Съдържание

[1. Увод: 3](#_Toc164653164)

[2. Python 4](#_Toc164653165)

[2.1. Приложение и разпространение 4](#_Toc164653166)

[2.2. За какво се използва Python? 5](#_Toc164653167)

[2.2.1. Какво можете да правите с Python? 5](#_Toc164653168)

[2.2.2. Софтуерно тестване и прототипиране: 6](#_Toc164653169)

[2.2.3. Ежедневни задачи: 6](#_Toc164653170)

[2.2.4. Автоматизация или скриптове: 6](#_Toc164653171)

[2.2.5. Уеб разработка: 7](#_Toc164653172)

[2.3. Версии 7](#_Toc164653173)

[2.4. Аритметични операции 7](#_Toc164653174)

[2.5. Преглед на функциите 8](#_Toc164653175)

[3. Що е алгоритъм? 11](#_Toc164653176)

[3.1. Представяне: 14](#_Toc164653177)

[3.2. Описание от високо ниво: 15](#_Toc164653178)

[3.3. Описание на реализацията: 15](#_Toc164653179)

[3.4. Формално описание: 15](#_Toc164653180)

[4. Django REST Framework 16](#_Toc164653181)

[5. Django 16](#_Toc164653182)

[6. Операционна система (ОС) 17](#_Toc164653183)

[6.1. Основни функции на операционната система 19](#_Toc164653184)

[6.2. Управление на процесора 19](#_Toc164653185)

[6.3. Управление на паметта 20](#_Toc164653186)

[6.4. Потребителски интерфейс 20](#_Toc164653187)

[7. Какво е HTTP? 21](#_Toc164653188)

[7.1. Уеб браузърът и уеб сървърът си говорят на един език – HTTP 21](#_Toc164653189)

[8. База данни: 22](#_Toc164653190)

[8.1. Преглед на база данни: 23](#_Toc164653191)

[8.2. Приложение на база данни: 24](#_Toc164653192)

[9. Django-admin и manage.py: 25](#_Toc164653193)

[9.1. URL адреси: 25](#_Toc164653194)

[10. Реализация 27](#_Toc164653195)

[10.1. Анализ на заданието 27](#_Toc164653196)

[10.2. Разработване на макет 27](#_Toc164653197)

[10.3. Създаване на функционалност 30](#_Toc164653198)

[11. Заключение 33](#_Toc164653199)

[12. Списък с използвана литература: 34](#_Toc164653200)

# Увод:

С развитието на модерните технологии онлайн пазаруването заема все по-голяма част от ежедневието ни. Електронна търговия е продаване на стоки и услуги онлайн през интернет, посредством сайт или приложение. Няма съмнение, че през последните години този вид търговия е направила много бизнеси печеливши. Електронната търговия промени модерния пазар през последното десетилетие. ‘’Aysel Fashion ‘’ е перфектният начин да намериш стилни и модерни облекла без да излизаш от дома!

Въпреки че е доминиращ модел, продажбата на стоки и услуги онлайн идва със собствен набор от предимства в сравнение с традиционните бизнеси. Онлайн може да се пазарува по всяко време на денонощието, когато на всеки е удобно. Доставката също се извършва в удобно за всички време. Всичко се доставя до вратата на нашия дом. Спестява се реденето на опашки, обикалянето между стелажите и носенето на тежки торби. Цените са добри. Спокойното сравняване на стоки и услуги, анализирането и оценката на продуктите, са действия, които се вършат много по-лесно от компютъра вкъщи или в офиса, отколкото в натоварените и препълнени магазини. Този плюс на онлайн пазаруването е от изключителна важност през студените зимни дни или навалиците около различните празници. Факт е, че за по-заетите хора, пазаруването онлайн е особено важна привилегия. Така, ако няма необходимото време за избор на дрехи, подаръци и стоки от всякакъв тип, то съвсем спокойно може да се съчетае със сутрешното кафе в офиса или вечерята вкъщи.

Видовете и естеството на продуктите и услугите в интернет пространството е безкрайно. То обхваща много по-голям обем, от който и да бил мол или търговски център. Затова изборът е в много по-широк спектър и стига човек да знае какво търси. Почти винаги може да го намери след обстойно проучване, съчетано с не много инвестирано време и почти никакви усилия. Един от най-големите плюсове на пазаруването онлайн е ниската надценка на стоките и услугите. Така, ако в даден сайт цената на продукт е една, то в друг магазин тя е съвсем различна и завишена. Това е една от причините за изключителната популярност на пазаруването по интернет. Освен ниските цени, разнообразни са и начините за заплащане, в брой при получаване, по сметка, чрез карта, чрез пощенски запис. Може да се разглежда разнообразие от дрехи, да сравняват цени и да се правят покупки с лекота.

# Python

Python е език за програмиране от високо ниво, интерпретативен, интерактивен, обектно ориентиран, създаден от Гуидо ван Росум в началото на 90-те години. Кръстен е на телевизионното шоу на BBC Monty Python’s Flying Circus. Често бива сравняван с Tcl, Perl, Scheme, Java и Ruby. Python има сходства с езици като Perl, C, C++ и Java. Той е език за програмиране с общо предназначение. Други негови отличителни черти са, че е обектно-ориентиран, интерпретируем и интерактивен. „Интерпретируем“ означава, че за да изпробвате даден код дали работи, не е нужно да го компилирате преди това. „Обектно-ориентиран“ – че Python поддържа технологията на програмиране, при която кода се обособява в групи, наречени „обекти“, които взаимодействат помежду си.

## Приложение и разпространение

В наши дни този език се прилага навсякъде: от 3D рендериране във филмовата или игралната индустрия, управление и автоматизация на сървъри, дори и програми за други операционни системи като Linux OS. Други приложения като Drop Box, Bit Torrent, Blender също са написани на Python. За да направят софтуера си по-бърз, редица компании използват Python. Facebook и NASA са две от най-популярните имена, които използват Python широко в своята дейност. Python e често използван за разработка на уебсайт – софтуерната рамка на Python за разработка на уеб сайтове – “Django”, е една от най използваните в наши дни. Или може би желаете да разработвате игри? Тогава кросплатформения набор от Python модули (Pygame) е това, от което имате нужда. Ако искате да разработвате заглавия за мобилни платформи, то тогава имате на разположение библиотеката с отворен код Kivy. Python намира приложение и в машинното обучение (machine learning), съответно и в разработките на изкуствен интелект. Python има висока степен на четимост (т.е. почти идентичен с нормалния английски език), което го прави достъпен за много хора. Привличането на голям брой хора, които да го използват от своя страна увеличава способностите на езика, увеличава ресурсите, които са налични, увеличава проблемите, които могат да се решават с този език. Но най-вече Python никога не е зависел от платформата, за която се разработват приложенията. Дори да се върнем назад във времето и тръгнем към наши дни, можем да видим за колко различни платформи съществуват приложения, писани на Python: Linux, Window, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, Z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE.

## За какво се използва Python?

Python обикновено се използва за разработване на уебсайтове и софтуер, автоматизация на задачи, анализ на данни и визуализация на данни. Тъй като е относително лесен за научаване, Python е възприет от много непрограмисти, като счетоводители и учени, за различни ежедневни задачи, като организиране на финанси. „Писането на програми е много креативна и възнаграждаваща дейност“, казва инструкторът от Мичиганския университет и Coursera Чарлз Р. Северънс в книгата си „ Python for Everybody“. „Можете да пишете програми по много причини, вариращи от изкарване на прехраната ви до решаване на труден проблем с анализ на данни до забавление до помагане на някой друг да реши проблем.“

### Какво можете да правите с Python?

Някои неща включват:

• Анализ на данни и машинно обучение

• Уеб разработка

• Автоматизация или скриптове

• Софтуерно тестване и прототипиране

• Ежедневни задачи

### Софтуерно тестване и прототипиране:

При разработването на софтуер Python може да помогне при задачи като контрол на изграждането, проследяване на грешки и тестване. С Python разработчиците на софтуер могат да автоматизират тестването за нови продукти или функции. Някои инструменти на Python, използвани за тестване на софтуер, включват Green и Requestium.

### Ежедневни задачи:

Python не е само за програмисти и специалисти по данни. Изучаването на Python може да отвори нови възможности за тези в професии с по-малко данни, като журналисти, собственици на малък бизнес или специалисти по маркетинг в социални медии. Python може също да позволи на непрограмисти да опростят определени задачи в живота си. Ето само няколко от задачите, които можете да автоматизирате с Python:

* Следете цените на фондовия пазар или крипто
* Изпратете си текстово напомняне да носите чадър винаги, когато вали
* Актуализирайте списъка си за пазаруване на хранителни стоки
* Преименуване на големи пакети от файлове
* Преобразуване на текстови файлове в електронни таблици
* На случаен принцип възлагайте задължения на членове на семейството
* Попълвайте онлайн формуляри автоматично

### Автоматизация или скриптове:

Ако установите, че изпълнявате задача многократно, можете да работите по-ефективно, като я автоматизирате с Python. Писането на код, използван за изграждане на тези автоматизирани процеси, се нарича скриптове. В света на кодирането автоматизацията може да се използва за проверка за грешки в множество файлове, конвертиране на файлове, изпълнение на проста математика и премахване на дубликати в данни. Python може да се използва дори от сравнително начинаещи за автоматизиране на прости задачи на компютъра - като преименуване на файлове, намиране и изтегляне на онлайн съдържание или изпращане на имейли или текстове на желани интервали.

### Уеб разработка:

Python често се използва за разработване на задната част на уебсайт или приложение - частите, които потребителят не вижда. Ролята на Python в уеб разработката може да включва изпращане на данни до и от сървъри, обработка на данни и комуникация с бази данни, URL маршрутизиране и гарантиране на сигурност. Python предлага няколко рамки за уеб разработка. Често използваните включват Django и Flask. Някои работни места за уеб разработка, които използват Python, включват back end инженери, full stack инженери, Python разработчици, софтуерни инженери и DevOps инженери.

## Версии

Езикът се обновява често. Версия 3 съдържа много промени и поради това е несъвместима с по-старите реализации на езика. Повечето налични библиотеки и програми са написани и работят за версия 2.

## Аритметични операции

Python притежава обичайните аритметични оператори (+, -, \*, /, %), а \*\* е допълнителен оператор, който се използва за степенуване, например 5\*\*3 == 125. Поведението на операцията за деление се променя значително с течение на времето:

* Python 2.1 и по-ранни версии заимстват делението от езика C. Операторът / изпълнява целочислено деление, ако и двата операнда са цели числа, и деление с плаваща запетая в противния случай. Целочисленото деление закръглява към нулата, например 7 / 3 == 2 и -7 / 3 == -2.
* Python 2.2 променя целочисленото деление, така че да осъществява закръгляването надолу (към минус безкрайност), например 7 / 3 == 2 и -7 / 3 == -3. В тази версия е въведен операторът //, който извършва деление със закръгляване надолу (към по-ниската целочислена стойност). Неговото поведение може да се подразбере от следните примери: 7 // 3 == 2, -7 // 3 == -3, 7.5 // 3 == 2.0 и -7.5 // 3 == -3.0.
* Python 3.0 променя оператора /, така че той винаги да изпълнява ролята на деление с плаваща запетая. По-общо: във версиите преди Python 3.0 / представлява „класическо деление“, докато във версия 3.0 / е деление на реални числа, а // е деление със закръгляване надолу (към по-ниската целочислена стойност).

Макар и в противовес с повечето други езици, закръгляването надолу (в посока минус безкрайност) придава постоянство на изпълняваните операции. Например това означава, че равенството (a + b) // b == a // b + 1 е винаги вярно. Също така означава, че равенството b \* (a // b) + a % b == a е валидно както за положителните, така и за отрицателните стойности на a. Въпреки това поддържането на валидността на представеното равенство означава, че докато резултатът от a % b е в интервала [0, b) при положителна стойност на b, то при отрицателна стойност на b трябва да бъде в интервала (b, 0]. Python предлага функция round за закръгляване на десетични към цели числа.

Версиите преди Python 3.0 използват закръгляване далеч от нула: round(0.5) би дало като резултат 1.0 (закръгляване нагоре), съответно round(-0.5) връща −1.0 (закръгляване надолу). Python 3.0 използва закръгляване към най-близкото четно число: round(1.5) връща като резултат 2, съответно round(2.5) отново е равно на 2. Python позволява използването на булеви изрази с множество релации на равенство по начин, който е в съответствие със стандартната употреба на равенства в математиката. Например изразът a < b < c проверява дали a е по-малко от b и дали b е по-малко от c. Езиците, произхождащи от C, интерпретират този израз по различен начин: първо се пресмята a < b, връщащо като резултат 0 или 1, след което тази стойност се сравнява с c.

Поради своята богата математическа библиотека Python често е използван като скриптов език с научна насоченост за решаване на различни задачи от приложната математитка, например обработка и управление на числови данни.

## Преглед на функциите

За разлика от много други езици, Python не изисква декларации на променливи, те се създават при инициализацията им. Python разполага с вградени в самия език структури като: комплект (tuple), списък (list) и речник \*\*(dictionary/map). Типът на променливата се определя от типа на присвоените стойности, в това отношение прилича на BASIC и се различава от много други програмни езици. Присвояването на различни стойности е коректно и това води до промяна на типа на променливата спрямо последната и присвоена стойност.

* Кортеж (tuple)

Кортежът се състои от елементи, които могат да бъдат произволни: прости константи или обекти. Елементите на кортежа не е задължително да са еднородни. Кортежите (подобно на символните низове) са непроменливи: не е възможно да се присвои стойност на даден елемент от комплекта. Възможно е създаването на комплект, който съдържа в себе си променливи обекти – например списък (list). Комплектите могат да бъдат вложени един в друг. Дефинират се чрез изброяване със запетайка, поставени в кръгли скоби.

(1, 4, 1, 78)

(3.14, ‘ string ’, -5)

Всички елементи са индексирани от нулата. За получаването на i-тия елемент трябва да се посочи името на комплекта и след това индексът i в квадратни скоби.

Пример:

t = (0, 1, 2, 3, 4)

print t[0], t[-1], t[-3]

Резултат: 0 4 2

По този начин комплектът може да се нарече вектор – константа, ако елементите му са били еднородни.

* Списък (list)

Списъците са съставен (compound) тип данни, групиращ стойности (елементи), които сами по себе могат да бъдат от различен тип и могат да бъдат подменени след тяхната декларация. Възможно е присвояване на изрезки, като това може да промени и размера на списъка. Може да се вмъква друг списък в настоящия, да се добавят единични елементи, премахват и т.н.

Дефинира се чрез изброяване на елементите със запетая, поставени между квадратни скоби.

Пример:

[2, 4.87, ‘s’] [‘string’, (2,4,1), [5,8]]

Достъп до елементите се осъществява както и в комплекта.

Пример:

list = [1, ‘w’, (2,8), [0,3,4]] print list[0], list[1], list[-2], list[-1][0]

Резултат: 1 w (2,8) 0

2.5.1 Речник (dictionary/map)

Речникът е неподредено множество от двойки (ключ, стойност), като ключа е константа от всеки тип (най-вече символни низове). Комплекти могат да бъдат използвани за ключове само в случай, че не използват променливи обекти. Не може да се използва списък (list), тъй като той притежава възможността да се променя.

Основните операции са:

* Съхраняване на стойност под някакъв ключ (съхраняването на две стойности под един и същи ключ е невъзможно, запазва се втората такава подадена със съответния ключ).
* Извличане на стойността при подаване на ключа.
* Премахване на двойка ключ – стойност чрез „del“.
* Методът keys() на обект от тип речник връща списък с всички използвани ключове, а has\_key() проверява дали даден ключ е в речника.
* Чифт големи скоби „{ }“ създават празен речник.
* Пример:
* d = {'a': 1, 'b': 3, 5: 3.14, 'name': 'John'}
* d['b'] = d[5]
* print d['a'], d['b'], d[5], d['name'] Резултат: 1 3.14 3.14 John
* За да добавите нова двойка „ключ“:„стойност“, е достатъчно да присвоите на елемент с нов ключ съответната стойност:
* d['new'] = 'new value'
* print d
* Резултат: {'a':1, 'b':3, 5:3.14, 'name':'John', 'new':'new value'}
* Python не поддържа указатели, динамична памет и адресна аритметика (подобно на Java).
* Присвояване от една променлива към друга (Пример: a = 2; b = a;) има своите особености в Python. В случаи на присвояване на обекти от тип числа или низове се създава нов обект (под „нов обект“ да се разбира памет за съхранение на стойност от всеки тип). Копираното съдържание се присвоява от променливата 'а' в променливата 'b'(семантика на копирането – променливите от двете страни на оператора значат два независими обекта).

Ако от дясната страна стои стойност от тип клас, списък, речник или комплект присвояваме само връзката (указателя) към адреса, където е стойността на обекта, двата обекта сочат към една и съща памет – при промяна на стойността на едната се променя и другата (семантика на указатели).

Пример:

a = 5; b = a; b = 3

print ' семантика на копирането: a=', a, 'b=', b

a = [2,5]; b = a; b[0] = 7

print ' семантика на указателите: a=', a, 'b=', b

Резултат:

семантика на копирането: a = 5 b = 3

семантика на указателите: a= [7,5] b= [7,5]

# Що е алгоритъм?

Алгоритъм (от името на учения ал–Хорезми) е термин от математиката, информатиката, лингвистиката и други области, с който се описва сложно действие чрез редица от елементарни действия, които изпълняващият може да извърши в последователни стъпки без допълнителни обяснения. Обикновено изпълнението на алгоритъма включва изчисление или обработка на данни.

По-строго дефинирано, алгоритъмът е ефективен метод за изчисляване на функция, който може да бъде изразен в рамките на крайно време и пространство и чрез добре дефиниран формален език. Започвайки от начално състояние и входни данни (понякога празни), инструкциите описват пресмятания, чието изпълнение преминава през краен брой добре дефинирани последователни състояния и завършва с крайно състояние, като в процеса се получават крайни резултати. Не е задължително преходът между състоянията да е детерминиран (еднозначно определен): някои алгоритми, известни като вероятностни алгоритми, съдържат елемент на случайност.

Концепцията за алгоритмите съществува от векове, но частичното формулиране на понятието започва с опитите да се реши 10-ия проблем на Хилберт – „Задача за разрешимост на диофантово уравнение“, поставен от Давид Хилберт през 1900 година на Втория световен конгрес по математика в Париж. Последващите формулировки, при които се цели дефинирането на „ефективна изчислимост“или „ефективен метод“, включват рекурсивните функции на Ербран-Гьодел-Клини от 1930, 1934 и 1935 година, ламбда смятането на Алонсо Чърч от 1936 година, „Формулировка 1“ на Емил Пост от 1936 година и машината на Тюринг от 1936 – 1937 и 1939 година.

Създаването на формална дефиниция на алгоритъм, съответстваща на интуитивното понятие, остава отворен въпрос и в наши дни. При все че няма общоприета формална дефиниция на алгоритъм, неформално понятието може да се определи като „набор от правила, които точно дефинират някаква поредица от операции“. Това определение обхваща всички компютърни програми, включително тези, които не извършват числени изчисления, стига те да прекратяват работа след краен брой операции.

Класически пример за „алгоритъм“ е алгоритъмът на Евклид чрез изваждане за намиране на най-големия общ делител (НОД) на две цели числа, по-големи от 1. При него се изпълнява следната поредица от стъпки: На стъпка i, се дели X на Y и остатъкът се означава с R. Ако R = 0, резултатът на задачата е Y. В противен случай, резултатът съвпада с НОД на числата R и Y. Такъв алгоритъм, при който за намирането на решение е необходимо да бъде решена аналогична, но „по-малка“ задача, се нарича рекурсивен. Американските учени Джордж Булос и Ричард Джефри дават следното неформално определение за алгоритъм в своя класически учебник по математическа логика:

Никое човешко същество не може да пише достатъчно бързо или достатъчно дълго, или достатъчно дребно, за да изброи всички елементи на дадено изброимо безкрайно множество, изписвайки техните имена, едно след друго, с една и съща нотация. Но хората могат да направят нещо също толкова полезно в случая с някои изброими безкрайни множества – те могат да дадат експлицитни указания за определяне на n-тия елемент на множеството за произволно крайно n. Такива указания могат да бъдат дадени съвсем експлицитно, във форма, в която могат да бъдат изпълнени от изчислителна машина или от човек, който може да извършва само съвсем елементарни операции със символи. Изброимо безкрайно множество е такова множество, за чиито елементи може да се дефинира биективно съответствие с множеството на естествените числа. Така според Булос и Джефри алгоритъмът съдържа инструкции за процес, който „създава“ резултатни цели числа от произволно входно цяло число (или входни числа), което на теория може да бъде произволно голямо. По този начин алгоритъмът може да бъде алгебрично уравнение, като y = m + n – две произволни „входни променливи“ m и n, които създават резултат y. В същото време опитите на различни автори да дефинират понятието показват, че под „алгоритъм“ се разбира много повече от това:

Точни указания (на език, разбираем за „компютъра“) за бърз, ефективен, „добър“ процес, който специфицира „движенията“ на „компютъра“ (машина или човек, оборудван с необходимата вътрешна информация и способности), за да намери, декодира и обработи произволни входни числа/символи m е n, символи + и = ... и „ефективно“ да произведе, за „разумно“ време, краен резултат y на определено място и в определен формат.

Концепцията за „алгоритъм“ се използва и за дефиниране на понятието решимост, което играе централна роля за обяснението как формалните системи възникват от малък набор аксиоми и правила.

В логиката времето, необходимо за изпълнение на даден алгоритъм, не може да се измери, тъй като то не е свързано с обичайното физично измерение. Подобни несигурности, които характеризират текущите изследвания в тази област, водят до липсата на дефиниция за „алгоритъм“, която да съответства както на конкретната, така и на абстрактната употреба на термина.

Алгоритмите са от съществено значение за начините, по които компютрите обработват информацията. Много компютърни програми съдържат алгоритми, които определят специфични инструкции, които компютърът трябва да изпълни в строго определен ред, за да се реши дадена задача. В този смисъл, алгоритъмът може да се разглежда като произволна поредица от операции, които могат да се симулират от пълна по Тюринг система. Към този възглед се придържат автори като Мински, Савидж и Гуревич:

Но ние също така ще твърдим, като Тюринг..., че всяка процедура, която може „естествено“ да бъде наречена ефективна, всъщност може да бъде реализирана от (проста) машина. Макар че това може да изглежда крайно, аргументите... в полза на това е трудно да се отхвърлят. Неформалният аргумент на Тюринг в полза на неговата теза доказва и по-силно твърдение: всеки алгоритъм може да бъде симулиран от машина на Тюринг... според Савидж [1987], алгоритъм е изчислителен процес, дефиниран от машина на Тюринг. Обичайно, когато един алгоритъм е свързан с обработка на информация, данните се четат от входа, изпращат се на изхода и/или се съхраняват за по-нататъшна обработка. Съхранените данни се разглеждат като част от вътрешното състояние на обекта, изпълняващ алгоритъма. На практика, състоянието се пази в една или повече структури от данни.

За всеки подобен изчислителен процес трябва строго да се дефинира един алгоритъм, като се определи така, че да е приложим при всички възможни обстоятелства, които могат да възникнат. Това ще рече, че систематично трябва да се разгледат всички условни разклонения, случай по случай, като критериите за всеки от случаите трябва да са ясно дефинирани и изчислими.

Тъй като алгоритъмът е точно определен списък от точно определени стъпки, редът на изчислението им винаги е от критично значение за работата на алгоритъма. Обикновено се предполага, че инструкциите са изрично изброени и са описани отначало-докрай, така както се изобразява една блок-схема. Това разбиране за формализацията на алгоритъма е основано на принципите на императивното програмиране. Други алтернативни концепции за алгоритмите предоставят функционалното и логическото програмиране.

## Представяне:

Алгоритмите могат да се представят с много различни видове нотация, в това число естествени езици, псевдокод, блок-схеми или програмни езици. Описанието на алгоритми на естествен език страда от обичайната склонност на езика към многословие и многосмисленост и поради това рядко се използва за формулирането на сложни алгоритми. Псевдокодът и блок-схемите представляват структурирани начини за изразяване на алгоритми, които избягват двусмислиците на естествения език и са независими от конкретния програмен език, на който алгоритмите се реализират. Програмните езици са главно насочени към изразяването на алгоритми в изпълним от компютър вид, но често се ползват и за да онагледяват, дефинират или документират алгоритмите.

Има голямо разнообразие от начини за представяне на алгоритмите – дадена програма за машина на Тюринг може да се опише като поредица от машинни таблици, като блок-схема, като рудиментарен машинен код или асемблерен код. Представянията на алгоритмите могат да се класифицират в три нива на описание на машината на Тюринг:

## Описание от високо ниво:

„...текст за описване на алгоритъм, игнориращ подробностите на реализацията му. На това ниво няма нужда да се отбелязва как машината управлява своята памет или глава.“

## Описание на реализацията:

„...текст, използван за дефиниране на начина, по който машината на Тюринг използва своята глава, и начина, по който съхранява данните в паметта си. На това ниво не се дават подробности за състоянията или преходните функции.“

## Формално описание:

Най-подробното, „на най-ниско ниво“, задава „таблица на състоянията“ на машината на Тюринг.

# Django REST Framework

Django REST Framework (изходен код), обикновено съкратено "DRF", е библиотека на Python за изграждане на интерфейси за програмиране на уеб приложения (API).

Много рамки ви позволяват лесно да създавате API за блог приложения, но ние ще използваме само една – Django REST framework (DRF).

Той е удобен по много начини и предлага следните предимства:

Неговият API за уеб сърфиране е огромна полза за вашите разработчици.

Правилата за удостоверяване включват пакети за OAuth1 и OAuth2 .

Сериализацията поддържа както ORM , така и не-ORM източници на данни.

Може да се персонализира докрай. Просто използвайте обикновени изгледи, базирани на функции, ако не се нуждаете от по- мощните функции .

Има богата документация и голяма подкрепа от общността .

Използва се и се ползва с доверие от международно признати компании, включително Mozilla , Red Hat , Heroku и Eventbrite .

REST, което означава „REpresentational State Transfer“, е една от програмните архитектури, които определят как работят Django API. Терминът „RESTful“ се използва за уеб услуги, които се придържат към ограниченията на REST. С две думи, докато DRF улеснява работата с Django, RESTful API работят според REST.

# Django

Django е уеб рамка на високо ниво на Python, която насърчава бързото развитие и чистия, прагматичен дизайн. Създаден от опитни разработчици, той се грижи за голяма част от неприятностите на уеб разработката, така че можете да се съсредоточите върху писането на вашето приложение, без да е необходимо да преоткривате колелото. Той е безплатен и с отворен код.

Django е широко използвана рамка за уеб приложения на Python с философия „включени батерии“. Принципът зад batteries-included е, че общата функционалност за изграждане на уеб приложения трябва да идва с рамката, вместо като отделни библиотеки.

# Операционна система (ОС)

Операционната система (ОС) е основна част от компютърния системен софтуер, която управлява и координира ресурсите на хардуера и софтуера и обслужва изпълняваните компютърни програми. Приложният софтуер обикновено има нужда от ОС, за да работи.

ОС разпределя задачите по време, планира ефикасното използване на ресурсите на системата и може да включва специализиран софтуер за изчисление на ресурси: стойност на процесорното време, използвана памет, ресурси за печат и други. При входно-изходнитеоперации и динамично разпределение на паметта ОС действа като посредник между приложния софтуер и хардуера, макар че приложните команди обикновено се изпълняват директно, чрез системни повиквания към функции на ОС. ОС се среща на повечето устройства с компютри – от мобилни телефони през игрални конзоли до уеб сървъри и суперкомпютри. В компютърната техника операционната система (ОС) е софтуер, който управлява компютърни хардуерни и софтуерни ресурси и предоставя общи услуги за компютърни програми. Ето някои често срещани операционни системи:

1. Windows: Разработена от Microsoft, Windows е широко използвана операционна система за персонални компютри. Той предоставя графичен потребителски интерфейс и поддържа широк набор от софтуерни приложения.

2.macOS: Разработена от Apple, macOS е операционната система, използвана на Macintosh компютрите на Apple. Той осигурява удобен за потребителя интерфейс и е известен със своята стабилност и сигурност.

3.Linux: Linux е безплатна операционна система с отворен код, която се използва широко в сървъри и суперкомпютри. Той предоставя широк набор от инструменти и помощни програми за разработчици и системни администратори.

4.Android: Android е популярна операционна система за мобилни устройства, разработена от Google. Базиран е на ядрото на Linux и предоставя лесен за използване интерфейс за мобилни устройства.

5.iOS: Разработена от Apple, iOS е операционната система, използвана на iPhone и iPad. Той осигурява удобен за потребителя интерфейс и е известен със своите функции за сигурност.

6.Chrome OS: Разработена от Google, Chrome OS е лека операционна система, предназначена за използване на Chromebook. Базиран е на ядрото на Linux и предоставя удобен за потребителя интерфейс за уеб сърфиране и приложения за продуктивност.

Това са само няколко примера от многото операционни системи, които съществуват днес. Всяка операционна система има свои уникални характеристики и силни страни и изборът на операционна система зависи от специфичните нужди на потребителя.

Терминът операционна система, най-вероятно ви е познат още от ерата на първите персонални компютри. Днес, умните устройства вече са десетки видове, тяхната работа нямаше да бъде толкова продуктивна, ако не разполагаха с операционни системи, които да ни позволяват не само бързодействие на процеса, който сме им задали, но и мултитаскинг, за да използваме времето си още по-ефективно. Всъщност не всеки компютър разполага с операционна система – този, който управлява микровълновата ни фурна, например, няма нужда от такава, тъй като наборът от задачи, които трябва да изпълнява никога не се променя. От операционна система, обаче, имат нужда всички останали устройства, които изпълняват различни цели, комуникират с потребителите по по-сложен начин и могат да бъдат надграждани с цел отговаряне на променящите се нужди на хората.

Както сами можете да се досетите, нашите [смартфони](http://revo.bg/telefoni.html) и [таблети](http://revo.bg/tablets.html) отговарят на тези условия, което обяснява и защо те не могат да бъдат интелигентни мобилни устройства без да разполагат с операционна система. И макар всеки да знае какво е операционната система на практика, тук ще погледнем към нея от теоретична гледна точка.

Всяко едно устройство, което разполага с операционна система (ОС), може да промени своята работа с модифициране на същата тази операционна система. Точно поради тази причина основният софтуер на устройствата (операционната система) е съставена от така наречен „портативен” код, който търпи изменения и може да бъде надграден без промяната да окаже каквото и да е въздействие върху хардуера.

## Основни функции на операционната система

Операционната система изпълнява две основни функции, описани по-долу:

1. Тя управлява хардуерните и софтуерните ресурси на системата, към които спадат неща като: централен процесор, памет, екран, списък с контакти, батерия, свързване към интернет мрежа и какво ли още не. Тази функция е жизненоважна, тъй като многобройни програми и методи за въвеждане на информация се съревновават едновременно за „вниманието” на централния процесор и едновременно изискват оперативна памет и пространство. Целта на ОС, следователно, е да разпределя наличните ресурси между всички тези приложения по възможно най-оптимален начин.
2. ОС осигурява стабилен и постоянен начин, по който приложенията да се справят с хардуера без да знаят подробности за този хардуер. Този „начин” е по-известен като приложно-програмен интерфейс (API), който позволява на един програмист да напише приложение на едно компютърно устройство и да бъде почти 100% сигурен, че приложението ще бъде разпознато от друго устройство от същия тип.

## Управление на процесора

Управлението на процесора се изразява в две неща: гарантиране, че всеки процес или приложение получава достатъчно от времето на процесора, за да функционира и използване на колкото се може повече цикли на процесора за реална работа. В зависимост от начина, по който ОС управлява споменатите процеси различаваме два вида операционни системи: single-tasking и multi-tasking. Най-просто казано, първият вид ОС ни оставя с впечатлението, че компютърът работи само по една задача, докато вторият – че работи едновременно по много. Мобилните операционни системи, които ползваме днес, са оборудвани с отлични възможности за мултитаскинг.

## Управление на паметта

Управлението на паметта се изразява в следните две задачи: 1) на всеки процес трябва да бъде осигурена достатъчно памет, без това да пречи на останалите процеси и 2) различните видове памет в системата трябва да се използват правилно, така че всеки процес да се изпълнява с оптимален ефект.

Управлението на капацитета на твърдия диск (при мобилните устройства това е вграденият сторидж) е най-бавно. Освен с него, обаче, ОС се занимава с още два вида памет:

Високоскоростен кеш: това е бързо, сравнително малко количество памет, до което централният процесор достига най-бързо. Контролерите на кеша предвиждат от кои данни ще има нужда процесорът и след това изтеглят тези данни от основната памет в кеша, за да ускорят производителността на системата.

Основна памет: позната още като RAM памет.

Целта на ОС е да балансира между всички тези видове памет, придвижвайки данните на блокове (наричани още страници) между наличната памет в съответствие с протичащите процеси.

## Потребителски интерфейс

Когато мислим за операционни системи, ние обикновено си представяме техния „външен вид” или потребителския интерфейс. Потребителският интерфейс всъщност е нещо като „горния слой” на операционната система, а управлението на всички основни функции се случва в „ядрото”. Връзката между ядрото на операционната система и потребителския интерфейс определя и разликите между съвременните операционни системи.

# Какво е HTTP?

[**HTTP**](https://bg.wikipedia.org/wiki/HTTP) (**H**yper**T**ext **T**ransfer **P**rotocol) е протокол за трансфер на [хипертекст](https://help.superhosting.bg/hypertex-world-wide-web.html) в интернет. Първата версия на протокола е от 1991 г., а през 2015 г. излиза новата и по-добра версия HTTP/2.

HTTP протоколът е измислен със създаването на [World Wide Web](https://help.superhosting.bg/world-wide-web.html) (световната информационна мрежа или уеб). Този протокол определя правилата, по които се извършва обмяната на информация между уеб сървър и уеб клиент (най-често уеб браузър).

Когато [Тим Бърнърс-Лий](https://bg.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BC_%D0%91%D1%8A%D1%80%D0%BD%D1%8A%D1%80%D1%81-%D0%9B%D0%B8%D0%B9) осъществява идеята си, за мрежа от взаимно свързана информация, наричана днес [уеб](https://help.superhosting.bg/world-wide-web.html), той създава и основните ѝ компоненти. Създаден е първият [уеб сървър](https://help.superhosting.bg/what-is-web-server.html), който предоставя първата уеб страница на първия уеб браузър. Комуникацията между уеб браузър и уеб сървър се извършва по създадения за тази цел протокол – HTTP.

В HTTP протокола [са описани](https://datatracker.ietf.org/doc/html/rfc7230) няколко [метода](https://datatracker.ietf.org/doc/html/rfc7231) за поискване (и изпращане) на информация от уеб браузъра към уеб сървъра. Тези HTTP методи се съдържат в заявките, които уеб браузърът изпраща към уеб сървъра.

Най-често срещаните HTTP методи са GET и POST. Съществуват още: HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT.

В HTTP протокола са описани и HTTP статус кодове, които се изпращат от уеб сървъра, в отговор на заявките от уеб браузъра.

## Уеб браузърът и уеб сървърът си говорят на един език – HTTP

Уеб браузърът се свързва с уеб сървъра чрез мрежова TCP връзка, по която изпраща HTTP заявки и приема HTTP отговори.

HTTP заявките от браузъра и HTTP отговорите на сървъра започват с тъй наречените HTTP хедъри. Това са редовете, съдържащи важните указания, по които протича HTTP обмена на информация.

В началото на HTTP заявката чрез HTTP хедърите браузърът описва какво търси и в какъв вид данни би могъл да го получи. Според това описание уеб сървърът ще знае какво и как точно да изпрати към уеб браузъра.

Например в следната HTTP заявка браузърът пита уеб сървъра за ресурса index.html, като го търси на хост help.superhosting.bg и заявява, че разговорът ще се проведе по протокол HTTP/1.1:

GET /index.html HTTP/1.1

Host: help.superhosting.bg

В следващите HTTP хедъри предоставя повече информация за себе си (User-Agent) и за възможностите и предпочитанията си. Подава информация на уеб сървъра, че може да приеме данните в определен формат (Accept) – text, html, xml, xhtml и че може да са компресирани (gzip, deflate).

User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml

Accept-Encoding: gzip, deflate

На тази HTTP заявка уеб сървърът връща HTTP отговор, започващ с код 200 OK. Това означава, че търсеният ресурс е наличен, заявката е приета и информацията се изпраща към уеб браузъра.

HTTP/1.1 200 OK

В следващите HTTP хедъри уеб сървърът предоставя повече информация за себе си (Server), за установената връзка (Connection) и за самата информация (Content), която изпраща към браузъра.

# 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.

# 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),

]

# Реализация

## Анализ на заданието

Заданието за дипломния проект предполага, реализиране на сайт за електронна търговия с дрехи с възможност за търсене и филтриране на артикули.

Уебсайтът трябва да показва наличните продукти заедно с техните описания. Както всеки друг уебсайт за електронна търговия, сайтът трябва да има система за колички. Клиентите трябва да могат да добавят артикули в количката си, да променят количеството, както и да се отказават от направения избор. Трябва да се предвидят необходимите формуляри за завършване на покупката и за плащане. Естествено, плащането ще бъде фиктивно

Системата трябва да може да се управлява от администратор, който наблюдава, редактира или добавя клиенти, следи поръчките, редактира и променя артикули и друга подходяща информация.

Дизайнът на сайта трябва да е адаптивен и да изглежда еднакво добре на различни устройства.

## Разработване на макет

Практическата разработка започва от разработване на визуален макет на приложение. По принцип, първата задача е да се избере дизайн на проекта. Понеже уеб дизайнът е отделна тема и в постановката на заданието не стои въпроса за разработване на дизайн, по тази причина избирам готов шаблон. Почти винаги се налага да направим известна модификация. И тъй като в заданието имам формулировка да е responsive дизайна, т.е. да може да работи еднакво добре на различни устройства, още от самото начало аз съм се насочила към bootstrap framework от страна на клиента. С оглед на това, че bootstrap сам по себе си вече е responsive дизайн, то използвайки bootstrap като framework за разработване на HTML-a, си гарантирам решаване на тази точка от заданието.

Трябва да се проверява дали всичко се визуализира добре на различни устройства и в различни браузъри.

Технологичната основа я има. Има и проект в PyCharm и съответната структура. За основен базов шаблон съм избрала шаблона “eshoper”. Създала съм си папка “base”, където съм изграждила своя макет. В общия случай прехвърляме само това, което ни е необходимо. По-добре е да добавим нещо в последствие, което сме пропуснали, отколкото да копираме едно съм едно всичко и след това да имаме доста ненужни неща, които заемат място на диска, както и на сървъра и пречат или най-малкото създават объркване. Променяме визията, където искаме да бъде променено.

Премахваме излишните секции, елементи от шаблона, които не са ми нужни и евентуално доразработваме, следвайки стила на шаблона.

<a href="" class="option1">

Add To Cart

</a>

<a href="" class="option2">

Buy Now

</a>

</div>

</div>

<div class="img-box">

<img src="images/p11.png" alt="">

</div>

<div class="detail-box">

<h5>

Създаваме това, което на нас ни е необходимо, според функционалността, която очакваме от приложението, без обаче да създаваме функционалност.

<div class="col-md-6">

<div class="subscribe\_form mt-10">

<div class="heading\_container heading\_center">

<h3>ВХОД</h3>

</div>

<p>Само регистрирани потребители могат да публикуват обяви</p>

<form action="">

<input id="f1" type="text" placeholder="Въведете потребителско име">

<input type="password" placeholder="Въведете парола">

<button>

Влизане

</button>

</form>

</div>

На този ме интересува само визията. На стилово ниво в CSS файла style съм променила, който е специфичен за шаблона, променили съм цвета на всички места където е било необходимо. Също така промених на първо място менюто и иконата на приложението. Менюто е разработено според очакваната функционалност.

Имаме слайд шоу, но съм работила единствено върху първия слайд, защото другите ще бъдат еднотипни и ще следват същия шаблон, също така ще се генерират автоматично, т.е. за да преценим визията ни е необходим само един слайд.

Същото е и за панелите за стоките, където модифицирам само първата по същите съображения, останалото ще се генерира програмно върху шаблона на първата от тях. Ползваме шаблона като концепция, но го разширяваме когато е необходимо. В оригиналния HTML шаблон няма такава секция, която да има локално меню, но понеже има главно меню ние ползваме него като шаблон и правим локално меню.

Естествено накрая приключвам с футер, така както е прието да има във всяко уеб приложение като още на този етап си представяме че футера и хедера са под формата на меню. Хедера ни е логото на хипотетичната фирма и главното меню. Футера и хедера са винаги активни, а останалите секции ще бъде активни една по една - само тази която според конкретната функционалност е извикана от потребителя.

На този етап е правилно да проверим как ще изглежда и на друго устройство за целта ползваме вградения във всеки браузър инструмент за програмисти за симулация на различни клиентски устройстваа.

## Създаване на функционалност

Пристъпваме към практическата изработка на отделните функции. Идеята е клиента си получава всичкия HTML, но и да се създаде възможност комуникация със сървъра на ниво данни. Използваме Python Django за backend и VUE за frontend разработката. След като избрах технологията Django реших да организирам application програмите в интерфейса API на базата на библиотека Django Rest Framework (DRF). Реших че frontend-a ще бъде базиран на HTML в комбинация с JavaScript библиотеката VUE. За целта в работната среда pycharm инсталираме Django, създаваме Django приложение, също така и проект MyProject в него създаваме приложение main, инсталираме библиотеката DRF, регистрираме приложението което ние сме създали main, регистрираме и Rest framework и сме готови да пристъпим към практическата работа. За бази данни съм избрала MySQL за целта съм инсталирала библиотеката MySQLclient и съгласно инструкциите за инсталиране на библиотеката така както е в документацията съм си настроила връзка към БД като предварително съм си създала БД - в моя случаи dzi-03.

Създала съм папката templates с името на приложението main. Слагам index, който е въпросният index.html. Всички статични ресурси и в Django се изисква статичните ресурси да се обособение в папката по подразбиране static. В папката static слагаме всички възможни ресурси. Пристъпвам към настройване на приложението. То става като първа стъпка задаваме директиви към шаблонизаторът, за да зареди статичните ресурси. Втората стъпка е навсякъде където се ползва статичен ресурс да се добави командата static.

<!-- bootstrap core css -->

<link rel="stylesheet" type="text/css" href="css/bootstrap.css" />

<!-- f<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/html">

<head>

<!-- Basic -->

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<!-- Mobile Metas -->

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

След като направим static, е време за първите проби. Следващата стъпка ми беше да си настроя реалната функционалност. За да се включи библиотеката VUE добавям няколко Java Script файла.

<script src="https://unpkg.com/vue@3"></script>

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

<script src="{% static 'js/logic.js' %}"></script>

</body>

</html>

Vue има механизъм, който да ги обвърже с някакви контроли в html кода. Обектът има 2 части, “data” връща променливите с които ще работя в шаблона, “methods” функции които аз ще обвързвам с мойте действия.

В главното меню имам категории. Те се попълват като от БД се извади категориите, които реално ги има и динамично се изграждат този списък категории в случая това са “Дамски, Детски, Мъжки”.

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" data-toggle="dropdown"

role="button" aria-haspopup="true" aria-expanded="true">

<span class="nav-label">

Категории <span class="caret"> </span>

</span>

</a>

<ul class="dropdown-menu">

<li><a href="#list" @click="set\_filter(0)">Всички</a></li>

<li v-for="(category, idx) in categories">

<a href="#list" @click="set\_filter(category.id)">

{{category.name}}

</a>

</li>

В модела за базата “cart\_item” има поле което препраща към таблицата “size”. Имам предварително създадена таблица с възможните размери и всеки клиент си избира от нея.

size = models.ForeignKey(Size, verbose\_name='Размер', on\_delete=models.CASCADE, null=True, related\_name='size')

description = models.TextField('Описание', default='', blank=True, help\_text='Описание на предлагания продукт')

remark = models.TextField('Забележки', default='', blank=True, help\_text='Текст на съобщение')

picture = models.ImageField('Снимка', upload\_to='post\_pics', blank=True)

user = models.ForeignKey(AppUser, verbose\_name='Потребител', on\_delete=models.CASCADE, null=True,

Изграждаме необходимите изгледи – те са два вида. Единият вид е класическият за Django когато извикваме определен шаблон. Вторият е за така наречените REST услуги.

И на последно място имам и административен панел, който Django сам го предлага.

.

# Заключение

Това е моят дипломен проект. Работата по него ми даде възможност да се запозная с уеб-базираните приложения за електронна търговия. Надявам се и в бъдеще да имам още успешни задачи. Самата разработка е по-скоро макет, илюстрация на идеята за мелетронен магазин. За създаването на реално работещо приложение е необходимо да се разработят и редица други компоненти. Така например, професионално изработените сайтове в тази област обиновенно имат модули за различни видове плащания, за следене на поръчки и пратките, поддържат складово стопанство, имат елементи на CRM, събират статистическа информация и т.н.

Но поставените в заданието задачи са решени и считам, че се получи едно добро приложение.

В разработката съм използвала различни инструменти и технологии:

* Python и Django: Използвани за създаване на сигурни и мащабируеми уеб приложения.
* HTML5 и CSS3: Основни технологии за създаване и стилизиране на уеб страници.
* Bootstrap: Библиотека за уеб дизайн, предоставяща готови компоненти за отзивчиви уеб интерфейси.
* JavaScript и библиотеки Axios и Vue.js: Добавят интерактивност и динамичност в уеб приложенията.

Тези технологии работят синергично, позволявайки ни да създаваме функционални и съвременни уеб приложения, които са привлекателни за потребителите. Комбинацията им осигурява цялостно решение за разработване на уеб софтуер.

Надявам се, разработеното от мене приложение да послужи за модел и основа на реално приложение, което да спечели интереса на потребителите и да намери своето приложение.

# 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/