Съдържание

[1. УВОД 3](#_Toc164574413)

[2. Софтуерни технологии и инструменти 4](#_Toc164574414)

[2.1 HTML 4](#_Toc164574415)

[2.2 Python 5](#_Toc164574416)

[2.3 CSS 6](#_Toc164574417)

[2.3.1 Видове селектори: 7](#_Toc164574418)

[2.3.2 Групиране на селектори 8](#_Toc164574419)

[2.4 JS 8](#_Toc164574420)

[2.5 MySQL 10](#_Toc164574421)

[2.6 JINJA 12](#_Toc164574422)

[2.6.1 Характеристика: 12](#_Toc164574423)

[2.6.2 Синтаксис: 12](#_Toc164574424)

[2.7 REST 13](#_Toc164574425)

[2.8 Django 14](#_Toc164574426)

[2.9 Vue 16](#_Toc164574427)

[2.9.1 Компоненти: 16](#_Toc164574428)

[2.9.2 Шаблони 17](#_Toc164574429)

[2.9.3 Реактивност 17](#_Toc164574430)

[2.9.4 Преходи 17](#_Toc164574431)

[2.10 API 18](#_Toc164574432)

[2.10.1 Библиотеки и рамки 19](#_Toc164574433)

[2.10.2 Операционна система 20](#_Toc164574434)

[3. Етапи на проекта 21](#_Toc164574435)

[3.1 Анализ на заданието 21](#_Toc164574436)

[3.2 Избор на конктерна технология 21](#_Toc164574437)

[3.3 Клиентски интерфейс 25](#_Toc164574438)

[3.4 Администативен интерфейс 27](#_Toc164574439)

[4. Заключение: 29](#_Toc164574440)

[5. Списък с използвана литература: 30](#_Toc164574441)

# УВОД

Онлайн сайтовете за мебели могат да се използват за различни цели – да се разглеждат и закупуват от клиентите. Предоставят възможност за получаване на идеи и вдъхновение за дома чрез представяне на дизайни, тенденции и стилове.

Някои сайтове предлагат възможност за персонализиране на мебелите, като избор материали, цветове и размери. Също предоставят услуги за онлайн интериорен дизайн, където клиентите могат да получат съвети за подреждане на мебелите в своя дом. Някои сайтове предоставят възможност за визуализация на мебелите във виртуално преди покупката.

Пазаруването онлайн е удобно и достъпно постоянно, позлолявайки да се правят покупки по график. Получава се достъп до отзиви от други клиенти, което помага да се направи информиран избор и да се разбере качеството на продукта.

Онлайн сайтовете за мебели също могат да послужат на клиентите да следят обявяване на различни събития, намаления и промоции, които да привличат нови клиенти и да задържат старите.

Пазаруването онлайн предоставя подробен виртуален каталог с разнообразие от мебели, снабден с ясни снимки, описания, и технически характеристики.

# Софтуерни технологии и инструменти

## HTML

HTML (Hyper Text Markup Language) е абревиатура за оформление. Това е основният език за създаване на уеб страници. В HTML се използват тагове за определяне на структурата и съдържанието на уеб страниците, чрез различни HTML елементи. Тези елементи могат да включват текст, снимки, линкове, форми, видео и друг медийни елементи, които се визуализират в уеб браузърите на потребителите. В повечето случаи HTML кодът е написан в текстови редактори, с файлов формат и се качва и хоства на сървъри, които са онлайн в интернет или са часр от мрежата. Тези .html файлове съдържат програмно на таговете на HTML и текстово съдържание с маркери и коментари. За да се илюстрира как се включва текст в HTML код, уеб браузърите са програмирани от своя страна така, в повечето случаи,макар 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. Основното предимство на HTML е, че уеб страниците, които го вкл‘чват в кода си, могат да се разглеждат чрез показването им от браузъра на екрана на повечето устройства. HTML може да прикрепя скриптове писани ная 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 ще през 1997г. Това допринася за разделяне на съдържанието и структурата на уеб страниците от тяхното визуално представяне. Създаването на HTML базирана уеб страниза може да се извърши с помощта на обикновен текстов редактор. Този начин изисква добро познаване на HTML тагове, така че те да бъдат в логическа последователнос, и в тях да бъдат интегрирани текстови сегменти, или текстът, който ще се показа на страницата.

Примерен HTML код:

< ! DOCTYPE html >

< html >

< head >

< meta charset = “windows - 1251”>

## Python

Python е език за програмиране от високо ниво, интерпретативен, интерактивен, обектно ориентиран, създаден от Гуидо ван Росум в началото на 90-те гофини. Python предлага добра структура и поддръжка за разработка на големи приложения. Той притежава вградени сложни типове данни като гъвкави масиви и речници, за които биха били необходими дни, за да се напишат ефикасно. Python позволява разделянето на една програма на модули, които могат да се използват отново в други програми. Също така притежава голям избор набор от стандартни модули, които да се използват като основа на програмите. Поради факта, че Python е интерпретативен език, се спестява значително време за разработка, тъй като не са необходими компилиране и свързване за тестването на дадено приложение. Всеки модул на Python се компилира преди изпълнение до код за съответната виртуална машина. Този код се записва за повторна употреба като .pyc файл. Програмите, написани на Python, са доста компактни и четими, като често те са и по – кратки от своите еквивалентни, написани на C/C++. Това е така, понеже:

* наличните сложни типове данни позволяват изразяването на сложни действия с един единствен оператор;
* групирането на изразите се извършва чрез отстъп, вместо чрез начални и крайни скоби или някакви ключови думи;
* не са необходими декларации на променливи или аргументи;

Това е изходния код на една примерна програма на Python, която отпечатва на екрана:Hello Python!

# ! /user/bin/python

Print (‘Hello Python!’)

Python използва вдлъбнатини за интервал, вместо къдрави скоби или ключови думи, за да разграничи блокове. Увеличаването на отстъпа идва след определени твърдения; намаляването на отстъпа означава края на текущия блок. По този начин визуалната структура на програмата точно представя нейната семантична структура. Тази функция понякога се нарича правило за засада. Някои други езики използват отстъп по този начин; но в повечето случаи отстъпът няма семантично значение. Препоръчителният размер на отстъпа е четири интервала.

Оператори и изрази:

* Операторът за присвояване, използващ един знак за равенство =
* Операторът if, който условно изпълнява блок от код, заедно с else и elif
* Операторът for, който итерира интерируем обект, улавяйки всеки елемент в локална променлива за използване от прикачения блок
* Инструкцията while, която изпълнява блок от код, докато условието му е вярно
* Операторът try, който позволява изключения, повдигнати в неговия прикачен кодов блок, да бъдат уловени и обработени от except клаузи
* Инструкцията raise , използвана за повдигане на определено изключение или повторно повдигане на уловено изключение
* Инструкцията def, която дефинира функция или метод
* Операторът break , който излиза от цикъл
* Операторът continue, който пропуска остатъка от текущата итерация и продължава със следващата
* Операторът pass , служещ като NOP, синтактично необходимо за създаване на празен кодов блок
* Операторът return, използван за връщане на стойност от функция
* Инструкциите import и from , използвани за импортиране на модули, чиито функции или променливи могат да се използват в текущата програма.

## CSS

CSS е език за описание на стилове – използва се основно за описание на онлайн представянето на уеббазиран документ, който написан на език за маркиране. Най-често се използва допълнително към чистия HTML, но се прилага и върху XML уебстраници и документи. Спецификата на CSS официално се поддържа от W3C. CSS още в началото на развитието на www започва да се добавя към стандартния HTML с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995г., съдържанието на сайтове и стила на техния дизайн са описани в една и съща HTML страница. В резултат на това HTML кодът се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един-единствен файл и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл. CSS позволява да се определя как да изглеждат елементите на една HTML страница – шрифове, размери, цветове, фонове и други. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, последван от свойства и стойности. Например в следния CSS код: p { font – size: 9pt ; } има едно правило. То се състои от селектора Р и свойството font-size, на което е зададена стойност 9pt.

### Видове селектори:

В CSS има няколко вида селектори:

**Селектори за класове** – с помощта на CSS може да създавате собствени класове за всеки един селектор. След дефиниране на класа, то всички селектори, на които е просвоен този клас, ще показват един и същ ефект. Използва се следния систаксис: селектор.клас {атрибут: стойност}

Пример:

Нека имаме клас h1 със син цвят и клас параграфи р, позиционирани вляво.

Името на класа е произволно, но за удобство ще използваме blue. Следователно целия код ще изглежда така:

h1.blue {color: #0000ff}

p.left {text-align: left}

С това класовете са дефинирани и трябва да се поместят в секцията head на основната HTML страница, като се затворят между таговете <style> , </style>:

<style type = “text / css” >

H1.blue {color: #0000ff}

p.left {text - align: left}

</style>

Може да се въведете и параграф без декларирания клас:

<p id = ‘rb’ Хубавото вино е червено, тъмно червено </p>

### Групиране на селектори

Няколко селектора могат да бъдат подредени, като се отделят със запетаи, и им се зададе еднакъв ефект, т.е. атрибути с еднакви стойности. Синтаксисът при групирането е: селектор1, селектор2, селектор3,……{атрибут: стойност}

Пример:

<head>

<title> Няколко ефекта на един ред код </title>

<meta http-equiv = Content – Type

Content = “text/html; charset = windows-1251” />

<style type=”text/css”>

H1, h2, h3, p , del {color: #ff0000}

</ style >

</head >

## JS

JS е интерпретируем език за програмиране, разпространен с повечето уеб браузъри. Поддръж обектно ориентиран и функционален стил на програмиране. Създаден е през 1995г. Най – често се прилага към HTML кода на интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се използва също за писане на сървърни скриптове JSON, както и за много други приложения. JavaScript не трябва да се бърка с Java, съвпадението на имената е резултат от маркетингово решение на Netscape. Функциите на JavaScript могат да се свържат със събития на страници. JavaScript е най – широко разпространеният език за програмиране в интернет. Прието е JavaScript да се наричат скриптове. JavaScript може да влияе на почти всяка част от браузъра. Браузъра изпълнява JavaScript кода в цикъла на събития т.е. като резултат от действия на потребителя или събитието в браузъра (например document.onLoad).

Основни задачи в повечето JavaScript приложения са:

* Зареждане на данни чрез AJAX.
* Ефекти с изображения и HML елементи: скриване/показване, пренареждане, влачене, слайд шоу, анимация и много други.
* Управление на прозорци и рамки.
* Разпознаване възможностите на браузъра.
* Използване на камерата и микрофона.

Функцията document.write не се препоръчва, защото може да има неочаквани резултати. В следващия пример използваме един от чистите начини да вмъкнем HTML таг:

< script >

Document . body . innerHTML +=

< h1 >Здравей!<h1.’’ ;

</ script >

JavaScript е слабо типизиран, което означава, че определени типове са имплицитно преодразувани в зависимост от използваната операция.

* Двойчниат + оператор преобразува и двата операнда в низ, освен ако и двата операнда не са числа. Това е така, защото операторът за добавяне се удвоява като оператор за конкатенация
* Двойчния – оператор винаги преобразува и двата операнда в число
* И двата унарни оператора(+, -) винаги преобразуват операнда в число

Стойностите се преобразуват в низове като следното:

* Низовете са оставени както са
* Числата се преобразуват в тяхното низово представяне
* Елементите на масивите се преобразуват в низове, след което се съединяват със запетаи(,)
* Други обекти се преобразуват в низа, където е името на обекта

Стойностите се преобразуват в числа чрез преобразуване в низове и след това преобразуване на низовете в числа. Тези процеси могат да бъдат модифицирани чрез дефиниране toString и valueOf функции върху прототипа съответно за преобразуване на низ и число. Често се споменава и {} + {} резултатът от 0(число). Това е подвеждащо: {} се интерпретира като празен кодов облак вместо празен обект и празния масив се образува в число от оставащия унарен + оператор. Изразът е обвит в скоби. Фигурните скоби се интерпретират като празен обект и резълтатът от израза е

“[object Object]” както се очаква.

## MySQL

MySQL е система за управление на бази данни (СУБД), която се използва широко за съхранение и управление на данни в уеб приложения, софтуерни приложения и други. Тя позволява на потребителите да създават, изтриват, променят и извличат данни от бази данни посредством структурирани заявки на езика SQL(Structured Query Language). MySQL е отворена система за управление на бази данни, което означава, че е безплатна за използване и с отворен код, като се предлага както в свободна, така и в комерсиална версия. Освен базовите функции за съхранение и управление на данни, можем да научим още за MySQL, че включва модифициране и управление на бази данни и техните таблици. Критиците смятат, че популярността на MySQL е учудваща при наличието на други проекти на бази данни с отворен код и със сходна производителност и по – близо до SQL стандартите. Защитниците на MySQL отговарят, че програмата върши очакваното от потребителите, желаещи да приемат нейните ограничения в замяна на скорост и бърза разработка. Има 3 вида на MySQL сървър:

1. Стандартна: MySQL – стандартна е препоръчителна за повечето от потребителите и включва InnoDB хранилище.

2. Максимална (Мах): (това не е MaxDB, която е разработвана в сътрудничество със SAP) mysqld - max Extended MySQL Server. MySQL – максималната включва допълнителни възможности, които могат да не бъдат достатъчно тествани и не са необходими обекновено.

3. MySQL – Debug е компилирана по начин, по който се улеснява трасировката на грешки, но не е предназначена за повседневно използване, защото наличието на допълнителни инструкции може да намали производителността.

MySQL се предлага в две различни издания:

MySQL Community Server с отворен код и патентован Enterprise Server. MySQL Enterprise Server се отлочава с поредица от патентовани разширения, които се инсталират като сървърни плъгини, но иначе споделя системата за номериране на версиите и е изграден от същата кодова база.

Основни функции налични в MySQL 5.6:

* Широко подмножество на ANSI SQL 99, както и разширения.
* Поддръжка на пазлични платформи
* Изгледи с възможност за актуализиране
* Информационна схема
* Схема за ефективност, която събира и обощава статическите данни за изпълнението на сървъра и производителността на заявките за целите на наблюдението.
* Набот от опции за SQL Mode за контролиране на поведението по време на изпълнение, включително строг режим за по – добро спазване на SQL стандартите.
* X/Open XA поддръжка на разпределена обработка на транзакции; двуфазен ангажимент като част от това, използвайки машината за съхранение на InnoDB по подразбиране.

Вградена поддръжка за репликация:

* Асинхронна репликация: главен- подчинен от един главен към много подчинени или много главни към един подчинен
* Полусинхронна репликация : репликация от главен към подчинен, където главния чака репликация
* Синхронна репликация : Мулти – главна репликация се предоставя на MySQL Cluster
* Виртуален синхрон : Самоуправляваните групи от MySQL сървъри с поддръжка на множество главни може да се направи с помощта на: Galera Cluster или вградения плъгин за репликация на групи
* Индексиране и търсене в пълен текст
* Вградена библиотека с бази данни
* Поддръжка на Unicode
* Разделени таблици с подрязване на дялове в оптимизатора
* Споделено нищо клъстериране чрез MySQL Cluster

## JINJA

JINJA е шаблонизатор за Python, който се използва за генериране на текст базиран на шаблони. Той позволява вграждане на изрази, контролни структури и променливи в текстови шаблони. Jinja шаблонизаторър също може да послужи за генериране на диманичен текст, като позволява вмъкване на данни и логика в текстови шаблони. Той се използва широко в уеб разработката за генериране на HTML, CSS и други типове файлове. Механизмът за шаблони Jinja позволява персонализиране на тагове, филтри, текстове и други. Освен това, за разлика от Django шаблонен двигател, Jinja позволява на дизайнера на шаблони да извиква функции с аргументи на обекти.

### Характеристика:

Някои от функциите на Jinja са:

* изпълнение в пясъчна среда
* автоматично екраниране на HTML за предотвратяване на атаки със скриптове между сайтове(XSS).
* наследяване на шаблони
* компилиране до оптималния код на Python точно навреме
* незадължително предварително компилиране на шаблон
* конфигурируем синтаксис

### Синтаксис:

Синтаксисът за отпечатване на изход в Jinja използва двойните фигурни скоби, например { { Hello, World! } }.

Подобно на горното,коментарите в Jinja могат да бъдат написани с хаштаг(#), вместо (%), например, {# helpful comment #).

Синтаксисът за създаване на тест в Jinja е ключовата дума, is както и условия за оценка на валидността на тест, като например { % if variable is divisibleby 10 % } do somethink { % endif % }

Циклите for могат да се използват за интериране на последователности, като същевременно се запазват техните обектни свойства. Следващият пример демонстрира повторение на списък с потребители с username и password полета.

{ % for user in users % }

{ { user.username } }

{ { user.password } }

{ % endfor % }

Въпреки, че break и continue не са разрешени вътре в цикли, последователностите могат да бъдат филтрирани

Ето малък пример за шаблонен файл

< ! DOCTYPE html >

< html >

< head >

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

< / head >

< body >

{ % for item in item\_list % }

{ { item } } { % if not loop. Last % } , { % endif % }

{ % - endfor % }

< / body >

< / html >

## REST

REST (Representational State Transfer) е стил софтуерна архитектура за реализация на уеб услуги. Основната идея е да се определи системен ресурс, който се променя в резултат на взаймодействието между доставчика на услуги и потребителя. Архитектурният модул REST включва взаймодействията между сървър и клиент, осъществени по време на трансфера на данни. Концепцията беше въведена за първи път от Рой Филдинг през 2000г. като част от неговата докторска дисертация. Филдинг е един от соновните автори на НТТР протокола, под който се изпълнява REST имплементациите в повечето случаи. Архитектурата REST е разработена успоредно с НТТР1.1. Въпреки това REST е обща архитектура, която може да бъде реализирана в други среди, а не само под НТТР. World Wide Web представлява най – голямото осъществяване на архитектурния стил на REST.

REST – стилът обикновено се състои от сървъри и клиенти. Клиентите инициират заявки към сървърите; сървърите преработват заявките и връщат подходящи отговори. Заявките и отговорите са създадени през прехвърляне на образа на ресурси. Ресурсът може да бъде всякаква ясна и смислена комцепция, която може да бъде адресирана. Представяне на ресурс обикновено е документ, който намира сегашното възнамерявано състояние на ресурса. Клиентът започва да изпраща заявки, когато е готов да направи преходът към ново състояние.Докато една или повече заявки са неизпълнени за клиента се смята, че е в преход. Представянето на всяко приложение се състои от линкове, които могат да бъдат използвани следващия път, когато клиентът избере да направи нови социални промени.

RESTful уеб API (също наричано RESTful уеб service) е уеб приложение, което използва принципите на HTTP и REST. Представлява колекция от ресурси с четири дефинирани аспекта.

* Internet media типът на данните поддържани от уеб приложенията. Това най – често е JSON, но може да бъде всеки друг валиден Интернет медиен тип, като се има предвид, че е валиден хипертекст стандарт.
* Операциите поддържани от уеб приложението използвайки HTTP методи(примерно: GET, PUT, POST или DELETE)
* Приложенията трябва да се задвижват от хипертекста.

## Django

Django е софтуерна рамка за създаване на уеб приложения на програмния език Python.

Системата е свободен софтуер, разработван от доброволци под егидата на регистрираната в САЩ. Базира се на шаблона Model-View-Template. Основното предназначение на Джанго е улесняване създаването на сложни уеб – системи, почти винаги включващи бази данни. Джанго предлага гъвкав ORM за моделиране и завършен административен интерфейс за управление на съдържанието. Въпреки, че има своя собствена номенклатура, като наименование на изкикваемите обекти, генериращи „изгледи“ на HTTP отговорите, основната рамка на Джанго може да се разглежда като MVC архитектура. Съсътои се от обектно – релационен картограф, който посредничи между модели на данни и релационна база данн, система за обработка на HTTP заявка със система за уеб шаблони и базиран на регулярен израз URL диспечер.

Също така в основната рамка са включени:

* лек и самостоятелен уеб сървър за разработка и тестване
* система за сериализиране и валидиране на формуляри, която може да превежда между HTML формуляри и стойности, подходящи за сънранение в базата данни
* шаблонна система, която използва концепция за наследяване, заимствана от обектно – ориентираното програмиране
* рамка за кеширане, която може да използва жсеки от няколко метода за кеширане
* поддръжка за класове на междинен софтуер, които могат да се наместват на различни етапи от обработката на заявката и да изпълняват персонализирани функции
* система за интернационализация, включително преводи на собствениците на Джанго на различни езици

Основната дистрибуция на Джанго също включеа редица приложения в своя пакет включително:

* разширяема система за удостоверяване
* диманичен административен интерфейс
* инструменти за генериране на RSS и Atom синдикирани емисии
* рамка „Сайтове“, която позволява на една инсталация на Джанго да стартира множество уебсайтове, всеки със собствено съдържание и приложения
* инструменти за генериране Google Sitemap
* вградено смекчаване на фалшифициране на заявка между сайтове, скриптове между сайтове, SQL инжектиране, кракване на пароли и други типични уеб атаки, повечето от които за включени по подразбиране.
* рамка за създаване на ГИС приложения

Конфигурационната система на Django позволява кодът на трета страна да бъде включен в обикновен проект, при условие че следва конвенциите за многократно използвано приложение. Повече от 5000 пакета са достъпни за разширяване на първоначалното поведение на рамката, предоставяйки решение на проблеми, с които оригиналният инструмент не се е занимавал: регистрация, търсене, предоставяне и потребление на API, CMS и др.

Django може да се използва във връзка с Apache, Nginx, използвайки WSGI, Gunicorn или Cherokee, използвайки flup (модул на Python). Django също така включва възможност за стартиране на FastCGI сървър, което позволява използване зад всеки уеб сървър, който поддържа FastCGI, като Lighttpd или Hiawatha. Възможно е също така да използва други уеб сървъри, съвместими с WSGI. Django официално поддържа пет бекенда на бази данни: PostgreSQL, MySQL, MariaDB, SQLite, Oracle. Microsoft SQL Server може да се използва с Django-mssql, докато подобни външни бекенд съществуват за IBM Db2, SQL Anywhere и Firebird.

Django може да се изпълнява заедно с Python на всеки Java EE сървър за приложение като GlassFish или JBoss. В този случай трябва да се инсталира Django-python , за да се осигурят JDBC драйвери за свързване на бази данни, което също може да предостави функционалност за компилиране на Django в .war, подходящ за внедряване.

## Vue

Vue.js е преден край на JavaScript – библиотека с отворен код модел – изглед – модел на изглед за изграждане на потребителски интерфейси и приложения с една страница. Създаден е от Evan You и се поддържа от него и останалите активни членове на оснивния екип.

### Компоненти:

Компонентите на Vue разширяват основните HTML елементи, за да капсулират код за многократна употреба. На високо ниво компонентите са персонализирани елементи, към които компилаторът Vue продава поведение. Във Vue компонентът е по същество екземпляр на Vue с предварително дефинирани опции. Кодовият фрагмент по – долу съдържа пример за Vue компонент. Компонентът представя бутон и отпечатва броя на кликванията върху бутона:

< template >

< div id = “tuto” >

### Шаблони

Vue използва базиран на HTML синтаксис на шаблон, койъо позволява обвързване на изобразения DOM към данните на основния екземпляр на Vue. Всички Vue шаблони са валиден HTML , който може да бъде анализиран от съвместими със спецификации браузъри и HTML анализатори. Vue компилира шаблоните във виртуални DOM функции за изобразяване. Виртуален обектен модел на документ или ‚DOM‘ позволява на Vue да визуализира компоненти в паметта си, преди да актуализира браузъра. В комбинация със системата за реактивност, Vue може изчисли минималното количество DOM манипулации, когато състоянието на приложението се промени.

Потребителоте на Vue могат да използват синтаксис на шаблон или да изберат директно да пишат функции за рендиране, използвайки хиперскрипт чрез извикване на функции.

### Реактивност

Vue разполага със система за реактивност, която използва обикновени JavaScript обекти и оптимизирано повторно изобразяване. Всеки компонент следи своите реактивни зависимости по време на изобразяването си, така че системата знае точно кога да изобрази отново и кои компоненти.

### Преходи

Vue предоставя различни начини за прилагане на ефекти на преход, когато елементи се вмъкват, актуализират или премахват от DOM. Това включва инструменти за:

* Автентично прилагайте класове за CSS преходи и анимации
* Използвайте JavaScript , за да манипулирате директно DOM по време на куките за преход
* Интегрирайте JavaScript аминационни библиотеки на трите страни, като Velosity.js

Когато елемент, обвит в преходен компонент бъде вмъкнат или премахнат, това се случва:

* Vue автоматично ще надуши дали целевия елемент има приложени CSS преходи или анимации. Ако това стане, класовете за преход на CSS ще бъдат добавени/премахнати в подходящи моменти.
* Ако компонентът на прехода предостави JavaScript кукички, тези кукички ще бъдат извикани в подходящ момент.
* Ако не бъдат открити CSS преходи/анимации и не са предоставени JavaScript кукички, DOM операциите за вмъкване и/или измъкване ще бъдат изпълнени незабавно на следващия кадър.

## API

Интерфейсът за приложно програмиране е начин две или повече компютърни програми или компоненти да комуникират помежду си. Това е вид софтуерен интерфейс, предлагащ услуга на други части от софтуера. Документ или стандарт, който описва как да се изгради или използва такава връзка или интерфейс, се нарича API спецификация. Казва се, че компютърна система, която отговаря на този стандарт, внедрява или излага API. Терминът API може да се отнася или до спецификация, или до изпълнението. Докато потребителския интерфейс на системата диктува как нейните крайни потребители взаймодействат с въпросната система, нейният API диктува как да се пише код, който се използва от възможностите на тази система.

За разлика от потребителския интерфейс, който свързва компютър с човек, интерфейсът за приложно програмиране свързва компютри или софтуерни части един с друг. Той не е предназначен да се използва дитектно от лице, различно от компютърен специалист, който го включва в софтуера. API често се състои от различни части, които действат като инструменти или услуги, които са достъпни за програмиста. Казва се, че програма или програмист, който използва една от тези части, извиква тази част от API. Извикванията, които съставят API, са известни също като подпрограми, методи, заявки или крайни точки. Спецификацията на API дефинира тези повиквания, което означава, че обяснавакак да ги използвате или реализирате.

Една от целите на API е да скрие вътрешните подробности за това как работи системата, като разкрива само онеза части, които програмистът ще намери за полезни, и ги поддържа последователни, дори ако вътрешните подробности се променят по – късно. Един API може да бъде създаден по поръчка за определена двойка системи или може да бъде споделен стандарт, позволяващ оперативна съвместимост между много системи.

Има API за езици за програмиране, софтуерни библиотеки, компютърни операционни системи и компютърен хардуер. Приложните програмни интерфейси(API) възникват през 40-те години на миналия век, но терминът се появява едва през 60-те и 70-те години на миналия век. Съвременната употреба на термина API често се отнася до уеб API, които позволяват комуникация между компютри, които са свързани чрез интернет. Последните разработки на API доведоха до повишаване на популярността на микроуслугите, които са слабо свързани услуги, достъпни чрез обществени API.

### Библиотеки и рамки

Интерфейсът към софтуерна библиотека е един тип API. API описва и предписва „очакваното поведение“(спецификация), докато библиотеката е „действително изпълнение“ на този набор от правила.

Един API може да има множество реализации(или нито една, тъй като е абстрактна) под формата на различни библиотеки, които споделят един и същ програмен интерфейс.

Определянето на API от неговата реализация може да позволи на програми, написани на един език, да използва библиотека, написана на друг. Например, тъй като Scala и Java се компилират в съвместим байт код, разработчиците на Scala могат да се възползват от всеки Java API.

API също може да бъде свързан със софтуерна рамка : рамката може да се основава на няколко библиотеки, прилагащи няколко API, но за разлика от нормалното използване на API, достъпът до поведението, вградено в рамката, се медиира чрез разширяване на нейното съдържание с нови класове включен в самата рамка.

Освен това цялостният прпграмен поток на контрол може да бъде извън контрола на повикващия и в ръцете на рамката чрез инверсия на контрола или подобен механизъм.

### Операционна система

API може да определя интерфейса между приложение и операционнта система. POXIS, например, предоставя набор от общи спецификации на API, коитп имат за цел да позволят приложение, написано за съвместимо с POXIS операционна система, да бъде компилирано за друга съвместима с POXIS API.

API се различава от двоичен интерфейс на приложение по това, че е базиран на изходния код, докато ABI е базиран на двоичен код. Например POXIS предоставя API, докато Linux Standard Base предоставя ABI.

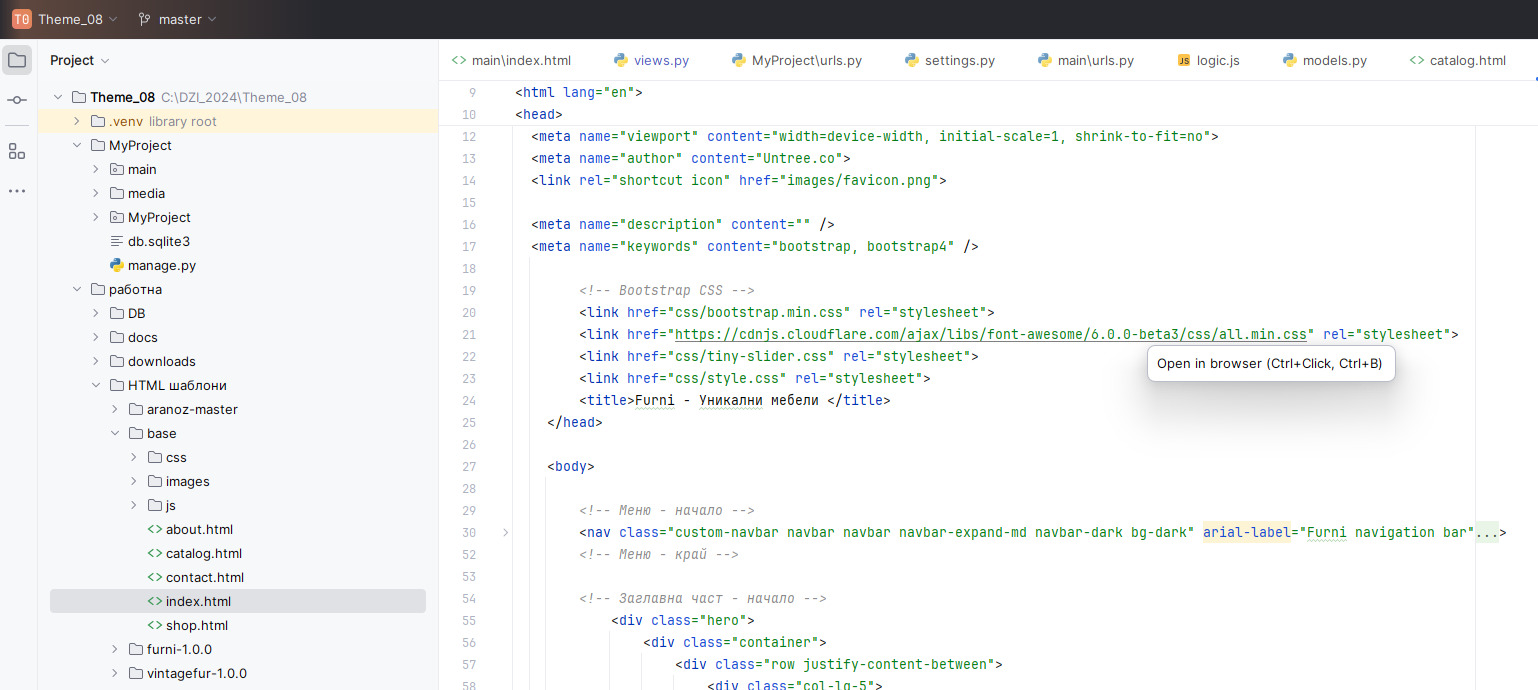
# Етапи на проекта

## Анализ на заданието

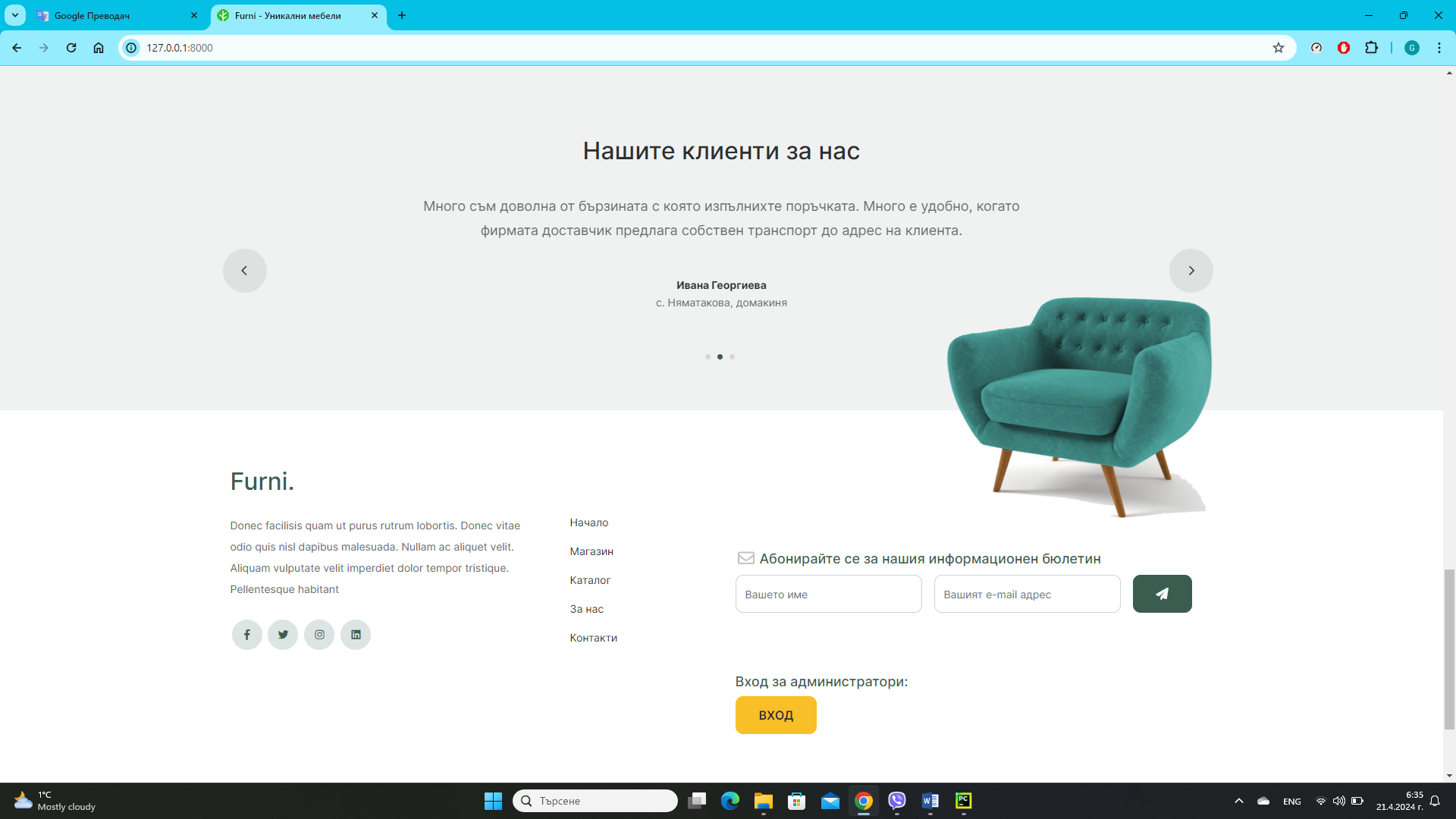
Анализирам означава чета го внимателно, гледам с какво разполагам и се опитвам да разбера кво всъщност трябва да постигна, т.е.определям си целите и задачите от една страна и сходните данни от друга страна. След това проучвам възможните решения, които ги има в света. Най – напред търсим аналогични решения . Например имаме да направим сайт за хотел и със сигурност има и други хотели и те евентуално имат сайтове. Правя система за оправление на складово стопанство. Обикновено не съм от първите и единствените и затова гледам другите хора какво са направили, но това не означава, че трябва да ги копирам едно към едно , но е правилно да се поуча от чуждия опит най – малкото да имам база за сравнение, да видя кое ми подхожда и кое неи. Следващото, което е мисля си за технологии, които реализират това. Примерно гледам и викам, че масово хората пишат на PHP, използват framework, масово правят динамични файлове, ми и аз ще правя диманични сайтове затова чета, чета, чета и викам да де ама сега напоследък пишат на python и то не просто на python а използват и Django. От там и аз започвам да гледам що е то python дали става за нас, гледам що е то Django как се инсталира, какви плюсове има, какви минуси, но всъщност се сещам , че само python и Django не стигат, липсва и нещо друго и тогава си казвам да направя динамичен интерфей. Сега е модерно да има програмиране и от страна на клиента, трислойни модели на разработените приложения, чета, чета, чета, и всичко това се нарича теортично изследване на проблема. И това всъщност е огромната част от същността на теоретичната част на дипломния проект. Изследвам проблема какво е подходящо да се ползва от другите хора, защо го ползвам. Трябва да търся нещо подходящо на моя проект и си гледам плюсове и минуси и в края на краищата правя избор.

## Избор на конктерна технология

Чуди ла съм се, мислила съм, имала съм много възможностино в крайна сметка немога всички. Избирам едно. Избрала съм технология, следващото, което трябва да направя е да тръгна да направя моя избор. За да приложа моя избор съобразявайки се с технологията, аз първо трябва вече ясно да очертая функционалните елементи на приложението, което правя. Всъщност аз винаги правя приложение, независимо от това дали е сайт или нещо друго. Тръгвам да реализирам приложението от изчистване на функциите, които ще реализирам. При това даже има такава стратегия по Софтуерно инженерство, която се нарича MVP стратегия. Затова тръгвам на най – важните неща. Първо правя минимално приложение, представям си функциите, и това как ще изглежда и си правя така наречения макет. Макета е визия на отделните стъпки във функциониранетона приложението.



Създаването на макет спестява много труд по отношение на реализация, защото се изчиства функционалността, махат се излишни функции. Започвам фактическата реализация вече като приложни програмисти.

До тук бях софтуерен архитект, след това бях софтуерен инженер а сега съм програмист. Започвам с бегенда. Инсталирам Django, създавам Django проект, след това създавам приложениe main в проекта и го регистрирам в settings.py.

*# Application definition*INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'main',  
 'rest\_framework',  
]

Създавам папка templates, в нея прехвърлям всички .html файлове от макета и папка static - в нея прехвърлям всички статични елементи. Това са .js , .css , и другите статични ресурси за шаблоните. Водя се от макета, който съм създала. Създавам и папката media, в която ще се съхраняват динамично зарежданите изображения (графични файлове). Указвам път до тази папка в настройките на Django:

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC\_URL = 'static/'

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media/')

Необходимо е във файла urls.py да се укаже и съответният път за проследяване на заявките към тези ресурси:

if settings.DEBUG:  
 urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

В самите шаблони трябва да се настроят съответните връзки. За целта във всеки шаблон, в самото му начало се поставя директивата load static

<!DOCTYPE html>  
{% load static %}  
<html lang="en">  
<head>

След което за всеки локален статичен ресурс с командата static се посочва неговит URL относително папката static:

<meta name="keywords" content="bootstrap, bootstrap4" />  
  
 *<!-- Bootstrap CSS -->* <link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">  
 <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css" rel="stylesheet">  
 <link href="{% static 'css/tiny-slider.css' %}" rel="stylesheet">  
 <link href="{% static 'css/style.css' %}" rel="stylesheet">  
 <title>Furni - Уникални мебели </title>  
</head>  
  
<body>

Т.к. хедъра и футера на всички страници са почти еднакви, прилагам идеята за разширяване на базов шаблон. Базовият шаблон съдържа в себе си общите компоненти. Разлките при извикване от различни страници се предават като контекст на извикващият изглед:

def index(request):  
 items = Item.objects.order\_by('-hits')  
 reviews = Review.objects.all()  
 menu = {  
 "home": 'active',  
 "shop": '',  
 "catalog": '',  
 "about": '',  
 "contact": '',  
 }  
  
 context = {  
 'page\_title': 'Съвременни мебели с уникален дизайн',  
 'page\_subtitle': 'Актуален дизайн, високо качество на изработката, качествени материали.',  
 'items': items,  
 'reviews': reviews,  
 'menu\_status': menu,  
 }  
 return render(request, 'main/index.html', context)

Разширяващит шаблон задължително започава с указване на базовия шаблон:

{% extends 'main/base.html' %}  
{% load static %}  
{% block page\_content %}  
 *<!-- Секция дизай и изработка - начало -->* <div class="product-section">  
 <div class="container">  
 <div class="row">

## Клиентски интерфейс

Естественият ми избор всъщност е Django и не просто Django а същата концепция на Rest, при която освен визуална информация към сървъра и клиента се обменя просто данни. Защото това приложние след един по внимателен анализ е всъщност съчетание на сайт представящ фирмата и на некакъв макар и малък сайт за магазин, който пък е съвсем друга тема. От една страна не се опитвам да създам пълноценен електронен магазин с кошници, плащания и други но имам основните елементи на електронната търговия и в същото време имам всички елементи, които са характерни за един представящ фирма сайт. Трябва да имам Каталог, Магазин, За нас, Контакти. Секциата Каталог всъщност е интерактивно приложение. Ако отида в моето приложение и погледна шаблоните, виждам, че за всяка страница си имам шаблон. Но ако се вгледам по подробно, страницата примерно about, страницата index, тоест началната, страницата shop магазин, те всичките наследяват една обща страница, която по същество представлява класически Django подход. Наследявам файла baсe.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 Магазин там е почти същото като Начало. До тук е Django такова каквото е , заради това, което го обичат неговите потребители. От тук нататък влизам в диманичния клиентски интерфейс. В моя проект има една страница, която не следва описанта по-горе схема. Това е шаблонът catalog.html. В него шаблонизатора на Django се маскира за сървъра с команда (логическите скоби verbatim – endverbatim) и се предава всичко едно към едно към клиента, което позволява на шаблонизатора View той да обработи въпросния шаблон и той да го раздвижи.

<body>  
 <script type="text/javascript"> window.CSRF\_TOKEN = "{{ csrf\_token }}"; </script>  
 {% verbatim %}  
 <div id="app">

....

</div>  
*<!-- Финализиране - край -->  
  
<!-- Футер - начало -->*<div id="s4" v-if="(status == 0)">  
{% endverbatim %}  
 <footer class="footer-section">  
 <div class="container relative">  
  
 <div class="sofa-img">  
 <img src="{% static 'images/sofa.png' %}" alt="Image" class="img-fluid">  
 </div>

....

Тоест поддаваме необходимата разработка, за да може да се реализира от страна на клиента.

Тази реализация предполага първо да VUE.js и разработения за него конкретен JavaScript код – файл logic.js:

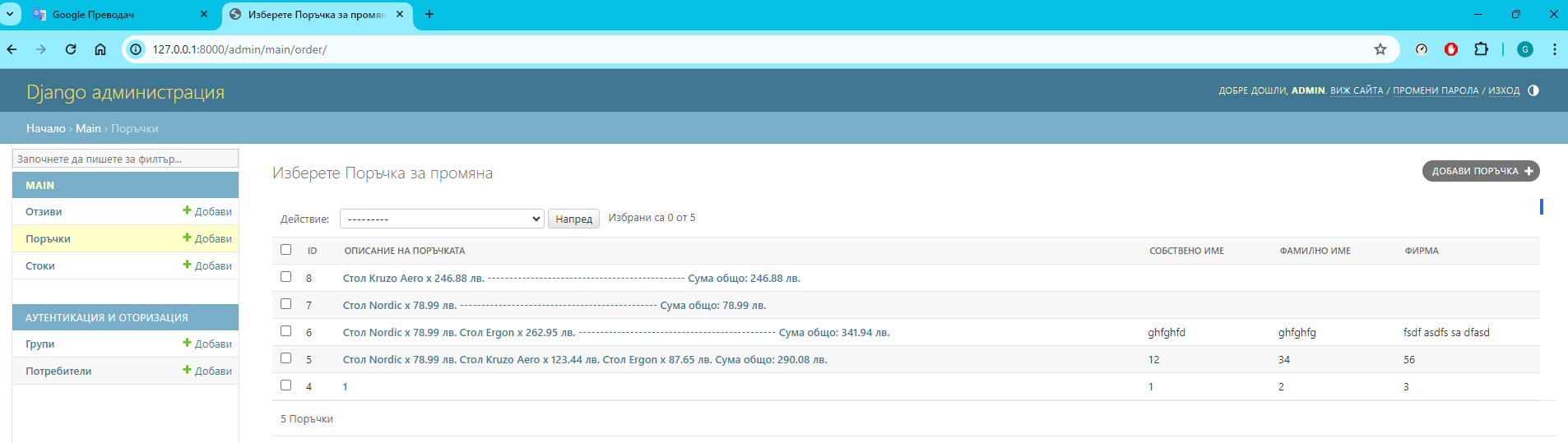
*<!-- включване на VUE js -->* <script src="https://unpkg.com/vue@3"></script>  
 <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>  
 <script src="{% static 'js/logic.js' %}"></script>  
  
</body>

## Администативен интерфейс

Административният интерфейс е „подарък“ от Django. Единственото, което се иска е във файла admin.py да си регистрирам тези таблици от базата данни, които искам администратора да управлява, създава, наблюдава и т.н.

from django.contrib import admin  
from .models import Item, Order, Review  
  
  
@admin.register(Item)  
class ImportantV(admin.ModelAdmin):  
 list\_display = ('id', 'name', 'price', 'hits')  
 list\_display\_links = ('name', )  
  
  
@admin.register(Review)  
class ImportantV(admin.ModelAdmin):  
 list\_display = ('id', 'name', 'txt', )  
 list\_display\_links = ('name', )  
  
  
@admin.register(Order)  
class ImportantV(admin.ModelAdmin):  
 list\_display = ('id', 'order\_details', 'first\_name', 'last\_name', 'company', )  
 list\_display\_links = ('order\_details', )

Като тук обаче съм избрала един разширен начин за регистриране на таблици. Не показвам всички полета а си избирам с опцията list\_display кои полета да покажа в изгледа на администраривния панел и си избирам кое поле да е активно и като цъкна върху него да ми отвори детайли.



Това не е задължително, но е полезно и често се използва при изграждане на администраторски интерфейси, особено когато се предполага, че той ще бъде често посещаван. По същество тази страница Каталог е еднодукоментна. Имам също и серизлизатор като той е един единствен. Това е изискване на библиотеката DRF. Имам файл models, който описва моделите я базата данни.

# Заключение:

Разработеният сайт за поръчки на мебели онлайн може да представлява голяма стъпка напред за бизнеса, като предоставя на клиентите по – добра и по – удобна пазарна среда, допринасяйки за разширяване на клиентската база и увеличаване на продажбите. Потребителите могат да разлеждат и поръчват мебели от вкъщи или от където и да е другаде, където имат интернет достъп. Това предоставя голямо удобство, като се избягват дългите пътувания до магазини и често разочароващите изживявания, свързани с липсата на налични артикули в магазина. Онлайн магазинът може да предложи по-голямо разнообразие от мебели в сравнение с физическия магазин, поради по-малките пространствени ограничения. Потребителите могат да избират между различни стилове, размери и цветове на мебелите. Може да има по-ниски разходи за операции в сравнение с традиционния физически магазин, което може да доведе до по-конкурентни цени за клиентите. Освен това, създаването на онлайн платформа премахва географските ограничения, позволявайки на магазина да достигне по-голяма аудитория, гъвкавост и скорост на обслужване, подобрена комуникация по-добро обслужване на клиентите.

# Списък с използвана литература:

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