Съдържание

[1. Въведение 4](#_Toc164726729)

[2. Цел на проекта 5](#_Toc164726730)

[2.1. Основни цели: 5](#_Toc164726731)

[3. Характеристика 6](#_Toc164726732)

[4. Защо Python и Django са най-подходящи за проекта? 6](#_Toc164726733)

[4.1. Python 6](#_Toc164726734)

[4.2. Django 7](#_Toc164726735)

[5. Какво е REST? 8](#_Toc164726736)

[5.1. Какво е API? 8](#_Toc164726737)

[5.2. Какво е Rest API? 9](#_Toc164726738)

[5.3. Защо Django REST API? 10](#_Toc164726739)

[6. Библиотеката Django REST Framework 11](#_Toc164726740)

[6.1. Сериализация на данни: 11](#_Toc164726741)

[6.2. Компоненти за изграждане на API: 11](#_Toc164726742)

[6.3. Автентикация и авторизация: 12](#_Toc164726743)

[6.4. Генериране на документация: 12](#_Toc164726744)

[6.5. Широка поддръжка и общност: 12](#_Toc164726745)

[6.6. Заключение за DRF и REST......................................................................12](#_Toc164726746)

[6.7. VUE 13](#_Toc164726747)

[6.7.1. Какво е Vue.js? 13](#_Toc164726748)

[6.7.2. Еднофайлови компоненти ..................................................................14](#_Toc164726749)

[6.7.3. API стилове: 15](#_Toc164726750)

[6.7.4. API за опции: 15](#_Toc164726751)

[6.7.5. Композиционен API: 16](#_Toc164726752)

[6.8. Кое да изберем?​ 17](#_Toc164726753)

[6.9. Характеристика на VUE 17](#_Toc164726754)

[6.9.1. Компоненти: 17](#_Toc164726755)

[6.9.2. Шаблони: 19](#_Toc164726756)

[6.9.3. Реактивност: 19](#_Toc164726757)

[6.9.4. Преходи: 19](#_Toc164726758)

[6.9.5. Маршрутизиране: 20](#_Toc164726759)

[6.10. Предимства: 22](#_Toc164726760)

[6.10.1. Документация и разработка 22](#_Toc164726761)

[6.10.2. Преизползване на кода 22](#_Toc164726762)

[6.10.3. Разработка на апликации за всички платформи 22](#_Toc164726763)

[6.10.4. Интеграции 23](#_Toc164726764)

[6.10.5. Недостатъци: 23](#_Toc164726765)

[6.10.6. Екосистема на Vue: 23](#_Toc164726766)

[6.10.7. Библиотеки на Vue: 24](#_Toc164726767)

[7. Защо избираме базата данни MySQL? 24](#_Toc164726768)

[7.1. Какво са бази данни и как се управляват те? 24](#_Toc164726769)

[7.2. MySQL базите данни са релационни. 24](#_Toc164726770)

[7.3. SQL и MySQL едно и също нещо ли са? 25](#_Toc164726771)

[7.4. MySQL е софтуер с отворен код. 25](#_Toc164726772)

[8. HTML и CSS 25](#_Toc164726773)

[8.1. Какво означава HTML? 25](#_Toc164726774)

[8.2. Как работи HTML? 26](#_Toc164726775)

[8.3. Какво е HTML5 и какви са неговите предимства? 27](#_Toc164726776)

[8.4. Какво е CSS? 27](#_Toc164726777)

[8.5. Защо да изучаваш HTML и CSS? 27](#_Toc164726778)

[9. Реализация 28](#_Toc164726779)

[12.1 Първи стъпки на проекта: 28](#_Toc164726780)

[12.2 Създаване на приложение: 28](#_Toc164726781)

[9.1. Използване на шаблони: 29](#_Toc164726782)

[9.2. Настройване на Django за работа с БД MySQL: 29](#_Toc164726783)

[9.3. Настройване на MySQL база данни: 29](#_Toc164726784)

[9.4. Настройване на Django проекта: 30](#_Toc164726785)

[9.5. Миграции 30](#_Toc164726786)

[9.6. Създаване на моделите: 30](#_Toc164726787)

[9.7. Създаване на администратор: 31](#_Toc164726788)

[9.8. Регистриране на таблица: 31](#_Toc164726789)

[9.9. Извеждане на данни от БД: 31](#_Toc164726790)

[10. Заключение 32](#_Toc164726791)

[11. Списък на използваната литература 34](#_Toc164726792)

# Въведение

 В настоящата ера на информационни технологии и дигитално развитие, интернетът е превърнал библиотеките във виртуални пространства за достъп до знание и информация. Онлайн библиотеките предоставят удобен и бърз достъп до разнообразни източници на информация, което ги прави отлично решение за образователни и научни нужди. В този контекст, разработването на онлайн библиотека представлява значим и актуален проект, който отговаря на нуждите на съвременното общество за удобен и ефективен достъп до знание.

 Актуалността на проекта е още по-изразена с нарастването на обема на информацията в онлайн средата и увеличаването на интереса към онлайн образование и научни изследвания. Онлайн библиотеката представлява не само средство за съхранение и предоставяне на информация, но и инструмент за обогатяване на образователния и научния процес, като предлага разнообразни ресурси и инструменти за учене и изследване.

 Целта на настоящия дипломен проект е да се разработи функционален и ефективен софтуерен продукт – платформа за управление на библиотека, която да отговаря на нуждите на потребителите за бърз и удобен достъп до различни видове информация. Проектът има за цел да създаде виртуално пространство, където потребителите могат да намират, изследват и използват разнообразни материали за своите образователни и научни интереси.

 Чрез анализ на изискванията и нуждите на потребителите, изследване на съвременните технологии за уеб разработка и бази данни, проектиране и разработване на архитектурата и функционалностите на системата, тестване и оптимизация на софтуерния продукт, както и документиране и подготовка за внедряване и експлоатация, дипломният проект ще представи комплексен подход за създаване на онлайн библиотека, която да отговаря на високите изисквания на съвременното общество за достъпност и качество на информацията.

# Цел на проекта

 Целта на настоящия дипломен проект е да се разработи функционален и ефективен софтуерен продукт - онлайн библиотека, която да отговаря на нуждите на потребителите за бърз и удобен достъп до различни видове информация.

## Основни цели:

 Създаване на удобно и достъпно пространство за знание: Проектът има за цел да създаде виртуална среда, където потребителите могат лесно да намират и използват разнообразни материали за образователни и научни нужди.

 Подобряване на образователния и научния процес: Чрез предоставянето на разнообразни ресурси и инструменти за учене и изследване, проектът се стреми да подобри качеството на образователния и научния процес на потребителите.

 Улесняване на търсенето и навигацията: Онлайн библиотеката ще предоставя функционалности за търсене и навигация, които да улеснят потребителите в тяхната работа с информацията.

 Подобряване на достъпността и гъвкавостта: Чрез използване на съвременни технологии и методи за уеб разработка, проектът цели да осигури гъвкавост и достъпност на системата за широк кръг от потребители.

Оптимизация на функционалностите и производителността: Проектът ще се стреми да оптимизира функционалностите и производителността на онлайн библиотеката, за да осигури бърз и ефективен достъп до информацията.

Чрез постигането на тези цели, дипломният проект ще предложи иновативно и усъвършенствано решение за създаване на онлайн библиотека, която да отговаря на изискванията на съвременното общество за удобен и качествен достъп до знание и информация.

# Характеристика

 Проектът, който ще разработим, е онлайн библиотека, предоставяща уеб базирано приложение за достъп до разнообразни източници на информация. За постигане на тази цел, ще използваме следните технологии и инструменти: Python за създаване на бизнес логиката на приложението, Django като уеб фреймуърк за разработване на уеб приложението, MySQL като релационна база данни за съхранение на информацията, HTML за създаване на структурата на уеб страници, CSS за стилизиране на уеб интерфейса и JavaScript за добавяне на интерактивност и функционалности.

 В процеса на разработка ще създадем модели за данни, които ще отразяват структурата на библиотеката, изгледи за представяне на информацията на потребителите и контролери за обработка на заявките от тях. Също така ще приложим MVC архитектурата, която ще ни помогне да разделим логиката за данните, представянето и обработката на заявките, което ще улесни разработката и поддръжката на приложението.

 В процеса на разработка ще отделим внимание на оптимизиране на интерфейса и производителността на приложението. Чрез правилно използване на CSS и JavaScript ще подобрим визуалния дизайн и интерактивността на уеб страниците, а с оптимизиране на бизнес логиката ще гарантираме бърз и ефективен достъп до информацията за потребителите.

 Този подход ни позволява да създадем функционална и удобна за използване онлайн библиотека, която отговаря на изискванията на съвременните потребители за бърз и лесен достъп до знание и информация.

# Защо Python и Django са най-подходящи за проекта?

## Python

Python е високо ниво, интерпретиран, гъвкав и лесен за научаване програмен език, който се използва широко за уеб разработка, научни изчисления, изкуствен интелект, обработка на данни и други приложения. Ето няколко основни причини, поради които Python е идеален за нашия проект:

Простота и чистота на синтаксиса: Python е известен със своя чист и елегантен синтаксис, който прави кода лесен за четене и разбиране. Това го прави идеален за разработка на уеб приложения, където яснотата и лесното поддръжане са от съществено значение.

Богатство от библиотеки и инструменти: Python разполага с широка гама от вградени библиотеки и инструменти, които правят разработката по-лесна и ефективна. За нашия проект можем да използваме библиотеки като Django, които ни предоставят готови решения за много от задачите, свързани с уеб разработката.

Гъвкавост и разнообразие: Python е гъвкав и разнообразен език, който позволява разработка на различни видове приложения - от уеб приложения и сценарии за автоматизация до научни изчисления и машинно обучение. Това означава, че ако в бъдеще искаме да разширим функционалността на проекта, Python ще ни позволи да го направим безпроблемно.

## Django

Django е популярен уеб фреймуърк за Python, който предоставя мощни инструменти и готови решения за бързо и лесно разработване на уеб приложения. Ето няколко причини защо Django е най-добрият избор за нашия проект:

Мощност и гъвкавост: Django предлага цялостно решение за уеб разработка, включително вградени модули за администрация, автентикация, сигурност и много други. Това прави разработката на уеб приложения бърза и лесна.

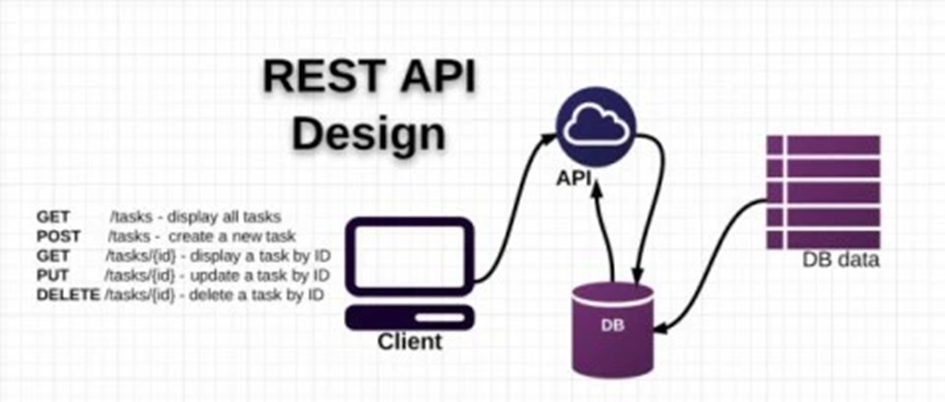
Сигурност и стабилност: Django е известен със своите вградени механизми за сигурност, които правят уеб приложенията, базирани на Django, надеждни и стабилни. Това е от решаващо значение, особено когато става въпрос за съхранение и обработка на чувствителна информация.

Голяма общност и поддръжка: Django е поддържан от голяма и активна общност от разработчици по целия свят. Това означава, че можем да се възползваме от безценното общество знание и ресурси, които са на разположение, за да подобрим и разширим нашия проект.

# Какво е REST?

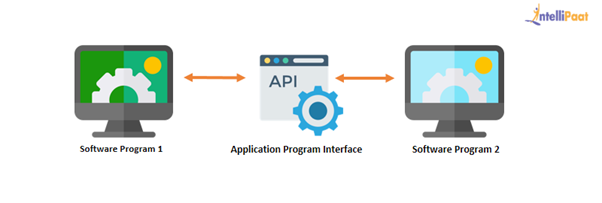
Точно както ние, хората, взаимодействаме с компютрите с помощта на потребителски интерфейс, уеб приложенията или софтуерните програми също трябва да взаимодействат помежду си. За да се направи това, се създава различен вид интерфейс.

REST е съкратено Representational State Transfer (за това понякога се изписва като “ReST“.) Разчита на клиент – сървър, кеширан, комуникационен протокол, като в почти всички случаи, се използва HTTP протокол. REST е архитектурен стил за проектиране на мрежови приложения. Идеята е, че вместо да използват сложни механизми за свързване между машини (като CORBA, RPC или SOAP), просто HTTP се използва за заявки между машините.



## Какво е API?

От поръчка на храна или резервация на хотелска стая до изтегляне на софтуер онлайн, цялата тази интерактивност е възможна с помощта на нещо, известно като API или интерфейс за програмиране на приложения. С други думи, API не е нищо друго освен интерфейс, който друг софтуер използва за достъп до данни или други приложения.



С прости думи, API е начин, по който софтуерните програми взаимодействат помежду си. Разработчиците създават API на сървъра и позволяват на клиенти или крайни потребители да взаимодействат с него.

Знаем, че даден уебсайт използва URL адреса, за да направи извикване на сървър и да изтегли уеб страница в браузър. API също улесняват обажданията към сървъра, но по по-опростен начин. API предлагат стандартен набор от инструкции, използвайки кои разработчици, софтуерни приложения и сайтове могат да имат достъп до уеб услуги или активи на уеб приложение.

Но отговорът на въпроса „Как да осъществя достъп до уеб услуги?“ беше само SOAP (Simple Object Access Protocol) за известно време. За разлика от акронима предполага, протоколът не е толкова лесен. SOAP, използван за извършване на комуникация, използвайки изключително XML файловата структура. За да работят със SOAP в JavaScript, разработчиците трябваше да напишат тон кодове, само за да изпълнят проста задача.

## Какво е Rest API?

REST означава “прехвърляне на представително състояние (REST)”. REST API използва прост URL адрес, за да прави заявки, вместо да използва XML. Основното правило на REST гласи, че човек трябва да може да върне набор от данни, когато е свързан към конкретен URL адрес. С други думи, URL се използва за изпращане на заявка и за получаване на отговор. REST може да използва различни HTTP методи като GET, POST, PUT и DELETE за изпълнение на задачи. За разлика от SOAP, в уеб услугите, базирани на REST, можем да получим изходи във форма, която е по-лесна за анализиране в рамките на езика, като JSON (JavaScript Object Notation) или CSV (стойности, разделени със запетая).

Въпросът е „Как да създадем REST API?“ - Има много опции за изграждане на REST API. Django REST Framework е един от най-добрите сред тях. Какво представлява Django REST Framework (DRF) и защо продължаваме с него.

## Защо Django REST API?

Django REST Framework (DRF) е мощен инструмент, който ни позволява да създадем гъвкав и мащабируем RESTful API в нашия проект. Ето няколко основни причини за използването на Django REST Framework:

Готови решения за изграждане на API: DRF предлага множество готови компоненти и инструменти, които улесняват изграждането на RESTful API. Това включва сериализатори за преобразуване на данните от Python обекти в JSON формат и обратно, гледища за дефиниране на структурата и логиката на API, и автентикация за сигурност.

Поддръжка на различни типове заявки и формати: DRF поддържа различни типове заявки, включително GET, POST, PUT, PATCH и DELETE, което прави управлението на данните лесно и удобно. Също така, DRF поддържа различни формати за обмен на данни, като JSON, XML, YAML и други.

Автоматична документация: DRF генерира автоматична документация за нашето API, която може да се използва за лесно разбиране и използване на API от страна на разработчиците. Това включва информация за наличните ендпойнти, параметрите на заявките и формата на върнатите данни.

Мащабируемост и гъвкавост: DRF е много гъвкав и мащабируем инструмент, който позволява лесно добавяне на нови функционалности и разширения към нашето API. Това ни позволява да адаптираме API според нашите нужди и изисквания.

Безплатно и с отворен код: DRF е безплатен и с отворен код, което означава, че можем да го използваме свободно и да го променяме според нашите нужди. Това осигурява голяма гъвкавост и контрол върху разработването на нашето API.

Съчетавайки тези преимущества, Django REST Framework е отличен избор за създаване на мощно, гъвкаво и сигурно RESTful API в нашия проект.

# Библиотеката Django REST Framework

Django REST Framework (DRF) е популярна библиотека за създаване на уеб приложения и RESTful API с помощта на Django, който е уеб фреймуърк за Python. DRF предоставя набор от инструменти и компоненти, които улесняват разработването на API, включително сериализация на данни, автентикация, авторизация, рутиране на заявки и документация.

Ето някои от основните характеристики и функционалности на Django REST Framework:

## Сериализация на данни:

DRF предоставя мощен механизъм за сериализация на данни, който позволява преобразуване на моделите на Django във формати като JSON или XML и обратно. Това улеснява комуникацията между клиента и сървъра чрез API.

## Компоненти за изграждане на API:

 DRF предлага различни компоненти за изграждане на API, включително класове за изгледи (viewsets), рутиращи класове (routers), сериализатори, филтри, пагинация и други. Тези компоненти улесняват разработването на API и добавянето на различни функционалности към него.

## Автентикация и авторизация:

DRF предоставя вградена поддръжка за различни методи на автентикация и авторизация, включително базова, токенова, сесионна и OAuth2 автентикация. Това позволява на разработчиците да установят права за достъп и сигурност на техните API.

## **Генериране на документация**:

DRF автоматично генерира документация за API-то, която се базира на използваните класове за изгледи и сериализатори. Това включва информация за наличните ендпойнти, техните параметри и върнатите данни, което прави API-то лесно разбираемо и управляемо.

## Широка поддръжка и общност:

DRF е една от най-използваните и популярни библиотеки за създаване на API в екосистемата на Django. Тя има широка общност от разработчици и поддръжка от страна на Django проекта.

Съчетавайки тези функционалности, Django REST Framework прави разработката на RESTful API бърза, лесна и ефективна, което го прави предпочитан избор за много уеб разработчици и компании.

## Заключение за DRF и REST

Изграждането на RESTful API може да бъде сложно, но Django REST framework се справя сравнително добре със сложността. Надявам се да се забавлявате да създавате нови API с помощта на Django REST framework.

В повечето случаи изграждането на архитектурен интерфейс RESTful API може да бъде сложно. Въпреки това, Django REST framework може да бъде сравнително подходяща за тази цел заради факта, че е достатъчно ефективна гъвкава да изгради качествен API само в рамките на два или три реда код.

## VUE

### Какво е Vue.js?

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.

Ето минимален пример:

import { createApp } from 'vue'  
createApp({  
 data() {  
 return {  
 count: 0  
 }  
 }  
}).mount('#app')

<div id="app">  
<button @click="count++">  
Count is: {{ count }}  
</button>  
</div>

Горният пример демонстрира двете основни характеристики на Vue:

* 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.

Може вече да имате въпроси - не се притеснявайте. Ще разгледаме всеки малък детайл в останалата част от документацията. Засега, моля, прочетете, за да можете да разберете на високо ниво какво предлага Vue

Прогресивният фреймуърк:

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. Ето предишния пример, написан в SFC формат:

<script>  
export default {  
 data() {  
 return {  
 count: 0

}   
 }  
}  
</script>

<template>  
 <button @click="count++">

Count is: {{ count }}

</button>

</template>  
<style scoped>  
button {  
 font-weight: bold;  
}  
</style>

### API стилове:

Компонентите на Vue могат да бъдат създадени в два различни API стила: Options API и Composition API.

### API за опции:

С API за опции ние дефинираме логиката на компонент, като използваме обект от опции като data, methods и mounted. Свойствата, дефинирани от опциите, са изложени на тази вътрешна функция, която сочи към екземпляра на компонента:

<script>  
export default {  
data() {  
return {  
count: 0  
 }  
 },  
  
 increment() {  
 this.count++  
 }  
 },  
  
 mounted() {  
 console.log(`The initial count is ${this.count}.`)  
 }  
 }  
 </script>  
  
<template>  
<button @click="increment">Count is: {{ count }}</button>  
</template>

### Композиционен API:

С Composition API ние дефинираме логиката на компонент, като използваме импортирани API функции. В SFCs Composition API обикновено се използва с <script setup>. Атрибутът за настройка е намек, който кара Vue да извършва трансформации по време на компилиране, които ни позволяват да използваме Composition API с по-малко шаблони. Например импортирането и променливите/функциите от най-високо ниво, декларирани в <script setup>, могат директно да се използват в шаблона.

Тук е същият компонент, с абсолютно същия шаблон, но вместо това използва Composition API и

<script setup> :

<script setup>  
import { ref, onMounted } from 'vue'  
  
const count = ref(0)  
  
function increment() {  
 count.value++  
}  
  
onMounted(() => {  
 console.log(`The initial count is ${count.value}.`)  
})

</script>  
  
<template>  
 <button @click="increment">Count is: {{ count }}</button>  
</template>

## Кое да изберем?​

И двата API стила са напълно способни да покрият обичайни случаи на употреба. Те са различни интерфейси, захранвани от една и съща основна система. Всъщност API за опции е внедрен върху API за композиция! Основните концепции и знания за Vue се споделят в двата стила.

Приложният програмен интерфейс (API) на опциите е съсредоточен около концепцията за „екземпляр на компонент“ (както се вижда в примера), който обикновено се привежда в съответствие с базиран на клас мисловен модел за потребители, идващи от ООП езиков произход. Освен това е по-удобен за начинаещи, като абстрахира детайлите за реактивност и налага организация на кода чрез групи опции.

Composition API е съсредоточен около декларирането на реактивни променливи на състоянието директно в обхват на функция и съставянето на състояние от множество функции заедно, за да се справи със сложността. Той е в по-свободна форма и изисква разбиране за това как реактивността работи във Vue, за да се използва ефективно. В замяна неговата гъвкавост позволява по-мощни модели за организиране и повторно използване на логиката.

## Характеристика на VUE

### 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.

Кодовият фрагмент по-долу съдържа пример за Vue компонент. Компонентът представя бутон и отпечатва броя на кликванията върху бутона:

<template>  
 <div id="tuto">  
 <button-clicked v-bind:initial-count="0"></button-clicked>  
 </div>  
</template>  
  
<script>  
Vue.component('button-clicked', {  
 props: ['initialCount'],  
 data: () => ({  
 count: 0,  
 }),  
 template: '<button v-on:click="onClick">Clicked {{ count }} times</button>',  
 computed: {  
 countTimesTwo() {  
 return this.count \* 2;  
 }  
 },  
 watch: {  
 count(newValue, oldValue) {  
 console.log(`The value of count is changed from ${oldValue} to ${newValue}.`);  
 }  
 },  
 methods: {  
 onClick() {  
 this.count += 1;  
 }  
 },  
 mounted() {  
 this.count = this.initialCount;  
 }  
});  
  
new Vue({  
 el: '#tuto',  
});  
</script>

### 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.

### 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.

### Преходи:

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

* Автоматично прилагайте класове за CSS преходи и анимации
* Интегриране CSS анимационни библиотеки на трети страни, като Animate.cs
* Използване JavaScript, за да манипулирате директно DOM по време на куките за преход
* Интегриране JavaScript анимационни библиотеки на трети страни, като Velocity.js

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

Vue автоматично ще надуши дали целевият 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 подходящи моменти.

Ако не бъдат открити CSS преходи/анимации и не са предоставени JavaScript кукички, DOM операциите за вмъкване и/или премахване ще бъдат изпълнени незабавно на следващия кадър.

### Маршрутизиране:

Традиционен недостатък на едностраничните приложения (SPA) е невъзможността да се споделят връзки към точната „под“ страница в рамките на конкретна уеб страница. Тъй като SPA обслужват своите потребители само с един базиран на URL отговор от сървъра (обикновено обслужва index.html или index.vue), маркирането на определени екрани или споделянето на връзки към конкретни секции обикновено е трудно, ако не и невъзможно. За да разрешат този проблем, много рутери от страна на клиента ограничават своите динамични URL адреси с "hashbang" (#!), напр. [page.com/#!/](https://l.facebook.com/l.php?u=http%3A%2F%2Fpage.com%2F%3Ffbclid%3DIwAR3NhWGb7MCEQY5x1wrE06s_ZnH2Vj3fz76GZT2-XzeDXh6dopGrcGbg3m0%23!%2F&h=AT2o607b1ESvgMlDek05dpqD3qH4SUCXMhvda59C_uJoDDJ6bDTfSYevcm-pnHC9ixKH33Yj-FvufqyrZ7ro6V_XxA_BdM6PUjM-PWc6u0FWoItpp30IqUdAeoB_h1RqQq2kPg). Въпреки това, с HTML5 повечето съвременни браузъри поддържат маршрутизиране без hashbang.

Vue предоставя интерфейс за промяна на това, което се показва на страницата въз основа на текущия URL път – независимо от това как е променен (дали чрез имейл връзка, опресняване или връзки в страницата). Освен това използването на рутер от предния край позволява умишлено преминаване на пътя на браузъра, когато определени събития на браузъра (т.е. щраквания) възникнат върху бутони или връзки. Самият Vue не идва с хеширано маршрутизиране в предния край. Но пакетът "vue-router" с отворен код предоставя API за актуализиране на URL адреса на приложението, поддържа бутона за връщане назад (хронология на навигацията) и нулиране на парола за имейл или връзки за потвърждение на имейл с URL параметри за удостоверяване. Той поддържа картографиране на вложени маршрути към вложени компоненти и предлага фин контрол на прехода. С Vue разработчиците вече композират приложения с малки градивни елементи, изграждащи по-големи компоненти. С vue-router, добавен към микса, компонентите трябва просто да бъдат картографирани към маршрутите, към които принадлежат, а родителските/коренните маршрути трябва да посочват къде трябва да се рендират децата.

<div id="app">  
 <router-view></router-view>  
</div>  
...  
  
<script>  
...  
const User = {  
 template: '<div>User {{ $route.params.id }}</div>'  
};  
  
const router = new VueRouter({  
 routes: [  
 { path: '/user/:id', component: User }  
 ]  
});  
...  
</script>

Кодът по-горе:

1. Задава преден маршрут на [websitename.com/user/](https://l.facebook.com/l.php?u=http%3A%2F%2Fwebsitename.com%2Fuser%2F%3Ffbclid%3DIwAR0HPf4xUHNWB9c3YZQcJLQQCr7Jxzg1jhv7krIsL4CrJGv3x1kW9iOHCXY&h=AT2o607b1ESvgMlDek05dpqD3qH4SUCXMhvda59C_uJoDDJ6bDTfSYevcm-pnHC9ixKH33Yj-FvufqyrZ7ro6V_XxA_BdM6PUjM-PWc6u0FWoItpp30IqUdAeoB_h1RqQq2kPg)<id>.

2. Което ще се рендира в потребителския компонент, дефиниран в (const User...)

3. Позволява на потребителския компонент да предаде конкретния идентификатор на потребителя, който е въведен в URL адреса с помощта на ключа за параметри на обекта $route: $route.params.id.

4. Този шаблон (различен според параметрите, предадени на рутера) ще бъде изобразен в <router-view></router-view> в div#app на DOM.

5. Окончателно генерираният HTML за някой, който въвежда: [websitename.com/user/1](https://l.facebook.com/l.php?u=http%3A%2F%2Fwebsitename.com%2Fuser%2F1%3Ffbclid%3DIwAR1-96A2gT4NA8qU_lO91IvId9c8Y99ekhPq-cdb16pH1Pv7ud8gre4MqPI&h=AT2o607b1ESvgMlDek05dpqD3qH4SUCXMhvda59C_uJoDDJ6bDTfSYevcm-pnHC9ixKH33Yj-FvufqyrZ7ro6V_XxA_BdM6PUjM-PWc6u0FWoItpp30IqUdAeoB_h1RqQq2kPg) ще бъде:

<div id="app">  
 <div>  
 <div>User 1</div>  
 </div>  
</div>

## Предимства:

### Документация и разработка

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 за 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.

### Библиотеки на Vue:

Vue Router, Vuex, Vue Loader, Vue Server Renderer, Vue Test Utils, Vue Dev-Tools

# Защо избираме базата данни MySQL?

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

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

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

## MySQL базите данни са релационни.

Релационните бази данни съхраняват данни в отделни таблици, вместо да поставят всичките данни в една голяма директория. За по-бърза работа структурите от бази данни са организирани във физически файловe. Логическият модел, с обекти като бази данни, таблици, изгледи, редове и колони предлага гъвкава програмна среда. Вие настройвате правила, управлявате връзките между различните полета с данни, като „one-to-one“, „one-to-many“, „unique“, “required” или „optional“, както и различни показалци между различните таблици. Базите данни налагат тези правила, така че с добре-проектирана база данни вашите приложения никога няма да се сблъскат с несъвместими, дублиращи се, остарели или липсващи данни.

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

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

## MySQL е софтуер с отворен код.

„Отворен код“ означава, че е възможно за всеки да модифицира софтуера. Всеки желаещ може да свали MySQL софтуера от Интернет и да го използва, без да заплаща нищо. Ако искате може да изучите изходния код и да го промените така, че да отговаря на вашите нужди. Разбира се, ако възнамерявате да използвате MySQL за комерсиални цели трябва да се сдобиете и със съответния лиценз.

# HTML и CSS

## Какво означава HTML?

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

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

## Как работи HTML?

**HTML** ни помага да зададем основната структура на дадена уеб страница. Също така е полезен, за да редактираме параграфи, заглави и други основни елементи на една уеб страница. Когато някой кликне на линк в страница, или въведе нов URL в адресното поле, браузърът изпраща заявка за документ от Web server-a, където е качена страницата, посредством HyperText Transport Protocol (HTTP).

Сървърът след това изпраща документа обратно до потребителя като се визуализира на дисплея от браузъра. Всички неща, които са на този документ – текст, снимки, аудио или видео файлове, всички те са **подредени посредством HTML структура**.

## Какво е HTML5 и какви са неговите предимства?

HTML5 е осъвременена версия на едноименния език. Подобренията на HTML5 спрямо предходните версии са много, но ето най-ключовите от тях:

* Подобрен код – HTML5 предлага на уеб дизайнерите да използват по-чист, по-хубав код. Вече може да се премахнат <div> таговете и да се заменят със семантични HTML5 елементи.
* По-елегантни формуляри – Опциите за полета за въвеждане на потребителски данни са много по-гъвкави в сравнение с преди.
* Поддръжка на мултимедия – Кажете „сбогом“ на „плъг-ин“-ите в кода за аудио и видео. HTML5 има вградени опции за работа с аудио и видео файлове.

## Какво е CSS?

CSS е език за описание на стилове. Преди създаването на езика, задаване на стилове с HTML тагове е било кошмарът на разработчиците. Това е било невероятно предизвикателство при изграждането на големи сайтове.

Именно това е причина да се разработи CSS – стиловете излизат от обхвата на HTML, улесняват процеса по работа и реално това позволява всички стилове да бъдат променени бързо и с лекота. Чрез CSS се отделя описанието на съдържанието и структурата на сайтовете от стилизирането им.

## Защо да изучаваш HTML и CSS?

Сега, когато знаеш какво е HTML и как се допълва от употребата на CSS, ползите от тези два езика. Те са водещи в уеб разработката и са приложими на практика във всяка индустрия, която се нужда от дигитално присъствие. И ако това не е достатъчно, ето още няколко причини защо си струва да ги овладееш:

* + 1. Позволяват ти да се отличиш – умението да използваш тези два езика ще ти позволи да създаваш сайтове с уникална визия, която да ги открои от конкурентните. Ще можеш да правиш красиви и привлекателни темплейти за имейли за корпоративна комуникация и нюзлетъри.
    2. Помагат ти с работата в екип – дори и да не си програмист, да знаеш какво е HTML и как се използва в комбинация със CSS ще ти бъде от голяма полза, когато трябва да работиш съвместно с програмисти и дизайнери за постигане на определени цели.
    3. Лесни за изучаване – от гледна точка на уеб разработката, това са два лесни за научаване езика. Структурата от тагове, която се използва от HTML е лесна за разбиране, дори и да нямаш никакви предишни познания и опит.

# Реализация

## Първи стъпки на проекта:

* 1. Отварям (или инсталирам) PyCharm и създавам нов проект
  2. Инсталирам Django – *pip install django*
  3. Създавам приложение – django-admin startproject e\_library
  4. Правя папка в (C:) диск, с име като името на проекта, където да държим проекта
  5. Разглеждам ulrs.py и settings.py
  6. Правя тест на сървъра – python manage.py runserver

## Създаване на приложение:

1. В settings.py, секция INSTALLED\_APPS регистрираме приложението
2. В папката на приложението създаваме празен файл urls.py
3. В главния urls.py пренасочваме проследяване на url-ите към нашето приложение:
4. Добавям include в импорта след path: from django.urls import path, include
5. Добавям още един път за проследяване

from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('main.urls'))  
  
]

1. В новия urls.py копирам от стария и коригирам съдържанието:

from django.urls import path  
from . import views  
urlpatterns = [  
 path('', views.index)  
]

1. във vies.py създавам новата функция index:

from django.shortcuts import render  
from django.http import HttpResponse  
  
# Create your views here.  
def index(request):  
 return HttpResponse('<h4> Главна страница </h4>')

## Използване на шаблони:

1. В папката main създавам папка templates. Името е важно да е точно такова. В тази папка създавам папка със същото име като това на приложението.
2. В новосъздадената папка създавам шаблон (.html файл) с име index.html и попълвам title и body с някакво съдържание.
3. Във vies.py коригирам функцията index:

def index(request):  
 return render(request, 'main/index.html')

## Настройване на Django за работа с БД MySQL:

При инсталиране Django установява по подразбиране настройки за работа с база данни sqlite. Така, без необходимост от допълнително конфигуриране, потребителят може с лекотата да съхранява и извлича на данни от БД. Sqlite е подходящa за малки уеб приложения, но за по-големи ще доведе до проблеми с производителността. За да се избегнат това, първата стъпка е да изберете подходяща база данни, способна да обслужва стотици или дори хиляди едновременни посещения. За тази цел може да се използва база данни MySQL.

## Настройване на MySQL база данни:

1. Инсталирам MySQL и достъп до него чрез XAMPP.
2. Създаваме нова (празна) база данни.
3. Създаваме нов потребител и му даваме права върху създадената база данни.

## Настройване на Django проекта:

1. Инсталирам MySQL клиент - *pip install mysqlclient*
2. Настройване на връзката към БД - Отваряме settings.py на проекта и настройваме DATABASES:

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'dzi\_15',  
 'USER': 'admin',  
 'PASSWORD': 'admin123',  
 'HOST': 'localhost',  
 'PORT': '3306',  
 'OPTIONS':{'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'"}  
 }  
}

## Миграции

В терминала и подавам команда за подготовка на миграциите

python manage.py makemigrations

## Създаване на моделите:

1. В models.py, създаваме клас за статиите:

class article(models.Model):  
 title = models.CharField('Заглавие', max\_length=200)  
 content = models.TextField('Съдържание')  
 author = models.CharField('Автор', max\_length=50)  
 published = models.DateField('Публикувано на', default=date.today)  
  
 def \_\_str\_\_(self):  
 return self.title  
  
 class Meta:  
 verbose\_name = 'Статия'  
 verbose\_name\_plural = 'Статии'

1. Мигрираме

python manage.py makemigrations

python manage.py migrate

## Създаване на администратор:

1. Стартирам тестовия сървър
2. Преминавам на страница /admin (от адресната лента на браузъра)
3. Променям препратката в base.html да стартира админ панела:

<li class="nav-item"><a class="nav-link px-lg-3 py-3 py-lg-4" href="admin">Admin</a></li>

1. Спираме тестовия сървър и в терминала пишем команда
2. python manage.py createsuperuser

## Регистриране на таблица:

В admin.py импортираме от models създадената таблица и я регистрираме:

from django.contrib import admin  
from .models import article  
  
admin.site.register(article)

## Извеждане на данни от БД:

1. Във views.py модифицираме изгледа на основната страница

from .models import article  
  
def index(request):  
 articles = article.objects.all()  
 params = {'page\_title': 'ГЛАВНА СТРАНИЦА',  
 'page\_header': 'Главна страница',  
 'page\_content': 'Текст на страницата',  
 'articles': articles,  
 }  
 return render(request, 'main/index.html', params)

1. В index.html добавяме визуализиране на параметъра articles

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

1. Модифицирам извеждането да е по един елемент на ред
2. Примери за различни селекции:

articles = article.objects.filter(title\_\_contains='1')  
articles = article.objects.order\_by('-id').filter(title\_\_contains='1')  
articles = article.objects.order\_by('-id')[:2]

1. Променям изгледите, така че едната страница да показва новите статии (срез по id), а другата – всички статии

# Заключение

Създаденото от мене приложение отговаря на изискванията на заданието за дипломен проект. То реализира зададената функционалност, има добре изглеждащ потребителски интерфейс, визуализира се правилно на устройства с различна резолюция.

Фактически моето приложение представлява трислойно приложение за управление на библиотека. Системата показва всички читатели и книгите, които те са взели. Всеки читател може да провери за наличието на определена книга или да види кои книги е взел. Администраторът от своя страна има достъп до всички данни за читателите - история, взети книги, данни за контакт.

Приложението предлага две обособени групи функционалност – за администратора на библиотеката и за читателите.

Администраторът може да добавя нови книги, да вижда целия списък с наличните книги и да премахва книга от базата данни, да вижда списъка с читатели, да добавя или премахва читател от списъка, да отдава определена книга на даден читател и да отбелязва книгата като върната.

Читателят може да види своя профил, да редактира профила си в частта лични данни и парола за достъп, да види кои книги е взел от библиотеката и в какъв срок трябва да ги върне.

Самото приложение може да се администрира с помощта на администраторско меню. Дизайнът на приложението е адаптивен и изглежда еднакво добре на различни устройства.

Приложението е разработено и достъпно за демонстриране на локаленния (тестовия) уеб сървър, който е част от Django.

Разработката на проекта демонстрира възможностите за създаване на съвременни уеб приложения, които се основават на силната комбинация от Python и Django. Бъдещите перспективи включват по-нататъшно разширяване на функционалността, оптимизиране на производителността и внедряване на допълнителни технологии за подобряване на потребителския опит.

В заключение, дипломната работа успешно демонстрира възможностите за разработка на трислойни приложения с използването на Python, Django и Vue.js. Тези технологии съчетани в съвременен стек от технологии предоставят ефективност, сигурност и гъвкавост, които са от съществено значение за разработването на уеб приложения в днешно време.

# Списък на използваната литература

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

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

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

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

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

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

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