Съдържание

[1. Увод 3](#_Toc164720501)

[2. Използвани технологии 5](#_Toc164720502)

[2.1. Python 5](#_Toc164720503)

[2.1.1. Основни характеристики на езика 5](#_Toc164720504)

[2.1.2. Предимства за начинаещи програмисти 5](#_Toc164720505)

[2.1.3. Четимост и изчистеност на езика 5](#_Toc164720506)

[2.1.4. Сфери на развитие с Python 6](#_Toc164720507)

[2.1.5. Професии за Python разработчици 7](#_Toc164720508)

[2.1.6. Ускорен старт в програмирането с Python 7](#_Toc164720509)

[2.2. HTML (HyperText Markup Language) 8](#_Toc164720510)

[2.3. CSS (Cascading Style Sheets) 9](#_Toc164720511)

[2.4. Бази данни 10](#_Toc164720512)

[2.4.1. SQL - Език за структурирани запитвания 10](#_Toc164720513)

[2.4.2. Какво са бази данни и как се управляват те? 11](#_Toc164720514)

[2.4.3. MySQL 11](#_Toc164720515)

[2.4.4. SQL и MySQL едно и също нещо ли са? 12](#_Toc164720516)

[2.5. JavaScript 12](#_Toc164720517)

[2.6. Django 14](#_Toc164720518)

[2.6.1. Заявки 14](#_Toc164720519)

[2.6.2. Какви са свойствата на Django? 15](#_Toc164720520)

[2.6.3. Какви са предимствата на Django? 15](#_Toc164720521)

[2.6.4. Django форми 16](#_Toc164720522)

[2.6.5. Връзка до страницата чрез формата 18](#_Toc164720523)

[2.6.6. URL 20](#_Toc164720524)

[2.6.7. Шаблон(Template) 21](#_Toc164720525)

[2.7. Запазване на формата 23](#_Toc164720526)

[2.8. Проверка на формите (Form validation) 27](#_Toc164720527)

[2.9. Редактиране на формата 27](#_Toc164720528)

[3. Заключение 31](#_Toc164720529)

[4. Списък с използвана литература: 32](#_Toc164720530)

# Увод

Всеки един хотел, независимо колко е популярен и успешен, почти винаги има свободни стаи. Хотела не желае да има свободни стаи, тъй като не реализират печалби за обекта, а трупат загуби. Хотела губи от свободни стаи, които има, тъй като така те не трупат печалби, и затова хотела се нуждае от т.н.сайт, който помага тези стаи да бъдат запълнени.

Чрез интегрирането на формулярите на Django, уебсайтът гарантира безпроблемно въвеждане на данни и валидация, подобрявайки преживяването на потребителите. Този проект изследва предимствата на използването на Django и неговите библиотеки за създаване на сайт за хотел, като подчертава скалируемостта, сигурността и разширяемостта му.

Хотелската индустрия все повече се полага на онлайн платформи за резервации на настанявания. Затова има нарастваща нужда от ефективни и потребителски приятни уебсайтове за резервации в хотели

Хотела определя колко незаети стаи има във всеки един момент от работното си време, а сайта ни, предлага на Вас място от което да резервирате. Хотелите, които имат собствен сайт, генерират директни онлайн резервации през сайта на хотела си. Това води до много производни позитиви, най-съществен от които е съкращаването на разходите за придобиване на нови резервации.

На второ място, използването на сайт, намалява времето, необходимо за управлението на множество онлайн платформи, премахва необходимостта от ръчно въвеждане на резервации в хотелската система и оптимизира цялостната онлайн дистрибуция.

На трето място, достъпът до клиентски данни дава по-добра представа за потребителите, как и кога резервират, както дали хотелският продукт може да се подобри, според нуждите на клиентите. Директните онлайн резервации имат най-висока среднодневната цена, след премахването на комисионните към трети страни и други рекламни разходи по придобиването на резервации.

Един от най-големите плюсове на хотелския сайт е възможността, която предоставя на бизнеса ви за приемане на резервации **24/7**. Няма нужда от физическо лице, което да бъде ангажирано. Представете си, че бъдещите гости на хотела решават да си запазят стая при вас късно вечерта в събота, но не могат да направят запитване по телефона. Чрез собствения канал за резервации, който им предлагате на сайта си, без значение от часова зона и място, във всеки един момент може да получите резервация – бързо, удобно и лесно за двете страни.

Потребителите днес търсят **информация за дестинации** и хотели основно на своите **мобилни устройства**. Когато намерят подходящ хотел, повечето потребители очакват и да имат възможност да резервират през самото мобилно устройство. Това е основна причина повечето бизнеси да предлагат удобен интерфейс, с който онлайн резервациите да стават лесно и бързо.

Сайта е напълно оптимизиран за всички мобилни устройства и е съвместим с всички мобилни браузъри, а това е от голямо значение, тъй като почти всеки ваш клиент разполага със смартфон. Възползвайте се от тази тенденция, за да може да резервира стая в хотела ви само с няколко клика чрез телефона си.

Чудили ли сте се дали онлайн пазарът е най-добрият пазар? Подобен директен канал помага изключително много за увеличаване на печалбата поради значително по-ниските комисионни и преференциалните условия за ползване. Все още огромна част от потребителите се доверяват на възможността за директна резервация през сайта на хотела, като смятат, че това е най-добрата цена, на която могат да резервират.

# Използвани технологии

## Python

**Python е сред най-използваните** и популярни програмни езици в света. Днес се радва на общност от над 17 милиона разработчици, които го имат в технологичния си стек.

### Основни характеристики на езика

Python е **интерпретативен**, **динамично типизиран**, **мултипарадигмен** език от високо ниво. Подходящ е както за **обектно-ориентирано**, така и за **функционално програмиране**, наред с други, а това, че е от високо ниво означава, че е изключително лесно четим.

Езикът разполага с т.нар. **garbage collector**. Езикът се обновява често, като последната стабилна версия е от октомври 2023. Дизайнът му е фокусиран върху четимостта и функционалността на езика, без излишен код и трудности при писането му. Любопитен факт е, че тази философия е залегнала дори в шеговит манифест с 19 принципа, наречен The Zen of Python.

### Предимства за начинаещи програмисти

Вече споменах накратко две от основните предимства на езика, от които като начинаещ програмист ще имаш най-голяма полза. Нека видим и защо:

### Четимост и изчистеност на езика

Че четимостта има значение е принцип, залегнал и в манифеста по-горе. Цялата философия и причина зад създаването на езика е това **да бъде лесен за четене и минималистичен**. Там, където много езици използват препинателни знаци, Python залага на ключови думи на английски.

За да изглежда кодът прилежно и подредено, се използват много отстъпи и нови редове. По този начин бялото пространство прави четенето на кода много по-лесно. Това е ключово за начинаещи програмисти, защото едно от нещата, които ще ти помогнат да навлезеш в света на програмирането, да започнеш да го разбираш, е и четенето на код, наред с писането му и практиката.

### Сфери на развитие с Python

Python бързо започна да набира популярност в последните години покрай развитието на полета като data science и machine learning. За последните 2 години общността е нараснала с 8 милиона разработчици, а над 60% от дивелъпърите в сферата на **машинното обучение и data science** използват именно този език в дейността си.

A graph of data in a computer

Description automatically generated with medium confidence

Източник: [JetBrains](https://www.jetbrains.com/lp/devecosystem-2022/python/)

Езикът е много популярен и за разработката на **решения за IoT** (Internet of Things, устройства за умни къщи и т.н.). Сред предпочитаните е за **уеб разработка** и в **системната администрация**, а намира значително приложение и в **автоматизираното софтуерно тестване**. За сметка на това, ако се вълнуваш предимно от разработката на мобилни приложения или игри, по-добре е да избереш алтернатива на Python – езикът не играе особена роля в тези области.

### Професии за Python разработчици

Предвид индустриите, в които Python намирана най-широко приложение, можеш да си създадеш относителна представа и какви професии можеш да преследваш в тях, овладявайки езика. След най-популярните са:

* Уеб дивелъпър – с този език ще работиш основно от сървърната страна, или т.нар. back-end. Ще работиш в тясна колаборация с дизайнери и front-end дивелъпъри, за да можеш да интегрираш работата им в проекта.
* Data Analysts и Data Scientist – и двете са тясно свързани със сферата на data science, с фокус върху обработката и анализа на събираните данни. Разбира се, като Data Scientist ще се нуждаеш от много повече и по-комплексни умения, включително и добро разбиране за бизнеса и процесите му, не само за IT аспекта на дейността ти.
* Machine Learning инженер – ако искаш да навлезеш в необятния и динамичен свят на изкуствения интелект, то машинното обучение е сферата, към която да се насочиш. Като ML инженер част от дейността ти ще изисква да боравиш с множество алгоритми и да осигуряваш статистически анализи.

По-необичайна реализация за Python специалисти е навлизане в света на финансите и търговията с финансови деривати. Т.нар. процес по web scraping осигурява огромни количества информация, а самият език разполага с редица инструменти, подпомагащи финансовия анализ и симулацията на различни подходи при търговията с финансови инструменти.

Езикът има и чисто практични ползи за професионалисти в други сфери. Дори и с базови познания ще можеш да си създадеш web scraper или crawler, които да подпомогнат дейността ти в момента, или да автоматизираш процеси в работата си. Затова и много хора от света на маркетинга, например, развиват базови умения с езика.

### Ускорен старт в програмирането с Python

Днес Python е много повече от език за програмисти. От него се ползват бизнес анализатори, финансови анализатори, специалисти по продажби и маркетинг, дори адвокати. Истината е, че **езикът предоставя огромни възможности** за оптимизация на работата, както и за впускане в предизвикателни и перспективни, високотехнологични сфери.

## HTML (HyperText Markup Language)

Зад абревиатурата HTML стои не по-разбираемото HyperText Markup Language. Какво обаче означава това?

* Hyper е противоположното на linear. Linear, що се касае до компютърните науки, означава, че всичко трябва да следва една точно определена последователност – дадено нещо преди друго и никога не се пренареждат в друг шаблон. HTML не се придържа към този шаблон и именно това позволява на потребителите да разглеждат уеб страници навсякъде и по всяко време – нещо, което днес считаме за даденост.
* Text – това е инструмента с който ще работиш. Обикновен, до болка познат английски език. Всички елементи на HTML всъщност са цели думи или части от такива, които няма да ти е трудно да се сетиш кои са. Например <ul> означава „unordered list“ или <ol>, което означава „ordered list”. Дори базови познания по английски език са достатъчни за да се ориентирате с лекота в синтаксиса на HTML.
* Markup е това, което ще правиш. Чрез текст ще изписваш отделни елементи. Тези елементи се наричат „тагове“ и имат различни свойства, които трябва да бъдат посочени, за да постигнеш търсения от вас ефект, при изгледа на вашата страница. Markup най-условно казано е процеса по работа с таговете, елементите и атрибутите в HTML.
* Language – защото разбира се, HTML е език. Със свои синтаксис и особености, със свое предназначение, но все пак вид език, и по-точно – markup език.

HTML представлява език за форматиране на хипертекст, даващ възможност за визуализиране на текст, график в браузърите или други приложения. Това е описателен език за форматиране, чрез който се създават документи, независещи от хардуерната или софтуерната платформа. Основните възможности на HTML са :

* Оформление на документи
* Включване на връзки към други Интернет страници
* Включване на графики, звук
* Създаване на таблици и други
* Създаване на списъци Езикът за форматиране на WEB страници HTML е универсалния език във WEB.

Той се използва за създаване на статични страници (в последните версии на езика, той е допълнен с т. нар. DHTML - Dynamic HyperText Language, който добавя известна интерктивност на страниците) и представлява обикновенни ASCII файлове т. е. това са обикновенни текстови файлове, които може свободно да се променят с помощта на Notepad. Тези файлове се състоят от текст, който бива визуализиран и команди (тагове, tags), които служат за указване на това как да бъде визуализиран текста.

## CSS (Cascading Style Sheets)

CSS е език за описание на стиловете - използва се основно за описване на представянето на документ, написан на език за маркиране. Най – често се използва заедно със HTML, но може също така да се приложи и върху XHTML, XML и производните му.

Официално спецификацията на CSS се поддържа от W3C. Създаден първоначално като средство за разделяне на съдържанието от представянето му, днес той се използва основно за визуално оформление на HTML страници. Kаскадните стилове позволяват в много по-добра степен контрол върху елементите на страницата, като добавят и нови възможности за форматиране на страницата.

Положителния ефект от използването на какскадни стилове е този, че ако се наложи да се промени дадено форматиране е достатъчно да се промени само на едно място, а именно файла съдържащ каскадния стил.

## Бази данни

### SQL - Език за структурирани запитвания

SQL(Structured Query Language) е непроцедурен език за създаване на заявки към база от данни. Непроцедурен означава „какво, а не как”. Например SQL описва какви данни трябва да бъдат извлечени, изтрити или вмъкнати, а не как трябва да бъде извършена тази операция.

SQL е разработен, за да обслужва релационните бази от данни. Всъщност може да се програмира на SQL само за релационни бази от данни, които поддържат SQL, като MSAccess, Oracle, Sybase и Informix. Въпреки, че реализациите на всеки потребител се различават един от друг, това не води до съществени изменения на SQL кода.

Една от най – мощните способности на SQL е възможността да събира и обработва данни от няколко таблици едновременно. Едно от големите преимущества на SQL е, че е напълно платформено и продуктово независим език. Той е от така наречените езици от високо ниво или четвърто поколение програмни езици, при които може да се свърши много работа с малко код.

SQL притежава много мощна система от команди, с които могат да се извършат всички необходими операции с бази от данни – от създаването на база от данни до манипулирането на данните и създаване на вложени процедури. SQL е стандартният език, използван за манипулиране и извличане на данни от релационните бази данни. SQL позволява на програмиста или администратора на бази данни да извърши следните неща:

* Модифициране на структурата на базите данни
* Промяна на системните настройки за сигурността
* Добавяна на разрешения за достъп на даден потребител до бази данни или таблици
* Разпитване на базата данни за информация
* Обновявяне съдържанието на базата данни Синтаксиса на SQL е много гъвкав, въпреки че има правила, които трябва да се спазват, както е при всеки програмен език.

Командата Select се използва много по-често от всяка друга SQL команда. Тя е незаменима при всяка комуникация с компютъра, отнасяща се за извличане на данни.

### Какво са бази данни и как се управляват те?

Бази данни наричаме структурирано количество от данни. Бази данни наричаме всичко от най-обикновен списък за пазаруване, през изображения в галерия до огромните количества информация в корпоративните мрежи. За да добавите, получите достъп и да работите с данни, които са съхранени в базите данни на компютъра, вие се нуждаете от система за управление на базите данни, като MySQL Server. Тъй като компютрите вече могат да боравят с огромни количества от данни, системите за управление на базите данни играят ключова и централна роля, било то като отделни приложения или като част от такива.

### MySQL

MySQL е най-популярната система за управление на SQL бази данни с отворен код. Тя се разпространява и поддържа от Oracle Corporation. Официалният сайт е http://www.mysql.com и на него можете да се сдобиете с най-новата информация за MySQL софтуера.

MySQL е open source релационна, управляваща бази от данни система (RDBMS), която използва SQL, най – популярния език за въвеждане, достъп и извършване на други процеси с данни в база данни. Това е open source, всеки може да изтегли MySQL и да добави към нея, в зависимост от общите публични правила.

MySQL е известен основно заради скоростта си, сигурността и гъвкавостта си. Системата за релационни бази данни MySQL, осигурява приложен програмен интерфейс (API) за C, C++, Java, Perl, PHP, Python и Tcl, допуска функциите в SELECT и WHERE частите на заявките.

Понастоящем MySQL работи върху Linux, UNIX и Windows платформи. Голям е интереса към MySQL като алтернатива за подходящи системи бази данни от Oracle, IBM и Informix. Невъзможно е да си представим модерен и високофункционален Уеб сайт, в основата на който не стои база данни.

Множество от функциите, които правят страниците привлекателни не биха могли да съществуват, услуги като онлайн банкиране, пазаруване и дори просто организиране на информацията, са немислими без някакъв вид система за бази данни. Няма да навлизаме в технологични и терминологични подробности, а ще покажем как да използвате напрактика една от най-популярните и разпостранени в Интернет системи за управление на бази данни - MySQL. Основната версия на системата е безплатна за използване, без значение с комерсиални или идеални цели, като това е една от основнит причини за широката й популярност. Това не означава, че възможностите, които предлага са недостатъчни. Системата може да изпълнява заявките на неограничен брой потребители, издържа до 50 милиона записа, предлага лесна за използване система за управление на правата на различните потребители, отлична скорост на обработка на заявките.

### SQL и MySQL едно и също нещо ли са?

Не. Тъй като по-горе обяснихме какво представлява в най-общи линии MySQL, нека погледнем SQL. Това е абревиатура, която означава „Structured Query Language” и както може би се сещате от името е програмен език. Всъщност това е най-често използвания програмен език, който се използва при достъп до бази данни. В зависимост от вашата програмна среда можете да влезете в SQL директно (например за да генерирате репорти), да вграждате SQL откъси в код изписан на друг език или да използвате специфични програмно-приложни интерфейси просто за да скриете синтаксиса на SQL.

## JavaScript

Javascript е най-широко разпространеният език за програмиране в интернет, след HTML. Въпреки че го нарекох "език за програмиране", с негова помощ не се пишат програми, а скриптове които се вмъкват в HTML документа.

В този смисъл JavaScript е език за писане на скриптове, докато JAVA е език за програмиране. Освен съвпадението в част от името, двата езика нямат кой знае какви прилики, дори са разработени от различни корпорации ( JAVA е дело на SUN, a JavaScript е разработка на Netscape).

JAVA е мощен език за програмиране не само на интернет приложения, но и на самостоятелни програми за различни платформи. Интернет приложенията на JAVA се наричат аплети. Те са файлове с разширение .class и се вмъкват в HTML документа между таговете . Тук няма да се спираме подробно на JAVA аплетите. Нека разгледаме възможностите на JavaScript, какво можете и какво не можете да правите с него:

* Ефекти с изображения. Rollover ефекти, слайд шоу, и много други.
* Уравление на прозорци и рамки. Отваряне и затваряне на прозорци, задаване на размера на прозорец, управление на един прозорец от друг и т. н.
* Разпознаване на типа на браузъра, операционната система, разделителната способност на екрана и дълбочината на цветовете.
* Много други работи Какво не можете да правите с помощта на JavaScript:
* Не можете да записвате информация на сървъра (не можете да организирате форуми, да обработвате бази данни) JavaScript кода се вмъква в HTML документа между двойката елементи.

Програмите създадени с JavaScript се наричат скриптове. Файлът, в който се съдържа кода на скрипта, е с разширение .js.

JavaScript скриптовете могат да се вграждат в HTML кода на уеб страницата, директно като код или да се извикват като външен .js файл. При зареждането на уеб страницата, външният .js файл се сваля локално на компютъра на потребителя и самото изпълнение на скрипта се извършва от уеб браузъра.

Една от разликите с езика за програмиране [Java](https://help.superhosting.bg/java.html) е, че за изпълнението на JavaScript скрипт не е нужно преди това кода да бъде компилиран, а директно се изпълнява/интерпретира от уеб браузъра.

Обикновено при работата на даден уеб сайт взимат участие две страни (сървър и клиент) – сървърът, на който се намира сайта, и уеб браузърът на потребителя, през който се разглежда сайта. Уеб браузърът можем да наричаме клиент. Някои от елементите на сайта се изпълняват от страна на клиента, като например HTML, CSS, JavaScript, картинки и всички други, които се свалят локално на компютъра на потребителя. От страна на сървъра се изпълняват елементите, които обработват заявките създадени от клиента към сървъра и връщат желаната информация, като например PHP, XML и други.

Когато срещне тагът . Този скрипт не е задължително да бъде написан на JavaScript. Има и други езици за писане на скриптове, например VBScript.

## Django

Django е безплатна и отворена уеб рамка, написана на Python. Уеб рамката е набор от компоненти, които ви помагат да развивате уеб сайтове по-бързо и по-лесно.

Когато създавате уебсайт, винаги се нуждаете от подобен набор от компоненти: начин за справяне с удостоверяването на потребителя (регистрация, влизане, излизане), панел за управление на вашия уебсайт, формуляри, начин за качване на файлове и т. н.

За ваше щастие, други хора отдавна забелязаха, че уеб разработчиците са изправени пред подобни проблеми при изграждането на нов сайт, така че те се обединиха и създадоха рамки (Django е една от тях), които ви дават готови компоненти за използване.

Рамките съществуват за да ви спестяват нуждата да преоткривате колелото и да помогнат за облекчаване на част от разходи, когато създавате нов сайт.

### Заявки

Какво се случва, когато някои изпрати заявка за сайт до вашият сървър, посредством Django?

Нека погледнем от по-близо сървърите, за да разберем за какво точно е Django. Първото нещо, което сървърът има нужда да знае е, че вие искате той да ви покаже дадена уеб страница.

Когато потребителската заявка достигне уеб сървър, тя се предава към Django рамката, която се опитва да разбере какво всъщност е поискано от страна на потребителя. За целта тя взема адреса на уеб страницата и се опитва да разбере какво да направи от тук нататък. Тази част се извършва от т.нар. Django URL Resolver. Не е нещо сложно – за целта се използва набор от шаблони и се опитва да открие съвпадение с изисквания URL адрес.

### Какви са свойствата на Django?

* С Django се изграждат леки и самостоятелни уеб сървъри за разработка и тестване;
* Django използва сериализация на формулярите и валидираща система, която превежда между HTML формулярите и стойностите, които се използват в базите данни.
* Има template система, която използва концепцията за наследяване от обектно-ориентираното програмиране (ООП);
* Основната Django дистрибуция върви в комплект с приложения в своя „contrib“ пакет, които включват неща като инструменти за генериране на RSS и Atom syndication feeds, допълнителна технологична рамка, която позволява на една Django инсталация да поддържа едновременно до няколко уебсайта, всеки от които със свое съдържание и приложения, инструменти за генериране на Google Sitemaps и още много други.

### Какви са предимствата на Django?

Django е проектиран по начин, който помага на разработчиците да създават дадено приложение възможно най-бързо. От идеята, през процеса на разработка, до завършения продукт, Django позволява да се реализира един проект, едновременно по един бюджетно-ориентиран начин, без това да се отрази на функционалността на продукта.

• **Пълен набор инструменти** - Django е подходящ за всеки етап от разработката на дадено приложение – потребителска верификация, администриране на съдържанието, RSS feed настройки и още много други. По този начин той е напълно достатъчен за процеса на разработка от начало до край.

• **Сигурност** - Django гарантира с една идея повече сигурност за вашия продукт и за неговите клиенти. Това се гарантира още на ниво “разработка”, като намалява възможностите за пропуски от страна на разработчиците до минимум. Някои от най-честите грешки се случват по отношение на прилагането на SQL injection, крос-сайт скриптирането и подобно.

• **Гъвкавост** - Без значение дали става въпрос за управление на съдържанието, специфични компютърни платформи или приложения, които ще се използват от големи организации – Django може да отговори на всяка от тези нужди.

Ето още предимства:

* **Бърз** - Django е проектиран по начин, който помага на разработчиците да създават дадено приложение възможно най-бързо. От идеята, през процеса на разработка, до завършения продукт, Django позволява да се реализира един проект, едновременно по един бюджетно-ориентиран начин, без това да се отрази на функционалността на продукта
* **Пълен набор инструменти** - Django е подходящ за всеки етап от разработката на дадено приложение – потребителска верификация, администриране на съдържанието, RSS feed настройки и още много други. По този начин той е напълно достатъчен за процеса на разработка от начало до край.
* **Сигурност** - Django гарантира с една идея повече сигурност за вашия продукт и за неговите клиенти. Това се гарантира още на ниво “разработка”, като намалява възможностите за пропуски от страна на разработчиците до минимум. Някои от най-честите грешки се случват по отношение на прилагането на SQL injection, крос-сайт скриптирането и подобно.
* **Гъвкавост** - Без значение дали става въпрос за управление на съдържанието, специфични компютърни платформи или приложения, които ще се използват от големи организации – Django може да отговори на всяка от тези нужди. Това демонстрира ясно неговата гъвкавост.

### Django форми

Последното нещо, което искаме да направим в нашия уеб сайт е да създадем начин за добавяне и редактиране на публикации в блога. Django администратора е супер, но е по-трудно да се персонализира и направи приятно на външен вид. С формите ще имаме абсолютно надмощие над нашия изглед - можем да правим почти всичко което може да си представим!

Хубавото на Django формите е, че може да дефинираме форма от самото начало или да създадем ModelForm, в която ще съхраняваме резултатът от формата на модела.

Точно това искаме да направим: ще създадем форма за нашия Post модел.

Като всяка важна част от Django, формите също имат собствен файл: forms.py.

Трябва да създадем файл с това наименование в директорията на blog.

blog

└── forms.py

И така, нека да отворим IDE и напишем следният код:

* blog/forms.py
* from django import forms
* from .models import Post
* class PostForm(forms.ModelForm):
* class Meta:
* model = Post
* fields = ('title', 'text',)

Първо трябва да въведем Django формите (from django import forms) и нашия Post модел (from .models import Post).

PostForm, както забелязахте е името на нашата форма. Трябва да кажем на Django, че тази форма е ModelForm (така че Django да направи малко магия за нас)- forms.ModelForm отговаря за това.

След това с class Meta, казваме на Django кой модел трябва да се използва за да се създаде тази форма (model = Post).

Най-накрая трябва да кажем с какви полета ще разполагаме в нашата форма. При този сценарий искаме само title и text да бъдат показани – author ще бъде този, който в момента е влязъл с профила си (ти!) и created\_date трябва автоматично да се създаде, когато започнем нова публикация (като в кода), нали?

И това е! Всичко от което имаме нужда в момента е да използваме формата в изгледа и да я покажем в шаблона.

И така още веднъж, ще създадем връзка до страницата чрез URL, view и template.

### Връзка до страницата чрез формата

Време е да отворим blog/templates/blog/base.html в редактора. В div наречен page-header ще добавим връзка:

blog/templates/blog/base.html

<a href="{% url 'post\_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>

Забележете, че искаме да извикаме нашия нов изглед post\_new. Класът "glyphicon glyphicon-plus" е предоставен от темата на bootstrap, която използваме и ще покаже знака плюс.

След добаване на този ред, вашия HTML файл трябва да изглежда ето така:

blog/templates/blog/base.html

{% load static %}

<html>

<head>

<title>Django Girls blog</title>

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">

<link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet' type='text/css'>

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

</head>

<body>

<div class="page-header">

<a href="{% url 'post\_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>

<h1><a href="/">Django Girls Blog</a></h1>

</div>

<div class="content container">

<div class="row">

<div class="col-md-8">

{% block content %}

{% endblock %}

</div>

</div>

</div>

</body>

</html>

След запазване и обновяване на страницата [http://127.0.0.1:8000](http://127.0.0.1:8000/) трябва да виждате познатата грешка NoReverseMatch. Така ли е? Добре!

### URL

Отваряме файла blog/urls.py в редактора и добавяме ред:

blog/urls.py

path('post/new/', views.post\_new, name='post\_new'),

И последния код трябва да изглежда така:

blog/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.post\_list, name='post\_list'),

path('post/<int:pk>/', views.post\_detail, name='post\_detail'),

path('post/new/', views.post\_new, name='post\_new'),

]

След като обновим сайта виждаме AttributeError, тъй като нямаме изглед за изпълнението на post\_new. Нека го добавим сега.

post\_new изглед

Време е да отворим файлът blog/views.py в редактора и да добавим следните няколко реда от код при останалите във формата

blog/views.py

from .forms import PostForm

И след това нашия изглед:

blog/views.py

def post\_new(request):

form = PostForm()

return render(request, 'blog/post\_edit.html', {'form': form})

За да създадем нова форма Post, трябва да извикаме PostForm() и да я препратим към шаблона (template). Ще се върнем отново на този изглед (view), но сега нека бързо да създадем шаблон (template) за формата.

### Шаблон(Template)

Трябва да създадем файл post\_edit.html в директория blog/templates/blog и да го отворим в редактора ни за код. За да направим така, че формата да работи са ни нужни няколко неща:

Трябва да покажем формата. Можем да направим това чрез (например)

{{ form.as\_p }}.

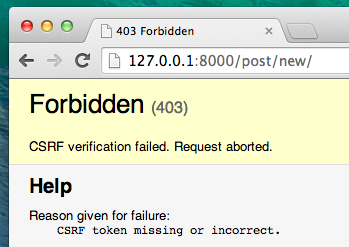
Горният ред трябва да бъде обвит с HTML form tag:

 <form method="POST">...</form>.

Трябва ни бутон Save. Това правим с HTML button:

 <button type="submit">Save</button>.

И най-накрая, точно след отварящия <form ...> tag трябва да добавим {% csrf\_token %}. Това е много важно, тъй като прави нашата форма надеждна! Ако забравите за това, Django ще се оплаче когато се опитате да запазите формата.



Добре, нека видим сега как трябва да изглежда HTML кода в post\_edit.html:

blog/templates/blog/post\_edit.html

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

{% block content %}

<h2>New post</h2>

<form method="POST" class="post-form">{% csrf\_token %}

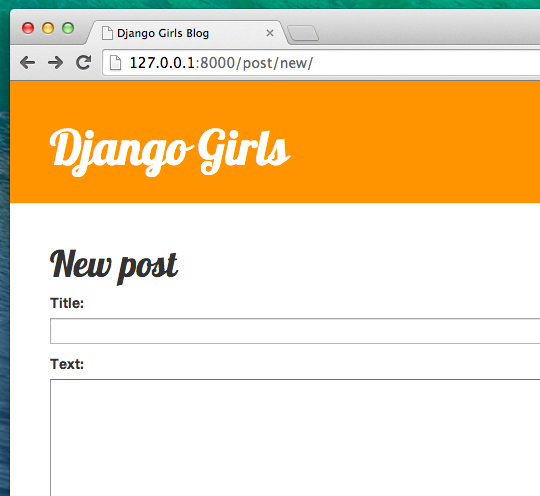
{{ form.as\_p }}

<button type="submit" class="save btn btn-default">Save</button>

</form>

{% endblock %}

Време е да обновим страницата! Формата се показа!



Но, чакай малко! Когато напишеш нещо в полетата title и text и се опиташ да го запазиш, какво ще стане?

Нищо! Пак сме на същата страница и нашият текст изчезна... и няма нова публикация. Какво се обърка?

Отговорът е: нищо. Трябва още малко да поработим върху нашия изглед (view).

## Запазване на формата

Отворете пак blog/views.py в редактора. В момента всичко, което имаме в изгледа на post\_new е следното:

blog/views.py

def post\_new(request):

form = PostForm()

return render(request, 'blog/post\_edit.html', {'form': form})

Когато изпратим форматата се връщаме пак на същият изглед, но този път имаме повече данни в запитването (request) или по-точно в request.POST (наименованието няма нищо общо с блога "post", а с факта, че "публикуваме" информация). Помните ли как в нашия HTML файл при дефинирането на <form> имахме променлива method="POST"? Всички тези полета са сега в request.POST. Не трябва да преименувате POST (друга валидна стойност за method е GET, но нямаме време да обясним каква е разликата).

И така в нашия изглед (view) имаме две отделни ситуации да разрешим: първо, когато достъпваме страницата за първи път и искаме празна форма, и второ, когато се върнем назад към изгледа (view) с всичките данни които сме въвели във формата. Тоест трябва да добавим условие (ще използваме if за тази цел):

blog/views.py

if request.method == "POST":

[...]

else:

form = PostForm()

Време е да попълним празното място [...]. Ако method-а е POST тогава искаме да изградим PostForm с данни от формата, нали? Ще направим това както следва:

blog/views.py

form = PostForm(request.POST)

Следващото нещо е да проверим дали формата е вярна (всики полета които се изискват да са определени и да няма неправилно въведени стойности). Правим това чрез form.is\_valid().

Проверяваме дали формата е валидна и ако да в запазваме!

blog/views.py

if form.is\_valid():

post = form.save(commit=False)

post.author = request.user

post.published\_date = timezone.now()

post.save()

Основно имаме две неща: запазваме формата с form.save и добавяме автор (тъй като няма поле author във формата PostForm а това поле се изисква). commit=False означава, че не искаме все още да запазим моделът Post - искаме да добавим първо автора. През повечето време ще използвате form.save() без commit=False, но в този случай трябва да се снабдим с него. post.save() ще запази промените (добавяне на автора) и новата публикация в блога е създадена!

Накрая, би било добре ако можем веднага да отидем на страницата post\_detail за нашата новосъздадена блог публикация, нали? За да направим това трябва ни трябва още едно въвеждане:

blog/views.py

from django.shortcuts import redirect

Добавете го в самото начало на вашия файл. И сега можем да кажем "отиди на страница post\_detail за новосъздадената публикация":

blog/views.py

return redirect('post\_detail', pk=post.pk)

post\_detail е името на изгледа, на който искаме да отидем. Помните ли, че този изглед (view) изискваше стойност pk? За да го предадем на изгледа използваме pk=post.pk, където post е новосъздадена публикация!

И така, говорихме много, но може би искаме да видим как изглежда всичко, нали?

blog/views.py

def post\_new(request):

if request.method == "POST":

form = PostForm(request.POST)

if form.is\_valid():

post = form.save(commit=False)

post.author = request.user

post.published\_date = timezone.now()

post.save()

return redirect('post\_detail', pk=post.pk)

else:

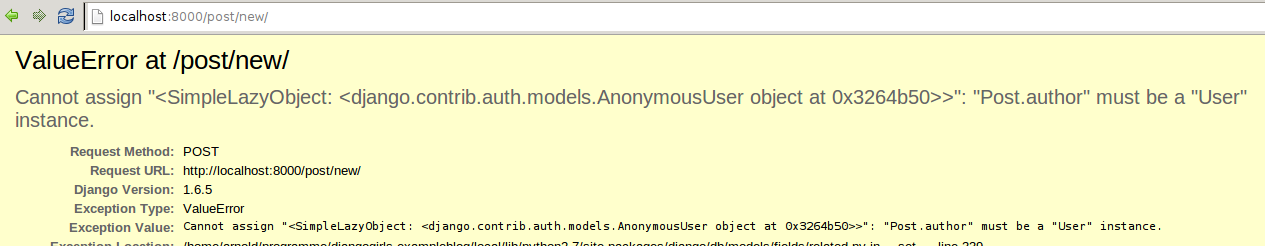
form = PostForm()

return render(request, 'blog/post\_edit.html', {'form': form})

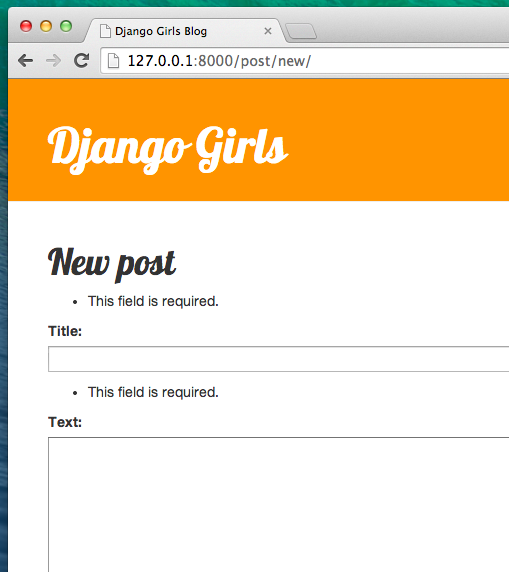
Нека видим дали работи. Отидете на страница <http://127.0.0.1:8000/post/new>, добвавете title и text, запазете..и voilà! Новата публикация е добавена и сме пренасочени към страница post\_detail!

Може би забелязахте, че избираме датата на публикуване преди да запазим публикацията. По-късно ще се запознаем с publish button in **Django Girls Tutorial: Extensions.**

Тъй като наскоро ползвахме административния интерфейс на Django, системата в момента си мисли, че сме влезнали в акаунта си. Има няколко ситуации, които могат да доведат до излизане от акаунта (затваряне на търсачката, рестартиране на базата данни, и др.). Ако при създаване на нова публикация получавате грешки свързани с потребител, който не е влязал в акаунта си, отидете на администраторската страница <http://127.0.0.1:8000/admin> и влезте пак в акаунта си. Това ще реши проблема временно. Има дълготрайно решение на проблема, което ви очаква в раздела **Домашна работа: добавяне на сигурност към уеб страницата ви!** намиращ се след главното ръководство.



## Проверка на формите (Form validation)

Сега ще ви покажем колко са готини формите на Django. Една публикация трябва да има полета за title и text. В нашия Post модел не казахме дали тези полета се изискват (както при published\_date), така че Django очаква те да са нагласени по подразбиране.

Опитайте се да запазите формата без title и text. Познайте какво ще стане!

Django се грижи за проверката дали всички полета във формата ни са правилни. Не е ли готино?

## Редактиране на формата

Сега знаем как да добавим нова публикация. Но какво ако искаме да променим вече съществуваща? Това много прилича на това, което тъкмо направихме. Нека набързо да създадем няколко важни неща. (Ако не разбирате нещо, попитайте инструктора си или вижте в предходните глави, тъй като вече минахме през всичко това.)

Отворете blog/templates/blog/post\_detail.html в редактора и добавете реда

blog/templates/blog/post\_detail.html

<a class="btn btn-default" href="{% url 'post\_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>

така че шаблона (template) да изглежда така:

blog/templates/blog/post\_detail.html

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

{% block content %}

<div class="post">

{% if post.published\_date %}

<div class="date">

{{ post.published\_date }}

</div>

{% endif %}

<a class="btn btn-default" href="{% url 'post\_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>

<h2>{{ post.title }}</h2>

<p>{{ post.text|linebreaksbr }}</p>

</div>

{% endblock %}

Отворете blog/urls.py в редактора и добавете този ред:

blog/urls.py

path('post/<int:pk>/edit/', views.post\_edit, name='post\_edit'),

Ще използваме пак шаблона blog/templates/blog/post\_edit.html, така че последното липсващо нещо е изгледа (view).

Нека отворим blog/views.py в редактора и да добваим това накрая на файла:

blog/views.py

def post\_edit(request, pk):

post = get\_object\_or\_404(Post, pk=pk)

if request.method == "POST":

form = PostForm(request.POST, instance=post)

if form.is\_valid():

post = form.save(commit=False)

post.author = request.user

post.published\_date = timezone.now()

post.save()

return redirect('post\_detail', pk=post.pk)

else:

form = PostForm(instance=post)

return render(request, 'blog/post\_edit.html', {'form': form})

Това изглежда почти еднакво с нашия изглед post\_new, нали? Но не изцяло. Първо зададохме допълнителен pk параметър от urls. После взехме Post модела, който искаме да променим с get\_object\_or\_404(Post, pk=pk) и накрая, когато създаваме форма, подаваме тази публикация като инстанция (instance), когето запазваме формата...

blog/views.py

form = PostForm(request.POST, instance=post)

... и когато отваряме формата с публикацията, която искаме да променим:

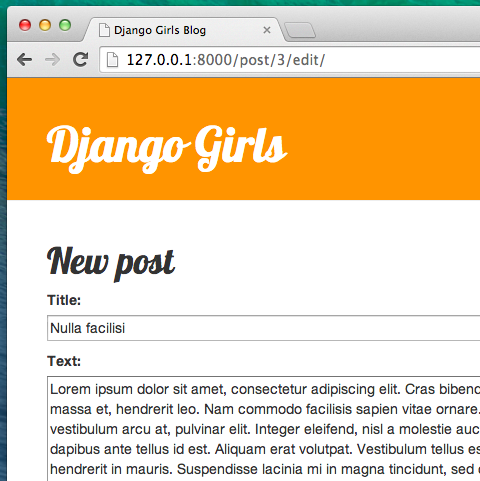
blog/views.py

form = PostForm(instance=post)

Добре! Нека проверим дали работи! Да отидем на страница post\_detail. Трябва да има бутон за редактиране в горния десен ъгъл:



Когато го натиснете, ще видите формата с нашата публикация:



Можете да промените заглавието или текста и да запазите промените!

Поздравления! Вашата апликация изглежда все по- и по-завършена!

# Заключение

Това е моят дипломен проект, надявам се занапред да имам още успешни задачи. Благодарности към господин Бориков, защото, ако не беше неговата помощ проектите ни нямаше да бъдат толкова добри! Трябва да се отбележи, че това е реално работещ сайт, който е тестван само върху тестови сървър. Предстои да се качи на реален хостинг, което ще позволи на реалните потребители да го достъпват и ползват. А както знаем, тестът в реална работна среда е най-сигурния тест.

Надявам се, разработеното от нас приложение, да спечели интереса на потребителите и да намери широко приложение.

# 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/