# Съдържание

[Съдържание 1](#_Toc133496430)

[1 Въведение 4](#_Toc133496431)

[2 Цел на проекта 5](#_Toc133496432)

[3 Характеристика 7](#_Toc133496433)

[4 Какво е Python/ Django и защо е най-добрият програмен език за нашият проект? 8](#_Toc133496434)

[4.1 Какво е Django? 8](#_Toc133496435)

[4.2 Какво се случва, когато някой изпрати заявка за сайт до вашият сървър, посредством Django? 8](#_Toc133496436)

[4.3 Какви са свойствата на Django? 9](#_Toc133496437)

[4.4 Какви са предимствата на Django? 9](#_Toc133496438)

[5 Какво е REST и защо избираме тази технология? 10](#_Toc133496439)

[5.1 Какво е API? 11](#_Toc133496440)

[5.2 Какво е REST API? 12](#_Toc133496441)

[5.3 Защо Django REST API? 13](#_Toc133496442)

[6 Какво е библиотеката Django REST Framework? 13](#_Toc133496443)

[7 Заключение за DRF и REST 15](#_Toc133496444)

[8 Какво е фреймуърка VUE и за какво го използваме? 16](#_Toc133496445)

[8.1 Какво е Vue.js? 16](#_Toc133496446)

[8.1.1 API стилове: 18](#_Toc133496447)

[8.1.2 API за опции: 18](#_Toc133496448)

[8.1.3 Композиционен API: 19](#_Toc133496449)

[8.2 Кое да изберем?​ 19](#_Toc133496450)

[8.3 Характеристика на Vue 20](#_Toc133496451)

[8.3.1 Компоненти: 20](#_Toc133496452)

[8.3.2 Шаблони: 21](#_Toc133496453)

[8.3.3 Реактивност: 21](#_Toc133496454)

[8.3.4 Преходи: 22](#_Toc133496455)

[8.3.5 Маршрутизиране: 22](#_Toc133496456)

[8.4 Предимства: 24](#_Toc133496457)

[8.4.1 Документация и разработка 24](#_Toc133496458)

[8.4.2 Преизползване на кода 24](#_Toc133496459)

[8.4.3 Разработка на апликации за всички платформи 25](#_Toc133496460)

[8.4.4 Интеграции 25](#_Toc133496461)

[8.4.5 Недостатъци: 25](#_Toc133496462)

[8.4.6 Екосистема на Vue: 25](#_Toc133496463)

[8.4.7 Библиотеки на Vue: 26](#_Toc133496464)

[9 Защо избираме базата данни MySQL? 26](#_Toc133496465)

[9.1 Какво са бази данни и как се управляват те? 26](#_Toc133496466)

[9.2 MySQL базите данни са релационни. 26](#_Toc133496467)

[9.3 SQL и MySQL едно и също нещо ли са? 26](#_Toc133496468)

[9.4 MySQL е софтуер с отворен код. 27](#_Toc133496469)

[10 Защо таксономията на Блум е най-добрият начин за оценяване на тестове? 27](#_Toc133496470)

[10.1 Когнитивна област 28](#_Toc133496471)

[10.2 Ефективна област 30](#_Toc133496472)

[10.3 Психомоторна област 31](#_Toc133496473)

[11 Какво е HTML и CSS и защо се нуждаем от него? 32](#_Toc133496474)

[11.1 Какво означава HTML? 32](#_Toc133496475)

[11.2 Как работи HTML? 33](#_Toc133496476)

[11.3 Какво е HTML5 и какви са неговите предимства? 33](#_Toc133496477)

[11.4 Какво е CSS и как помага на HTML да блести? 34](#_Toc133496478)

[11.5 Защо да изучаваш HTML и CSS? 34](#_Toc133496479)

[12 Реализация 35](#_Toc133496480)

[12.1 Първи стъпки на проекта: 35](#_Toc133496481)

[12.2 Създаване на приложение: 35](#_Toc133496482)

[12.3 Използване на шаблони: 36](#_Toc133496483)

[12.4 Настройване на Django за работа с БД MySQL: 36](#_Toc133496484)

[12.5 Настройване на MySQL база данни: 36](#_Toc133496485)

[12.6 Настройване на Django проекта: 36](#_Toc133496486)

[12.7 Подготвяме и изпълняваме миграциите: 37](#_Toc133496487)

[12.8 Създаване на моделите: 37](#_Toc133496488)

[12.9 Създаване на администратор: 38](#_Toc133496489)

[12.10 Регистриране на таблица: 38](#_Toc133496490)

[12.11 Извеждане на данни от БД: 38](#_Toc133496491)

[12.12 Реализиране на тестовата система: 39](#_Toc133496492)

[13 Списък на използваната литература 42](#_Toc133496493)

# Въведение

Същността на човешкото учене обикновено се дефинира като усвояване на определени знания и обусловени от тях действия и постъпки в определени ситуации. През целия живот на човека протичат процеси на учене – съзнателно и планомерно или недостатъчно осъзнато и непреднамерено. Ученето предполага не откъслечни, макар и успешни, еднократни актове със случаен или мимолетен характер, а системни и сравнително устойчиви действия. Те засягат личността и същевременно ѝ дават възможност за относително трайна изява в тях. Образователният модел носи в себе си ценностите на хуманистичната психология – достойнството, правата и свободите на личността, принципи на съвременната педагогика за отношение към децата и съвременната психология за процеса на учене.

Процесът на учене е съпътстван както с получаването на нова знания, така и с упражненията, които ще помогнат за качественото запаметяване на новата за ученика информация. Обучението от своя страна обаче не се ограничава само с времето прекарано в училище. За постигане на отличен резултат, ученикът трябва да притежава както интерес, така и желание сам да търси и намира нова информация. Умението за самостоятелно учене е компетентност, която се формира в процеса на обучение и учене по всички учебни предмети в целия цикъл на обучение в училище. Това изисква постоянно внимание от страна на учителите, за да се формират основните ѝ елементи и да се развият като умения и най-вече като положително отношение на учениците към ученето – любопитство към нови решения, приложение на вече наученото в нови ситуации, нагласа за учене и постоянство в този процес.

Умението за самостоятелно учене включва както получаването, обработката и усвояването на нови знания и умения, така и търсенето и използването на различни насоки, наръчници и справочници, които да улеснят усвояването. Това дава възможност на учещите се да надграждат върху предишните познания и житейски опит с цел използване и прилагане на знанията и уменията в различен контекст.

Самостоятелното учене е цялостен психолого-педагогически подход за осъществяване на процес на обучение, в центъра на който е поставен ученикът. „Независимото“ учене не означава абсолютна индивидуална свобода на избора, нито учене, изцяло извън общия контекст на образователната система.

Моделът изисква създаването на цялостна, гъвкава, отворена активност, в съответствие с индивидуалните особености на учещите се. Основното изискване е осигуряване на достъп до разнообразни ресурси и източници на познание за всекиго. Преподавателят направлява и контролира ученето на своите ученици индиректно, като изпълнява ролята на партньор, посредник, помощник.

Програмата, която изработих съдържа голям набор от тестове с въпроси както с избираем, така и със свободен отговор с цел за проверка на вече изучен от учениците учебен материал. Чрез тези тестове обучаващите се ще имат възможност да проверят знанията си в дадена сфера, както и да самоанализират както вярно, така и грешно маркираните си отговори. По този начин те ще имат възможността автоматизирано да получават оценка и в последствие самостоятелно да работят върху пропуските в знанията си.

# Цел на проекта

Преди ясно да дефинираме целта на проекта, трябва да се замислим какви бихме искали да бъдат резултатите от работата ни. В случая ние бихме желали по възможно най-ефективен, интерактивен и будещ интерес и желание за черпене на знания у учениците начин да имаме възможността да наблюдаваме постигнат прогрес в качественото запаметяване на заучения учебен материал от страна на учениците, както и постигнат напредък в успеха на обучаващите се след изграждането на програма, достатъчно ефективна да помогне за затвърждаването на знанията, получени по време на учебните занятия. Други важни фактори, като одобрението и съгласната за осъществяването на подобен проект от страна на учителите, които преподават в даденото учебно заведение също има важна роля в определянето на целта на проекта, както и начините с които той би бил реализиран. Не на последно място при определянето на главната причина за създаването на проекта е съобразяването с нуждите, желанията, способностите на учениците също както и облагите, които хората, използващи програмата биха могли да имат.

След като вече сме определили най-значимите фактори при изграждането на програма, търсеща приложение в обучението на учащите се, бихме могли да заявим че, целта на проекта е разработване на функционален модул, който съдържа бланка с тестови въпроси, базирани на вече изучен учебен материал, които да бъдат полезни на учениците при обучението им. Бланката съдържа въпроси от различно естество, както с избираем, така и със свободен отговор, които имат за цел да стимулират умствената дейност на учениците, да зародят способност за критично мислене в процеса на решаване на задача, да помогнат при затвърдяването и преговора на вече изучен учебен материал и да развият умения за учене при младежите. Уменията за учене се формират у децата веднага след раждането им, но могат несъзнателно да бъдат изгубени, в случай, че не бъдат редовно развивани. Ключово за формирането на умения за учене е притежаването на мотивация за учене, осъзнаването на ученето като процес с отделни, разграничим стъпки и наличието на инструменти, които да помогнат на учещия се да разбере същността на процеса учене.

Уменията за учене са важни по няколко причини:

* Първо, развитите умения за учене подобряват представянето на учениците по всеки един предмет и помагат за създаването на позитивна обучителна среда.
* Второ, те осигуряват на учениците инструменти за учене през целия живот, които подпомагат за успешното полагане на всякакъв вид изпити и тестове.
* Трето, тези умения са емпирично различни от общата интелигентност и дори могат да компенсират дефицит в общата грамотност и/или предварителни познания по дадена тема при решаване на проблеми.

Обикновено в категорията “умения за учене” включваме основните инструменти, които са необходими за учене в училище или университет, като например способността на ученика да води записки, да управлява времето си или да се концентрира върху дадена задача. Същинските умения за учене обаче включват не само прилагането на тези инструменти, но и осъзнатост на процеса на учене. С други думи, уменията за учене са насочени към развиване на мислене от по-висш порядък, което позволява да наблюдаваме и оценяваме процесите, чрез които учим. Именно с тези цел, програмата дава възможност на всеки ученик да провери своите знания върху даден учебен материал и да определи нивото си на качествено владеене на дадени умения с теоретична или практическа насоченост.

След решаването на един такъв тест и анализирането както на вярно посочените отговори, така и сгрешените такива, ученикът може да направи собствена самооценка, да открие пропуските в познанията си и следователно да работи за отстраняването им. За помощ ученикът може да се обърне с молба към своя учител , който да ясно и точно да изясни неразбирането при определена тема, а ученикът от своя страна в последствие да обърне повече внимание на пропуските си и да ги поправи, за да постигне отлични резултати през учебно време.

Програмата е публикувана онлайн и налична за всеки ученик в ПГЕЕ, град Банско. Тя може да бъде намерена в приложението "електронна библиотека" от сайта на ПГЕЕ Банско и ще бъде активна по всяко време в денонощието, с цел всеки един ученик да гимназията да има лесен и удобен достъп до нея и следователно да има възможност да се упражнява и провери знанията си по всяко време, удобно за него самия.

# Характеристика

За да достигнем до резултата на проекта ще използваме известни и надеждни езици, библиотеки, технологии и база данни.

Използваме технологията REST, за да имаме по-бърза и опростена връзка за проектиране на мрежовото приложение. В последствие с помощта на Django REST Framework (DRF), ние реализираме нашият REST проект като предостави набор от мощни инструменти, за да улесни процеса на разработка. С Vue разработваме на потребителски интерфейс с Python/Django. Проектираме необходимите таблици в базата данни MySQL на сайта и разработваме съответните модели и сереализатори. Принципът за оценяване, използван в тестовите задачи се подчинява пряко на „Таксономията на Блум“.

# Какво е Python/ Django и защо е най-добрият програмен език за нашият проект?

## Какво е Django?

Django е безплатна технологична уеб framework с отворен код, написана на Python. Технологичните уеб рамки ви позволяват да разработвате уебсайтове по-бързо и по-лесно. Когато изграждате уебсайт вие винаги се нуждаете от един и същи набор от компоненти: начин да вградите потребителска заверка (регистрация, вписване, отписване), контролен панел за вашият сайт, формуляри, системи за качване на файлове и т.н. Технологичните рамки съществуват за да ви спестят „преоткриването на колелото“ всеки път когато създавате нов сайт.

## Какво се случва, когато някой изпрати заявка за сайт до вашият сървър, посредством Django?

Нека погледнем от по-близо сървърите, за да разберем за какво точно е Django. Първото нещо, което сървърът има нужда да знае е, че вие искате той да ви покаже дадена уеб страница. Когато потребителската заявка достигне уеб сървър, тя се предава към Django рамката, която се опитва да разбере какво всъщност е поискано от страна на потребителя. За целта тя взема адреса на уеб страницата и се опитва да разбере какво да направи от тук нататък. Тази част се извършва от т.нар. Django URL Resolver. Не е нещо сложно – за целта се използва набор от шаблони и се опитва да открие съвпадение с изисквания URL адрес.

Django проверява шаблоните от край до край и ако намери съвпадение предава заявката на асоциирана функция, която се нарича „view“. Всички интересни неща се случват в тази view функция, например потребителската заявка съдържа искане за промяна в някои налични данни. View функцията проверява ако имате право да извършвате изисканото действие и ако имате – го извършва. След това view генерира отговор и Django го изпраща обратно към потребителския браузър.

## Какви са свойствата на Django?

* + - Django предоставя лек и самостоятелен уеб сървър за разработка и тестване;
    - Django използва сериализация на формулярите и валидираща система, която превежда между HTML формулярите и стойностите, които се използват в базите данни.
    - Има шаблонизатор, който използва концепцията за наследяване от обектно-ориентираното програмиране (ООП);

Основната Django дистрибуция върви в комплект с приложения в своя „contrib“ пакет, които включват неща като инструменти за генериране на RSS и Atom syndication feeds, допълнителна технологична рамка, която позволява на една Django инсталация да поддържа едновременно до няколко уебсайта, всеки от които със свое съдържание и приложения, инструменти за генериране на Google Sitemaps и още много други.

## Какви са предимствата на Django?

Ако изучавате уеб разработка с Python, то най-доброто направление за развитие на уменията, е да усвоите Django. Django е безплатен **framework**, който е с отворен код. Специфичното при него е, че е написан изцяло на python и по този начин ви позволява да изграждате уебсайтове чрез Python много по-бързо. Разбира се предимствата на Django не свършват дотук.

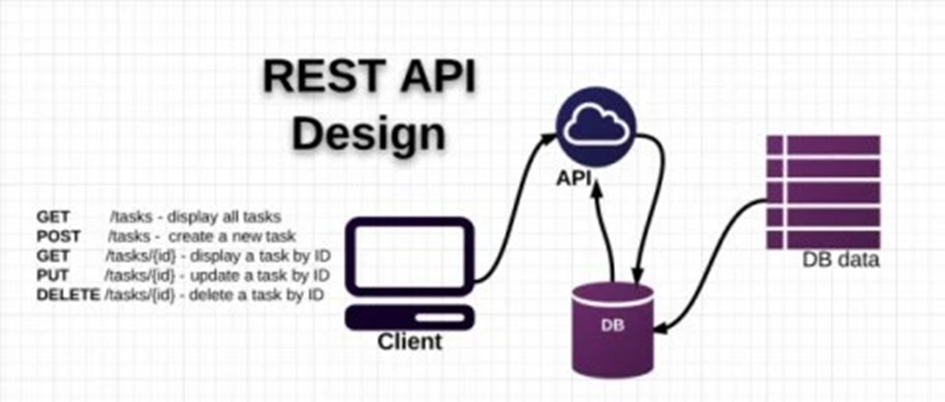
Ето още предимства:

* **Бърз** - Django е проектиран по начин, който помага на разработчиците да създават дадено приложение възможно най-бързо. От идеята, през процеса на разработка, до завършения продукт, Django позволява да се реализира един проект, едновременно по един бюджетно-ориентиран начин, без това да се отрази на функционалността на продукта
* **Пълен набор инструменти** - Django е подходящ за всеки етап от разработката на дадено приложение – потребителска верификация, администриране на съдържанието, RSS feed настройки и още много други. По този начин той е напълно достатъчен за процеса на разработка от начало до край.
* **Сигурност** - Django гарантира с една идея повече сигурност за вашия продукт и за неговите клиенти. Това се гарантира още на ниво “разработка”, като намалява възможностите за пропуски от страна на разработчиците до минимум. Някои от най-честите грешки се случват по отношение на прилагането на SQL injection, крос-сайт скриптирането и подобно.
* **Гъвкавост** - Без значение дали става въпрос за управление на съдържанието, специфични компютърни платформи или приложения, които ще се използват от големи организации – Django може да отговори на всяка от тези нужди. Това демонстрира ясно неговата гъвкавост.

# Какво е REST и защо избираме тази технология­­­?­­­

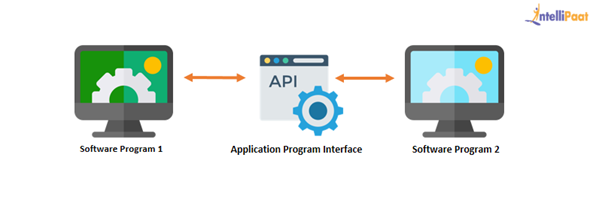
Точно както ние, хората, взаимодействаме с компютрите с помощта на потребителски интерфейс, уеб приложенията или софтуерните програми също трябва да взаимодействат помежду си. За да се направи това, се създава различен вид интерфейс.

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?

Ето няколко ключови момента, поради които разработчиците избират Django REST Framework:

* DRF е лесен за използване, но същевременно ефективен и усъвършенстван.
* Той предоставя уникални функции за изграждане на API за уеб сърфиране.
* Той предоставя редица готови схеми за удостоверяване, които могат да идентифицират идентификационните данни, с които е направена заявката.
* Той предлага функционалност за сериализиране, която поддържа както ORM (обектно-релационното картографиране е техника, която преобразува данни между системи с несъвместим тип, използвайки обектно-ориентирани езици за програмиране), така и източници на данни, които не са ORM.
* DRF предоставя функцията за персонализиране. Ако не се нуждаем от по-мощни функции, можем просто да използваме обикновени изгледи, базирани на функции.
* DRF е подкрепен от набор от обширна документация и страхотна общност.
* Той придоби популярност и защото процесът на уеб разработка се извършва с език за програмиране Python.

# Какво е библиотеката Django REST Framework?

Django REST Framework или DRF е качествена и лесна за използване платформа, която позволява версия на API за уеб сърфиране. DRF е изграден върху Django. Той предоставя набор от разнообразни инструменти, за да улесни процеса на разработка на RESTful API.

REST e архитектура, където всяка информация или понятие, която може да се обособи, се разглежда като ресурс и се идентифицира чрез идентификатор на ресурси URI. В много отношения, World Wide Web сама по себе си, основана на HTTP, може да се разглежда като REST архитектура. Общо казано, може да се гледа на интернет страница на като ресурс с ресурсен идентификатор URI = URL RESTful приложения използват HTTP заявки за публикуване на данни (създаване и / или обновяване), четене на данни (заявки), и изтриване на данни. По този начин REST използва HTTP за всички четири CRUD (Create/Read/Update/Delete) операции. REST е лека алтернатива на механизми като RPC (отдалечено извикване на процедури) и уеб услуги (SOAP, WSDL, и др.). Въпреки, че е прост, REST е пълнофункционален, всичко което може да се направи с Web Services, може да се направи и с RESTful архитектура. REST не е “стандарт“. Никога няма да има W3C recommendataion за REST. Съществуват програмни рамки за REST, но REST е толкова прост, че най-често проектите създават „своя собствена“ със стандартни функции (в езици като Perl , Java или C #).

Използвайки методологията на REST за уеб услуга включва използването на следните компоненти:

* XML/jSON: формат за представяне на ресурса.
* HTTP: методите GET, HEAD, POST, PUT и DELETE посочват какви действия да бъдат предприети.
* URI: URI, обикновено URL, е идентификатор на ресурс, който локализира уеб услугата.

REST сървисите са лесни за имплементация и консумиране. Работят върху HTTP/HTTS протокол. Поддържат различни видове формат на данните XML, JSON, HTML, др. Операциите са stateless, т.е. не се съхраняват никакви данни между отделните REST –заявки с цел по-нататъшно използване.

REST сървисите могат да бъдат консумирани от всеки клиент - дори браузър ползвайки Java Script. Те са много лесни за имплементиране. В случай, че не се ползва XML-формат на данните, се избягва тежкото XML-парсване.

REST е добър за:

1. уеб услуги;
2. ресурси, които не са в XML-формат;
3. предаване на данни използвайки Интернет, създаване на съдържание в уеб браузъра от много различни източници (т.е. викайки различни сървиси);

Някои причини, поради които може да искате да използвате REST framework:

* + API за сърфиране в мрежата е огромна полза за разработчиците.
  + [Правила за удостоверяване](https://www.django-rest-framework.org/api-guide/authentication/), включително пакети за [OAuth1a](https://www.django-rest-framework.org/api-guide/authentication/#django-rest-framework-oauth) и [OAuth2](https://www.django-rest-framework.org/api-guide/authentication/#django-oauth-toolkit).
  + [Сериализация](https://www.django-rest-framework.org/api-guide/serializers/), която поддържа както [ORM](https://www.django-rest-framework.org/api-guide/serializers#modelserializer).
  + Възможност за персонализиране докрай - просто използвайте [обикновени изгледи, базирани на функции,](https://www.django-rest-framework.org/api-guide/views#function-based-views) ако не се нуждаете от [по-](https://www.django-rest-framework.org/api-guide/generic-views/)[ефективните](https://www.django-rest-framework.org/api-guide/viewsets/)[функции](https://www.django-rest-framework.org/api-guide/routers/).
  + Обширна документация и [голяма подкрепа от общността](https://groups.google.com/forum/?fromgroups#!forum/django-rest-framework).
  + Използва се и се ползва с доверие от международно признати компании, включително [Mozilla](https://www.mozilla.org/en-US/about/), [Red Hat](https://www.redhat.com/), [Heroku](https://www.heroku.com/) и [Eventbrite](https://www.eventbrite.co.uk/about/).

# 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.

# Какво е фреймуърка VUE и за какво го използваме?

## 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" \t "_blank). 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.

# Защо таксономията на Блум е най-добрият начин за оценяване на тестове?

Провеждането на изпит върху определен учебен материал е систематично придобиване на информация и обратна връзка по отношение на използването, качествата и въздействието на обект, програма или процес. От друга страна, процесът на оценяване е измерване на степента на постигане на очакваните учебни резултати. Поставянето на оценка на ученика е систематично придобиване на информация и обратна връзка по отношение на използването, качествата и въздействието на обект, програма или процес, използвайки критерий регулиран чрез набор от стандарти. Двата процеса са взаимно допълващи се и необходими при обективната оценка на знанията и уменията на ученика. Принципът на оценяване, който е използван при направата на тестовете в програмата, пряко се подчинява на "Таксономията на Блум".

Таксономията на Блум е набор от три йерархични модела, използвани за класифициране на образователните учебни цели в нива на сложност и специфичност. Трите списъка обхващат учебните цели в когнитивната, ефективната и психомоторната област. Списъкът с когнитивни домейни е основният фокус на повечето традиционно образование и често се използва за структуриране на учебни цели, оценки и дейности в учебната програма.

Моделите са кръстени на Бенджамин Блум , който председателства комисията от педагози, които създават таксономията. Той също така редактира първия том на стандартния текст, Таксономия на образователните цели: Класификация на образователните цели.

Таксономията на Блум се състои от 3 области: когнитивна област, ефективна област и психомоторна област

## Когнитивна област

Когнитивната област от таксономия на Блум е най-добрият начин за правене на тестове. Всеки въпрос трябва да има ниво на трудност както в таксономията на Блум, въпросите с най-ниско оценяване и най-лесни по трудност са знанията са на дъното на пирамидата. Знанието включва разпознаване или запомняне на факти, термини, основни понятия или отговори, без непременно разбиране какво означават. Някои характеристики могат да включват:

* Познаване на спецификата - терминология, конкретни факти
* Познаване на начините и средствата за справяне със специфики - конвенции, тенденции и последователности, класификации и категории
* Познаване на универсалиите и абстракциите в дадена област - принципи и обобщения, теории и структури

Например:

* Назовете три често срещани сорта ябълки.

Второто ниво на пирамидата и малко по-високо оценените въпроси е разбирането. Разбирането включва демонстриране на разбиране на факти и идеи чрез организиране, обобщаване, превод, обобщаване, даване на описания и посочване на основните идеи.

Например:

* Обобщете отличителните характеристики на ябълка Golden Delicious и ябълка Granny Smith.

Третото ниво на пирамидата е за оценяване на това което можем да приложим, като вече сме показали знания и разбиране. Приложението включва използване на придобитите знания за решаване на проблеми в нови ситуации. Това включва прилагане на придобитите знания, факти, техники и правила. Обучаемите трябва да могат да използват предишни знания за решаване на проблеми, идентифициране на връзки и взаимоотношения и как те се прилагат в нови ситуации.

Например:

* Ще предотвратят ли ябълките скорбут, заболяване, причинено от недостиг на витамин С?

Четвъртото и много по-оценени въпроси и ниво на пирамидата е анализа. Анализът включва изследване и разделяне на информацията на съставни части, определяне как частите са свързани една с друга, идентифициране на мотиви или причини, правене на изводи и намиране на доказателства в подкрепа на обобщения. Характеристиките му включват:

* + Анализ на елементите
  + Анализ на взаимоотношенията
  + Анализ на организацията

Например:

* Сравнете и съпоставете четири начина за сервиране на храни, приготвени с ябълки, и проучете кои имат най-големи ползи за здравето.

Пети и последен етап по-сложност и най-високо оценяване въпрос е синтеза. Синтезът включва изграждане на структура или модел от различни елементи; то също така се отнася до акта на сглобяване на части, за да образуват цяло, или обединяване на части от информация, за да формират ново значение. Характеристиките му включват:

* + Изработка на уникална комуникация
  + Изготвяне на план или предложен набор от операции
  + Извеждане на набор от абстрактни отношения

Например:

* Преобразувайте „нездравословна“ рецепта за ябълков пай в „здравословна“ рецепта, като замените избраните от вас съставки. Аргументирайте ползите за здравето от използването на съставките, които сте избрали, спрямо оригиналните.

Последната стъпка от структурата на един тест и последната стъпка от таксономията на Блум е оценката. Оценяването включва представяне и защита на мнения, като се правят преценки относно информацията, валидността на идеите или качеството на работата въз основа на набор от критерии. Характеристиките му включват:

* Присъди от гледна точка на вътрешни доказателства
* Преценки по външни критерии

Например:

* Кои видове ябълки са подходящи за печене на пай и защо?

## Ефективна област

Уменията в ефективната област описват начина, по който хората реагират емоционално и способността им да чувстват болката или радостта на други живи същества. Ефективните цели обикновено са насочени към осъзнаването и растежа на нагласите, емоциите и чувствата.

Има пет нива в ефективната област, преминаващи през процесите от най-нисък ред към най-високия.

|  |  |
| --- | --- |
| Получаване | Най-ниското ниво; ученикът пасивно обръща внимание. Без това ниво не може да се получи обучение. Получаването също е свързано с паметта и разпознаването на ученика. |
| Отговарям | Ученикът активно участва в учебния процес, а не само обръща внимание на даден стимул; ученикът също реагира по някакъв начин. |
| Оценяване | Ученикът придава стойност на обект, явление или част от информацията. Ученикът свързва ценност или някои ценности със знанията, които е придобил. |
| Организиране | Ученикът може да комбинира различни ценности, информация и идеи и да ги приспособи към собствената си схема; ученикът сравнява, свързва и доразвива наученото. |
| Характеризиращ | Ученикът на това ниво се опитва да изгради абстрактно знание. |

## Психомоторна област

Уменията в психомоторната област описват способността за физическо манипулиране на инструмент или инструмент като ръка или чук. Психомоторните цели обикновено се фокусират върху промяната или развитието на поведението или уменията.

Блум и колегите му никога не са създавали под категории за умения в психомоторната област, но оттогава други преподаватели са създали свои собствени психомоторни таксономии. Симпсън (1972) предлага таксономия от седем нива.

Възприятие - Способността да се използват сензорни сигнали за насочване на двигателната активност: Това варира от сензорна стимулация, през избор на сигнал, до превод.

Например:

- Открива сигнали за невербална комуникация.

- Преценява къде ще падне топката, след като бъде хвърлена, и след това се премества на правилното място, за да хване топката.

- Регулира топлината на печката до правилната температура чрез миризмата и вкуса на храната.

- Регулира височината на вилиците на мотокара, като сравнява къде се намират вилиците спрямо палета.

Комплект - Готовност за действие: включва умствени, физически и емоционални настройки. Тези три набора са предразположения, които предопределят реакцията на човек към различни ситуации (понякога наричани нагласи). Това подразделение на психомоториката е тясно свързано с подразделението „реагиране на явления“ на ефективната област.

Например:

- Познава и действа според последователност от стъпки в производствения процес.

- Разпознава своите способности и ограничения.

- Показва желание за усвояване на нов процес (мотивация).

Насочван отговор - Ранните етапи на изучаване на сложно умение, което включва имитация и проба и грешка: Адекватността на изпълнението се постига чрез практикуване.

Например:

- Изпълнява математическо уравнение, както е показано.

- Следва инструкциите за изграждане на модел.

- Реагира на сигнали с ръка на инструктора, докато се учи да работи с мотокар.

Механизъм - Междинният етап в усвояването на сложно умение: Научените реакции са станали обичайни и движенията могат да се изпълняват с известна увереност и опитност.

Например:

- Използвайте персонален компютър.

- Поправете течащ кран.

- Карам кола.

# Какво е HTML и CSS и защо се нуждаем от него?

## 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 и как помага на HTML да блести?

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:

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

## Създаване на приложение:

* В settings.py, секция INSTALLED\_APPS регистрираме приложението
* В папката на приложението създаваме празен файл urls.py
* В главния urls.py пренасочваме проследяване на url-ите към нашето приложение:
* Добавям include в импорта след path: from django.urls import path, include
* 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'**))  
  
]

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

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

* 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:

* 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.
* В новосъздадената папка създавам шаблон (.html файл) с име index.html и попълвам title и body с някакво съдържание.
* 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:

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

## Настройване на Django проекта:

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

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

## Подготвяме и изпълняваме миграциите:

* В терминала и подавам команда за подготовка на миграциите

python manage.py makemigrations

## Създаване на моделите:

* 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'**

* Мигрираме

python manage.py makemigrations

python manage.py migrate

## Създаване на администратор:

* Стартирам тестовия сървър
* Преминавам на страница /admin (от адресната лента на браузъра)
* 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>

* Спираме тестовия сървър и в терминала пишем команда

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:

* 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)

* В index.html добавяме визуализиране на параметъра articles

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

* Модифицирам извеждането да е по един елемент на ред
* 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]

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

## Реализиране на тестовата система:

* Правя шаблон с помощта на HTML и добавям инструкции
* С помощта на *div* дефинирам раздел или секция в HTML документ. Използвам го като контейнер за HTML елементи - който след това се стилизира с CSS.

Настройване на JavaScript към проекта:

* JS файла ни, се състои от 3 елемента: данни(data), методи(methods) и направени функции(created: functions)
* Данните включват дефиниране на оценка, тема, въпроси и предмети:

const App = {  
 data() {  
 **return** {  
 status: 0,   
 timer: {  
 h: 0,  
 m: 0,  
 s: 0,  
 id: 0,  
 },  
 test: [],  
 rate: 0.00,  
 tasks\_total: 0,  
 theme\_id: 0,  
 theme\_name: **''**,  
 subject\_id: 0,  
 subject\_name: **''**,  
 total: 0,  
 points\_total\_max: 0,  
 }  
 }

* Методите включват стартиране на теста, избиране на случаен начин тема, отброяване на времето за решаване под формата на таймер, оценяване на отговорите и приключване на теста:

methods: {  
 startTest(){…},  
 loadTheme(){…},  
 testTimer(){…},  
 valueTasks(){…},  
 stopTest(){…},

* Функциите представляват команди на подадените заявки от methods:

created: **function**(){  
 **this**.status = 0  
 **this**.theme\_id=$(**"#theme\_id"**).text();  
 **this**.theme\_name=$(**"#theme\_name"**).text();  
 **this**.subject\_id=$(**"#subject\_id"**).text();  
 **this**.subject\_name=$(**"#subject\_name"**).text();  
 **this**.tasks\_total=$(**"#tasks\_total"**).text();  
 **this**.loadTheme()  
 console.log(**this**.status)  
}

Обобщение:

Съществуват 2 ситуации:

1. когато изберем тест от целия предмет

2. когато изберем тема и тест по темата

Ако потребителят има регистрация той може да разгледа тест и да го редактира.

За да се направи въпрос минимално са нужни 10 точки, ако е между 10 и 30 точки светва жълто и ако е над 3 пъти от минимума свети зелено. Ако имаме под 10 въпроса не може да се генерира функционален тест.

Тестът има 3 етапа:

1. запознаване потребителя с условията и начина на провеждане на изпита

2. стартиране на теста- всеки потребител сам стартира теста чрез бутон "старт"

3. оценка и резултатите от теста- когато потребителят реши, че е готов натиска бутон "край на теста" и на екрана му се появя броят на точките, които е получил.

Оценката е цветово определена:

* слаб 2 - червено
* среден 3 -оранжев
* добър 4 - жълт
* много добър 5 - син
* отличен 6 - зелен

Как е реализиран?

От страната на платформа е използван Jango, който предоставя попълване на шаблони от стана на съвъра с информация.

Когато потребителят реши да прави тест по дадена тема сървъра праща запитване с номера на темата.

Имаме едновременно 2 framework-a:

1. от стана на сървъра Jango

2. от страна на клиента View

# Списък на използваната литература

1. <https://softuni.bg/blog/what-is-django>
2. <https://cynoteck.com/bg/blog-post/flask-vs-django/>
3. <https://bg.savtec.org/articles/coding/the-basics-of-rest-and-restful-api-development.html>
4. [https://urocibg.eu/](https://urocibg.eu/%D0%BA%D0%B0%D0%BA%D0%B2%D0%BE-%D0%B5-api/)
5. <https://www.django-rest-framework.org/>
6. <https://softuni.bg/blog/vue-js-january-2021>
7. <https://bg.wikipedia.org/wiki/MySQL>
8. <https://priobshti.se/article/strategii-v-pomosht-na-prepodavaneto/taksonomiya-na-blum-ili-kak-da-napravim-uroka-v>
9. https://html.w3schools.bg
10. <https://www.inventum.bg/web-design/css/>