Съдържание:

[1. Увод 3](#_Toc164643523)

[2. Django: 3](#_Toc164643524)

[2.1. Изработка на сайт с Django: 3](#_Toc164643525)

[2.2. Ключови елементи на Django: 4](#_Toc164643526)

[2.3. Поддръжка на Django приложение: 4](#_Toc164643527)

[2.4. Хостинг за Django: 4](#_Toc164643528)

[2.5. Какви предимства има Django?: 5](#_Toc164643529)

[2.6. Django-admin и manage.py: 5](#_Toc164643530)

[2.7. URL адреси: 6](#_Toc164643531)

[2.8. Преобразуватели на пътища: 7](#_Toc164643532)

[3. База данни: 7](#_Toc164643533)

[3.1. Преглед на база данни: 8](#_Toc164643534)

[3.2. Приложение на база данни: 9](#_Toc164643535)

[3.3. MySQL 9](#_Toc164643536)

[4. HTML 11](#_Toc164643537)

[4.1. Обща информация: 11](#_Toc164643538)

[4.2. Общо представяне: 13](#_Toc164643539)

[5. GitHub: 15](#_Toc164643540)

[5.1. Отдалечено хранилище на Git: 15](#_Toc164643541)

[5.2. Репозитории в GitHub: 16](#_Toc164643542)

[6. Създаване на „web-приложение за резервации ”: 18](#_Toc164643543)

[6.1. Анализ на заданието 18](#_Toc164643544)

[6.2. Архитектуран на приложението 18](#_Toc164643545)

[6.3. Подготовка за реализация на приложението: 19](#_Toc164643546)

[6.4. Клиентски интерфейс 20](#_Toc164643547)

[6.5. Използване на шаблони: 21](#_Toc164643548)

[6.6. Оформяне: 21](#_Toc164643549)

[6.7. Миграции: 24](#_Toc164643550)

[6.8. Извеждане на данни от БД: 25](#_Toc164643551)

[6.9. Заявки за резервация 27](#_Toc164643552)

[6.10. Административен панел: 29](#_Toc164643553)

[7. Заключение 29](#_Toc164643554)

[8. Списък с използвана литература: 31](#_Toc164643555)

# Увод

Всеки един ресторант или клуб, независимо колко е популярен и успешен, почти във всеки момент от своето работно време има незаети маси. Заведението няма интерес тези маси да стоят празни, тъй като не реализират приход за обекта, а трупат загуби. Ресторантът губи от незаети маси, които има, тъй като той вече е платил наем за помещението, ток за светлината и музиката в залата, заплата на обслужващия персонал и т.н. този сайт помага тези маси да бъдат запълнени. Ресторантът определя колко незаети маси има във всеки един момент от работното си време, а сайта ги предлага на Вас. Така заведението не трупа загуби от незаетите си маси и въпреки отстъпката отново реализира печалба и запълва ресторанта си с доволни клиенти. ReserveEasy реализира мечтата на всеки един ресторантьор - да види заведението си пълно с клиенти и усмивки. А Вие просто се възползвате да хапвате вкусно в най-добрите ресторанти и да се забавлявате в най-добрите клубове. Откривате нови кухни и места и взимате голяма отстъпка от сметката.

# 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 за малки динамични 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.

## Ключови елементи 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.

## 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).

## Какви предимства има 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;

## 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().

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 като задължителен аргумент екземпляр на HttpRequest.

Ако съответстващият URL 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 за 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.

# 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.

## MySQL

MySQL е една от най-популярните релационни бази данни, използвани в софтуерната индустрия днес. Той предоставя надеждна, бърза и мащабируема система за съхранение и управление на данни, което го прави предпочитан избор за много уеб и приложения.

В основата си, MySQL използва структурата на релационните бази данни, което означава, че данните се съхраняват в таблици с редове и колони. Той предлага богат набор от функции и възможности, които позволяват на разработчиците да създават сложни и мощни приложения. Ето някои от ключовите характеристики на MySQL:

1. **SQL (Structured Query Language):** MySQL използва SQL за манипулиране на данните. Това включва изпълнение на заявки за създаване, четене, обновяване и изтриване на данни от базата данни. SQL е стандартизиран език, който позволява на разработчиците да комуникират с базата данни по един и същ начин, независимо от технологията на базата данни.
2. **Мащабируемост и производителност:** MySQL е много ефективен при обработката на големи обеми данни и много заявки едновременно. Той предлага различни методи за оптимизация на заявките и индексиране на данните, което осигурява висока производителност и отзивчивост на приложенията.
3. **Транзакции и ACID съответствие:** MySQL поддържа транзакции, което позволява на разработчиците да изпълняват група операции като едно атомарно действие. Това осигурява цялостност и съхранение на данните в случай на неуспешно изпълнение на транзакцията. MySQL е съвместим с ACID (Atomicity, Consistency, Isolation, Durability) принципите за контрол на транзакциите.
4. **Сигурност на данните:** MySQL предлага различни механизми за сигурност на данните, включително управление на достъпа, шифроване на данните и мониторинг на активността на потребителите. Това помага за защита на чувствителната информация и предотвратяване на неоторизиран достъп до данните.
5. **Отворен код и общност:** MySQL е разработен като отворен код проект и разпространява своят софтуер под GNU General Public License (GPL). Това означава, че всеки може да използва, модифицира и разпространява MySQL безплатно. Освен това, MySQL разполага с активна общност от програмисти и потребители, които споделят знания и опит за улесняване на работата с базата данни.
6. **Репликация и кластериране:** MySQL предлага възможности за репликация и кластериране, които позволяват на разработчиците да създават високодостъпни и отказоустойчиви системи. Репликацията позволява на данните да бъдат копирани и синхронизирани между различни сървъри, като се осигурява по-голяма надеждност и производителност. Кластерирането позволява на разработчиците да създават група от сървъри, които работят заедно като едно цяло, като се увеличава мащабируемостта и отказоустойчивостта на системата.
7. **Характеристики на управление на транзакции:** MySQL предоставя различни характеристики за управление на транзакции, включително поддръжка на изолация на транзакции, контрол на конкурентността и оптимизация на изпълнението на транзакциите. Тези характеристики позволяват на разработчиците да създават стабилни и ефективни уеб приложения, които обработват голям обем данни.
8. **Интеграция с други технологии:** MySQL се интегрира лесно с други технологии и платформи, като например програмни езици като PHP, Python, Java и други. Това прави MySQL подходящ избор за разработчиците, които работят с различни езици за програмиране и технологии.
9. **Обработка на географски данни:** MySQL предлага различни възможности за обработка на географски данни, включително поддръжка на географски индекси, функции за географски заявки и интеграция с географски данни. Това прави MySQL подходящ избор за създаване на географски приложения и системи за управление на географски данни.
10. **Поддръжка и развитие:** MySQL е активно поддържан и развиван от Oracle Corporation и голямо общество от разработчици по целия свят. Нови версии и актуализации се публикуват редовно, като се въвеждат нови функции, подобрения на производителността и корекции на грешки. Това гарантира, че MySQL е винаги актуален и сигурен избор за разработчиците.

# HTML

## Обща информация:

HTML(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.

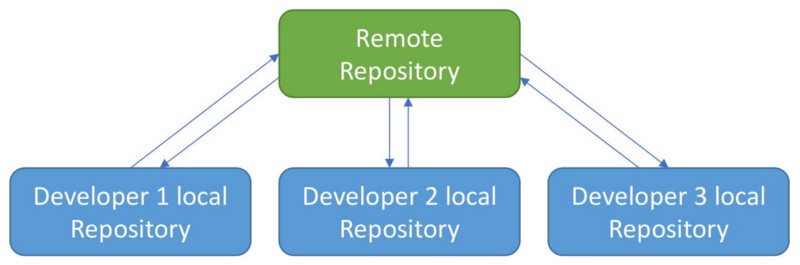
Дълго този стандарт е толкова добре работещ, че не се разглежда необходимост от подобряването му, но междувременно Microsoft оттеглят работата си по flash, като Adobe продължават да го поддържат за уеб, Майкрософт пуска проприетарна версия на flash, която обаче не става популярна, въпреки качеството 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.

# 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.

## Репозитории 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** - Изтласкваме (изпращаме, Push) вашия branch, за да го качим някъде другаде:

git push origin <branch-name>

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 репозитории:

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.

# Създаване на „web-приложение за резервации ”:

## Анализ на заданието

Заданието за дипломния проект предполага разработване на WEB-базирано приложение за резервации на места в ресторант. Клиентите трябва могат да проверяват наличието на свободни места за определена дата и заведение, да научат повече за предлаганите условия и менюта и да запазват места. Това всъщност означава, че трябва да се разработи сайт, който по своята структура ще следва концепцията за изграждане на динамичен информационен сайт. Информационен, защото по задание трябва да има няколко страници и да предлага информацция за различни ресторанти. Динамичен – защото се предполага че предлаганата информация може да се променя във времето.

Страниците които сайтът трябва да предоставя са „Начало“, „За нас“, „Резервации“ и „Контакти“. Секцията „Резервации“ следва да се оформи on-line форма за заявка на дата и час, ресторант и брой места и да се предвиди функционалност за проверка дали ресторанта е в състояние в посочения час да предостави на клиента заявената услуга или да я откаже. Необходимо е също така да се реализира и администраторско меню, което да позволи добавяне, редактиране или премахване на информация за съответните раздели(страници). Наложено е изискване сайтът да е адаптивен и да изглежда еднакво добре на различни устройства.

## Архитектуран на приложението

Със сигурност има различни начини, средства и технологии за решаване на поставената задача. Но моят избор е Python, Django и Bootstrap.

Bootstrap е библиотеката, която ще позволи практически бе специални усилия да се изгради т.н. адаптивен (responsive) дизайн, като по този начин ще се реализира условието от заданието сайта да изглежда добре на различни устройства.

Изборът на Python и Django всъщност избор на backend технология за решаване на поставената задача. Логино е цялата логика на приложението да бъде изнесена на сървъра (в backend-а), а за комуникация да се използва механизма на обвързване на форми с модели, който е заложен в Django. Реализацията на администраторски достъп също се решава практически автоматично, т. к това е вградена функционалност на избраният framework.

## **Подготовка за реализация на приложението:**

Преди всечко трябва да създадем проект (в смисъла на Django и PyCharm), в който ще работим. За целта:

* 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

Всеки Django проект може да съдържа в себе си няколко различни приложения, но задължително трябва да има поне едно. В случая под приложение се разбира функционално обособен модул на проекта. За тази цел:

* 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-файловете 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

• 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>')

## Клиентски интерфейс

Изборът на Django предопределя и подхода за изграждане на клиентския интерфейс. По задание сайтът трябва да има страници „Начало“, „За нас“, „Резервации“ и „Контакти“ и следователно трябва да има съответните шаблони Но ако се вгледам по подробно, страницата примерно about, страницата index, тоест началната, страницата резервации, те всичките могат да наследяват една обща страница, което по същество представлява класически Django подход. Наследявам файла base.html и там е класически шаблон. Всеки изглед генерира някакъв контекст, в случая имам меню, определям коя страница е активна, защото шаблона е общ но в зависимост от избора на потребителя активна е или страницата Начало или страницата За нас и т.н. Активната страница я подавам като контекст. Като контекст подавам името на таблицата, подзаглавието на таблицата и някои други неща. Иначе имам начало, като то всъщност представлява заглавие или подзаглавие, меню и форма за абонамент. Отивам на другата страница - там е почти същото като Начало. Това е Django такова каквото е , заради това, което го обичат неговите потребители.

## Използване на шаблони:

Ако изгледите често са нариччани контролери, заради логиката на приложението, която те реализират, то оформянето на външния вид е невъзможно без използването на HTML 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 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 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)

**Избираме тема –** темата всъщност е визуалната основа на нашето приложение. В идеалния случай екипът битрябвало да включва или да наеме уеб-дизайнер, който да разработи дизайна и съответните html, CSS и JS елементи на темата. Но на практика за малки проекти се използват готови теми (както платени, така и безплатни).

В конкретния случай след дълго лутане в Интернет съм избрала една тема на „HTML Codex“, която се казва „restoran-1.0.0“. Намирам я за най-близка до замисъл, който имам. Свалям архива и го разархивирам в някаква папка (например test). В папката на приложението (main) създавам папка static и в нея копирам всички общи файлове за темата. Създавам шаблон за общите части на различните страници – base.html. 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' %}" />

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** – това е следващата важна стъпка. За целта отваряме settings.py на проекта и настройваме DATABASES по следния начин:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': ‘dzi\_02’,

'USER': 'admin',

'PASSWORD': 'admin\_2024',

'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. Базата даанни е свързана и следва да създадем необходимите модели. Те се разполагат във файла models.py и представляват класове, всеки от които описва структурата на една таблица в БД:

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 = 'Резервация'

verbose\_name\_plural = 'Резервации'

И задължителни създаваме и изпълняваме миграциите:

Python manage.py makemigrations

Python manage.py migrate

**Създаване на администратор** – Django предлага автоматично изграждане на администраторски интерфейс, чрез който може да се управлява сайта като цяло, в т.ч. и да се създават и премахват потребители. Но преди това е необходимо да се създаде т.н. суперпотребител, което дава достъп до админ-панела:

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>

4. Спираме тестовия сървър и в терминала пишем команда

Python manage.py createsuperuser

5. 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:

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)

Параметърът params e всъщност контекстът, който се подава на шаблона. Представлява речник, към ключовете на който можем да се обръщаме в шаблона като към имена на променливи. Например, в index.html визуализиране на параметъра articles би изглеждало така:

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

Създаваме изгледи за всяка от страниците на сайта, обвързваме ги със съответните шаблони и създаваме и подаваме необходимия контекст.

В една база данни отделните таблици рядко са напълно независими една от друга. Между тях се изграждат връзки чрез т.н. ключови полета. Типичните връзки са „едно към едно“, „едно към много“ и „много към много“. Създаването на свързани таблици в Django e лесно, независимо от типа на връзката. За целта модулът models предлага типове полета, като най-използваното вероятно е models.ForeignKey:

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'

За да станат достъпни в админ-панела, създадените модели трябва да се регистрират в него. Това се прави рез съответните записи в admin.py:

from django.contrib import admin

from .models import Article, Comments

admin.site.register(Article)

admin.site.register(Comments)

## Заявки за резервация

За изпращане на запитвания към сървъра решавам да изплзвам класическият подход в уеб-програмирането – форми. От страна на клиента това просто използваме механизма на HTML5 за създаване и използване на форми(таг form):

<form>  
 <div class="row">  
 <div class="col-6">  
 <div class="form-group">  
 <label class="text-black" for="fname">Име</label>  
 <input type="text" class="form-control" id="fname">  
 </div>  
 </div>  
 <div class="col-6">  
 <div class="form-group">  
 <label class="text-black" for="lname">Фамилия</label>  
 <input type="text" class="form-control" id="lname">  
 </div>  
 </div>  
 </div>  
 <div class="form-group">  
 <label class="text-black" for="email">Email адрес</label>  
 <input type="email" class="form-control" id="email">  
 </div>  
  
 <div class="form-group mb-5">  
 <label class="text-black" for="message">съобщение/въпрос</label>  
 <textarea name="" class="form-control" id="message" cols="30" rows="5"></textarea>  
 </div>  
  
 <button type="submit" class="btn btn-primary-hover-outline">Изпрати</button>  
</form>

Обикновенно, формата извиква някакъв скрипт от страна на сървъра, който приема, валидира и обработва данните. Django има вградени механизми за работа с форми. Нещо повече, възможно е конкретна форма да се обвърже директно с таблица от базата данни. За целта създавам файл forms.py и в него задавам съответните връзки:

from django import forms  
from .models import Stock  
  
class StockForm(forms.ModelForm):  
 def \_\_init\_\_(self, \*args, \*\*kwargs):   
 *# used to set css classes to the various fields* super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.fields['name'].widget.attrs.update({'class': 'textinput form-control'})  
 self.fields['quantity'].widget.attrs.update({'class': 'textinput form-control', 'min': '0'})  
  
 class Meta:  
 model = Stock  
 fields = ['name', 'quantity']

class LoginUserForm(AuthenticationForm):  
 username = forms.CharField(label='Потребителско име: ', widget=forms.TextInput(attrs={'class':'form-control'}))  
 password = forms.CharField(label='Парола: ', widget=forms.PasswordInput(attrs={'class':'form-control'}))

За регистрирането наследявам формата за влизане на потребителя от идентификацонната система на Django. Наследявам класа потребители, респективно в базата данни, която се създава при създаването на Django проект има съответна таблица. Ннаследявам готовите форми, изграждам връзките на мойте форми с базата данни и следвам модела описан в документацията.

Това беше за формите, а в изгледите създавам изглед lоgin, който всъщност автоматично се опитва да логне програмно потребителя, задавам му success\_url, към който ще бъде насочен потребителя в случай на успешно влизане в системата и този изглед всъщност връща правилния url в случай, че потребителят е влязъл. Това прави Django толкова популярно, защото има такива удобни инструменти.

Третият елемент от тази система е в шаблоните. В login формата имам форма, но аз не я конструирам, а директно наследявам във файла forms, тоест шаблонизаторът като срещне променливата form ще генерира автоматично HTML формата, която пасва на описанието, което съм създал във forms, в моя случай съм казал, че искам само username и password. След като влезе потребителят можем даже да проверяваме за нива на достъп или за идентификационни данни – име, фамили и т.н Има такива ситуации, при които цялото приложение е изградено като интерактивно, еднодокументно или в случай, когато в рамките на някакъв сайт се вгражда някаква интерактивност на базата, на rest технологията. Но има и такива случай, при които се прави доста интерактивно приложение с използване само на форми, модели. С други думи – възможно е изграждане на пълноценни трислойни приложения само със средствата на Django каъвто е моят избор за конкретния проект.

## 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.

# Заключение

Представената разработка представлява едно завършено, пълнофункционално приложение. Основната цел – да се демонстрира определен подход за изграждане на потребителски интерфейс за уеб-базирано приложение я считам за постигната.

Има такива ситуации, при които цялото приложение е изградено като интерактивно, еднодокументно с интерактивност на базата, на rest технологията. Но има и такива случай, при които се прави интерактивно приложение с използване само на форми и модели. С други думи – възможно е изграждане на пълноценни трислойни приложения само със средствата на Django каъвто е моят избор за конкретния проект.

Python и Django успешно е една безспорно добра кобинация. Не случайно в редица класации те заемат челни позиции. Успешното съчетаване на подходящи технологии позволява даже и на програмисти с по-малък опит да създадът динамично и интерактивно приложение, което изглежда добре, функционира правилно и реализира всички поставени в заданието изисквания.

Естествено, има какво да се желае още и да се развива в бъдеще. Би могло, на първо място да се доразвие функционалността. Например, може да се добави автоматично уведомяване по е-mail или със SMS на подалият запитваане за резервация, с което да бъде информиран за действията, свързани със запитването. Би могло да се добавят и други функции, но това ще надхвърли значително изискванията на заданието.

Във вида в който е представено приложението е напълно функционално и би могло да се използва и в реална среда.

# Списък с използвана литература:

1 Django: https://bg.wikipedia.org/wiki/Django

2 Django-admin и manage.py: https://docs.djangoproject.com/en/4.2/ref/django-admin/

3 URL адреси: https://docs.djangoproject.com/en/4.2/topics/http/urls/

4 Debugger: https://bg.wikipedia.org/wiki

5 Бази данни: https://bg.wikipedia.org/wiki

6 HTML: https://bg.wikipedia.org/wiki/HTML

7 Admin panel: https://www.webdesigngroup.biz/