Съдържание

[1. Увод 3](#_Toc164568043)

[2. Какво е Python. 3](#_Toc164568044)

[3. Библиотека DRF - (Django REST Framework) 5](#_Toc164568045)

[Характеристики 5](#_Toc164568046)

[4. Django 6](#_Toc164568047)

[5. HTML, CSS, JS 10](#_Toc164568048)

[5.1 Структура на HTML 11](#_Toc164568049)

[5.2 CSS 13](#_Toc164568050)

[5.3 JavaScript 14](#_Toc164568051)

[6. REST 15](#_Toc164568052)

[6.1 Обзор 15](#_Toc164568053)

[6.2 Изграждане на RESTful API. 17](#_Toc164568054)

[6.3 Сериализатори и изгледи (view) 18](#_Toc164568055)

[7. VUE 19](#_Toc164568056)

[8. Jinja. 24](#_Toc164568057)

[9. XAMPP: 25](#_Toc164568058)

[9.1 Нужда от XAMPP: 26](#_Toc164568059)

[9.2 Предимства на XAMPP: 26](#_Toc164568060)

[9.3 Компоненти на XAMPP: 26](#_Toc164568061)

[9.4 Приложения на XAMPP: 27](#_Toc164568062)

[10. Реализация 28](#_Toc164568063)

[10.1 Анализ на заданието 28](#_Toc164568064)

[10.2 База данни и Модели 28](#_Toc164568065)

[10.3 Елемент от JAVASCRIPT кода 29](#_Toc164568066)

[10.4 HTML шаблони 30](#_Toc164568067)

[11. Заключение: 32](#_Toc164568068)

[12. Литература: 33](#_Toc164568069)

# Увод

В света на Интернет има огромно количество сайтове, насоччени в най различни предметни области. И една от най-развиващите се групи е електроннта търговия. Един специфичен, но все почесто срещан вид уеб-базирани приложения са сайтовете за резервации. Това по същество също е електронна търговия, но продаваната стока е определен вид услуга. Такава е и моята тема - разработване на система за резервации на автобусни билети през интернет. Пътниците трябва да могат да проверяват разписанията на автобусите, да научат повече за автобусите и да запитват за места в автобуса. На примера на една въображаема фирма за автобусни превози, която иска да създаде онлайн система за резервация на билети за нейните линии, съм се опитал да покажа как сравнително лесно може да се разработи такова онлайн приложение. Използвал съм възможностите на едни от най-популярните и всъщото време най-мощните софтуерни инструменти – фреймуърка Django и езика за програмиране Python.

# Какво е Python.

Python е един от най-популярните програмни езици в света и представлява отличен начин за начинаещи да се научат да програмират. Python е език от високо ниво, интерпретируем, обектно-ориентиран програмен език. Той е лесен за усвояване, съчетавайки чист синтаксис с голяма функционалност. В Python можем да създаваме променливи, които пазят стойности. Променливите не се декларират с определен тип, тъй като Python е динамично типизиран език. Някои от основните типове данни включват числа (цели, дробни), символни низове и булеви стойности. Той е разпространен в уеб разработката, научните изследвания, разработката на софтуер и други области. Условните оператори (if, else, elif) позволяват да вземате решения въз основа на определени условия. Те са мощен инструмент за управление на изпълнението на програмата в зависимост от състоянието на променливи или външни . Циклите (for и while) са структури, които ви позволяват да повтаряте изпълнението на блок от код няколко пъти. Това е полезно, когато искате да изпълните еднакъв код за множество стойности или докато определено условие е вярно.Списъците са структури, които позволяват да съхранявате последователност от стойности в Python. Речниците са асоциативни структури, които позволяват да свързвате ключове със стойности. Тези структури са полезни при работата с големи набори от данни или при организиране на информацията по определен начин. Функциите са блокове от код, които могат да бъдат използвани повторно в програмата. Те се дефинират с ключовата дума "def" и могат да приемат параметри и да връщат стойности. Използването на функции улеснява четимостта и поддръжката на програмния код. Python има голямо разнообразие от модули и библиотеки, които предоставят допълнителна функционалност. Обработката на изключения позволява на програмата да се справя с грешки по време на изпълнение, които могат да се случат. Python позволява разделянето на една програма на модули, които могат да се използват отново в други програми. Също така притежава голям набор от стандартни модули, които да се използват като основа на програмите. Съществуват и вградени модули, които обезпечават такива събития и елементи като файлов вход/изход (*I/O*), различни системни функции, сокети (*sockets*), програмни интерфейси към GUI библиотеки като Tk и поради факта, че Python е интерпретативен език, се спестява значително време за разработка, тъй като не са необходими компилиране и свързване  (*linking*) за тестването на дадено приложение. Програмната му идеология е сходна с тази на Java и всяко приложение, написано на него, е сравнително лесно преносимо и в други платформи. Програмите, написани на Python, са доста компактни и четими, като често те са и по-кратки от своите еквиваленти, 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 (друг език, използващ такъв начин на подредба, е Haskell);

Не са необходими декларации на променливи или аргументи; Python съдържа прости конструкции, характерни за функционалния стил на програмиране, които му придават допълнителна гъвкавост. Всеки модул на Python се компилира преди изпълнение до код за съответната виртуална машина. Езикът е строго типизиран (strong typing) – при несъответствие между типовете е необходимо изрично конвертиране. Езикът поддържа и динамична типизация (dynamic typing) – типовете на данните се определят по време на изпълнението. Работата се основава на принципа duck typing – типът на обектите се оценява според техните свойства.

Поддържа се garbage collector – вътрешната реализация на езика се грижи за управлението на паметта. Блоковете се формират посредством отстъп. Като разграничител между програмните фрагменти се използва нов ред. 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.

# Библиотека DRF - (Django REST Framework)

Библиотека DRF - (Django REST Framework) е библиотека за Python и Django, която предоставя инструменти за лесно създаване на RESTful API. DRF предлага множество възможности за конфигуриране и персонализиране на API-то, както и за автоматично генериране на документация.

DRF е с отворен код и има голямо и активно общество от разработчици. Библиотеката е лесна за употреба и предоставя множество функции за управление на автентикацията, разрешаване на достъпа, сериализация на данни и други.

Характеристики:

* **Гъвкавост** - DRF позволява да конфигурирате API-то по различни начини в зависимост от вашите нужди. Можете да използвате различни сериализатори, аутентикационни методи и маршрутизатори, за да създадете точно това, което ви трябва.
* **Сериализация на данни** - DRF предоставя мощен механизъм за сериализация на данни, който може да преобразува сложни Python обекти в JSON, XML и други формати, които могат да бъдат изпратени към клиентските устройства.
* **Поддръжка на аутентификация и авторизация** - DRF предлага набор от вградени методи за аутентификация и авторизация, които могат да се използват за защита на API-то. Можете да използвате JWT, OAuth2, BasicAuthentication и други методи за аутентификация.
* **Документация** - DRF може автоматично да генерира документация за API-то, която може да се визуализира в браузъра. Това прави API-то лесно за използване и разбиране.
* **Поддръжка на формати** - DRF поддържа множество формати за обмен на данни, включително JSON, XML, YAML и други.

Модел-изглед-контролер архитектура - DRF използва модел-изглед-контролер (MVC) архитектура, която разделя логиката за данни, бизнес логиката и управлението на потребителския интерфейс. Това позволява по-добра организация на кода и по-лесно пренасочване на заявките към подходящия изглед. Класове на изгледите - DRF предоставя множество готови класове на изгледите, които могат да се използват за създаване на API-то. Тези класове на изгледите предоставят много от функциите, които са необходими за изграждане на RESTful API, като например CRUD операции (Create, Read, Update, Delete), валидация на данни и други. Поддръжка на множество формати - DRF може да работи с множество формати за обмен на данни, включително JSON, XML, HTML, CSV, YAML и други. Това прави API-то лесно достъпно за различни клиентски устройства. Автоматично генериране на документация - DRF може да генерира автоматична документация за API-то, която може да бъде достъпна в браузъра. Това прави API-то лесно за използване и разбиране, като се предоставят информации за всички налични ендпойнти, параметри, възможни статус кодове и други. Поддръжка на многонишковост - DRF поддържа многонишковост, което позволява да се обработват множество заявки едновременно. Това води до подобрена производителност и по-добро управление на заявките. Поддръжка на различни бази данни - DRF може да работи с различни бази данни, включително PostgreSQL, MySQL, SQLite и други. Това дава възможност за гъвкавост в избора на базата данни. Разширяемост - DRF е много разширяема библиотека, която може да се допълва с множество плъгини и разширения, които предоставят допълнителни функционалности.

# Django

Django е безплатна и отворена уеб рамка, написана на Python. Уеб рамката е набор от компоненти, които ви помагат да развивате уеб сайтове по-бързо и по-лесно. Когато създавате уебсайт, винаги се нуждаем от подобен набор от компоненти: начин за справяне с удостоверяването на потребителя (регистрация, влизане, излизане), панел за управление на вашия уебсайт, формуляри, начин за качване на файлове . За наше щастие, други хора отдавна забелязаха, че уеб разработчиците са изправени пред подобни проблеми при изграждането на нов сайт, така че те се обединиха и създадоха рамки (Django е една от тях), които ви дават готови компоненти за използване. Рамките съществуват за да ви спестяват нуждата да преоткривате колелото и да помогнат за облекчаване на част от разходи, когато създавате нов сайт. За да разберем за какво всъщност е Django, трябва да разгледаме по-отблизо сървърите. Първото нещо е, че сървърът трябва да знае, че искате той да ви обслужва уеб страница. Представете си пощенска кутия (порт), която се следи за входящи писма (заявки). Това се прави от уеб сървър. Уеб сървърът чете писмото и след това изпраща отговор с уеб страница. Но когато искате да изпратите нещо, трябва да имате някакво съдържание. А Django е нещо, което ви помага да създавате съдържанието. Kогато заявката стигне до уеб сървър, тя се предава на Django, който се опитва да разбере какво всъщност се иска. Първо взема адреса на уеб страница и се опитва да разбере какво да прави. Тази част се прави от **urlresolver** на Django (обърнете внимание, че адресът на уебсайт се нарича URL - Uniform Resource Locator - така че името urlresolver има смисъл). Не е много умно - взема списък от модели и се опитва да съответства на URL адреса. Django проверява шаблоните отгоре надолу и ако нещо съвпада, Django предава заявката на свързаната функция (която се нарича view).

Представете си пощенски превозвач с писмо. Тя върви по улицата и проверява всеки домашен номер спрямо този на писмото. Ако съвпада, тя поставя писмото там. Ето как работи urlresolver!

Във функцията view се правят всички интересни неща: можем да разгледаме база данни, за да потърсим някаква информация. Може би потребителят иска да промени нещо в данните? Като писмо, в което се казва: „Моля, променете описанието на моята работа“. view може да провери дали ви е позволено да го направите, след това актуализира описанието на работата за вас и изпраща обратно съобщение: „Готово!“ Тогава view генерира отговор и Django може да го изпрати до уеб браузъра на потребителя.

Описанието по-горе е малко опростено, но все още не е необходимо да знаете всички технически неща. Наличието на обща идея е достатъчно.

Така че вместо да се гмурнем твърде много в детайли, ще започнем да създаваме нещо с Django и ще научим всички важни части по пътя!

Admin Panel (Django) - Админ панелът на Django (Django admin panel) е вграден административен интерфейс, който позволява управление на данните във вашия Django проект. Той ви позволява да създавате, редактирате и триете данни от базата данни без да е необходимо да пишете ръчен SQL код.

За да използвате админ панела на Django, трябва да създадете администраторски акаунт. Това може да се направи чрез команда от конзолата на проекта:

След като създадете администраторски акаунт, можете да влезете в админ панела чрез браузъра си на адрес ‘http://адрес-на-вашия-сайт/admin/’ и да въведете вашият администраторски потребител и парола.

Админ панелът на Django има много полезни функции като търсене, филтриране, сортиране, групиране и множество други. Можете да настроите изгледа на данните за да отговаря на вашите нужди и да добавите/премахнете колони по ваш избор.

Освен това, админ панелът позволява да се правят масови операции, като изтриване на множество записи едновременно. Също така, вие можете да добавите допълнителни функционалности чрез създаване на собствени изгледи за админ панела, като използвате функционалностите на Django.

Админ панелът на Django е много полезен инструмент за администриране на вашия уеб сайт, който може да ви спести много време и усилия в управлението на данните във вашето приложение.

Административният панел (admin panel) на Django е вграден административен интерфейс, който предоставя удобен начин за управление на данните във вашия Django уеб сайт. Той е изграден с помощта на Django framework и предоставя автоматичен интерфейс за добавяне, редактиране и изтриване на данни от моделите на вашата база данни.

За да използвате административния панел на Django, трябва да дефинирате моделите на вашата база данни във файловете на приложението си. Тези модели определят структурата на вашите данни и съдържат логиката за достъп до тях.

Административният панел на Django предлага множество функции и възможности за управление на вашите данни, като:

* Избор на модел за управление
* Добавяне, редактиране и изтриване на обекти
* Филтриране и търсене на обекти
* Изглед на детайли за определен обект
* Създаване на свързани обекти
* Управление на потребителите и групите, имащи достъп до административния панел

Административният панел може да бъде персонализиран и разширен по множество начини, като се използва Django Admin site API. Можете да създадете свои собствени изгледи, форми и действия, които да бъдат вградени в административния панел, за да отговарят на нуждите на вашето приложение.

За да се използва административният панел на Django, трябва да се дефинират модели на данните във вашия проект. Моделите представляват структура на данните, които ще бъдат запазени в базата данни. С помощта на административния панел можете да изпълнявате CRUD операции (Create, Read, Update, Delete) върху тези данни.

За да активирате административния панел в Django, трябва да включите приложението ‘django.contrib.admin’ във вашия ‘INSTALLED\_APPS’ списък във файла ‘settings.py’ на вашия проект. След като приложението е включено, можете да дефинирате административния интерфейс чрез дефиниране на класове наследяващи ‘django.contrib.admin.ModelAdmin’.

Пример за дефиниране на административен интерфейс за модел ‘Product’:

from django.contrib import admin

from .models import Product

class ProductAdmin(admin.ModelAdmin):

list\_display = ('id', 'name', 'price', 'created\_at')

list\_filter = ('created\_at',)

search\_fields = ('name',)

admin.site.register(Product, ProductAdmin)

В горния пример, ‘ProductAdmin’ е клас, който наследява ‘admin.ModelAdmin’. Вътре в класа са дефинирани опции за административния интерфейс като ‘list\_display, list\_filter и search\_fields’.

‘list\_display’ определя списъка с полета, които ще бъдат показани за всяка запис в административния списък. ‘list\_filter’ позволява да се филтрират записите по определени полета, а ‘search\_fields’ позволява да се търсят записи по определени полета.

Накрая, като използвате ‘admin.site.register()’ метода, регистрирате модела ‘Product’ в административния панел, заедно със създадения ‘ProductAdmin’ клас.

В админ панела на Django могат да се извършват и други операции като търсене, филтриране и сортиране на данните, добавяне на нови потребители и групи, настройка на права за достъп до определени данни за потребителите, и много други.

Админ панелът може да се персонализира и разширява по различни начини, като се използват допълнителни пакети и модули. Освен това, може да се използва и на различни езици, включително на български.

За да използвате админ панела на Django на български език, трябва да настроите езиковите настройки на приложението във файловете settings.py и urls.py. В settings.py трябва да зададете настройката за езика на 'bg', а в urls.py трябва да добавите ред за включване на езиковите файлове:

from django.conf.urls.i18n import i18n\_patterns

from django.utils.translation import gettext\_lazy as \_

urlpatterns = [

# ... тук добавяте вашите url patterns ...

]

# Patterns за включване на езиковите файлове

urlpatterns += i18n\_patterns(

path('admin/', admin.site.urls),

prefix\_default\_language=False,)

# HTML, CSS, JS

**HTML *(HyperText Markup Language*)** – преведено език за маркиране на хипертекст, е език за кодиране, който казва на уеб браузъра как да показва съдържанието и структурата на уеб страницата. Както подсказва името, той се използва за форматиране или маркиране на обикновен текст с прекъсване на абзаци, получер шрифт или курсив, цветни шрифтове, списъци, таблици и много друго. В дефиницията на HTML „хипертекст“ означава, че текстът може да включва връзки към други ресурси в мрежата. Това е пример за текст, който не е форматиран с HTML5 код. Сега започвам да се запознавам с различните начини на форматиране: получерен шрифт, курсив, подчертан, цветен. Все още не знам как да започна нов абзац или да напиша списък от задачите си за деня: 1. Събуждане в 6°°. 2. Утринна гимнастика 15 мин. Ето това е пример за текст, който е форматиран с HTML5 код. Сега започвам да се запознавам с различните начини на форматиране: получерен шрифт, *курсив*, подчертан, цветен.

Познаването на основните тагове и използването им дава свобода на блогъра в работата му със съдържанието. Ето защо ни трябват познания по HTML: Разбирането на HTML може да ни помогне да научим как е структуриран нашият блог или уебсайт.

Познания по HTML кода ще ни помогнат да станем по-самостоятелни, по-уверени в себе си като блогъри и ще ни спести време. Ще можем да отстраняваме някои проблеми при форматирането на WordPress без нужда от странична помощ. Ще оформяме някои части на страницата си по свое желание. Ще се справяме по-лесно със SEO – оптимизацията на нашият блог за търсачките.

## Структура на HTML

HTML елементите са ограничени от **тагове**, които могат да имат **атрибути** – например цвят, размер. Tаг за начало – определя началото на един HTML елемент – заграден в < и >;таг за край – определя края на HTML елемента. За разлика от началния таг, крайният има наклонена надясно черта пред името си – заграден в </ и >.В случая, когато е необходимо да се постави таг за край, не трябва да се забравя, за да се спре продължаването на определеното форматиране. Забележка: HTML таговете са контейнерите за различните видове HTML елементи. Те ограждат всяко съдържание на уеб страницата, в противен случай браузърът няма да знае как да го представи. Meta тагове <meta> таговете дефинират метаданни за HTML документ. Метаданните са данни (информация) за данни. Те се намират винаги в елемента <head> и обикновено се използват за задаване на набор от символи, описание на страница, ключови думи, автор на документа и настройки на прозореца на визуализиране. Особено важен за представянето в търсачката е атрибутът name със стойност description, който определя кратко описание на съдържанието на твоята уеб страница. Точно това описание най-често се използва от търсачките, за да запознае потребителя за какво се отнася статията. Таг title-Таг <title> определя заглавието на документа. Заглавието се показва в заглавната лента на браузъра или в раздела на страницата. Този таг е задължителен. Освен това е много важен за оптимизацията за търсачките (SEO). Съдържанието на елемента title се показва в търсачката и участва в класирането на уеб страниците при зададено търсене по ключова дума или фраза. Заглавия-Сред HTML елементите, които трябва да знаеш на всяка цена, е списъкът със заглавия, чрез които се организира структурата на уеб страницата. Основното заглавие се отбелязва с тага <h1>…</h1>.Заглавията на раздели в публикацията са в йерархия. Те се определят с таговете <h2>…</h2>,…,<h6>…</h6>, като тези тагове не са задължителни да присъстват всичките и <h6> дефинира най-маловажното заглавие. Форматиране на текст- За да добавиш курсив, го вмести между таговете <em> и </em>. За да добавиш удебелен текст (текст, който считаш за **важен**) в HTML документ, го обгради с таговете <strong> и </strong>. За да добавиш маркиран текст, го обгради с таговете <mark> и </mark>. За дефиниране на текст под черта се използва <sub> и </sub> таг. Параграфи- Тагове <p> и </p> които означават начало и край на абзац. Браузърите автоматично добавят един празен ред преди и след всеки елемент <p>. Този HTML елемент е ключов за структурата на една уеб страница.

Вероятно не е необходимо да използваш таговете за абзац, когато пишеш статиите си, защото на повечето платформи те автоматично се вмъкват в текста, който въвеждате. Сигурно ще се запиташ, защо ти трябва да знаеш този HTML елемент, щом като платформите сами го организират. Рано или късно нещо няма да ти хареса в оформянето на абзаците, в подравняването им. Тогава ще се наложи да действаш на ниво HTML. Таг(Pre)- Има и друга възможност за изписване на текст с вмъкнати празни редове – да се използва HTML елемента <pre>…</pre>:Тагът <pre> дефинира предварително форматиран текст (pre-formatted). Той ще се покаже точно, както е написан в изходния HTML код – с фиксирана големина на шрифта и запазвайки интервалите и прекъсванията на редове. Можеш да го използваш, за да покажеш примери за код, поезия или друг текст, в който са важни точното разстояние и прекъсванията на редовете. Основният недостатък на форматирания текст с помощта на <pre> е ширината. За разлика от нормалния HTML, той няма да се преоразмери, за да съответства на размера на прозореца на потребителя. Ще създаде хоризонтална лента за превъртане. Цитати - За означаване на текст, който е цитат, се използва елемента <blockquote>…</blockquote>. Той може да бъде форматиран по различен начин, използвайки CSS стил.

Връзки в текста (таг) <a>.Връзките в текста са изключително необходими за външни и вътрешни препратки, към различни части от същата страница, в целеви страници и дрyги. Трябва да познаваш добре таг <a>…</a>, за да вмъкваш нужните линкове. Този елемент е от особено значение за оптимизацията на публикациите за търсачката, осигурявайки чрез връзките необходимата допълнителна информация за обяснение на части от съдържанието. Елементът за връзка съдържа **anchor текст (котва),** състоящ се от една или повече думи, чрез който създаваш хипертекстовата връзка в документа. С този таг можеш да направиш възможно кликването върху изображение, в което да има връзка .**href** – атрибут, който указва дестинацията на връзката; **rel** – атрибут, който указва връзката между текущия документ и документа, към който е връзката. Използва се само ако присъства атрибут ***href***. target – атрибут, който указва къде да се отвори свързаният документ. Когато му се даде стойност target="\_blank", ще отвори свързания документ в нов прозорец или раздел. Тагът <a> с атрибут href е същия като основната връзка, с изключение, че е връзка към кодова дума (след id=), а не към URL адрес. Пред кодовата дума (в случая label5)се поставя знак **#**, за да се обозначи, че е вътрешна връзка. Без знака **#**браузърът ще търси извън страницата тази кодова дума .Вмъкване на изображения-В списъка с важните за един блогър HTML тагове, трябва да бъде поставен и таг <img>, който дефинира изображение в HTML страницата. <img> няма затварящ таг. Използва се винаги с поне два (задължителни) атрибута:  src и alt.***src*** – атрибутът служи за локализиране и извикване на permalink на медията – адреса на изображението; ***alt***– указва алтернативен текст на картинката, в случай че не се покаже визуалното изображение;

**Таг <code>-** Ако ни е необходимо да вмъкнем код в текста си, това става с <code>… </code> таг, който форматира текста с подходящ шрифт и например. Като алтернатива можем да използваме таг <pre>…</pre>, който ще покаже кода по различен начин от останалия текст. Неразбиващо пространство в HTML - Понякога в HTML на уеб страницата си можем да срещнем групата символи &nbsp; – non-breaking space. Обектът &nbsp; създава неразбиващо се пространство, което се използва, когато не искаме автоматично прекъсване на линията в тази позиция. Това е удобно, когато разбиването на думите e неприемливо.

## CSS

Какво е CSS JavaScript ? CSS e спецификация за стилове на елементите на една уеб страница. Той позволява разработчиците да определят стилове за цветове, шрифтове, размери, позиции и други атрибути на елементите. Това позволява програмистите да създаваме уеб страници, които са ефектни и изглеждат добре. Има няколко начина за прилагане на CSS на една уеб страница: a). Вграждане на CSS в HTML файла: Това се прави с добавяне на CSS код в тага <style> вътре в тага <head> на HTML документа. Този метод е подходящ, когато искаме да приложим стиловете само за една уеб страница. Използване на външен CSS файл: Създаваме CSS файл с разширение .css и дефиниране стиловете в него. След това трябва да вмъкнем файла към HTML документа с помощта на тага <link>. Този метод е подходящ, когато искаме да приложим стиловете за много уеб страници. Използване на inline CSS: Добавяме стиловете към style атрибутa на HTML елементите. В повечето браузъри може да инспектираме CSS като използваме developer tools функциите на самия browser. Най-популярните браузъри като Chrome, Firefox, Edge, и Safari имат добри инструменти за целта.

## JavaScript

JavaScript е програмен език, който позволява динамична промяна на поведението на браузъра в рамките на дадена HTML страницата. JavaScript се зарежда, интерпретира и изпълнява от уеб браузъра, който му осигурява достъп до Обектния модел на браузъра. JavaScript функции могат да се свържат със събития на страницата (например: движение/натискане на мишката, клавиатурата или елемент от страницата, и други потребителски действия). JavaScript е най-широко разпространеният език за програмиране в интернет. Прието е JavaScript програмите да се наричат скриптове. JavaScript може да влияе на почти всяка част от браузъра. Браузъра изпълнява JavaScript кода в [цикъла на събития](https://bg.wikipedia.org/w/index.php?title=Event_loop&action=edit&redlink=1) като резултат от действия на потребителя или събития в браузъра (например document.onLoad). JavaScript кодът може да се вмъква в HTML документа между двойката елементи <script> и</script>. Когато срещне тага <script>, браузърът разбира, че трябва да спре интерпретирането на HTML кода и да започне да обработва скрипта, намиращ се между <script> и </script>. Този скрипт не е задължително да бъде написан на JavaScript. Има и други езици за писане на скриптове, например VBScript. Но езикът по подразбиране е JavaScript. Другият начин да заредите JavaScript е да го поставите в отделен файл. Променливи в JavaScript са динамични и могат да съдържат стойности от всеки тип. Декларират се чрез ключовата дума var следвана от името на променливата или списък от променливи. Възможно е също да присвоим стойност на променливата при декларация чрез оператора =. По подразбиране стойността на променливата ще бъде undefined. Имената на променливите в JavaScript не трябва да започват с цифра, могат да започват с малка или голяма буква, \_ или $ и да съдържат само тези знаци както и числа. Всички останали символи са забранени. Името на променливата не може да е ключова дума или оператор. В JavaScript има два вида обхват за променливите: функционален и глобален. Функционален обхват важи за всяка променлива обявена чрез var в рамките на дадена функция. Тази променлива е видима единствено в рамките на тази функция. Глобален обхват важи за всяка променлива обявена извън функция или без var, дори и във функция. Тази променлива е видима от всеки друг код на страницата.

# REST

## Обзор

REST е стил софтуерна архитектура за реализация на уеб услуги. Основната идея е да се определи системен ресурс, който се променя в резултат на взаимодействието между доставчика на услуги и потребителя. Архитектурният модел REST включва взаимодействията между сървър и клиент, осъществени по време на трансфера на данни. Концепцията беше въведена за пръв път от Рой Филдинг през 2000 г. като част от неговата докторска дисертация. Филдинг е един от основните автори на HTTP протокола, под който се изпълняват REST имплементациите в повечето случаи.

Архитектурата REST е разработена успоредно с HTTP 1.1. Въпреки това, REST е обща архитектура, която може да бъде реализирана в други среди, а не само под HTTP. World Wide Web представлява най-голямото осъществяване на архитектурния стил на REST.

REST – стилът обикновено се състои от клиенти и сървъри. 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 концепция, която може да бъде адресирана. Представяне (анг. Representational) 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 официални промени.

REST е архитектурен стил за уеб приложения, който се основава на принципите на HTTP протокола. REST е акроним от английските думи "Representational State Transfer" (пренасяне на състоянието на ресурсите) и описва начина, по който клиентските приложения и уеб услуги могат да комуникират едно с друго чрез уеб протокола HTTP.

REST използва основни HTTP методи като GET, POST, PUT и DELETE, за да комуникира с ресурсите в уеб приложението. REST използва ресурси (например, потребители, статии, снимки) като централен елемент на архитектурата, като всяко ресурсно действие може да бъде извършено чрез HTTP заявка към този ресурс.

REST е популярен в уеб разработката, защото е лесен за разбиране и използване. RESTful API-та (уеб услуги, които използват REST) са съвместими с голям брой клиентски устройства и програми, като мобилни приложения, уеб браузъри, IoT устройства и други.

Важно е да се отбележи, че REST е само един от множество архитектурни стилове за уеб приложения, като всеки стил има своите предимства и недостатъци. Затова, при избора на архитектурен стил, е важно да се вземат предвид нуждите на проекта и целите на приложението.

Ето някои от основните принципи на REST, които помагат за изграждане на ефективни уеб приложения:

Ресурсно ориентирано програмиране - REST се базира на използването на ресурси като централен елемент на архитектурата. Всяко действие в уеб приложението може да бъде извършено чрез HTTP заявка към определен ресурс.

Използване на HTTP методи - REST използва основните HTTP методи като GET, POST, PUT и DELETE, за да комуникира с ресурсите в уеб приложението. Например, GET заявката се използва за извличане на информация от ресурс, а POST заявката се използва за добавяне на информация към ресурс.

Използване на URL адреси - REST използва URL адреси за идентифициране на ресурсите в уеб приложението. URL адресът на ресурса трябва да бъде постоянен и лесен за разбиране.

Предаване на данни във формат JSON или XML - REST използва форматиране на данните като JSON или XML, за да осигури ефективно предаване на информацията между клиента и уеб услугата.

Независимост от състоянието - REST е "безсъстоятелен" архитектурен стил, което означава, че всеки HTTP заявка към ресурса е независима от предишните заявки. Това прави REST уеб услугите много скалируеми и лесни за управление.

Restful Api- RESTful API е мощен инструмент за изграждане на уеб приложения, които могат да бъдат използвани от различни клиентски устройства. Важно е да следвате принципите на REST и да изградите API-то така, че да бъде лесно за използване и поддръжка. RESTful API може да бъде изграден с различни програмни езици и frameworks, включително Django, Express и други. Важно е да се вземат предвид нуждите на проекта и уменията на екипа при избора на конкретен frameworks за изграждане на RESTful API.

## Изграждане на RESTful API.

* Използвайте версии - когато правите промени в RESTful API, е важно да използвате версии, за да не нарушавате съвместимостта със старите клиенти. Можете да добавите номер на версията в URL адреса на API-то, например /api/v1/...
* Използвайте HTTP статус кодове - HTTP статус кодовете са важни за указване на резултата от изпълнението на заявката към RESTful API. Например, 200 означава успешна заявка, 400 означава грешка в заявката, а 404 означава, че ресурсът не е намерен.
* Използвайте HTTP хедъри - HTTP хедърите могат да бъдат използвани за предаване на допълнителна информация за заявката или отговора. Например, може да използвате хедър за авторизация, който съдържа токен за достъп.
* Използвайте кеширане - кеширането може да ускори отговора на RESTful API, като се избягва излишното извличане на информация от базата данни. Можете да използвате HTTP хедъра "Cache-Control" за указване на времето, за което резултатът на заявката може да бъде кеширан.
* Тествайте API-то - тестването на RESTful API е важна стъпка за увереност в правилната му работа. Можете да използвате инструменти като Postman или curl за изпращане на заявки към API-то и проверка на резултатите.

## Сериализатори и изгледи (view)

· Сериализаторите и изгледите (views) са ключови компоненти във фреймуърка Django за изграждане на уеб приложения. Ето как могат да бъдат обяснени:

Сериализаторът в Django се използва за преобразуване на сложни обекти (например модели на база данни) във формат, който може да бъде лесно използван за изпращане по мрежата. Това може да бъде JSON, XML или друг формат, който е необходим за приложението. Основната цел на сериализатора е да направи данните достъпни за приложенията, като ги преобразува в нужния формат за тях.

Изглед (View): Изгледът в Django е компонент, който обработва HTTP заявки и връща HTTP отговори. Той представлява логиката на приложението и извлича информация от базата данни, която е необходима за обработката на заявката. Изгледът може да извлича информация и от други източници, като например външни API-та или файлове. Основната цел на изгледа е да обработва заявките на потребителите и да връща нужните им отговори.

Сериализаторите и изгледите (views) в Django са мощни инструменти, които позволяват на разработчиците да създават гъвкави и мащабируеми уеб приложения. Сериализаторите са отговорни за преобразуването на обекти от Python във формат, който може да се предаде по мрежата, като например JSON или XML. Това е често използвано при създаването на RESTful API-та в Django. Сериализаторите позволяват на разработчиците да контролират точно какво ще бъде включено в изходящия формат, като допускат и вложени структури, които могат да бъдат свързани с моделите на базата данни.

Изгледите в Django са отговорни за обработката на HTTP заявки и генерирането на HTTP отговори, като HTML страници или JSON обекти. Изгледите предоставят множество функции за обработка на данните, включително филтриране, сортиране, пагинация и валидация. Django предлага няколко типа изгледи, включително функционални изгледи, клас-базирани изгледи и обобщени изгледи. В зависимост от нуждите на приложението, разработчиците могат да изберат подходящия тип изглед за реализацията на конкретна функционалност.

Сериализаторите и изгледите в Django могат да бъдат написани на български език, като е важно да се запази еднакъв стил на програмиране и да се спазват добрите практики

за оформление на кода. В Django също така съществуват и документация и ръководства на български език, които могат да бъдат полезни при работа с тези инструменти.

· Сериализатори и изгледи (view) са две ключови концепции в уеб разработката с Python и Django. Ето как може да ги обясним на български език:

**Сериализатори:**

Сериализаторите са класове в Django REST Framework (DRF), които трансформират данни от моделите в нашия Django проект във формат, който може да бъде изпратен към клиентите във формата на JSON, XML и други. С други думи, сериализаторите преобразуват данни от Python обекти във формат, който може да бъде използван от клиента. Също така, те позволяват да се валидират данните, които се изпращат от клиента към сървъра, като проверят дали данните отговарят на нужните условия.

**Изгледи (VIEW)**

В Django изгледите са функции или класове, които обработват HTTP заявки, получени от клиента. Те извличат или обработват данните, необходими за отговор на заявката и връщат HTTP отговор във формата на HTML, JSON, XML и други. Изгледите могат да бъдат класове или функции, като в последния случай се използва декоратора @api\_view, за да се конвертират функциите в изгледи, които могат да бъдат използвани в Django REST Framework.

# VUE

Vue е прогресивен фреймуърк, създаден за разработка на потребителски интерфейс (UI). В зависимост от целите на разработчиците, Vue може да бъде използван, както за фреймуърк, така и като библиотека от компоненти. Той се състои от главна библиотека, която се фокусира върху визията на апликациите и екосистема от допълнителни библиотеки, предоставящи на платформата множество функционалности.

Vue е създаден от Evan You като прогресивен JavaScript фреймуърк, а целта му е била да комбинира най-добрите функции на Angular и React.

Vue е предпочитан фреймуърк сред много от разработчиците по много причини. Той е лесен за научаване, има ясна структура и отлична документация, която го прави подходящ за използване, както от начинаещи, така и от напреднали, които лесно могат да преминат от React и Angular, към Vue. Той се основава на стандартните HTML, CSS и JavaScript и предоставя декларативен и базиран на компоненти програмен модел, който ви помага ефективно да разработвате потребителски интерфейси, независимо дали са прости или сложни. Използва се и от големи компании като Gitlab, Alibaba, Xiaomi, Adobe, Nintendo, Euronews и много други. · Декларативно изобразяване: Vue разширява стандартния HTML със синтаксис на шаблон, който ни позволява да опишем декларативно HTML изход въз основа на състоянието на JavaScript.

Vue автоматично проследява промените в състоянието на JavaScript и ефективно актуализира DOM, когато настъпят промени.

Vue е framework и екосистема, която покрива повечето от общите характеристики, необходими при разработката на интерфейса. Но мрежата е изключително разнообразна - нещата, които изграждаме в мрежата, могат драстично да се различават по форма и мащаб. Имайки това предвид, Vue е проектиран да бъде гъвкав и постепенно адаптируем. В зависимост от случая на употреба, Vue може да се използва по различни начини:

* Подобряване на статичен HTML без стъпка на изграждане
* Вграждане като уеб компоненти на всяка страница
* Едностранично приложение (SPA) Fullstack/Рендиране от страна на сървъра (SSR)
* Jamstack / Генериране на статичен сайт (SSG)
* Насочване към настолен компютър, мобилно устройство, WebGL и дори терминал

**Еднофайлови компоненти:**

В повечето Vue проекти с активиран инструмент за изграждане ние създаваме Vue компоненти, използвайки HTML-подобен файлов формат, наречен Single-File Component (известен също като \*.vue файлове, съкратено като SFC). Vue SFC, както подсказва името, капсулира логиката на компонента (JavaScript), шаблона (HTML) и стиловете (CSS) в един файл.

Vue е предпочитан фреймуърк сред много от разработчиците по много причини. Той е лесен за научаване, има ясна структура и отлична документация, която го прави подходящ за използване, както от начинаещи, така и от напреднали, които лесно могат да преминат от React и Angular, към Vue. Той се основава на стандартните HTML, CSS и JavaScript и предоставя декларативен и базиран на компоненти програмен модел, който ви помага ефективно да разработвате потребителски интерфейси, независимо дали са прости или сложни. Използва се и от големи компании като Gitlab, Alibaba, Xiaomi, Adobe, Nintendo, Euronews и много други.

* Декларативно изобразяване: Vue разширява стандартния HTML със синтаксис на шаблон, който ни позволява да опишем декларативно HTML изход въз основа на състоянието на JavaScript.
* Реактивност: Vue автоматично проследява промените в състоянието на JavaScript и ефективно актуализира DOM, когато настъпят промени.

Vue е framework и екосистема, която покрива повечето от общите характеристики, необходими при разработката на интерфейса. Но мрежата е изключително разнообразна - нещата, които изграждаме в мрежата, могат драстично да се различават по форма и мащаб. Имайки това предвид, Vue е проектиран да бъде гъвкав и постепенно адаптируем. В зависимост от случая на употреба, Vue може да се използва по различни начини:

* Подобряване на статичен HTML без стъпка на изграждане.
* Вграждане като уеб компоненти на всяка страница.
* Едностранично приложение (SPA) Fullstack/Рендиране от страна на сървъра (SSR).
* Jamstack / Генериране на статичен сайт (SSG).
* Насочване към настолен компютър, мобилно устройство, WebGL и дори терминал

В повечето Vue проекти с активиран инструмент за изграждане ние създаваме Vue компоненти, използвайки HTML-подобен файлов формат, наречен Single-File Component (известен също като \*.vue файлове, съкратено като SFC). Vue SFC, както подсказва името, капсулира логиката на компонента (JavaScript), шаблона (HTML) и стиловете (CSS) в един файл.

1. Компоненти :

Компонентите на Vue разширяват основните HTML елементи, за да запазят код за многократна употреба. На високо ниво компонентите са персонализирани елементи, към които компилаторът на Vue придава поведение. Компонентът е по същество екземпляр на Vue с предварително дефинирани опции.

1. Шаблони:

Vue използва базиран на HTML синтаксис на шаблон, който позволява обвързване на изобразения DOM към данните на основния екземпляр на Vue. Всички шаблони на Vue са валиден HTML, който може да бъде анализиран от съвместими със спецификацията 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 да изчисли минималния брой компоненти за повторно изобразяване и да приложи минималното количество DOM манипулации, когато състоянието на приложението се промени.

Потребителите на Vue могат да използват синтаксис на шаблон или да изберат директно да пишат функции за рендиране, използвайки хиперскрипт или чрез извикване на функции, или JSX. Функциите за изобразяване позволяват изграждането на приложения от софтуерни компоненти.

1. 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 да изобрази отново.

Предимства:

Vue.js има една от най-добре написаните документации, от която разработчиците могат да се учат бързо и структурирано. Това прави научаването и от начинаещите изключително лесно, в сравнение с React и Angular.

Разработчиците обичат да създават апликации с Vue, тъй като той им дава свобода и комфорт на работа, предлагайки всички компоненти, които биха били необходими.

Преизползване на кода:

Подходът, който Vue използва, базирайки се на компоненти, е вдъхновен от React. Кодът се пише под формата на компоненти, които могат да бъдат импортирани и преизползвани, когато и където е необходимо при създаването на апликациите.

Разработка на апликации на всички платформи:

Още едно предимство на Vue е неговата функция за създаване на приложения за всички платформи. Точно както при React, Vue позволява написването на един единствен код, който може да бъде използван на която и да е от съществуващите платформи и операционни системи.

Интеграции:

Разработчиците могат да интегрират Vue с други популярни структури, например React. Това им позволява да променят проекта, спрямо техните нужди и изисквания. Благодарение на тези интеграции, Vue става популярен избор за разработчиците, тъй като може да бъде използван с голяма част от съществуващите уеб апликации.

Ако back end-а е разработен с JavaScript, програмистите биха могли да използват тази технология при създаването на други JavaScript апликации.

Разнообразието от компоненти позволява създаването на различни типове уеб апликации, както и модифицирането на съществуващите структури.

Освен React, Vue може да използва HTML, благодарение на MVVM архитектурата, която позволява двустранна комуникация.

Недостатъци:

Въпреки всички предимства на Vue.js, той има и няколко недостатъка. Например, стабилността му е проблем още от обявяването на първата версия през 2015 година. Това означава, че платформата е чудесна при създаване на малки и лични проекти, но трябва да се внимава при работата с по-големи проекти, тъй като тази нестабилност, може да доведе до значителни загуби.

Екосистема на VUE:

Vue дава възможност за използване на множество инструменти и библиотеки, които значително улесняват разработката на апликациите.

Vue има богата екосистема от библиотеки, инструменти, както и свързани библиотеки за Vue JS UI компоненти и структури като Vuetify, Bootstrap Vue, Element, Quasar и др., сред които разработчиците могат да избират и да използват, спрямо нуждите на техните апликации.

# Jinja.

Jinja е мощен шаблонен език за Python, който позволява лесно и ефективно създаване на HTML, XML и други видове текстови изходи. Той е част от стандартната библиотека на фреймуърка за уеб програмиране Flask и е възможно да се използва с други уеб фреймуърци.

Jinja използва синтаксис, който наподобява на този на Django шаблоните. Един от основните принципи на Jinja е този да бъде "непрекъснат", т.е. няма да има код в шаблона, който да не може да се компилира и изпълни.

Пример :

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматичноВ този пример използваме Jinja за да изградим прости HTML шаблон. Използваме фигурните скоби **{{ }}** за да обозначим местата, където ще се поставят стойностите на променливите. В този случай променливите са **title**, **name** и **date**.

Ето някои от основните функции на Jinja:

* Условно изпълнение на код чрез конструкцията if-elif-else;
* Итерация през обекти чрез конструкцията for-in;
* Филтриране на данни чрез вградени филтри или допълнителни филтри, дефинирани от потребителя;
* Използване на макроси за повтаряеми блокове код;
* Използване на наследяване на шаблони за реализиране на шаблони на различни нива на вложеност;
* Интегриране на Python код в шаблоните чрез конструкцията {{ ... }} или {% ... %}.

Пример:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматичноВ този пример използваме наследяване на шаблони за да наследим основен шаблон **base.html**. Използваме цикъл **for-in** за да изпълним блока код за всеки елемент в списъка **users**, и печатаме съобщение, което съдържа името на всеки потребител.

# XAMPP:

XAMPP е крос-платформен уеб сървър, който е безплатен и с отворен код. XAMPP е кратка форма за крос-платформа, Apache, MySQL, PHP и Perl. XAMPP е популярен междуплатформен уеб сървър, който позволява на програмистите да пишат и тестват своя код на локален уеб сървър. Той е създаден от Apache Friends и обществеността може да ревизира или модифицира неговия собствен изходен код. Той включва MariaDB, Apache HTTP сървър и интерпретатори за PHP и Perl, наред с други компютърни езици. Поради простотата на разгръщане на XAMPP, разработчикът може бързо и лесно да инсталира WAMP или LAMP стек на операционна система, с допълнителното предимство, че могат да се зареждат и общи приложения за добавки като WordPress и Joomla.

## Нужда от XAMPP:

* XAMPP е просто локален хост или сървър.
* Този локален сървър работи на вашия персонален компютър, независимо дали е настолен или лаптоп.
* Използва се за тестване на клиенти или уебсайтове, преди да ги публикува на отдалечен уеб сървър.
* На локален компютър сървърният софтуер XAMPP осигурява подходяща среда за тестване на MYSQL, PHP, Apache и Perl проекти. Тъй като повечето внедрявания на уеб сървъри в реалния свят споделят същите компоненти като XAMPP, преминаването от локален тестов сървър към сървър на живо е лесно.

## Предимства на XAMPP:

В сравнение с други уеб сървъри като WAMP, той е лесен за настройка.

Той е Multi Cross-Platform, което предполага, че работи както на Windows, така и на Linux.

С една команда можете да стартирате и спирате целия стек на уеб сървъра и базата данни.

Налични са както пълна, така и стандартна версия на XAMPP.

Той има контролен панел, който можете да видите, че съдържа бутони за стартиране и спиране за специфични механизми, като например Apache, който работи през своя контролен панел.

Той също така включва OpenSSL, phpMyAdmin, MediaWiki, Joomla, WordPress и много допълнителни модули.

## Компоненти на XAMPP:

Компонентите, които са включени в XAMPP, са дадени по-долу:

* Крос-платформа: Различни операционни системи се инсталират в отделни конфигурации на различни локални системи. Компонентът за различни платформи е включен, за да подобри функционалността и обхвата на този пакет за разпространение на Apache. Работи с различни платформи, включително Windows, Linus и MAC OS пакети.
* Apache: Apache е междуплатформен HTTP уеб сървър. Използва се за транспортиране на уеб материали по целия свят. Ако някой поиска файлове, снимки или документи, използвайки своя браузър, HTTP сървърите ще обслужват такива активи на клиентите
* MariaDB база данни: XAMPP използва за включване на MySQL DBMS; обаче MariaDB вече зае мястото си. MySQL е една от най-широко използваните системи за управление на релационни бази данни. Той предоставя услуги за съхранение, манипулиране, извличане, управление и изтриване на данни чрез интернет.
* PHP: Пълната форма на PHP е Hypertext Preprocessor. PHP е backendезик за програмиране, който най-често се използва в уеб разработката. Потребителите могат да използват PHP за изграждане на динамични уебсайтове и приложения. Той поддържа различни системи за управление на бази данни и може да бъде инсталиран на всяка платформа. Написан е на езика за програмиране C.
* Perl: Perl често се нарича "общ" език за програмиране. Този език Perl е динамичен и интерпретируем. Този език се използва за уеб разработка, разработка на GUI, системна администрация и други неща. HTML, XML и други езици за маркиране се поддържат от Perl.
* phpMyAdmin: Това е инструмент за администриране на база данни за MariaDB.
* OpenSSL: OpenSSL е реализация с отворен код на SSL и TLP.
* Контролен панел на XAMPP: Контролният панел на XAMPP е панел, който подпомага работата и регулирането на други компоненти на XAMPP.
* Webalizer: Това е софтуерно решение за уеб анализ, което следи потребителски регистрационни файлове и отчети за използването.
* Mercury: Това е пощенски сървър, който помага при управлението на имейли в интернет.
* Tomcat: Това е JAVA-базиран сървър, който осигурява JAVA функционалност.
* Filezilla: Това е сървър за протокол за прехвърляне на файлове (FTP сървър), който улеснява и поддържа процесите на прехвърляне на файлове.

## Приложения на XAMPP:

* Създателите на XAMPP възнамеряват да се използва като инструмент за разработка, позволяващ на уеб дизайнерите и програмистите да тестват работата си на персоналните си компютри без нужда от интернет връзки. Много ключови елементи за сигурност са деактивирани по подразбиране, за да направи това възможно най-лесно. XAMPP се използва за обслужване на уеб страниците в Интернет.
* Може също да се използва за създаване и манипулиране на бази данни в MariaDB и SQLite, наред с други бази данни.
* След като XAMPP бъде инсталиран, FTP клиент може да се свърже с локален хост и да го третира като отдалечен хост. Когато инсталирате система за управление на съдържание като Joomla или WordPress, използвайте инструмент като FileZilla. Можете също да използвате HTML редактор, за да се свържете с локален хост чрез FTP.

# Реализация

## Анализ на заданието

Заданието за дипломния проект предполага, реализиране на софтуерна платформа за резервация на на автобусни билети през интернет. Тя трябва да предоставя информация за самите филми, за датата и часа на предстоящите прожекции и да позволява на потребителите онлайн да резервират билети за тях.. Платформата трябва да представлява уеб-базирано приложение, което реализира следната функционалност: панел „Касиер“, добавяне на линия, разписание, подробности за резервация, промяна на цените на билетите, създаване на резервация с възможност за избор на място в автобуса.

Приложението трябва да може да се администрира с помощта на администраторско меню (мениджърски панел), което да позволи добавяне, редактиране или премахване на линии и рейсове и определяне (задаване) на цени на билетите.

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

## База данни и Модели

За съхранение на данните съм избрал БД MySQL. Настройването на Дйанго да работи с тази база данни става в следните стъпки:

Инсталраме библиотеката mysqlclient.

Във файла settings.py, секциата databases се променя така:

DATABASES = {

'default': {

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

'NAME': 'dzi',

'USER': 'admin',

'PASSWORD': 'admin\_2024',

'HOST': 'localhost',

'PORT': '3306',

'OPTIONS': {'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'"}

}

}

Слоят за данни в Django е представен от файла models.py на приложението, който представя моделите които го свързват с базата данни. В конкретният случай съм избрал БД MySQL. Всеки модел в този файл е клас, който описва съответна таблица в БД. Например, моделът Line описва структурата на таблица, която ще се съхраняват данните за различните автобусни линии.

## Елемент от JAVASCRIPT кода

const App = {

data() {

return {

status: 0, // условeн номер на текущо визуализираната секция (или група секции)

categories: [],

items: [],

filter: {

... },

grid\_mode:3,

selected\_item:{},

selected\_item\_id:0,

cart: [],

...

loadCategories(){

vm = this

axios.get('/api/categories/')

.then(function(response){

vm.categories = response.data;

})

},

loadItems(){

vm = this

let url = 'api/items/'+this.filter.brand\_id+'/'+this.filter.model\_id+'/'+this.filter.cat\_id+'/'

axios.get(url)

.then(function(response){

vm.items = response.data;

this.filter.searching = false

})

},

},

created: function(){

this.status = 0

this.loadBrands()

this.loadCategories()

this.loadItems()

this.selected\_item=0

}

}

Vue.createApp(App).mount('#app')

Това е елемент от Javascript кода, който реализира функционалността на Vue, а Vue от своя страна е ядрото, което реализира клиентския интерфейс, което пък е темата на моя дипломен проект.

## HTML шаблони

Шаблоните са основата, върху кято се изграждат изгледите. В моят случай шаблоните са и връзката с VUE, т. е. те са визуалната част на динамичния потребителски интерфейс. Статичните ресурси се зареждат от страна на сървъра – команда сstatic.

{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta charset="utf-8">  
 <meta content="width=device-width, initial-scale=1.0" name="viewport">  
  
 <title>Кино ГЛОБУС</title>  
 <meta content="" name="description">  
 <meta content="" name="keywords">  
  
 *<!-- Favicons -->* <link href="{% static 'assets/img/favicon.png' %}" rel="icon">  
 <link href="{% static 'assets/img/apple-touch-icon.png' %}" rel="apple-touch-icon">  
  
   
 *<!-- Vendor CSS Files -->* <link href="{% static 'assets/vendor/aos/aos.css' %}" rel="stylesheet">  
 <link href="{% static 'assets/vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">  
 <link href="{% static 'assets/vendor/bootstrap-icons/bootstrap-icons.css' %}" rel="stylesheet">  
 <link href="{% static 'assets/vendor/boxicons/css/boxicons.min.css' %}" rel="stylesheet">  
 <link href="{% static 'assets/vendor/glightbox/css/glightbox.min.css' %}" rel="stylesheet">  
 <link href="{% static 'assets/vendor/remixicon/remixicon.css' %}" rel="stylesheet">  
 <link href="{% static 'assets/vendor/swiper/swiper-bundle.min.css' %}" rel="stylesheet">  
</head>

След което с команда verbatim изключваме обработката на шаблона от страна на сървъра и „предаваме“ управлението на клиентската страна – на шаблонизатора на VUE.

<body>  
 {% verbatim %}<div id="app">  
 …

<div class="modal-footer">  
 <button class="btn btn-danger" type="button" data-dismiss="modal" @click="reloadItem()">Затвори</button>  
 <button class="btn btn-success" type="button" data-dismiss="modal" @click="save()">&nbsp;Запази&nbsp;</button>  
 </div>  
 </div>  
 </div>  
 </div>

Интеграцията на Vue на ниво HTML.

<!-- 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 'assets/js/vue\_actions.js' %}"></script>

<!-- Main JS -->

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

Съчетавайки тези две неща се получава еднодокументен клиентски интерфейс, който е реализиран на Vue и черпи данни през DRF от сървърната част. От гледна точка на потребителя това е пълнофункционален интерфейс с различни страници

# Заключение:

Това е моят дипломен проект. Работата по него ме въведе в едина огомна, интересна и много переспективна област на приложното програмиране – уеб-базираните приложения. Надявам се и в бъдеще да имам още успешни задачи. Самата разработка е реално работещо приложение, тествано върху тестов сървър.

В разработката съм използвал 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 широко приложение.

# Литература:

1. Колисниченко, Денис, Адаптивен уеб дизайн с Bootstrap, Асеневци, С 2019
2. D.K Academy, Python – практическо програмиране. Асеневци, С 2022
3. Донълдсън, Тоби. Бързо ръководство:Програмиране с Python.АлексСофт, С 2017
4. Есканази, Аврам. Софтуерни техологии. КЛМН, С 2006
5. https://softuni.bg/blog/what-is-django
6. 2. https://cynoteck.com/bg/blog-post/flask-vs-django/
7. 3. https://bg.savtec.org/articles/coding/the-basics-of-rest-and-restful-api-development.html
8. 4. https://urocibg.eu/
9. 5. https://www.django-rest-framework.org/
10. 6. https://softuni.bg/blog/vue-js-january-2021
11. 7. https://bg.wikipedia.org/wiki/MySQL
12. 8. https://priobshti.se/article/strategii-v-pomosht-na-prepodavaneto/taksonomiya-na-blum-ili-kak-da-napravim-uroka-v
13. 9. https://html.w3schools.bg
14. 10. https://www.inventum.bg/web-design/css/