват на принципите на определен модел на базата данни. „Система база данни“ се отнася общо за модел на база данни, система за управление на база данни, както и база данни.

## Приложение на база данни:

Бази данни се използват за поддръжката на вътрешни операции в организации и са в основата на онлайн взаимодействия с клиенти и доставчици.

Бази данни се използват за съхранение на административна информация и за по-специализирани данни, като инженерни данни или икономически модели. Примери за приложения, използващи бази данни са компютъризирани библиотечни системи, системи за самолетни резервации, автоматизирани системи за инвентаризации, както и много системи за управление на съдържанието, които съхраняват уеб сайтове като колекции от уеб страници в база от данни.

## СУБД с общо и специално предназначение:

СУБД са се развили сложна софтуерна система и нейното разработване обикновено отнема хиляди години човешко усилие. Някои СУБД с общо предназначение като Adabas, Oracle, DB2 постоянно се надграждат от седемдесетте години насам. СУБД с общо предназначение се стреми да покрие нуждите на колкото може повече апликации, което ги прави по-сложни. Обаче, фактът, че стойността на тяхното развитие може да се разпредели между много потребители, означава, че те често са най-ефективни по стойност. СУБД с общо предназначение не винаги е оптимално решение: в някои случаи СУБД с общо предназначение могат да доведат до ненужно препълване с информация. Затова има много примери за системи, които използват бази данни със специално предназначение. Чест пример за това е email система, която изпълнява много от функциите на СУБД с общо предназначение като прибавяне и изтриване на имейли, съставени от различни типове данни или асоцииране на имейли с точно определен имейл адрес, но тези функции за ограничени до това, което се изисква да се работи с имейли и не осигуряват на потребителя с цялата функционалност която би била налична когато се използва СУБД с общо предназначение.

Много други бази данни имат приложен софтуер, който прониква в базата данни от името на крайните потребители без да показва директно интерфейсът на СУБД. Приложните програмисти могат да използват директно с кабелен протокол или по-вероятно програмен интерфейс. Дизайнерите на бази данни и администраторите им си взаимодействат със СУБД чрез специализирани интерфейси, за да построят и поддържат базите данни на приложението, и затова имат нужда от знания и разбиране как СУБД оперират външните интерфейси на СУБД и променящи се параметри.

# Шаблонизатор Jinja.

## Функционални възможности.

Jinja е машина за уеб шаблони за езика за програмиране Python . Създаден е от Armin Ronacher и е лицензиран под BSD лиценз . Jinja е подобна на машината за шаблони Django , но предоставя изрази, подобни на Python, като същевременно гарантира, че шаблоните се оценяват в пясъчна кутия . Това е текстов език за шаблони и по този начин може да се използва за генериране на всякакви маркировки, както и за изходен код. Механизмът за шаблони Jinja позволява персонализиране на тагове, филтри, тестове и глобални стойности. Освен това, за разлика от шаблонния двигател на Django, Jinja позволява на дизайнера на шаблони да извиква функции с аргументи на обекти.

## Функции на Jinja:

* Наследяване и включване на шаблони.
* Дефиниране и импортиране макроси в шаблони.
* HTML шаблоните могат да използват автоматично екраниране, за да предотвратят XSS от ненадеждно въвеждане от потребител.
* В пясъчна среда може безопасно да рендира ненадеждни шаблони.
* Асинхронна поддръжка за генериране на шаблони, които автоматично обработват синхронизиране и асинхронни функции без допълнителен синтаксис.
* Поддръжка на I18N с Babel.
* Шаблоните се компилират към оптимизиран код на Python точно навреме и се кешират или могат да бъдат компилирани предварително.
* Изключенията сочат към правилния ред в шаблоните, за да улеснят отстраняването на грешки.
* Разширяеми филтри, тестове, функции и дори синтаксис.

## Инсталиране на Jinja:

Препоръчваме да използвате най-новата версия на Python. Jinja поддържа Python 3.7 и по-нови. Също така препоръчваме да използвате виртуална среда, за да изолирате зависимостите на вашия проект от други проекти и системата.

Инсталирайте най-новата версия на Jinja с помощта на pip:

$ pip install Jinja2

Примери:

* Шаблонен файл example.html.jinja:

<!DOCTYPE html>

< **html** >

< **глава** >

< **заглавие** > {{ променлива | escape }} </ **title** >

</ **head** >

< **body** >

{% - **for** item **in** item\_list %}

{{ item }}{% **if not** loop .last %} , {% **endif** %}

{% - **endfor** %}

< / **тяло** >

</ **html** >

Шаблонен код:

**от jinja2 импортиране на** шаблон

**с** отворен ( 'example.html.jinja' ) **като** f :

tmpl = шаблон ( f . четене ())

печат ( tmpl . рендериране (

променлива = 'Стойност с <unsafe> данни' ,

item\_list = [ 1 , 2 , 3 , 4 , 5 , 6 ]

))

HTML низ:

<!DOCTYPE html>

< **html** >

< **head** >

< **title** > Стойност с **<** опасно **>** данни </ **title** >

</ **head** >

< **body** >

1,

2,

3,

4,

5,

6

</ **body** >

</ **html** >

# HTML

## обща информация:

(съкращение от термина на английски: ***H****yper****T****ext****M****arkup****L****anguage*, в превод „*език за маркиране на хипертекст*“) е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в интернет, а неговите стандарти се определят от международния консорциум W3C. Текущата версия на стандарта е HTML 5.0 (от 28 октомври 2014 г.), а предходната стабилна версия е HTML 4.1. Описанието на документа става чрез специални елементи, наречени HTML елементи или техните маркери, които се състоят от тагове и съответстващите етикети (HTML tags) и ъглови скоби (като например елемента <html>). HTML елементите са основната градивна единица на кода, който изграждат уеб страниците. Чрез тях се форматира, графично оформя текста и неговите отделните части в рамките на една уеб страница, като например заглавия, цитати, текстови раздели, хипертекстови препратки и т.н. Най-често HTML елементите са групирани по двойки <h1> и </h1>. В повечето случаи HTML кодът е написан в текстови редактори, с файлов формат .html, .htm, dhtml и се качва и хоства на сървъри, които са онлайн в интернет или са част от [www](https://bg.wikipedia.org/w/index.php?title=Www&action=edit&redlink=1) мрежата. Тези .html файлове съдържат програмно на таговете на HTML и текстово съдържание със маркери и коментари – също инструкции за браузъра, за това какъв точно тип е .html страницата, а също за това как да се показва текстът, особено що се отнася до езиковите характеристики. За да се илюстрира как се включва текст в HTML код: <маркер> Някакъв текст. </край на маркера>. уеб браузърите са програмирани от своя страна така, в повечето случаи, макар че някои браузъри могат да имат съответно проблеми на версията, за да могат да прочетат HTML документите и да ги покажат на екрана като уеб страници. Браузърите не показват самите HTML тагове, освен ако не се отиде в менюто за да се направи това, така че те „интерпретират“ (тоест [парсват](https://bg.wikipedia.org/w/index.php?title=Parsing&action=edit&redlink=1)) съдържанието на страницата като код и текст за да могат след работа на процесора да покажат желаното уеб-съдържание. Основното предимство на HTML е, че уеб-страниците, които са го включват в кода си, могат да се разглеждат чрез показването им от браузъра на екрана на повечето устройства. Уебстраницата може да има дизайн, който дори изглежда с добър дизайн с помощта на CSS или „правилно оформен“ (например с помощта на C#), както върху монитора на персоналния компютър, но също и върху миниатюрния дисплей на пейджър или дисплея на мобилен телефон.

HTML може да прикрепя скриптове писани на езици като JavaScript, който е помощен за HTML, и това променя поведението на дадена уеб страница. Cascading Style Sheets (CSS) се използват, като това се прави за да се определя изгледа и оформлението на текста и други включени в страницата изображения и илюстриращи материали. World Wide Web Consortium (W3C) поддържа както HTML, така и CSS, и насърчава използването на CSS в HTML страниците още от 1997. Това допринася за разделяне съдържанието и структурата на уеб страниците от тяхното визуално представяне.

## Общо представяне:

Първото публично достъпно описание на HTML е документът „HTML тагове“, споменат за първи път в интернет от Тим Бърнърс-Лий в края на 1991 г. Той описва 18-те елемента, които съставляват оригиналния, сравнително прост HTML дизайн. С изключение на маркера за хипервръзка, те бяха силно повлияни от SGMLguid, вътрешен формат на документация, базиран на стандартния обобщен език за маркиране (SGML), в CERN. Единадесет от тези елементи все още съществуват в HTML 4.

Първоначално HTML езикът е замислен и създаден като средство за структуриране и форматиране на документи, без да ги обвързва със средствата за възпроизвеждане (показване). В идеалния случай, текст с маркиране на HTML трябва да е без стилистични и структурни изкривявания се играе на оборудване с различни технически оснащенностью (цветен екран модерен компютър, монохромен екран организатор, ограничен от размерите на екрана на мобилен телефон или устройство и програма за гласово възпроизвеждане на текстове). Съвременното приложение на HTML обаче е много далеч от първоначалната му задача. Например, тагът <table> е предназначен за създаване на таблици в документи, но понякога се използва и за проектиране на поставянето на елементи на страницата. С течение на времето основната идея на платформата за независимост на езика HTML беше жертвана на съвременните нужди от мултимедиен и графичен дизайн.

HTML е език за хипертекстово маркиране, който уеб браузърите парсват („интерпретират“), за да покажат текст и съдържание на уебстраницата като изображения и други уеб-приложения и материали. Оригиналните характеристики за HTML са част от парсинговите дефиниции и методи на браузъра, като за да се допълнят, техните характеристики могат да бъдат променяни (подобрявани) в самия код на страницата, на страницата на разработчика или локално на потребителския компютър, чрез поддръжка на локален код, тоест с допълнително включване на разширяващият възможностите на HTML – CSS (Cascading Style Sheets), които да подобрят начините за покзаване на текстовото съдържание. Голяма част от определящите текста елементи датират от 1988 ISO technical report TR 9537 *Techniques for using SGML*, които представляват характеристиките *за* тогавашните форматиращи езици, като такъв използван в TYPЕSET програмата за текстообработка от 1960-те, създадена за CTSS (Compatible Time-Sharing System) операционна система. Въпреки че, концепцията на SGML е също базирана на елементи (вложени анотирани обхвати с атрибути) и по-малко на принтиращи ефекти със структурно разделение, HTML e неговото подобрение, а CSS допълва HTML в способността на браузъра да дийсплейва съдържанието.

Internet Engineering Task Force (IETF) официално обявяват HTML като SGML базиран език през 1993 г. Като публикували първия план за HTML спецификация: „Hypertext Markup Language (HTML)“ Internet-Draft от Тим Бърнърс – Лий и Daniel Connolly който включвал и SGML Document Type Definition за да дефинира граматиката ([синтаксиса](https://bg.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8_%D0%B5%D0%B7%D0%B8%D1%86%D0%B8)&action=edit&redlink=1)) на езика. По същия начин Dave Raggett създава Internet-Draft, „HTML+ (Hypertext Markup Format)“, в края на 1993, предлагайки стандарти за таблици и форми за попълване.

През 1994 IETF създава работеща група по HTML която през 1995 завършва HTML 2.0 (първата спецификация за HTML, която трябва да се счита за постоянна и бъдещите спецификации да я надграждат).

От 1996 спецификациите се поддържат от World Wide Web Consortium (W3C). През 2000, HTML е вече световен Интернет стандарт (ISO/IEC15445:2000), като в браузърите се ползват различни стандарти като HTML 3 и HTML 4.01, която версия е обявена през 1999.

Дълго този стандарт е толкова добре работещ, че не се разглежда необходимост от подобряването му, но междувременно Microsoft оттеглят работата си по flash, като Adobe продължават да го поддържат за уеб, Майкрософт пуска проприетарна версия на flash, която обаче не става популярна, въпреки качеството на графиката, и така след известни проблеми с flash-а, през 2012 се повдига въпроса за обновяване на стандарта, например с подобрения с видео-включването в страници.

## Версии на HTML:

* 24 ноември 1995 г. – HTML 2.0 е представен като RFC 1866;
* 25 ноември 1995 г. – качване на файлове с формуляри;
* Май 1996 г. – таблици;
* Август 1996 г. – клиентски картови изображения;
* През януари 1997 г. е представен HTML 3.2. Това е първата версия, разработена и стандартизирана от World Wide Web Consortium;
* През декември 1997 г. е представен HTML 4.0, отново от W3C (World Wide Web Consortium) в три вариации:
  + Строг, отпадналите елементи са забранени;
  + Междинен, отпадналите елементи са разрешени;
  + Фреймов, най-често рамкови елементи са позволени;
* април 1998 г. – версия 4.0 претърпява леки промени, без смяна на номера;
* декември 1999 г. – версия 4.01 е налице;
* януари 2008 г. - версия 5.0, първо публично публикуване. От 2014-та става основен стандарт.

## Алтернативни версии на HTML:

Тъй като HTML се смята за донякъде отворен стандарт, макар и поддържан от W3C и Internet Engineering Task Force, той е направен така, че да работи като част от самия браузър, като част от инструкциите към браузъра и парсъра, а също така е такъв за разработчиците, които свободно могат да го използват за своите нужди, някои нови версии на HTML като HTML5 имат проприетарен подход, което спъва работата на разработчиците и блокира стандартизиращите проекти на W3C и Internet Engineering Task Force, както и други стандартизиращи организации.

* януари 2008 г. – Трейдмарковия HTML5 на Apple е представен като работен проект от W3C;
* май 2011 г. – версия 5 е в процес на развитие на техническите спецификации. Пълната спецификация се очаква до 2014 г.
* октомври 2014 – HTML5 е публикувана като препоръка на W3C.

Представеният нов HTML, който също така има различно изписване от предните, има тенденции за трейдмарк, както и лого HTML5, според групата, работила по него започва още през 2004 от Web Hypertext Application Technology Working Group (WHATWG), а по-късно през 2008 W3C се включва за да одобри стандарта, с идеята, че той ще е еквивалентно добър на този от предните версии на HTML, и двете групи завършват стандарта на 28 октомври 2014.

Все пак става дума за Apple стандарт, който максимално подобрява работата на Macintosh настолни компютри и тяхната синхронизация с мобилни устройства.

## Създаване на HTML страници:

Създаването на HTML базирана уеб страница може да се извърши с помощта на обикновен текстов редактор. Този начин изисква добро познаване на HTML тагове, така че те да бъдат в логическа последователност, и в тях да бъдат интегрирани текстовите сегменти, или текстът, който ще се показва на страницата. Също така, често срещани са по-приятелски настроените инструменти, които не изискват от потребителя да пише ръчно HTML, макар и да притежава познания по HTML, което му позволява да създаде страница по метода WYSIWYG. Основен инструмент за тази цел Notepad, но за тази цел може да се използва дори текстообработваща програма Word, която разбира се позволява количеството html код да бъде умножено по експоненциала, но гарантира показването на един текст като уебстраница този път без каквито и да е познания в програмирането, парсинга и други подробности, тъй като код, създаден на Word позволява на браузъра да спре с часове, почти независимо от процесора.

Така че Word е програма, която позволява да запазите документ като HTML и да го редактирате като текст.

Въпреки това, в действителност компании като Microsoft и други специализирани в създаването на програмни инструменти и дори среди за програмиране, пускат за правещите уебстраници по-специализирани инструменти, които не правят процесора да спре, и тези програмни инструменти са също така способни при писане на код, от страна на създаващия уеб-страницата, или дори при движение на графични елементи от дизайнер, да се създавт HTML страници.

Такива са FrontPage, професионалният ASP.NET, а също и Notepad++, Macromedia Dreamweaver, Sublime Text и други.

Примерен HTML код:

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** charset="windows-1251">

<**title**>Заглавие на документа</**title**>

</**head**>

<**body**>

<**p**>Здравей, свят!!</**p**>

<**p**>Validate <**a** href="http://validator.w3.org/check/referer">XHTML Basic 1.0</**a**></**p**>

<**p**>

<**a** href="http://validator.w3.org/check?uri=referer"><**img**

src="http://validator.w3.org/images/vxhtml-basic10"

alt="Valid XHTML Basic 1.0!" height="31" width="88" /></**a**>

</**p**>

</**body**>

</**html**>

Структура и основни елементи на един HTML документ:

HTML таговете са най-малката съставна част на един HTML документ. Те, заедно със своите атрибути (като цвят, размер и т.н), съставят т.нар. HTML елементи.

Таговете са ключови думи, заградени в ъглови скоби. Обикновено са по двойки:

* таг за начало – маркира началото на един HTML елемент;
* таг за край – маркира края на HTML елемента. За разлика от началния таг, крайният има наклонена надясно черта пред името си:

<**b**> ... </**b**>

Таговете се влагат един в друг, като първия отворен таг се затваря последен.

Основни HTML елементи:

#### <html> елемент:

Указва на браузъра, че това е HTML документ. Отбелязва началото и края на документа и съдържа всички други негови елементи (с изключение на <!DOCTYPE> елемента).

<**html**>

...

</**html**>

#### <head> елемент:

Съдържа заглавието на документа, и може да съдържа стилове, скриптове, енкодинг и т.н.

<**html**>

<**head**>

<**title**>Заглавие на документа</**title**>

</**head**>

...

</**html**>

#### <body> елемент:

Съдържа форматиране видимо за потребителя – текст, хиперлинк, картинки, таблици, бутони, параграфи и т.н.

<**html**>

<**head**>

<**title**>Заглавие на документа</**title**>

</**head**>

<**body**>

Съдържание на документа...

</**body**>

</**html**>

#### <!DOCTYPE> елемент:

Декларира се първи, още преди <html> тага. Валидира документа. <!DOCTYPE> не е HTML таг. Той е инструкция за уеб браузъра – указва HTML версията, на която е написана страницата:

<!DOCTYPE HTML PUBLIC „-//W3C//DTD HTML 4.01//EN“ „http://www.w3.org/TR/html4/strict.dtd“>

<!DOCTYPE html>

<**html**>

<**head**>

<**title**>Заглавие на документа</**title**>

</**head**>

<**body**>

<**h1**>Моето първо заглавие.</**h1**>

Съдържание на документа......

</**body**>

</**html**>

Основни HTML тагове:

#### <b> или <strong> таг:

Удебелява текста.

Това е нормален текст**&nbsp;**– <**b**>а това е удебелен</**b**>.

Това е нормален текст – **а това е удебелен**.

#### <i> или <em> таг:

Задава курсив/наклон на текста.

Той кръсти своята кола <**i**>Светкавицата</**i**>, защото беше много бърза.

Той кръсти своята кола *Светкавицата*, защото беше много бърза.

#### <u> таг:

Подчертава текста.

Това е <**u**>подчертан текст</**u**>.

Това е подчертан текст.

#### <strong> таг:

Указва важен текст.

<**strong**>Важен текст</**strong**>

**Важен текст**

#### <sub> таг:

Дефинира текст под черта.

Текстът съдържа <**sub**>subscript</**sub**> текст.

Текстът съдържа subscript текст.

#### <sup> таг:

Дефинира текст над черта.

Текстът съдържа <**sup**>superscript</**sup**> текст.

Текстът съдържа superscript текст.

#### <del> таг:

Указва изтрит текст.

Моят любим цвят е <**del**>червен</**del**> син!

Моят любим цвят е ~~червен~~ син!

#### <br> таг:

Указва нов ред. Няма таг за край.

Този текст съдържа <**br**> нов ред.

Този текст съдържа  
нов ред.

#### <blockquote> таг:

Указва част от текста, който е цитат.

<**blockquote** cite="http://www.worldwildlife.org/who/index.html">

Когато често се срещаме с дадени хора, те стават част от живота ни.

И като станат част от живота ни, започват да се опитват да го променят.

И се сърдят, когато не правим това, което те изискват от нас.

Понеже всеки си мисли, че знае как другият трябва да живее живота си, но всъщност никой не знае как трябва да живее своя собствен.

#### <mark> таг:

За отбелязване на текст, като му се променя фона.

Този таг <**mark**>имитира истинските маркери</**mark**>.

Този таг имитира истинските маркери.

#### <a> таг:

Указва линк към друга страница. Най-важният атрибут на този таг е *href*. Той посочва URL адреса, към който сочи линка:

<**a** href="https://bg.wikipedia.org">Уикипедия на Български</**a**>

Уикипедия на Български

#### <img> таг:

Дефинира картинка в HTML страницата. Има два задължителни атрибута: *src* и *alt*. Атрибутът *src* указва URL адреса на картинката, *alt* – указва алтернативен текст на картинката, а *hight* и *width* – указват съответно височината и ширината на картинката в пиксели.

<**img** src="smiley.gif" alt="Smiley face" height="42" width="42">

Заглавия и параграфи:

#### Заглавия:

Таговете от <h1> до <h6> дефинират заглавия в HTML документа. <h1> дефинира най-важното заглавие. <h6> дефинира най-маловажното заглавие:

<**h1**>Заглавие 1</**h1**>

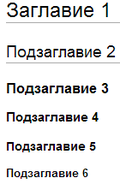
<**h2**>Подзаглавие 2</**h2**>

<**h3**>Подзаглавие 3</**h3**>

<**h4**>Подзаглавие 4</**h4**>

<**h5**>Подзаглавие 5</**h5**>

<**h6**>Подзаглавие 6</**h6**>

[](https://bg.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:HTMLHeadersExample.png)

#### Параграфи:

Тагът <p> указва параграф. Браузърите автоматично добавят по един празен ред преди и след текста, маркиран като параграф:

<**p**>Това е параграф.</**p**>

<**p**>Това е друг параграф.</**p**>

Това е параграф.

Това е друг параграф.

Повечето браузъри показват параграфа коректно, дори да не е сложен тага за край.

<**p**>Това е параграф.

<**p**>Това е друг параграф.

Това е параграф.

Това е друг параграф.

За нов ред в самия параграф, без да се започва нов, се използва тагът <br>.

<**p**>Това е<**br**>пара<**br**>граф с нови редове.</**p**>

Това е  
пара  
граф с нови редове.

Списъци:

#### Подредени списъци:

Подредените списъци започват с тага <ol>, а всеки елемент на списъка – с тага <li>:

<**ol**>

<**li**>Биричка</**li**>

<**li**>Още една биричка</**li**>

</**ol**>

1. Биричка
2. Още една биричка

Атрибутът **reversed** указва, че подреждането ще е в низходящ ред, а присвоената му стойност – показва началото на номерацията:

<**ol** reversed>

<**li**>Едно</**li**>

<**li**>Две</**li**>

<**li**>Три</**li**>

</**ol**>

3. Едно

2. Две

1. Три

Атрибутът **start** задава началото на номерацията:

<**ol** start="4">

<**li**>Едно</**li**>

<**li**>Две</**li**>

<**li**>Три</**li**>

</**ol**>

1. Едно
2. Две
3. Три

<**ol** reversed start="5">

<**li**>Едно</**li**>

<**li**>Две</**li**>

<**li**>Три</**li**>

</**ol**>

5. Едно

4. Две

3. Три

Атрибутът **type** задава вида на маркерите (букви или цифри), например 1, a, A, i, I:

<**ol** type="I">

<**li**>Едно</**li**>

<**li**>Две</**li**>

<**li**>Три</**li**>

</**ol**>

1. Едно
2. Две
3. Три

#### Неподредени списъци:

При тези списъци номерацията не е с цифри или букви, а с кръгли точки. Всеки списък започва с тага <ul>, а всеки елемент на списъка – с тага <li>:

#### Описателни списъци:

Това са списъци от термини/имена с описание на всеки термин/име. Такъв списък се указва с тага <dl> в съчетание с таговете:<br>

**Таблици:**

Дефинират се с тага <table>. Таблицата е разделена на редове, чрез тага <tr> („table row“), а всеки ред е разделен на клетки с данни (чрез тага <td>, „table data“). Всяка клетка може да съдържа текст, линкове, картинки, списъци, форми, други таблици и т.н. Атрибутът border задава рамка на таблицата.

<**table** border="1">

<**tr**>

<**td**>ред 1, колона 1</**td**>

<**td**>ред 1, колона 2</**td**>

<**td**>ред 1, колона 3</**td**>

</**tr**>

<**tr**>

<**td**>ред 2, колона 1</**td**>

<**td**>ред 2, колона 2</**td**>

<**td**>ред 2, колона 3</**td**>

</**tr**>

</table>

|  |  |
| --- | --- |
|  | |
| ред 1, колона 1 | | ред 1, колона 2 | ред 1, колона 3 |
| ред 2, колона 1 | | ред 2, колона 2 | ред 2, колона 3 |

Заглавната информация в таблицата се указва чрез тага <th>. Текстът обикновено се показва удебелен и центриран:

<**table** border="1">

<**tr**>

<**th**>Колона 1</**th**>

<**th**>Колона 2</**th**>

<**th**>Колона 3</**th**>

</**tr**>

<**tr**>

<**td**>ред 1, колона 1</**td**>

<**td**>ред 1, колона 2</**td**>

<**td**>ред 1, колона 3</**td**>

</**tr**>

<**tr**>

<**td**>ред 2, колона 1</**td**>

<**td**>ред 2, колона 2</**td**>

<**td**>ред 2, колона 3</**td**>

</**tr**>

</table>

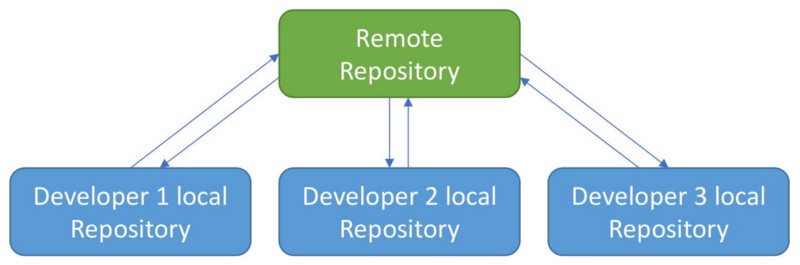
|  |  |  |
| --- | --- | --- |
| Колона 1 | Колона 2 | Колона 3 |
| ред 1, колона 1 | ред 1, колона 2 | ред 1, колона 3 |
| ред 2, колона 1 | ред 2, колона 2 | ред 2, колона 3 |

# GitHub:

GitHub e уеб базирана услуга за разполагане на софтуерни проекти и техни съвместни разработки върху отдалечен интернет сървър в т.нар. хранилище (software repository). Базира се на [Git](https://bg.wikipedia.org/wiki/Git_(%D1%81%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80)) системите за контрол и управление на версиите. Услугата може да бъде както платена за частни проекти, така и безплатна за т.нар. проекти с общодостъпен код, като и в двата случая потребителите могат да ползват всички възможности на услугата. Към май 2011 г. GitHub се счита за най-популярния сайт за разполагане на съвместни проекти с общодостъпен или наречен още отворен код**.**

Сайтът предоставя функционалност за изграждане на социални мрежи за споделяне на софтуерни програми и за работа с тях, като например индивидуално участие на всеки програмист в групови разработки, както и възможност за наблюдение върху работата на всеки участник в екипа.

## Отдалечено хранилище на Git:



Досега работихме само в локалното хранилище. Всеки разработчик ще работи в своето локално хранилище, но в крайна сметка идва момент, в който ще трябва да “избута” кода в някое отдалечено хранилище. След като кодът попадне в отдалеченото хранилище, вече и други разработчици ще могат да го виждат и модифицират.

## Репозитори в GitHub:

След регистрацията, трябва да кликнем върху „**Стартиране на проект”**, за да създадем ново Git хранилище в GitHub. Даваме име на хранилището и отиваме върху „Създаване на репозитори“.  
Например може да дадем име като git-blog-demo.

Това ще създаде отдалечено хранилище в GitHub и когато отворим хранилището, ще се отвори страницата с хранилището:

URL адресът на хранилището е маркираната част: [**https://github.com/aditya-sridhar/git-blog-demo.git**](https://github.com/aditya-sridhar/git-blog-demo.git)

За да насочим локалното си хранилище към отдалеченото хранилище, използваме следната команда:

git remote add origin [repository url]

Git Push:

git push origin <branch-name>

Изтласкваме (изпращаме, Push) вашия branch, за да го качим някъде другаде:

Това ще качи нашия branch в отдалеченото репо, наименувано origin (трябва да знаем, че това е URL, който може да е дефиниран първоначално по време на клонирането – за това повече по-долу, или при създаването на ново, отдалечено репо).

След успешно натискане, нашите съотборници вече ще могат да изтеглят нашия branch, за да преглеждат нашите commit-и.

Като пряк път можем да въведете думата HEAD вместо име на branch, за да използваме автоматично branch-а, в който се намираме в момента. HEAD винаги се отнася до най-новата ви контролна точка, тоест най-новия commit за текущия branch.

Както бе споменато по-рано, всичко в git може да се разглежда като контролна точка. Ето списък на видовете контролни точки, за които знаем вече (отново технически се наричат ​​„препратки“ и „ревизии“):

* HEAD
* <branch-name>, e.g. master
* <commit-hash>, e.g. e093542d01d11c917c316bfaffd6c4e5633aba58 (or e093542 for short)

Има още и:

* <tag-name>, напр. v1.0.0
* stash

И накрая, специални символи като ^, ~ и @ {} могат да се използват за промяна на препратки. Те са доста полезни.

За да изтласкаме целия код от локалното хранилище в отдалеченото хранилище, използваме следната команда:

git push -u origin master

Това изтласква кода от главния клон в локалното хранилище към главния клон в отдалеченото хранилище.

Git clone:

Клонираме вече съществуващи репозитори:

git clone [https://github.com/cooperka/emoji-commit-messages.git](https://github.com/cooperka/emoji-commit-messages)

Това ще изтегли .git хранилище от интернет (в нашият случай от GitHub) на нашия компютър и ще извлече най-новата/последна “снимка” на репото (всички файлове) в нашата работна директория. По подразбиране всичко ще бъде запазено в папка със същото име като на репото (в този случай emoji-commit-messages).

URL адресът, който ще посочим тук, се нарича remote origin / отдалечен източник (мястото, от което първоначално са изтеглени файловете). Този термин ще бъде използван и по-късно.

## Git status:

Трябва да видим текущото състояние на нашия проект:

git status

Това ще отпечата основната информация, като например кои файлове са модифицирани наскоро.

Трябва да проверяваме състоянието си по всяко време. Git ще отпечата допълнителна информация в зависимост от това какво се случва в момента, за да ни помогне да се ориентираме.

Git merge (с отдалечено репо):

Обединяване на промените на някой друг:

Това ще вземе всички commits, които съществуват в branch-а (other-branch-name), и ще ги интегрира във нашия собствен текущ branch.

Това използва всички данни за branch-ове, които се съхраняват локално, така че първо трябва да изпълним git fetch, за да изтеглим най-новата информация.

Например, ако някой друг добави няколко commits към главния branch – master на origin, можем да направим следното, за да изтеглим промените им и да актуализираме собствения си локален главен branch:

git checkout master # Make sure you’re on the right branch.

git fetch # Download any new info from origin.

git merge origin/master # Merge the ‘origin/master’ branch

into your current branch.

Името origin/master тук буквално означава контролна точка за origin/master на нашия компютър. Git използва тази нотация, за да разграничи едноименните клонове (напр. Master), разположени на различни места (например нашите собствени branches спрямо origin’s branches).

Допълнителни команди:

### Git pull:

Като пряк път можем да използваме командата pull за едновременно извличане и обединяване на всичко. Това е по-често използвано от обединяването ръчно, както е по-горе:

git pull origin master

Тук разделяме думите origin и master. Не искаме да използваме origin/master контролната точка, намираща се на собствения ни компютър. Вместо това искаме да извлечем директно от главния клон на отдалечената крайна точка, наречен origin.

Git pull:

Се използва за изтегляне на последните промени от отдалеченото хранилище в локалното хранилище. Кодът на отдалеченото хранилище се актуализира непрекъснато от различни разработчици, поради което git pull е необходимо:

git pull origin master

### Git clone:

Се използва за клониране на съществуващо отдалечено хранилище в нашия компютър. Командата за това е:

git clone [repository url]

### Git fetch:

Вземане на най-актуалната информация за дадено репо:

git fetch

Това ще изтегли последната информация за репото от произход (като всички различни клонове, съхранявани в GitHub).

Това не променя нито един от нашите локални файлове – просто актуализира данните за проследяване, съхранени в папката .git.

# Административен панел:

Всеки динамичен уеб сайт се състои от 2 части - публична и административна.  
Административната част на сайта е така наречената „content management system” - CMS (Система за управление на съдържанието или административен панел за управление на съдържанието) - **уеб базиран модул**. **Административния панел** на уеб сайт служи на потребителите извършват управление на съдържанието на интернет страниците (администраторите на сайта). Чрез него се организира за публикуване голям обем от всякакво съдържание - текстове, картинки и мултимедия. От админ панела може да се променя цялата информация на сайта - от корекции на съдържанието, добавяне или изтриване на елементи, променяне на ключови думи използвани за **SEO Оптимизация**.  
  
 Правилната работа с **админ панела** е много важна. На никого не допада, ако вместо снимка се появи празно квадратче или текстът, който чете е с някакъв неясен шрифт или неефектен цвят. Всяка една промяна през административния панел в реално време променя съответната информация в публичната част на **уеб сайта**, така че работата с него е много отговорна.

# Списък с използвана литература:

1. Django: <https://bg.wikipedia.org/wiki/Django>
2. ORM технология: <https://vinss.files.wordpress.com/2010/10/uprajnenie-1-orm.pdf>
3. Django-admin и manage.py: <https://docs.djangoproject.com/en/4.2/ref/django-admin/>
4. URL адреси: <https://docs.djangoproject.com/en/4.2/topics/http/urls/>
5. Debugger: https://bg.wikipedia.org/wiki
6. Бази данни: https://bg.wikipedia.org/wiki
7. Jinja: <https://en.wikipedia.org/wiki/Jinja_(template_engine)>
8. HTML: <https://bg.wikipedia.org/wiki/HTML>
9. GitHub: <https://bg.wikipedia.org/wiki/GitHub>
10. Admin panel: https://www.webdesigngroup.biz/