Съдържание

[1. Увод: 3](#_Toc133581855)

[2. Функционалните възможности на framework-а VUE: 4](#_Toc133581856)

[2.1 Библиотеката Axios 6](#_Toc133581857)

[2.1.1 История за създаването на VUE: 7](#_Toc133581858)

[2.1.2 Компоненти: 7](#_Toc133581859)

[2.1.3 Шаблони: 7](#_Toc133581860)

[2.1.4 Реактивност: 8](#_Toc133581861)

[3. Технологиите за комуникация и обмен на данни със сървъра 8](#_Toc133581862)

[3.1 Описание на HTTP сървър и браузър 9](#_Toc133581863)

[3.2 Common Gateway Interface (CGI) 10](#_Toc133581864)

[3.3 Работа на Web приложенията с бази от данни: 10](#_Toc133581865)

[3.4 HTML 11](#_Toc133581866)

[3.5 Развитие на HTML 12](#_Toc133581867)

[3.5.1 Маркиране 13](#_Toc133581868)

[3.5.2 Елементи 14](#_Toc133581869)

[4. Възможности за интегриране на Vue и Python/ Django/ DRF. 14](#_Toc133581870)

[4.1 Python 14](#_Toc133581871)

[4.2 Django 15](#_Toc133581872)

[4.2.1 Тагът „Verbatim“ 16](#_Toc133581873)

[5. Технологии 17](#_Toc133581874)

[5.1 DRF 17](#_Toc133581875)

[5.2 Jinja 18](#_Toc133581876)

[5.3 REST 19](#_Toc133581877)

[5.4 SOAP 20](#_Toc133581878)

[5.5 REST срещу SOAP 21](#_Toc133581879)

[6. Проектиране на необходимата функционалност: 22](#_Toc133581880)

[6.1 АНАЛИЗ НА ЗАДАНИЕТО 22](#_Toc133581881)

[6.2 МОДЕЛИ 22](#_Toc133581882)

[6.3 ЕЛЕМЕНТ ОТ JAVASCRIPT КОДА 22](#_Toc133581883)

[6.4 ТЕМПЛЕЙТИ 24](#_Toc133581884)

[7. Разработка на минимална версия на горното приложение, като се реализират: 26](#_Toc133581885)

[7.1 Регистриране на нови потребители: 26](#_Toc133581886)

[7.2 Оторизиране достъпа на потребител: 28](#_Toc133581887)

[8. Разглеждане на регистрираните обяви/заявки 29](#_Toc133581888)

[9. Заключение: 30](#_Toc133581889)

[10. Списък на използваната литература: 31](#_Toc133581890)

# Увод:

Участвам в общ проект с моята съученичка Константина, с която създаваме уебсайт, а именно „Борса за учебници“. Важно е училищата да имат такива онлайн библиотеки, тъй като всеки ученик се нуждае от онлайн място, от което да купува или където да продава своите неизползваеми вече учебници. В проекта аз участвам като frontend програмист, казано с други думи създавам визуалната част на проекта. Проектът ми е създаден с помощта на framework-а Vue. Той предоставя лесен и ефективен начин за създаване на модерни и динамични уеб приложения. Използваната библиотека във Vue е Axios, която е библиотека за изпращане на HTTP заявки от клиентската част на приложението.

Използвала съм също и HTML, който е основният маркиращ език за описание и дизайн на уеб страници. Python е програмен език, който също участва в създаването на моята част от проекта. Той е създаден да бъде език за програмиране, който да е удобен и ефективен за разработка на софтуер. Python има много използвани и популярни фреймуърка, като в моя случай използвам Django, който предоставя базовия шаблон и улеснява разработката на уеб приложения. Друг framework, без който проектът ми нямаше как да просъществува е Django REST framework (DRF). Той предоставя много инструменти и функции за създаване на уеб API. Тези инструменти включват сериализация на данни, валидация на данни, автоматично генериране на документация, авторизация и управление на потребители, аутентикация и много други. DRF също така предлага много вградени изгледи (views), които покриват много от общите изисквания на уеб услуги.

Jinja е още един мощен шаблонен език за Python, използван в практическата ми част, който позволява лесно и ефективно създаване на HTML 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.

Архитектурният стил, който влезе в употреба при създаването на проекта ми е REST. Той се използва за разработване на уеб услуги и уеб приложения. Основан е на използването на различни HTTP методи, като GET, POST, PUT и DELETE, за достъп до ресурси.

SOAP е абревиатура от "Simple Object Access Protocol" (прост протокол за достъп до обекти) и е протокол за обмен на данни в мрежата, който се използва в програмирането. SOAP е базиран на XML и използва HTTP за пренос на съобщенията. Това също учатва в създаването на нашия уебсайт.

# Функционалните възможности на framework-а VUE:

Vue.js е изключително популярна библиотека за създаване на уеб приложения, която е базирана на JavaScript. Vue.js предоставя едно от най-добрите решения за създаване на модерни, динамични и многофункционални уеб приложения. Това е поради факта, че Vue.js е много лека библиотека, която може да бъде използвана за разработване на малки до средно големи приложения.

Vue.js е изключително гъвкава и лесна за употреба библиотека, която се интегрира лесно с други технологии като Bootstrap, Materialize, Bulma и др.

Vue.js предоставя много полезни функции, като например реактивност, компонентно базирана архитектура, двустранно свързване на данни, компонентна рутинг система, директиви, филтри и много други.

Vue.js се използва от много уеб разработчици по света, защото тя предоставя много ефективни инструменти и функции за създаване на уеб приложения. Тя се използва за създаване на различни уеб приложения, като електронни магазини, социални мрежи, блогове, портали за новини и много други.

Vue.js е съвместима с много браузъри и може да бъде интегрирана със съвременните инструменти за разработка на уеб приложения като Webpack, Babel, ESLint и други. Това прави Vue.js много популярна сред уеб разработчиците и един от най-използваните инструменти за създаване на уеб приложения в момента.

Vue.js е съвременна библиотека за създаване на уеб приложения, която се фокусира върху декларативността, компонентната архитектура и лесната употреба. Тя предоставя лесен и ефективен начин за създаване на модерни и динамични уеб приложения.

Vue.js се базира на реактивна система, която позволява бързо обновяване на UI, когато се променят данните. Това означава, че приложенията, създадени с Vue.js, могат да бъдат изградени по такъв начин, че да бъдат лесно мащабируеми и поддръжаеми.

Основната концепция в Vue.js е компонентната архитектура. Компонентите са изградени от HTML, CSS и JavaScript, което ги прави много лесни за използване и многофункционални. Това означава, че с Vue.js можете да създавате уеб приложения, които са съставени от много малки и преизползваеми части.

Vue.js предоставя много полезни функции и инструменти за уеб разработка, като например компонентна рутинг система, двустранно свързване на данни, директиви, филтри и други. Това прави Vue.js една от най-популярните библиотеки за създаване на уеб приложения в момента.

С Vue.js можете да създавате различни видове уеб приложения, като електронни магазини, социални мрежи, блогове и много други. Vue.js е съвместима с много браузъри и може да бъде интегрирана лесно с други инструменти за разработка на уеб приложения като Webpack, Babel, ESLint и други.

В заключение, Vue.js е много популярна и ефективна библиотека за създаване на уеб приложения в JavaScript. Тя предоставя много инструменти и функции за лесна и ефективна уеб разработка и се използва от много уеб разработчици по света. Vue.js е лесна за употреба, гъвкава и може да бъде интегрирана с други технологии, което прави тази библиотека един от най-добрите избори за създаване на уеб приложения.

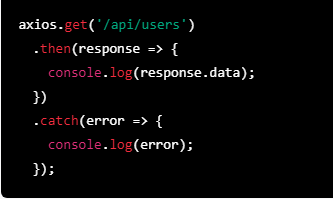
Vue позволява разширяване на HTML с HTML атрибути, наречени директиви. Директивите предлагат функционалност на HTML приложенията и идват като вградени или дефинирани от потребителя директиви.

## Библиотеката Axios

Axios е библиотека за изпращане на HTTP заявки от клиентската част на приложението. За да използвате Axios във Vue приложение, трябва да я инсталирате и да я импортирате във вашия код.

Методът axios.get() се използва за изпращане на GET заявки към сървър.

Пример:



В този пример изпращаме GET заявка към **/api/users** и използваме **.then()** за да обработим успешния отговор и да изведем данните в конзолата. Ако заявката срещне грешка, използваме **.catch()** за да обработим грешката и да я изведем в конзолата.

Методът axios.post() се използва за изпращане на POST заявки към сървър.

Пример:



В този пример изпращаме POST заявка към **/api/users** с данните **firstName**, **lastName** и **age**. Данните се предават в обекта **data**. Използваме **.then()** за да обработим успешния отговор и да изведем данните в конзолата. Ако заявката - --срещне грешка, използваме **.catch()** за да обработим грешката и да я изведем в конзолата.

### История за създаването на VUE:

Vue е създаден от Evan You след работа за Google, използвайки AngularJS в няколко проекта. По-късно той обобщи мисловния си процес: „Помислих си, какво ще стане, ако мога просто да извлека частта, която наистина харесвам в Angular, и да създам нещо наистина леко.“ Първият ангажимент за изходния код към проекта беше от юли 2013 г. и Vue беше публично обявен за първи път през февруари 2014 г.

Имената на версиите често произлизат от манга и аниме, повечето от които са в жанра научна фантастика.

### Компоненти:

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

### 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 точно кои компоненти.

# Технологиите за комуникация и обмен на данни със сървъра

Web технологията се развива въз основа на приложения от типа клиент/сървър. При този модел имаме изпълними модули, които не е задължително да са част от една програма или пък да работят на една и съща машина. Отделните процеси могат да правят заявки към други процеси. В подобна среда процесите, които изискват дадени услуги наричаме клиенти, а тези процеси или програми, които ги доставят описваме като сървъри.

Клиентските и сървърните процеси могат да бъдат разпределени. С други думи, процесите могат да работят на различни машини, под различни операционни системи и реализиращ ги софтуер. Например сървърът за управление на база от данни може да е стартиран върху UNIX машина във Великобритания, докато Windows приложението, което прави заявките към базата от данни да е стартирано на PC в България. Детайлите по изпълнението на заявките остават скрити за потребителя в България - неговото клиентско приложение се грижи да осигури комуникацията със сървъра и да получи необходимите данни.

Web сървърите и брaузърите използват клиент/сървър архитектурата с помощта на протокола HTTP. Клиентът (в случая с WWW e брaузър) прави заявка за документ към сървъра просто с натискане на мишката или клавиш, и сървърът връща необходимия документ. Приложенията се грижат за цялата комуникация, заявки и обработване на документа. Крайният потребител се грижи единствено да заяви и получи необходимата информация.

## Описание на HTTP сървър и браузър

HTTP сървърите се основават на Hyper Text Transfer Protocol и осигуряват достъп до разпределени хипермедийни документи, приложения и бази от данни. Те работят на принципа заявка/отговор. Kлиентът осъществява връзка със сървъра и изпраща заявка. Сървърът отговаря със служебна информация, съдържаща версията на протокола на текущото съобщение и код за успех или грешка, последвана от MIME-кодирано съобщение.

Основните функции на HTTP сървърите включват комуникация по HTTP/1.0 съвместим протокол, няколко вида разпознаване на потребителски заявки, поддръжка на виртуални места, договаряне на съдържанието на съобщенията. В допълнение се поддържат изцяло MIME типовете, формати за графична информация като GIF и JPEG, собствен Application Programming Interface (API), кодиране на данните и проверка на съобщенията с използване на Secure Socket Layer (SSL) протокол.

Web брaузърите (например Netscape Navigator или Communicator, Microsoft Internet Explorer, Opera, Lynx и т.н.) са интерфейса на крайните потребители към World Wide Web. Браузърите комуникират с HTTP сървъра, за да получат хипермедийни документи като следят намесата на потребителя, инициират необходимия трансфер на документи и ги обработват по съответен начин, за да бъдат форматирани и изведени на екрана.

Основните им функции включват приемане на HTML документи, обработка и показването им на екрана като при избиране на хипервръзка се зарежда нов документ. В допълнение, почти всички поддържат кеширане на документите, запис на адреси на важни или често използвани документи, показване на части от документ или картина, възможност за частни комуникации чрез протоколи за сигурност, интерфейс от типа drag and drop, Java и JavaScript приложения, конкретни настройки за всеки потребител.

Plug–in са допълнителни програмни модули, инсталирани към всеки Web браузър. Работейки от страната на клиента, тяхната задача е да обработят правилно и бързо определени текстови, графични, звукови, видео и други видове формати, които са вмъкнати в Web страницата. Те са в състояние да подпомогнат разнообразяването на HTML страниците. Тяхното предназначение е не да обменят информация със сървъра, а да визуализират по подходящ начин подадената такава. При изработката на Web страници трябва да се има в предвид, че plug-ins са специфични за всеки браузър, който може да използва клиента, както и за съответната компютърна платформа и е необходимо да бъдат инсталирани в повечето случаи допълнително.

## Common Gateway Interface (CGI)

CGI се използва за комуникация между Web сървъра и приложения, които не са част от софтуера, който реализира функциите на HTTP сървър. Обикновено това е процес, работещ на сървъра и осигуряващ връзката между Web сървъра и останалите приложения, информационни ресурси, бази от данни.

CGI програмите служат за разширяване функциите на сървъра и добавят възможности за обработване на разнообразни клиентски заявки, които стандартно не се поддържат. Реализират се на произволен език, изпълним на сървъра, като: C, C++, Java, JavaScript, Visual Basic, VBScript, Perl, UNIX shells, Awk, Tcl. Използват се основно за връзка към информационни ресурси (обикновено поддържани в бази от данни) и за динамично генериране на HTML документи.

## Работа на Web приложенията с бази от данни:

* + Web клиентът осъществява връзка със сървъра на посочения в Uniform Resource Locator (URL) адрес и порт. Ако не е указан порт се избира стандартния за протокола HTTP, който обикновено е 80.
  + Web клиентът изпраща заявка за извличане, добавяне или промяна на информация. Потребителят генерира необходимата заявка, използвайки т. нар. форми. Те са част от езика HTML и са описани на следващите страници.
  + С тяхна помощ могат да се извикват CGI приложения, работещи на сървъра и да се предават или приемат данни.
  + Клиентските данни се изпращат чрез сървъра към CGI програма, чието име е описано в URL.
  + CGI програмата чете данните и извършва обработката им по предварително заявен начин.
  + На базата на обработените данни CGI програмата осъществява връзка с указания сървър на база от данни и изпраща исканите заявки.
  + CGI програмата извлича данните от заявките, прекратява връзката със сървъра на базата от данни и ги обработва.
  + CGI програмата генерира отговор до клиента. Обикновено това е HTML документ, който се изпраща до клиента чрез Web сървъра.
  + След изпращане на отговора HTTP сървърът приключва връзката.

## HTML

HTML е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в интернет, а неговите стандарти се определят от международния консорциум W3C.

HTML елементите са градивните елементи на HTML страниците. С HTML конструкции, изображения и други обекти като интерактивни форми могат да бъдат вградени в изобразената страница. HTML предоставя средства за създаване на структурирани документи чрез обозначаване на структурна семантика за текст като заглавия, параграфи, списъци, връзки, цитати и други елементи. HTML елементите са очертани от тагове, написани с ъглови скоби. Тагове като **<img />** и **<input />** директно въвеждат съдържание в страницата. Други тагове като <p> и </p> обграждат и предоставят информация за текста на документа и могат да включват тагове на поделементи. Браузърите не показват HTML таговете, но ги използват, за да интерпретират съдържанието на страницата.

HTML може да вгражда програми, написани на скриптов език като JavaScript, който засяга поведението и съдържанието на уеб страниците. Включването на CSS определя външния вид и оформлението на съдържанието. World Wide Web Consortium (W3C), бивш поддържащ HTML и настоящ поддържащ на CSS стандартите, насърчава използването на CSS вместо изричния презентационен HTML от 1997 г. Форма на HTML, известна като HTML5, се използва за показване на видео и аудио, основно с помощта на елемента <canvas>, заедно с JavaScript.

## Развитие на HTML

През 1980 г. физикът Тим ​​Бърнърс-Лий, изпълнител в CERN, предложи и създаде прототип на ENQUIRE, система за изследователи от CERN, които да използват и споделят документи. През 1989 г. Бърнърс-Лий написва меморандум, в който предлага интернет базирана хипертекстова система. Бърнърс-Лий уточнява HTML и написва браузъра и сървърния софтуер в края на 1990 г. Същата година Бърнърс-Лий и инженерът на системите за данни на CERN Робърт Кайо си сътрудничат по съвместна молба за финансиране, но проектът не е официално приет от CERN. В личните си бележки от 1990 г. Бърнърс-Лий изброява "някои от многото области, в които се използва хипертекст";

Първото публично достъпно описание на HTML беше документ, наречен "HTML тагове", споменат за първи път в Интернет от Тим ​​Бърнърс-Лий в края на 1991 г. Той описва 18 елемента, съставляващи първоначалния, относително прост дизайн на HTML. С изключение на маркера за хипервръзка, те бяха силно повлияни от SGMLguid, вътрешен стандартен генерализиран език за маркиране (SGML), базиран на документационен формат в CERN. Единадесет от тези елементи все още съществуват в HTML 4.

HTML е език за маркиране, който уеб браузърите използват, за да интерпретират и композират текст, изображения и други материали във видими или звукови уеб страници. Характеристиките по подразбиране за всеки елемент от HTML маркирането се дефинират в браузъра и тези характеристики могат да бъдат променени или подобрени чрез допълнителното използване на CSS от дизайнера на уеб страница. Много от текстовите елементи са споменати в техническия доклад на ISO от 1988 г. TR 9537 Техники за използване на SGML, който описва характеристиките на ранните езици за форматиране на текст като този, използван от командата RUNOFF, разработена в началото на 1960 г. за CTSS (съвместимо споделяне на време система) операционна система. Тези команди за форматиране са извлечени от командите, използвани от наборчиците за ръчно форматиране на документи. Концепцията на SGML за обобщено маркиране обаче се основава на елементи (вложени анотирани диапазони с атрибути), а не само на печатни ефекти, с отделна структура и маркиране. HTML прогресивно се придвижи в тази посока с CSS.

### Маркиране

HTML маркирането се състои от няколко ключови компонента, включително тези, наречени тагове (и техните атрибути), базирани на знаци типове данни, препратки към символи и препратки към обекти. HTML таговете най-често идват по двойки като **<h1>** и **</h1>**, въпреки че някои представляват празни елементи и затова не са сдвоени, например **<img>**.Първият таг в такава двойка е началният таг, а вторият е краен таг (те се наричат ​​също отварящи тагове и затварящи тагове).

Друг важен компонент е декларацията за тип на HTML документ, която задейства рендиране в стандартен режим.

Текстът между **<html>** и **</html>** описва уеб страницата, а текстът между **<body>** и **</body>** е видимото съдържание на страницата. Текстът за маркиране **<title>**Това е заглавие **</title>** дефинира заглавието на страницата на браузъра, показано в разделите на браузъра и заглавията на прозорците, а тагът **<div>** дефинира разделяне на страницата, използвано за лесно оформяне. Между **<head>** и **</head>** елемент **<meta>** може да се използва за дефиниране на метаданни на уеб страница.

Декларацията за тип документ **<!DOCTYPE html>** е за HTML5. Ако не е включена декларация, различните браузъри ще се върнат към „режим на странности“ за изобразяване.

### Елементи

HTML документите предполагат структура от вложени HTML елементи. Те са посочени в документа чрез HTML тагове, оградени в ъглови скоби по следния начин: **<p>.**

В простия общ случай степента на даден елемент се обозначава с двойка тагове: "начален таг**" <p>** и "краен таг**" </p>**. Текстовото съдържание на елемента, ако има такова, се поставя между тези тагове.

Етикетите могат също така да включват допълнително маркиране на етикети между началото и края, включително смес от тагове и текст. Това показва допълнителни (вложени) елементи като деца на родителския елемент.

Началният таг може също да включва атрибутите на елемента в тага. Те показват друга информация, като идентификатори за секции в документа, идентификатори, използвани за обвързване на информация за стила към представянето на документа, и за някои тагове като **<img>**, използвани за вграждане на изображения, препратката към ресурса на изображението в формат като този: **<img src="example.com/example.jpg">**

# Възможности за интегриране на Vue и Python/ Django/ DRF.

## Python

Python е програмен език, който е много лесен за научаване и ползване. Той е създаден да бъде език за програмиране, който да е удобен и ефективен за разработка на софтуер. Python е интерпретируем език, което означава, че програмният код не се компилира преди изпълнение, а се интерпретира от интерпретатора на езика в реално време.

Python има богата стандартна библиотека и множество модули и пакети, които могат да бъдат сваляни и използвани за различни нужди. Python има много използвани и популярни фреймуърци, като Django и Flask, които улесняват разработката на уеб приложения.

Python е много популярен за научни изследвания, тъй като има богата поддръжка на библиотеки за обработка на данни, машинно обучение и изкуствен интелект. Библиотеки като NumPy, Pandas, SciPy и TensorFlow са някои от най-използваните в тази област.

Python е крос-платформен език, което означава, че програмите, написани на Python, могат да бъдат използвани на различни операционни системи като Windows, macOS и Linux.

## Django

Django е популярен Python уеб фреймуърк за бързо и лесно разработване на уеб приложения. Той е създаден от Адриан Холовати и Саймън Уилсон през 2003 година и е наречен на името на групата "Джанго Рейнхарт" поради любовта на авторите към джаза.

Django е проектиран да бъде лесен за употреба, бърз и сигурен фреймуърк за уеб разработка. Той предоставя много инструменти и функции, които улесняват разработката на уеб приложения, като например автоматично генериране на административен интерфейс, ORM (Object-Relational Mapping) за работа с бази данни, механизъм за управление на сесии и много други.

Django е модулен фреймуърк, който позволява лесното добавяне на разширения и библиотеки. Той включва много стандартни библиотеки, които покриват широк спектър от функционалности, като работа със сесии, кеширане, форми, сигнали, съобщения и много други.

Django е известен със своята висока производителност и сигурност. Той има много вградени механизми за защита от атаки като CSRF (Cross-Site Request Forgery), SQL injection, XSS (Cross-Site Scripting) и др.

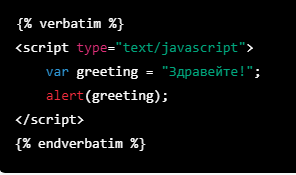
Django е много популярен в света на уеб разработката и се използва от много големи уеб сайтове като Instagram, Pinterest, Mozilla, Disqus и др.

В заключение, Django е мощен и лесен за употреба фреймуърк за бързо разработване на уеб приложения на Python. Той предоставя много инструменти и функции, които улесняват работата на уеб разработчиците и ги правят по-продуктивни.

### Тагът „Verbatim“

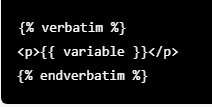
В Django шаблоните, тагът **{% verbatim %}** се използва за създаване на блок на код, който ще бъде изобразен директно като текст, без да бъде обработван от шаблонния език.

Този таг е полезен, когато искате да вмъкнете JavaScript код в шаблон, който може да съдържа специални символи, като например **{%** или **{{**, които са използвани от Django шаблоните. Когато използвате **{% verbatim %}** тага, всички символи в блока ще бъдат изобразени точно така, както са написани, без да бъдат обработвани от шаблонния език.



В този пример, блокът **{% verbatim %}** създава JavaScript код, който ще бъде изобразен точно така, както е написан, без да бъде обработван от Django шаблоните. Така, дори ако в кода има символи, които са използвани от Django шаблоните, те няма да бъдат интерпретирани като част от шаблона.

За да използвате тага **{% verbatim %}**, трябва да поставите кода или текста, който искате да се изобрази, между двата тага **{% verbatim %}** и **{% endverbatim %}**. Всичко, което е поставено между тези два тага, ще бъде изобразено точно така, както е написано, без да бъде обработвано от шаблонния език. Ето пример:



В този пример, тагът **{% verbatim %}** се използва за да се изобрази блок от HTML код, който съдържа символите **{{ variable }}**, които са специални за Django шаблоните. Блокът ще бъде изобразен точно така, както е написан, без да бъде обработван от шаблонния език.

Тагът **{% verbatim %}** може да се използва с други тагове и филтри в Django шаблоните, за да се създаде по-сложен код или текст, който да се изобразява точно така, както е написан.

# Технологии

## DRF

DRF е съкращение от Django REST framework, което е библиотека за създаване на RESTful API (API, базиран на REST архитектурата) с помощта на Django. Това е мощен инструмент, който позволява на разработчиците да създават уеб услуги с лесното използване на HTTP протокола.

Django REST framework (DRF) предоставя много инструменти и функции за създаване на уеб API. Тези инструменти включват сериализация на данни, валидация на данни, автоматично генериране на документация, авторизация и управление на потребители, аутентикация и много други. DRF също така предлага много вградени изгледи (views), които покриват много от общите изисквания на уеб услуги.

DRF е много популярен в областта на създаването на API и се използва от много големи уеб сайтове. Той предлага много гъвкавост и се интегрира перфектно с Django, което го прави много лесен за употреба.

В заключение, Django REST framework е мощна библиотека за създаване на RESTful API в Python, която предлага много инструменти и функции за лесна и ефективна уеб разработка. DRF е изключително популярен сред уеб разработчиците и се използва от много големи уеб сайтове като Mozilla, Dropbox, Red Hat и много други.

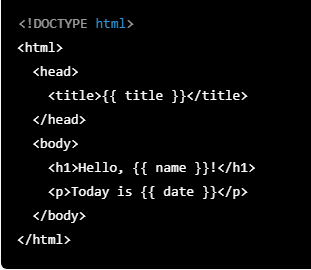
DRF предоставя множество изгледи (views), които покриват много от общите изисквания на уеб услуги. Тези изгледи включват готови имплементации за обработка на HTTP заявки, като GET, POST, PUT и DELETE. DRF също предлага много готови модули, като например Pagination, Filtering, Ordering и много други, които правят създаването на уеб услуги много по-лесно.

## 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:

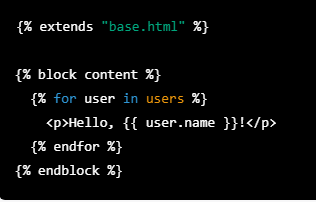


В този пример използваме Jinja за да изградим прости HTML шаблон. Използваме фигурните скоби **{{ }}** за да обозначим местата, където ще се поставят стойностите на променливите. В този случай променливите са **title**, **name** и **date**.

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:



В 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.

## REST

REST е архитектурен стил, който се използва за разработване на уеб услуги и уеб приложения. Той е основан на използването на различни HTTP методи, като GET, POST, PUT и DELETE, за достъп до ресурси.

REST се състои от няколко ключови компонента, включително:

* Ресурси - Това са обектите или услугите, към които може да се достъпи през HTTP заявки.
* HTTP методи - REST използва HTTP методите за определяне на типа на заявката, която се изпраща към сървъра. GET се използва за четене на ресурси, POST за създаване на нови ресурси, PUT за обновяване на съществуващи ресурси и DELETE за изтриване на ресурси.
* Представяне на ресурсите - Ресурсите се изпращат към клиента във формат, който е лесен за разбиране и обработка. Обикновено това е JSON или XML.
* URI - Всеки ресурс има уникален идентификатор във формата на URI (Uniform Resource Identifier). URI позволява на клиента да идентифицира ресурса, който иска да достъпи.
* REST е една от най-популярните архитектурни стилове в програмирането и е основен компонент в разработката на много уеб приложения и услуги.

## SOAP

SOAP е абревиатура от "Simple Object Access Protocol" (прост протокол за достъп до обекти) и е протокол за обмен на данни в мрежата, който се използва в програмирането. SOAP е базиран на XML и използва HTTP за пренос на съобщенията.

SOAP позволява на приложенията да извикват функции на други приложения или услуги по мрежата, като използват стандартизиран формат на съобщенията. SOAP се използва предимно в системите за обмен на данни между компютърни системи и е често използван за разработка на услуги на основата на SOA (Service-Oriented Architecture).

SOAP се състои от няколко ключови компонента, включително:

* Envelope (Обвивка) - Това е основният елемент на SOAP съобщението и съдържа информация за версията на SOAP и съдържанието на съобщението.
* Header (Заглавие) - Това е опционален елемент на SOAP съобщението и съдържа допълнителна информация за съобщението, като например сигурността или управлението на транзакции.
* Body (Тяло) - Това е елемент на SOAP съобщението, който съдържа реалното съдържание на съобщението.
* Fault (Грешка) - Това е елемент на SOAP съобщението, който съдържа информация за грешките, които могат да възникнат по време на изпълнението на съобщението.

SOAP е стандартизиран протокол, който се използва широко в програмирането за обмен на данни между системи. Въпреки че SOAP е бил използван широко в миналото, в последно време е набелязана тенденция на преминаване към по-леки и бързи протоколи като REST.

## REST срещу SOAP

REST (Representational State Transfer) и SOAP (Simple Object Access Protocol) са два протокола за комуникация в мрежата, които се използват за изграждане на услуги на основата на SOA (Service-Oriented Architecture).

REST и SOAP имат различни подходи към изграждането на услуги и са подходящи за различни видове приложения. Ето някои от главните разлики между REST и SOAP:

* **Архитектура:** REST е архитектурен стил, който се базира на HTTP протокола и използва HTTP методите (GET, POST, PUT, DELETE и др.) за работа с ресурсите. SOAP е протокол, който използва XML формат за обмен на съобщения между клиенти и услуги.
* **Сложност:** REST е по-лесен за разбиране и използване, тъй като използва прости HTTP методи и форматиране на данните, като JSON или XML. SOAP е по-сложен и изисква по-голяма конфигурация и поддръжка.
* **Скорост:** REST е по-бърз от SOAP, тъй като използва по-малко ресурси и не изисква толкова голям обем от съобщения.
* **Съвместимост:** REST е по-съвместим с уеб браузъри и уеб услуги, тъй като използва HTTP протокола, който е поддържан от всички браузъри. SOAP е по-подходящ за използване в корпоративни приложения, тъй като предоставя по-голяма сигурност и контрол върху изпращаните съобщения.

Въпреки че REST и SOAP имат различни подходи, те могат да бъдат използвани за реализиране на сходни функционалности. Изборът на протокол зависи от конкретните нужди на приложението и услугата, която трябва да бъде изградена.

# Проектиране на необходимата функционалност:

## АНАЛИЗ НА ЗАДАНИЕТО

Заданието за дипломния проект предполага, реализиране на софтуерна платформа (наречена „Борса за учебници“). Платформата има необходимата функционалност за регистриране на потребители, оторизиране на достъпа до платформата, регистриране на заявки за предлагане и/или търсене на учебници.

## МОДЕЛИ

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 избрала БД MySQL. Всеки модел в този файл е клас, който описва съответна таблица в БД. Например, моделът Klas описва структурата на таблица, която ще се съхраняват отделните отделните класове.

## ЕЛЕМЕНТ ОТ JAVASCRIPT КОДА

const App = {

data() {

return {

status: 0, //0 - показва се списък; 1 - показва се единична обява

}

},

methods: {

reloadItem(){

if (this.status == 0){this.status = 1} else {this.status = 0}

},

triggerModal(){

$('#ModalReview').trigger('focus')

},

},

created: function(){

this.status

}

}

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

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.

## ТЕМПЛЕЙТИ

В началото на всеки темплейт зареждаме статичните файлове

{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta charset="utf-8">  
 <meta content="width=device-width, initial-scale=1.0" name="viewport">  
  
 <title>Борса за учебници</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">  
  
 *<!-- Template Main CSS File -->* <link href="{% static 'assets/css/style.css' %}" rel="stylesheet">  
  
 *<!-- =======================================================  
 \* Template Name: Gp - v4.10.0  
 \* Template URL: https://bootstrapmade.com/gp-free-multipurpose-html-bootstrap-template/  
 \* Author: BootstrapMade.com  
 \* License: https://bootstrapmade.com/license/  
 ======================================================== -->*</head>

След което за всеки елемент от интерфейса създаваме съответната му HTML структура и командите към шаблонизатора. Например модалният прозорец за отзиви:

<body>  
 *<!-- ======= модалени прозорци ======= -->  
 <!-- ======= отзив ======= -->* <div class="modal fade"  
 id="ModalReview"  
 data-backdrop="static"  
 tabindex="-1"  
 role="dialog"  
 aria-labelledby="ModalReviewTitle"  
 aria-hidden="true">  
 <div class="modal-dialog modal-dialog-centered modal-xl modal-dialog-scrollable" role="document">  
 <div class="modal-content">  
 <div class="modal-header">  
 <h5 class="modal-title" id="ModalReviewTitle">Отзив</h5>  
 <button class="close" type="button" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">×</span></button>  
 </div>  
 <div class="modal-body">  
  
 </div>  
 <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>  
 *<!-- ======= редактиране на обява ======= -->*

Това е интеграцията на Vue на ниво HTML.

*<!-- интеграция на VUE -->* <script src="https://code.jquery.com/jquery-3.4.1.min.js" crossorigin="anonymous"></script>  
 <script>  
 $('#ModalReview').on('shown.bs.modal', function () {  
 $('#ModalReview').trigger('focus')  
 })  
 </script>  
 <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/borsa\_list.js' %}"></script>  
  
</body>

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

* + - Списък на офертите;
    - Разглеждане на конкретна оферта;
    - Отзиви към конкретна оферта.

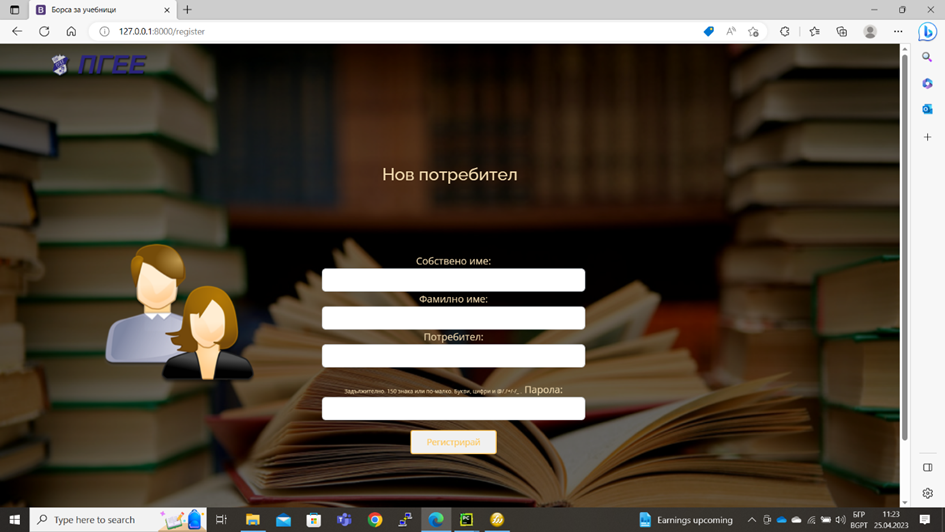
# Разработка на минимална версия на горното приложение, като се реализират:

## Регистриране на нови потребители:

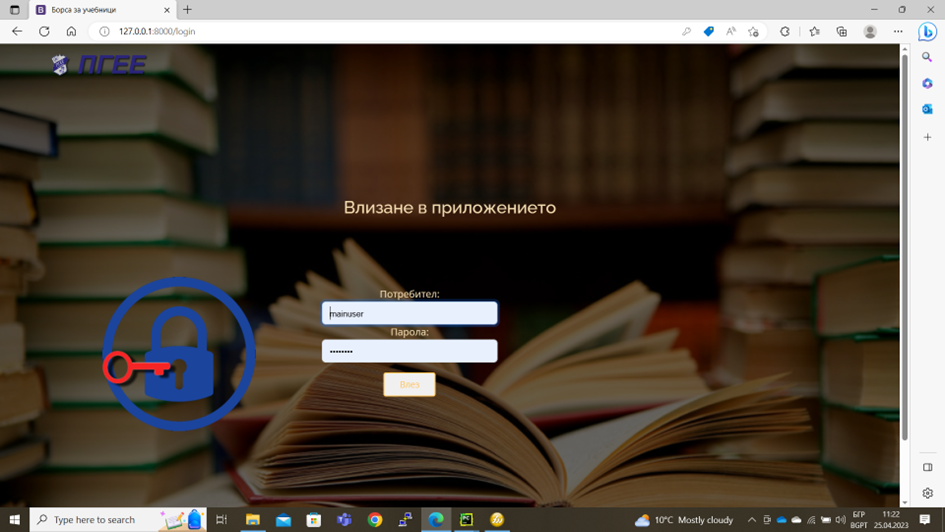
Интерфейсът за регистрация и орторизация на потребители използва вградената в Django система. Необходимо е само да наследим правилните модели на форми и да изградим съответните изгледи. Показаният по-долу код илюстрира този процес:

**from** django.contrib.auth.forms **import** AuthenticationForm  
**from** django.contrib.auth.models **import** User  
**from** django **import** forms  
  
  
**class** AuthUserForm(AuthenticationForm, forms.ModelForm):  
 **class** Meta:  
 model = User  
 fields = (**'username'**, **'password'**)  
  
 **def** \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 **for** field **in** self.fields:  
 self.fields[field].widget.attrs[**'class'**] = **'form-control'  
  
  
class** RegisterUserForm(forms.ModelForm):  
 **class** Meta:  
 model = User  
 fields = (**'first\_name'**, **'last\_name'**, **'username'**, **'password'**)  
  
 **def** \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 **for** field **in** self.fields:  
 self.fields[field].widget.attrs[**'class'**] = **'form-control'  
  
 def** save(self, commit=**True**):  
 user = super().save(commit=**False**)  
 user.set\_password(self.cleaned\_data[**"password"**])  
 **if** commit:  
 user.save()  
 **return** user

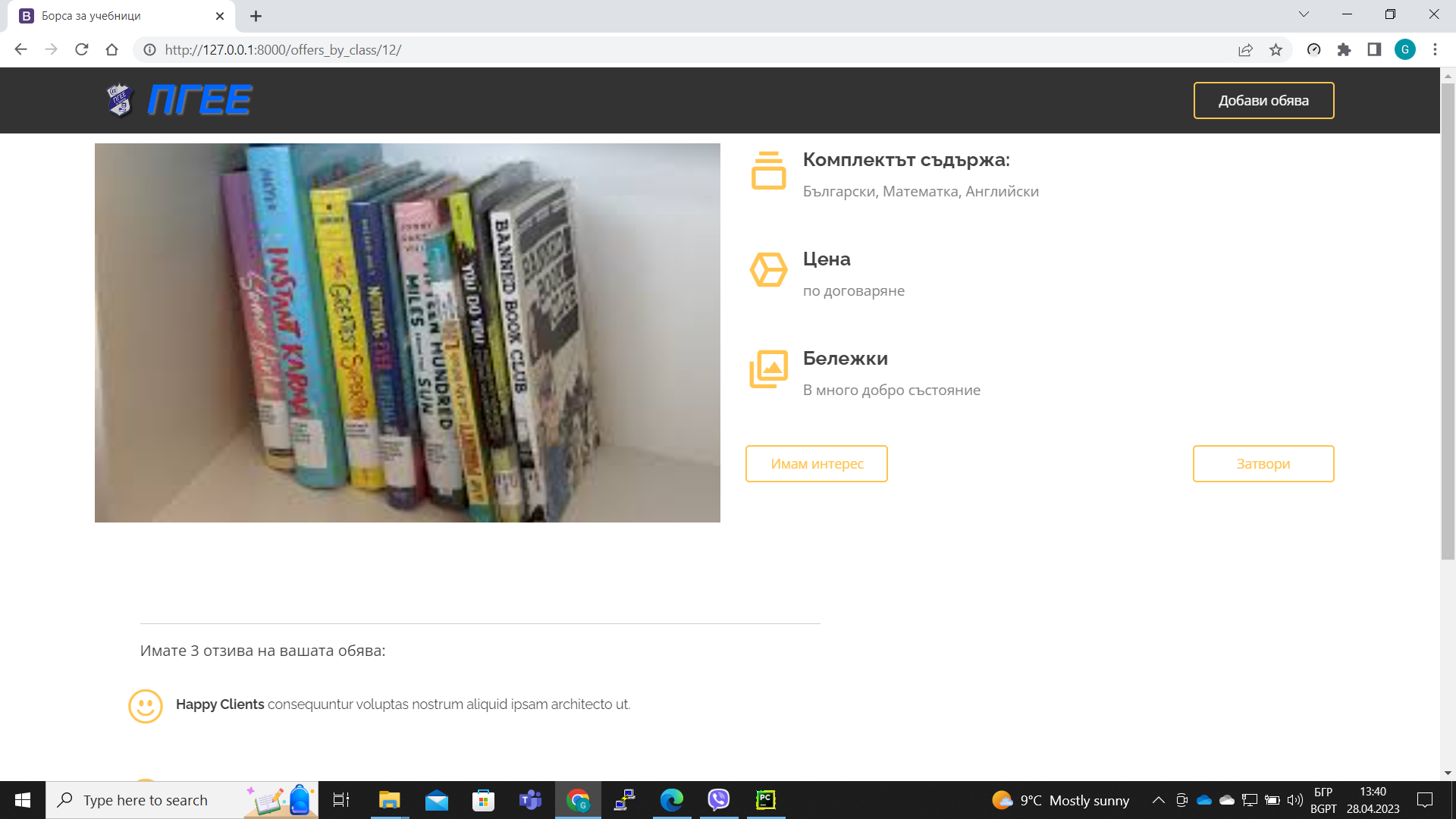
*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# регистрация и оторизация на потребители  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****class** MyprojectLoginView(LoginView):  
 template\_name = **'main/login.html'** form\_class = AuthUserForm  
 success\_url = reverse\_lazy(**'home'**)  
  
 **def** get\_success\_url(self):  
 **return** self.success\_url  
  
  
**class** RegisterUserView(CreateView):  
 model = User  
 template\_name = **'main/register.html'** form\_class = RegisterUserForm  
 success\_url = reverse\_lazy(**'home'**)  
 success\_msg = **'Регистриран е нов потребител'  
  
 def** form\_valid(self, form):  
 form\_valid = super().form\_valid(form)  
 username = form.cleaned\_data[**"username"**]  
 password = form.cleaned\_data[**"password"**]  
 aut\_user = authenticate(username=username, password=password)  
 login(self.request, aut\_user)  
 **return** form\_valid  
  
  
**class** MyProjectLogout(LogoutView):  
 next\_page = reverse\_lazy(**'home'**)



## Оторизиране достъпа на потребител:



# Разглеждане на регистрираните обяви/заявки

****

# Заключение:

Това е моят дипломен проект, надявам се занапред да имам още успешни задачи. Благодарности към господин Бориков, защото, ако не беше неговата помощ проектите ни нямаше да бъдат толкова добри! Трябва да се отбележи, че това е реално работещо приложение, което е тествано само върху тестови сървър. Предстои да се качи на реален хостинг (хостингът на училището), което ще позволи на реалните потребители да го достъпват и ползват. А както знаем, тестът в реална работна среда е най-сигурния тест.

Надявам се, разработеното от нас приложение, да спечели интереса на потребителите и да намери широко приложение.

# Списък на използваната литература:

1. <https://softwareacademy.bg/index.php?q=info&info=django+now/>

2. <https://softuni.bg/blog/advantages-of-django-framework>

3. <https://www.pro-soft.bg/web-saitove/django/>

4. <https://bg.wikipedia.org/wiki/REST>

5. <https://www.python.org/>

6. <https://chibisov.github.io/drf-extensions/docs/>

7. <https://bg.altlibs.net/lib/django-drf-filepond>

8. <https://softuni.bg/blog/what-is-html5-and-how-it-works>

9. <https://html.start.bg/%D0%9A%D0%B0%D0%BA%D0%B2%D0%BE+%D0%B5+HTML%3F-19832>

10. <https://www.sourcetrail.com/bg/python/django/solved-django-serializer-method-field-read-write/>

11. <http://javac.bg/?p=522>

12. <https://www.django-rest-framework.org/api-guide/serializers/>