Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Вахрушина Єгора Валентиновича*
(ПІБ)

академічної групи *121-17-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка серверної частини для функціонування навчальної платформи під керуванням операційної системи Android*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Сироткіна О.І.*			
розділів:				
спеціальний	*доц. Сироткіна О.І.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-17-1* (група) *Вахрушина Єгора Валентиновича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка серверної частини для функціонування навчальної платформи під керуванням операційної системи Android.*

затверджена наказом ректора НТУ «ДП» від *07.06.2021* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Сироткіна О.І*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Вахрушин Є.В..*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 68 с., 17 рис., 3 дод., 20 джерел.

Об'єкт розробки: Розробка серверної частини для функціонування навчальної платформи під керуванням операційної системи Android.

Мета кваліфікаційної роботи: створення бібліотеки WEB API, яка може використатися не лише сайтами, а й іншими клієнтськими програмами, такими як: мобільні платформи, комп'ютерні програми тощо.

У вступі проведено огляд предметної галузі та описано сучасний стан проблеми, визначено мету кваліфікаційної роботи й галузь її застосування, обґрунтовано актуальність обраної теми та сформульовано постановку завдання.

У першому розділі проведено аналіз предметної галузі, виконано дослідження існуючих аналогів та визначено їх недоліки, сформульовано причину необхідності розробки, призначення розробки, доведено її актуальність, зазначені вимоги до реалізації програмного забезпечення, технологій та програмних засобів.

У другому розділі сформульовано функціональне та експлуатаційне призначення програми, обрано інструментарій для розробки та визначено особливості процесу конструювання програмного забезпечення, виконано проектування та розробку програми, визначено вхідні і вихідні дані, описаний процес завантаження, вигляд інтерфейсу користувача та основні принципи функціонування програми.

В економічному розділі був проведений розрахунок трудомісткості розробленої інформаційної системи, було підраховано вартість роботи по створенню програми та визначено час на його створення.

Практичне значення полягає у створенні бібліотеки WEB API, яка надає доступ до зручного викладання навчальних матеріалів для вчителів, та легку взаємодію із навчальними матеріалами для учнів.

Актуальність сервісу визначається зростанням попиту на системи онлайн освіти, що пояснюється зростанням кількості вакансій на ринку праці, що потребують лише конкретних навиків, які можливо отримати без відвідування вищих навчальних закладів.

Список ключових слів: ВЕБ-КЛІЄНТ, ВЕБ-САЙТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА, ВЕТЕРИНАРНА КЛІНІКА, ВЕТЕРИНАРНА МЕДИЦИНА, ЕЛЕКТРОННА МЕДИЧНА КАРТКА ТВАРИНИ, КАЛЕНДАР ЩЕПЛЕНЬ ТВАРИНИ, VUE.JS, MVVM.

ABSTRACT

Explanatory note: 68 p., 17 figs., 3 apps., 12 sources.

The object of development: Web-client for information system of veterinary clinicDevelopment of a server part for operating a training platform using the Android operating system.

The purpose of the graduation project: creation of the WEB API library, which can be used not only by sites, but also by other client programs, such as: mobile platforms, computer programs, etc.

The introduction reviews the subject area and the current state of the problem, determines the purpose of the qualification work and its field of application, substantiates the relevance of the chosen topic and formulates the task statements.

In the first section the subject area was analyzed, research of existing analogues and determination of their disadvantages was done, the reason and the purpose of development was formulated, the relevance of the chosen topic was proved, the requirements for the implementation of program, technologies and software was determined.

In the second section the functional and operational purpose of the program was formulated, tools for development were chosen, features of the software design process were determined, the design and development of the program was performed, the input and output data was defined, the download process, user interface and main principles of functionality was described.

In economic section labor input and duration of software product development was defined, the estimation of expenses for its creation was executed.

The practical significance of the project is creation of a WEB API library, which provides access to convenient teaching materials for teachers, and easy interaction with learning materials for students.

The relevance of the service is determined by the growing demand for online education systems, which is explained by the growing number of vacancies in the labor market, which require only specific skills that can be obtained without attending higher education.

Keywords: WEB-CLIENT, WEBSITE, SOFTWARE, INFORMATION SYSTEM, VETERINARY CLINIC, VETERINARY MEDICINE, ELECTRONIC MEDICAL CARD, PET VACCINATION CALENDAR, VUE.JS, MVVM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – application programming interface;

MVT – Model-View-Template;

ПЗ – програмне забезпечення;

REST – Representational State Transfer;

DRF – Django REST Framework;

MVC – Model-View-Controller;

HTTP – Hyper Text Transfer Protocol;

JSON – JavaScript Object Notation;

UX/UI – User Interface/User Experience.

ВСТУП

У наш час розвиток інформаційних технологій дозволяє полегшити та перевести в онлайн працю в багатьох сферах. Поступово з’являється все більш і більш проєктів з онлайн навчанням.

Тематика даної кваліфікаційної роботи присвячена розробці серверної частини начальної системи.

Зараз навчальні платформи мають декілька недоліків, які не дозволяють отримати ефективні та якісні знання.

Однією з головних проблем є те, що існуючі проєкти не враховують індивідуальні властивості клієнтів. Як наслідок, після проходження онлайн курсів, є можливість, що людина не отримає бажаних знань. Окрім цього, на деяких платформах, через монетизацію, не можливо навіть продивитися склад курсу. Окрім цього, присутня проблема з мотивацію публікації навчальних матеріалів у вчителів, через незручний інтерфейс платформи.

Враховуючи всі вищезазначені фактори та попит на ринку ветеринарних клінік, можна зробити висновок, що обрана тематика кваліфікаційної роботи є актуальною.

Метою даної кваліфікаційної роботи є створення зручного API інформаційної навчальної системи яка надає наступні можливості учням:

– визначення схильності до конкретного напрямку та надання відповідних рекомендацій для проходження курсів;

– проходження курсів та відстежування прогресу навчання;

– виконання завдань, потребуючих компіляції коду, безпосередньо через API серверу.

В свою чергу для викладачів є наступні можливості:

– створення курсів в онлайн та офлайн форматах, як для групових так і для приватних занять;

– створення зручного розкладу для лекцій та практичних занять і можливість його редагування;

–створення завдань та тестів для покращення практичних навиків учнів.

Адміністратор проекту будуть мати наступні можливості:

– має доступ до адміністративної панелі;

– може редагувати та видаляти певні об’єкти системи.

– додавати нові жанри курсів;

Для реалізації поставленої мети необхідно виконати наступні завдання:

- проаналізувати предметну галузь;

- дослідити існуючі аналоги та виділити їх недоліки;

- визначити основні вимоги до програмного продукту;

- обрати архітектуру системи, шаблони проектування та методологію життєвого циклу розробки;

- проаналізувати варіанти та вирішити які інструменти використовувати для розробки;

- розробити сервер для інформаційної навчальної системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1.Загальні відомості з предметної галузі

Сучасний ринок освітніх послуг в Україні та світі рясніє безліччю пропозицій. Тут є все починаючи з традиційних навчальних закладів, що протягом довгого часу працюють з використанням одних і тих самих освітніх методик, й закінчуючи новітніми онлайн проєктами, котрі пропонують численні навчальні матеріали різного штибу. Кожен охочий, з урахуванням своїх можливостей, здібностей та вмісту гаманця, може знайти собі навчання до вподоби. Але таке різноманіття породжує проблему вибору. Люди часто не розуміють, що і як обрати, якими критеріями користуватись при виборі й що робити, якщо вибір виявився хибним.

І перша проблема, з якою зазвичай стикаються шукачі освіти, це проблема того, яку саме її форму обрати: очну чи віддалену, індивідуальну чи групову. Під час здійснення такого вибору варто враховувати цілу купу факторів. Наприклад, освіта обов'язково має виховну функцію. Відповідно, у випадку, якщо людина потребує не тільки знань, а, ще й корекції поведінки чи комунікативних навичок, варто обрати очну форму навчання.

Але віддалена освіта в будь-якому випадку може стати у пригоді. Вона або цілком здатна замінити традиційну очну форму, або може непогано її доповнити. Серед явних переваг такої форми навчання є те, що учень може навчатися коли завгодно, чому завгодно та в кого завгодно. При цьому, звісно, виникає проблема якості навчального матеріалу та його відповідності вимогам часу. Але конкуренція на цьому ринку дуже гарно впливає на його розвиток, оскільки нежиттєздатні освітні ініціативи, не підкріплені позбавленим сенсу використанням державних коштів майже миттєво зникають, заміщені не завжди якісним, але завжди більш зрозумілим й затребуваним контентом.

Віддалена освіта так само буває різна. Серед її засобів можна виділити два

основні різновиди: проведення занять у реальному часі, що вимагає особистої присутності учня і його занурення в навчальну подію, та дистанційне навчання за допомогою записів, текстів та інших освітніх матеріалів, що надаються учневі у зручний для нього спосіб. Обидва варіанти мають переваги та недоліки, з чого можна зробити висновок, що їхнє спільне використання може мати більшу ефективність, аніж використання поодинці. І це яскраво доводить історія дистанційного навчання.

Як зазначено у джерелі[1], на початку XVIII століття в Європі виникло так зване кореспондентське навчання. У 1728 році К. Філіпс подав у бостонську газету оголошення про набір студентів для вивчення стенографії в будь-якій точці країни шляхом обміну листами. Це стало початком надання освітніх послуг на відстані. Тоді учні поштою отримували матеріали від педагогів, відправляли на перевірку свої роботи і отримували коментарі.

Величезний внесок у розвиток дистанційного навчання вніс І. Пітман. У 1840-х роках він запропонував ввести в навчання стенографічний лист: британський вчений посилав шифровані тексти з уроками своїм учням і отримував назад роботи на перевірку.

Наступними були Ч. Тусен і Г. Панченштейдт, які в 1856 році заснували інститут заочної форми освіти в Берліні. Навчання проводилося все також розсилкою листів з навчальними матеріалами, контрольними роботами тощо.

В 1873 році були створені перші заочні школи в США. Незабаром після цього, 1892 році, університет Чикаго створив першу дистанційну програму, ставши тим самим, першим дистанційним навчальним закладом США. З 1899 року в Канаді Королівський університет став навчати студентів на відстані.

У 1914 році такі школи з'явилися у Великобританії, а незабаром поширилися в Австралії, Новій Зеландії.

В 1938 році відбувся перший з'їзд Міжнародної ради з кореспондентського навчання (International Council for Cortespondence Education).

Протягом першої половини XX століття з появою нових технологій прискорився процес розвитку дистанційної освіти. Вона була запропонована в

різних форматах і для широкого кола учнів

Отже, з винаходом радіо розвиток навчання на відстані прискорився, стали доступні нові форми роботи з учнями.

Вважається, що перший університет, який запровадив радіо в навчання, був Державний університет Пенсільванії, в 1934 році. Державний університет Айови запустив перший в світі освітній канал, який працює і сьогодні.

З появою телебачення в 1950-х роках розвинулися телевізійні курси. Отже, до 1953 року телевізійне мовлення курсів стало поширеним серед університетів США і Європи.

Університет Вісконсіса в 1965 році запровадив масштабну освітню програму для лікарів, використовуючи при цьому формат телефонного викладання. В 1968 році можна було отримати Кредитований диплом на базі дистанційної освіти в університеті Лінкольна (США, штат Небраска).

У 1960-і роки дистанційна освіта отримала міжнародне визнання і стала активно розвиватися за підтримки ЮНЕСКО. У 1963 році прем'єр-міністр Великої Британії Г. Вільсон оголосив про створення ефірного університету, який передбачав поєднання усіх навчальних закладів країни, що використовували дистанційну освіту. В 1969 році у Великій Британії був створений Відкритий університет (Open University). На сьогодні цей університет користується великою популярністю. В ньому навчається понад 200000 студентів з різних країн за різними напрямками.

У 1970 році була створена каліфорнійська робоча група метою якої була розробка навчальних телевізійних курсів. Пізніше була створена ціла організація Coastline Community College, що запропонувала навчальні фільми університетам, бібліотекам і каналам суспільного телебачення. В 1976 році було відкрито перший віртуальний коледж, який навчав за програмою Coastline. Але з плином часу, технології змінювалися. Тож незабаром з'явилася можливість викладати за допомогою супутникових каналів телебачення.

Згодом для дистанційного навчання стали використовувати комп'ютери. Ще в 60-х рр компанія IBM розробила унікальну програму дистанційного навчання

Coursewriter. Її можна було налаштовувати на різні види занять і вона використовувалася в університеті Альберти з 1968 по 1980-і р. на 17 різних курсах.

З винайденням інтернету людство зробило ще один крок вперед в освітніх технологіях.

Протягом 80-х років технології навчання в режимі реального часу удосконалювалися. Вони завойовували популярність в компаніях і в освітніх закладах. У 1981 році інститут стратегії та управління в США почав розробляти програму онлайн-курсів. В 1985 році південно-східний університет запропонував акредитовані дипломи, одержані через систему онлайн-курсів. У 1989 році виник університет Фенікса, навчання проводилося в режимі реального часу.

Протягом 1990-х років освітні установи використовували різні технології дистанційної освіти як в режимі синхронного, так і асинхронного навчання.

В 1992 році університет штату Мічиган розробив за допомогою комп'ютера індивідуальний підхід до онлайн навчання. У 1994 році університет запропонував віртуальну школу навчання (VSS) деяким своїм студентам психологам. Крім цього, в 1994 році компанія в Нью-Гемпширі, що займається дистанційною освітою, розробила програму CAL.Campus, яка надає можливість навчання, адміністрування та пересилання матеріалів виключно через інтернет

У 1997 році компанія Blackboard розробила стандартну платформу для управління та надання курсів. Зараз компанія є світовим лідером у сфері дистанційних технологій, продукти якої використовують понад 10000 організації по всьому світу.

У 2000-х роках дистанційне навчання стало домінуючим. Система інтернету вдосконалювалася, ставала більш доступною, також розвивалися і технології дистанційної освіти. В результаті кількість університетів, що використовують інтернет-технології зросла.

Зараз дистанційне навчання набуває все більшої популярності через те, що спостерігається збільшення кількості людей, які бажають здобути освіту, але не

мають можливості це зробити в формі денного навчання[1].

Таким чином питання використання комп'ютерної техніки та штучного інтелекту в освітніх технологіях не є вичерпним. Біль того, воно набуває нової актуальності, оскільки з’являється можливість використання комп'ютерів не тільки як допоміжного інструментарію, але і як повноцінних вчителів. Такі поєднання освітніх технологій потребують побудови новітніх освітніх платформ, що враховують такі зміни в освітній галузі, є безумовно актуальними.

На ринку вже існує багато різних навчальних платформ для онлайн освіти. Після проведення аналізу існуючих аналогів, було виділено певну схожість між найбільш популярними платформами. Через те, що в даній роботі не передбачено WEB інтерфейсу, порівняння можливо робити, базуючись на наповненні платформ матеріалами, ігноруючи лендінг. Порівнюючи наступні платформи: Coursera, edX та SkillShare, можливо виділити, що Coursera одразу пропонує список безплатних курсів для проходження(рис. 1.1). В аналогічному випадку, на платформі edX є перелік усіх курсів доступних на платформі(рис. 1.2). Але на відміну від Coursera, в edX є гнучкий фільтр для пошуку цікавлюючих курсів. SkillShare, на відміну від конкурентів потребує одразу оформити платну підписку для проглядання доступних курсів(рис. 1.3).

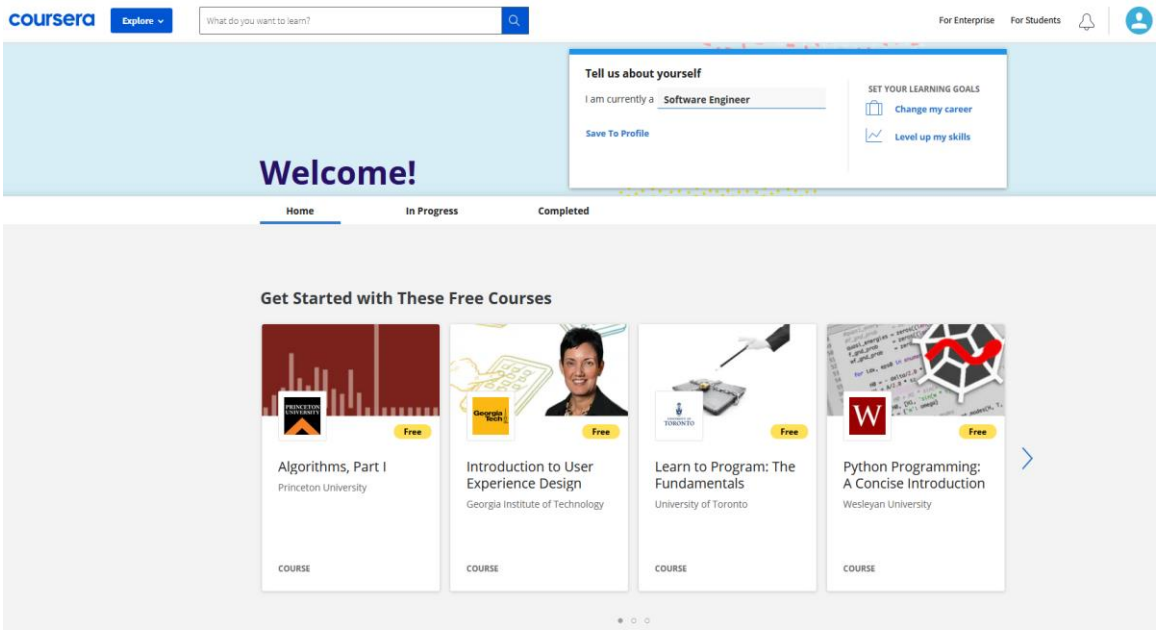


Рис 1.1 Перелік курсів на сайті Coursera

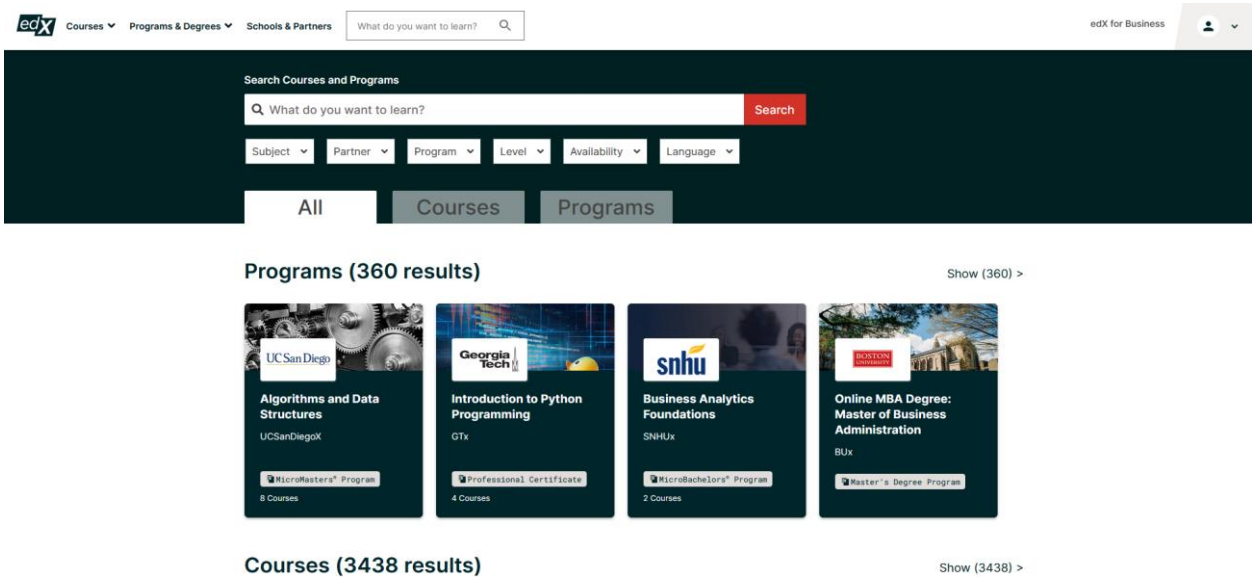


Рис. 1.2 Перелік курсів на сайті edX

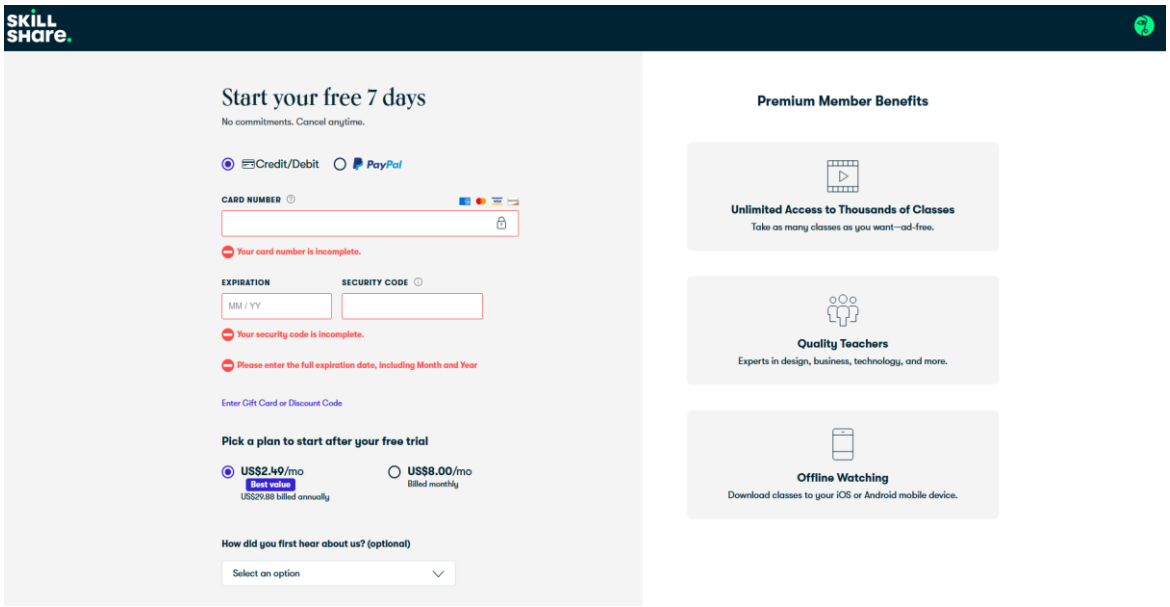


Рис. 1.3 Перелік курсів на сайті SkillShare

Також, молживо порівняти інформацію, яка надається користувачу, перед записом на курс. Оскільки SkillShare не дає можливості перевірити наповнення курсу, порівнюватися будуть платформи Coursera та edX.

Як можливо побачити на рисунку 1.4, Coursera надає цілу купу інформації п курс, що дозволяє краще зрозуміти, чи потрібен він клієнту.

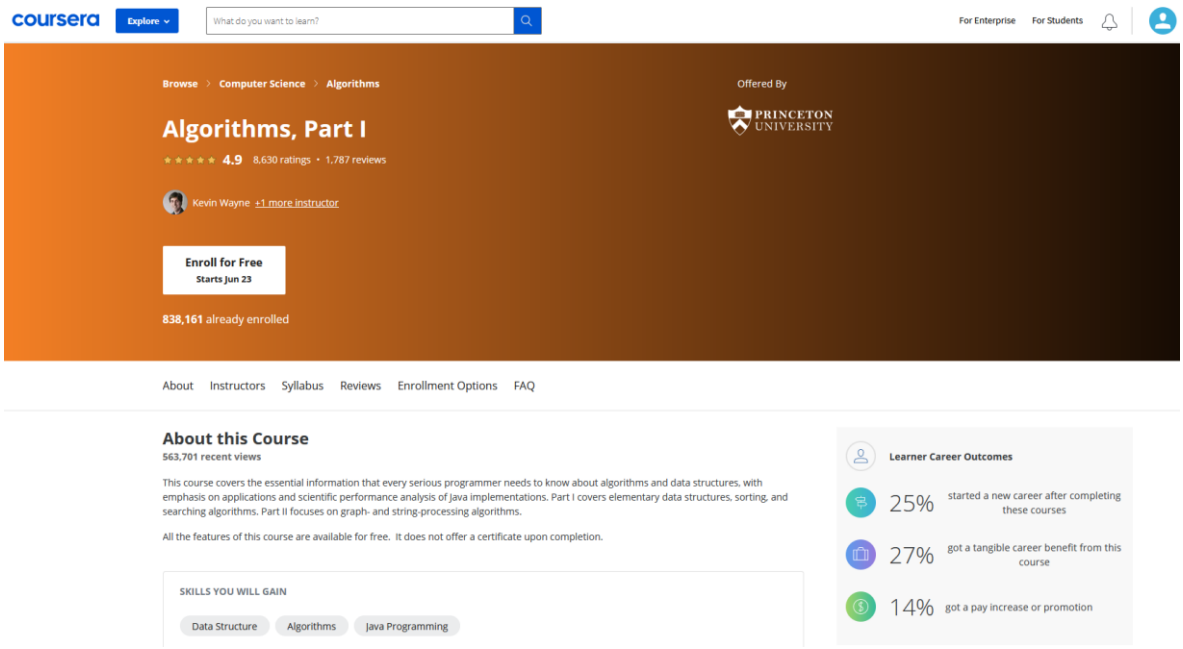


Рис 1.4. Інформація про курс на сайті Coursera

На відміну від Coursera, edX надає менше інформації(рис. 1.5)[17, 18, 19].

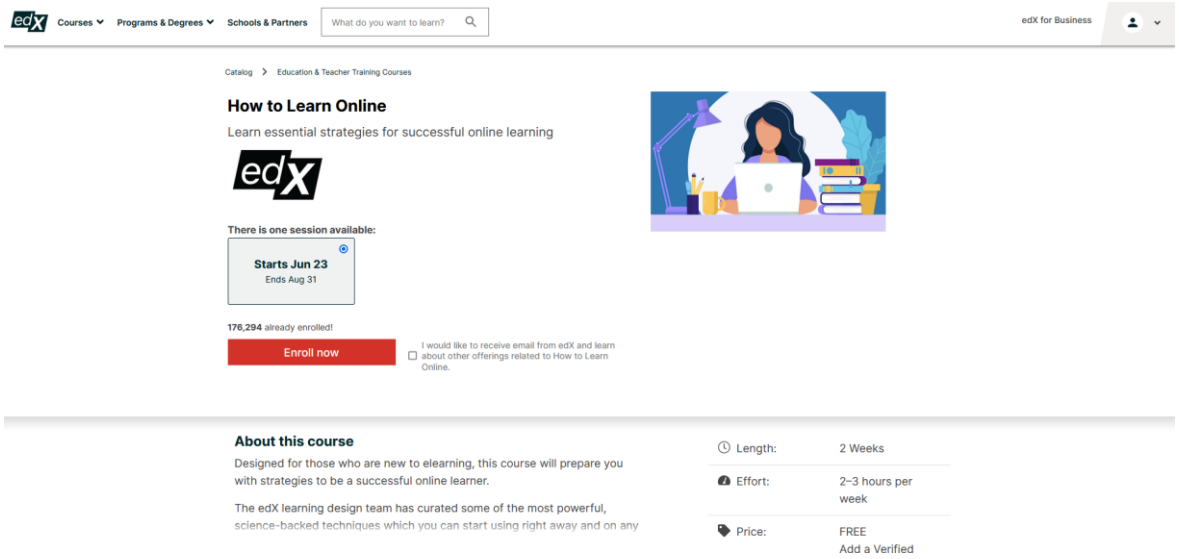


Рис 1.5. Інформація про курс на сайті edX

Після порівняння та аналізу конкурентів, були виділені основні особливості майбутнього проєкту.

1.2. Призначення розробки та галузь застосування

Під час виконання кваліфікаційної роботи було поставлено завдання розробити програмний продукт серверної частини мобільного навчальної платформи.

Необхідності розробки цього програмного забезпечення полягає в тому, що в існуючих сервісах, немає індивідуального підходу до навчання клієнтів, через що, падає ефективність та знижуються результату.

Галузь застосування даного продукту – освіта, а саме – ринок онлайн систем для навчання.

Призначення даної розробки – це надання зручного API з системою розподілення прав для наступних користувачів:

- учень;
- викладач;
- адміністратор системи.

1.3. Підстави для розробки

В кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником проєкту, випусковою кафедрою.

Підставою для розробки кваліфікаційної роботи на тему «розробка серверної частини для функціонування навчальної платформи під керуванням операційної системи Android» є наказ по Національному технічному університету «Дніпровська політехніка» від 07.06. 2021р. № 317-с.

1.4. Постановка завдання

Метою кваліфікаційної роботи є створення бібліотеки WEB API для організації роботи мобільної навчальної платформи, що реалізує можливість реєстрації нових користувачів системи(учнів та викладачів), проходження тесту

новим учнем для виявлення його схильностей та надання індивідуальних рекомендацій, створення та публікації нових курсів від викладачів за попередньої перевірки якості, створення практичних завдань для кращого засвоєння матеріалу учнями, як у форматі тестів, так і в форматі завдань, що потребують написання та компіляції коду.

Продукт призначений для створення навчальної платформи, яка завдяки використанню технологій WEB API зможе мати не тільки сайт, але й мобільний додаток та може бути інтегрована в інші навчальні проєкти.

Даний програмний продукт є серверною частиною навчальної інформаційної системи, що має клієнт-серверну архітектуру. Особливості даної архітектури добре описані в [4].

Основним недоліком існуючих на освітньому ринку сервісів є те, що вони просто надають учням матеріал для навчання, не аналізуючи та не відслідковуючи їх успішність оскільки технічно зробити це не дуже просто. Таким чином ефективне навчання за допомогою цих сервісів можливе виключно якщо учень має добру попередню підготовку, здібності та мотивацію до вивчення наданого матеріалу.

Якість наданих матеріалів на більшості сервісів також не бездоганна. Це пов'язано з тим, що модель співпраці з авторами навчальних курсів, не мотивує їх до більш якісного виконання своєї роботи. Таким чином можливо зробити висновок, що навчальні сервіси часто призначені для підтримки навчального процесу, а не для здобуття начального результату. Тому потрібно створити таку навчальну платформу, яка б дозволила учням отримувати якісно підготовлені матеріали й практичні завдання, які дозволять якісніше засвоїти пройдений матеріал, а викладачам та авторам курсів, дозволить отримати більший матеріальний зиск від якісного виконання свої роботи.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Програмний продукт загалом повинен виконувати наступні функції:

- задокументовані зручні API;
- реєстрації нових користувачів системи;
- різні права доступу для студента, адміністратора та вчителя;

бібліотека WEB API повинна надавати наступні можливості учням:

- реєстрація нових учнів;
- використання індивідуальних рекомендацій;
- отримання списку доступних курсів;
- зарахування на конкретний курс;
- отримання інформації за поточними курсами(зокрема відстежування прогресу);
- отримання розкладу занять з активних курсів;
- виконання практичних завдань в онлайн засобах розробки;

бібліотека WEB API повинна надавати наступні можливості вчителю:

- реєстрація нових вчителів;;
- створення нових курсів;
- створення та редагування розкладу занять;
- отримання результатів виконання завдань активних користувачів курсів;

бібліотека WEB повинна надавати наступні можливості адміністратору:

- керування обліковими записами користувачів;
- керування курсами;
- керування наборами практичних завдань;
- підключення та відключення онлайн засобів розробки;

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення безпеки персональних даних користувачів система має відповідати наступним вимогам:

- доступ до функцій, які надає серверна частина повинен мати криптографічний захист, тобто відбуватись за допомогою HTTPS[5];

- керування доступом різного рівня, наприклад, учень, вчитель чи системний адміністратор повинно бути реалізовано, за допомогою вбудованих засобів обраного фреймворку;

- авторизація в системі може бути виконана за допомогою сторонніх сервісів ідентифікації-аутентифікації;

1.5.3. Вимоги до складу та параметрів технічних засобів

Для функціонування серверної частини навчальної системи, потрібно обчислювальна платформа наступної конфігурації:

- Intel Core i3;
- 8 Гб оперативної пам'яті;
- мінімум 100 Мб вільного дискового простору.

1.5.4. Вимоги до інформаційної та програмної сумісності

Розроблене програмне забезпечення повинне бути кросплатформенним, тобто правильно функціонувати під управлінням будь-якої серверної операційної системи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Оскільки розроблений програмний продукт призначений для використання користувачами різних ролей, то функціональне призначення для кожної конкретної ролі є різним.

Бібліотека WEB API має наступні функціональні призначення для учня:

- надання інформації про доступні курси;
- надання інформації про викладачів конкретних курсів;
- отримання матеріалів для навчання за обраним напрямом;
- отримання розкладу занять;
- виконання практичних завдань та отримання результатів;
- отримання інформації про свої успіхи в курсах;
- отримання відгуків викладачів про виконане завдання;
- реєстрація та авторизація учнів в системі;
- редагування особистих даних.

Бібліотека WEB API має наступні функціональні призначення для викладача:

- реєстрація та авторизація в системі;
- отримання розкладу навчань;
- редагування розкладу своїх занять;
- отримання списку активних учнів;
- отримання особистих результатів активних учнів;
- створення нових курсів;
- створення нових завдань.

Бібліотека WEB API має наступні функціональні призначення для адміністратора:

- авторизація в системі;

- видалення курсів;
- додавання нових курсів;
- отримання списку всіх вчителів;
- отримання списку всіх учнів.

Розроблений програмний продукт реалізує наступне експлуатаційне призначення:

- надає клієнтським програмам доступ до навчально інформаційної системи;
- оптимізує процес онлайн освіти й дозволяє навчатися в заочній формі.

2.2. Опис застосованих математичних методів

У розробленому програмному продукті не використовувались математичні методи, тому що вимогами до розробки не було регламентовано ніяких задач математичного характеру.

2.3. Опис використаної архітектури та шаблонів проєктування

Розроблений програмний продукт є бібліотекою WEB API, що дозволяє клієнтським програмам взаємодіяти з системою онлайн навчання завдяки REST системі. Застосунок є сервером для клієнтських програм.

Архітектура REST описується обмеженнями, які вперше були представленні Р. Філдінгом в докторській дисертації [2]:

- єдиний інтерфейс;
- відсутність станів;
- кешування відповіді;
- клієнт-сервер;
- багаторівнева система;
- «код на вимогу».

В основі взаємодії клієнта з сервером лежить принцип того, що цю

взаємодію завжди починає клієнт. Таким чином, спочатку клієнт відправляє на сервер HTTP-запити. Сервер приймає ці запити, обробляє та повертає певну відповідь. HTTP-запити класифікуються в залежності від того яку логічну операцію вони мають виконати. Розрізняють такі види HTTP-запитів: GET, POST, PUT, DELETE тощо. Серверна частина отримуючи відповідний запит, виконує відповідні дії, при чому найчастіше, різні види HTTP-запитів зіставляють з операціями CRUD-доступу до даних. Тобто, GET зіставляється з READ, POST - з UPDATE, PUT - з CREATE та DELETE - з DELETE відповідно.

Після обробки запиту, сервер відправляє відповідь клієнту. У відповіді сервера завжди є статус виконання запиту клієнта в форматі трьох значного числа, перша цифра якого, завжди ранжується від 1 до 5. Статуси запиту поділяють на 5 типів відповідно:

- 1xx: інформаційний статус;
- 2xx: успіх;
- 3xx: перенаправлення;
- 4xx: помилка клієнта;
- 5xx: помилка серверу.

Завдяки цьому, між клієнтом і сервером можливо налагодити гнучку взаємодію[7].

Сервер може повертати інформацію в різних форматах, в таких як text/plane, XML, HTML тощо. Але REST системі, практично завжди використовують формат JSON, який протягом останнім років, практично повністю витіснив XML.

На рис 2.1 зображена взаємодія між клієнтами та платформою завдяки WEB API.

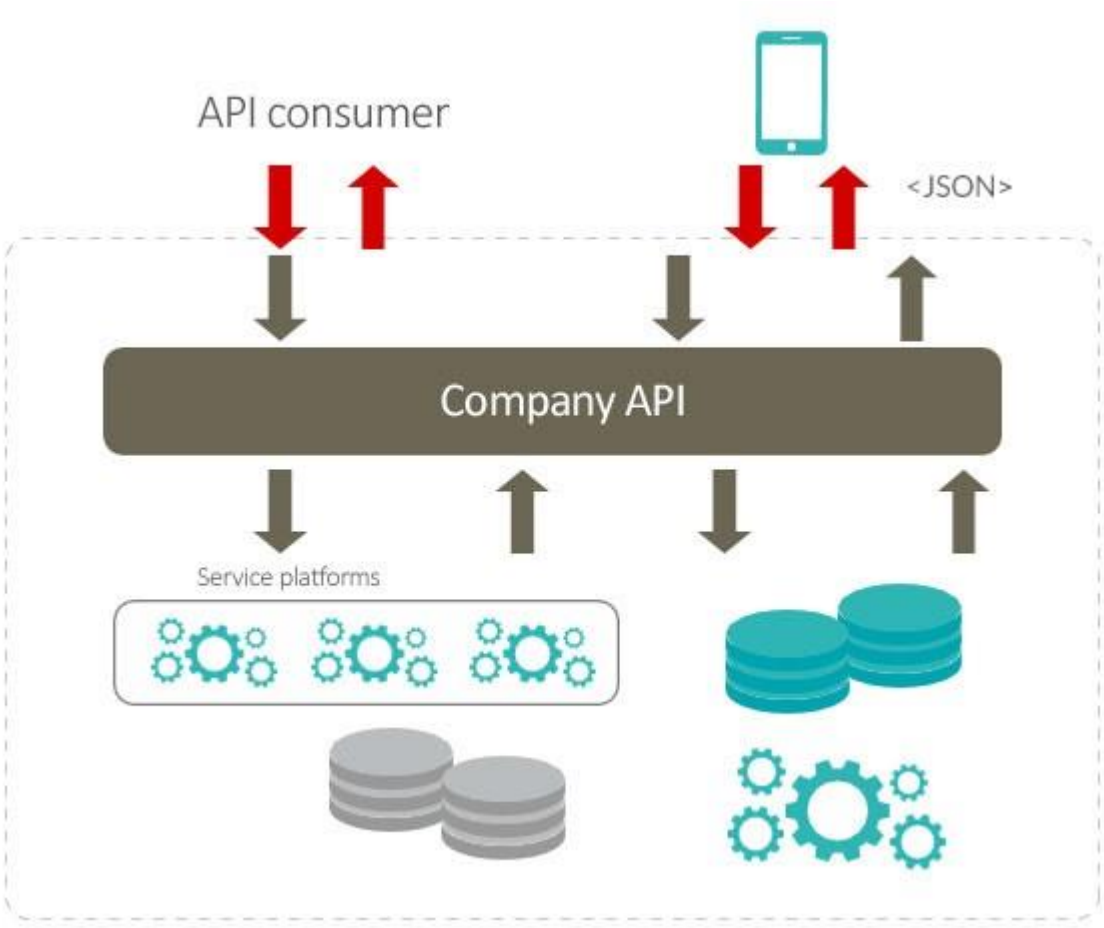


Рис. 2.1. Схема взаємодії між клієнтами та сервером завдяки WEB API

Як показано на рисунку, у WEB API може бути багато клієнтів. Це можуть бути мобільні додатки, сайти, комп'ютерні програми тощо. В свою чергу WEB API взаємодіє із самою платформою, яка опрацьовує запит клієнта і взаємодіє з базою даних.

Для реалізації серверної частини, використовується патерн MVC(Модель-Представлення-Контролер). Цей шаблон передбачає організацію коду, шляхом виділення блоків, що відповідають за різні здачі. Таким чином у MVC є наступні блоки:

- Модель - це компонент, якій відповідає за дані, а також за структуру додатку. Модель взаємодія напряму із базою даних. Зазвичай, цей компонент використовується для додавання, редагування або видалення записів із бази даних;

- Представлення - цей компонент відповідає за взаємодію з користувачем. В

нашому випадку - це API оформленні за принципом REST;

- Контролер - відповідає за взаємодію між минулими двома компонентами.

Розділення логіки програми за таким патерном надає багато переваг. Серед них:

- Швидка розробка. Завдяки суворим рамкам по структурі коду, модульності й наявності великої кількості середовищ для розробки з реалізованою базовою архітектурою MVC, можливо швидко розробляти програму.
- Легка командна взаємодія при розробці. Модуляція різних рівнів програми дає командам розробників можливість розробляти та вдосконалювати продукт набагато легше, адже вони можуть змінювати різні шари програми, при цьому, уникаючи конфліктів.
- Легше вносити зміни в програму. Завдяки модульності, можливо вносити зміни в код, не переживаючи за інші частини програми, оскільки конкретний компонент може бути змінений з мінімальним впливом на інші компоненти.
- Легке налагодження коду. Через те що програма роздріблена на різні рівні, розуміючи в якому рівні є несправність, можливо налагоджувати лише конкретний компонент для швидкого виявлення помилки.

Недоліками даної архітектури є:

- Складність освоєння при навчанні.

Життєвий цикл розробки програмного забезпечення – це безперервний процес, який починається з моменту часу, коли було прийнято рішення про створення проєкту і закінчується в момент, коли програмне забезпечення повністю виведене з експлуатації [7].

Модель життєвого циклу розробки програмного забезпечення – це концептуальна основа, що визначає послідовність виконання і взаємозв'язок всіх етапів розробки протягом життєвого циклу.

На сьогодні найвідомішими та найбільш використовуваними є наступні моделі:

- Модель водоспаду;
- Ітеративна модель;
- Спіральна модель;
- V-подібна модель;
- Модель Agile.

Для розробки проєкту було обрано використовувати модель Agile. Agile - це гнучкий підхід до розробки[8].

2.4. Опис використаних технологій та мов програмування

Для розробки бібліотеки WEB API був використаний Django, що є вільним фреймворком для розробки WEB додатків, мовою Python. Завдяки зрозумілій та легкій архітектурі проєкту, цей фреймворк дозволяє швидко створювати потужні системи, а завдяки модульності, їх легко масштабувати. Крім цього, він гарно оптимізований, завдяки чому, може змагатися в продуктивності з такими гігантами як ASP NET та Spring.

Також, для розробки REST було використано набір інструментів Django REST framework, який дозволяє швидко розробляти гнучі API для системи.

Для створення системи були використані наступні технології:

- Python3 – мова програмування, третя версія.
- Django – фреймворк для розроблення WEB додатків, третя версія[13].
- DRF (Django REST Framework) – набір інструментів, що використовується разом з Django для створення WEB API;
- PostgresQL – відкрита, об'єктноорієнтована система баз даних.
- SimplJWT – це бібліотека для Django, яка допомагає в авторизації та реєстрації нових користувачів, завдяки JWT.
- Social oauth2 – це бібліотека Django для авторизації користувачів через сторонні сервіси, такі як Google, Facebook, Instagram тощо.
- Pillow – це Django бібліотека обробки зображень.
- Gunicorn – це популярна бібліотека налаштування взаємодії між сервером

Django та вебсервером.

- Nginx - популярний вебсервер і проксі сервер, що працює на Unix системах.

Однією з особливостей Django є те, що проєкт ділиться не лише на компоненти MVC, а й ще на більші модулі, в кожному з яких є своя реалізація MVC. Так, наприклад, проєкт можливо розділити на модуль контролю користувачів, модуль курсів та модуль монетизації й усі ці модулі будуть незалежними. Таким чином є можливість замінити, наприклад, модуль монетизації на інший, і при цьому не потрібно буде вносити зміни в інші модулі. Ця особливість додає більше гнучкості при розробці й дозволяє більш легко розробляти продукт.

На відміну від таких популярних фреймворків, як Spring та ASP.NET, в Django є велика кількість вже реалізованих компонентів, з яких складається MVC. Для реалізації шару Модель, в Django є компонент Model, в якому є велика купа реалізованих методів для взаємодії із базою даних. Методи для додавання, змінення та видалення записів, вже реалізовані в цьому компоненті. Завдяки цьому достатньо лише описати структуру бази даних, для подальшої роботи з нею.

Для реалізації шару Контролера, в Django використовується класи та підкласи Serializer. Цей клас реалізований в модулі DRF і дозволяє швидко та легко зробити де серіалізацію та серіалізацію об'єктів бази даних. В цьому компоненті проходить перевірка правильності вхідних даних перед записом до бази даних. А в випадку серіалізації, проходить формування JSON з даних пам'яті, наприклад моделей.

Шар Представлення реалізується завдяки підкласам APIView. Для кожного класу, що наслідує APIView вказується вибірка бази даних, з якою буде працювати цей клас і серіалайзер, що буде опрацьовувати цю вибірку. За замовчуванням, в класі вимкнені всі команді. Для їх підключення, потрібно перевизначити метод, який називається відповідно до команди, маленькими літерами. Наприклад, для підключення команди GET потрібно перевизначити

метод get(). За замовчуванням, для доступу до будь-яких методів класу представлення не потрібно авторизуватися в системі. Але для того, щоб частково або повністю захистити вміст класу, використовується конструкція permission_class. В ній можливо визначити, які з користувачів системи будуть мати доступ до конкретних команд.

Для авторизації використовуються технології JWT і OAuth. Технологія JWT дозволяє створювати авторизаційні токени для зареєстрованих в системі користувачів. Завдяки цим токенам, користувач може отримати доступ до ресурсів, які потребують авторизації. Таким чином, користувачу не потрібно постійно вводити свої авторизаційні дані, при взаємодії з системою. Технологія JWT передбачає створення двох токенів: токен доступу та токен відновлення. Токен доступу - зазвичай має короткий термін існування. Тобто після закінчення відведеного часу, він стає не правильним і його треба поновити, використовуючи токен відновлення. Натомість токен відновлення має більш довгий термін актуальності, а коли й він стає не правильним, користувач повинен знову вводити ім'я користувача та пароль. Принцип праці JWT описаний схемою на рис. 2.2.

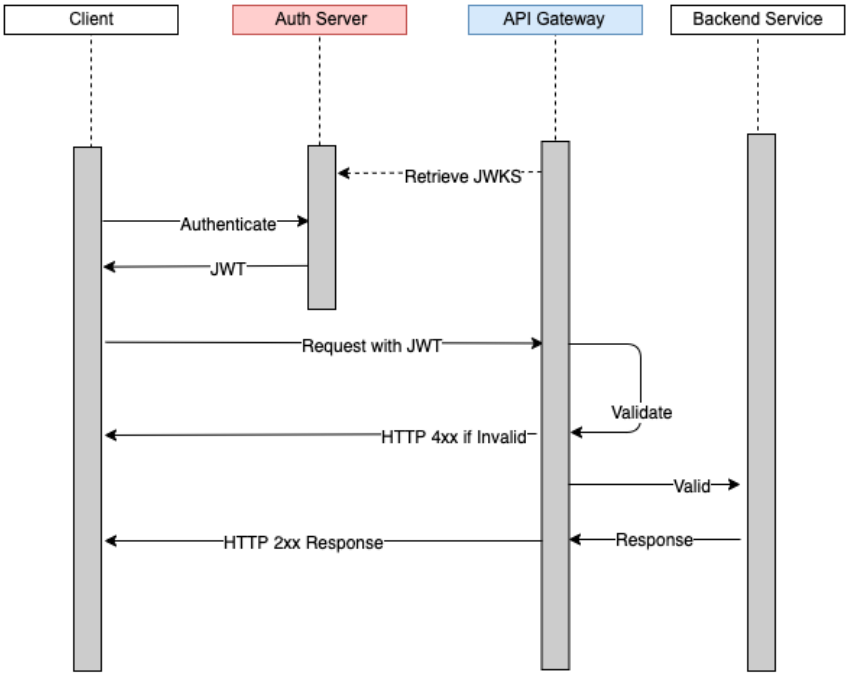


Рис 2.2. Схема праці JWT

OAuth працює за схожим принципом(рис. 2.3). OAuth може використовувати як власний, так і сторонній сервіс авторизації для отримання токену, тому є можливість використовувати зовнішній сервіс без створення додаткових облікових записів в системі.

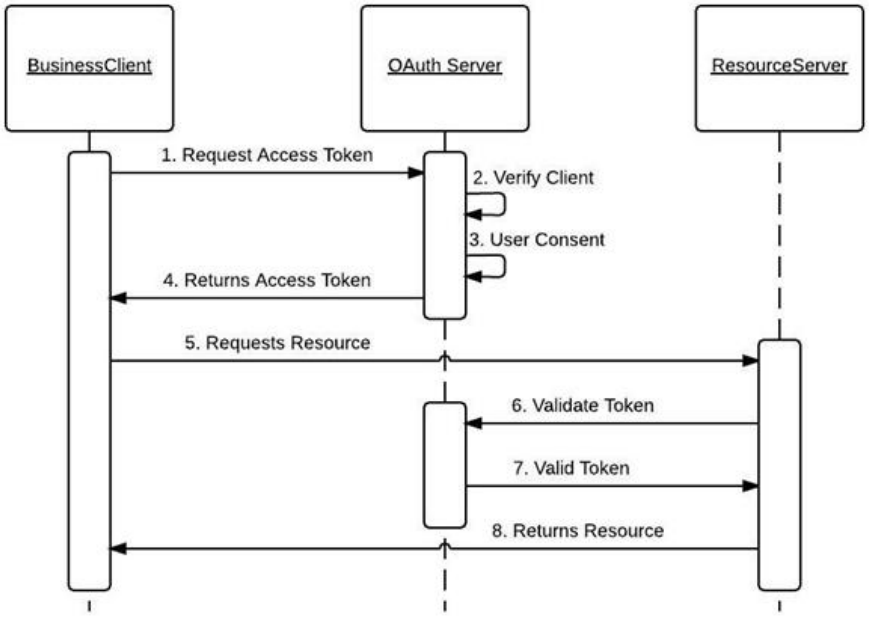


Рис 2.3. Принцип праці OAuth

2.5. Опис структури програми та алгоритмів її функціонування

Типова структура проєкту розробленого програмного продукту має структуру, що представлена на рис. 2.4.

Директорія env містить різні бібліотеки Python, включаючи Django, що використовуються для роботи проєкту[16].

Директорія study_platform містить в собі модулі з яких складається проєкт і також є точкою запуску програми. В неї входять наступні директорії та файли:

−__pycache__ – директорія в якій знаходяться попередньо компільовані класи.

−courses – модуль проєкту в якому реалізована частина з курсами;

−media – директорія в якій зберігається мультимедійний контент;

−static – директорія в якій знаходиться статичні файли проєкту, такі як: CSS та javascript файли, та файли зображень вбудованих бібліотек django;

−staudy_platform – директорія з файлами глобальної конфігурації проєкту;

−tasks – модуль проєкту в якому реалізовані функції роботи з учнівськими завданнями;

−userManagement - модуль в якому реалізовані класи користувачів системи та система авторизації;

Кожен модуль проєкту має схожу архітектуру. На прикладі модуля courses, розглянемо загальну архітектуру модулів(рис 2.5):

−__pycache__ – директорія в якій знаходяться попередньо компільовані класи;

−migrations – директорія до якої зберігаються міграції бази даних, які належати до конкретного модуля;

−__init__.py – це файл, який вказує, що поточна директорія містить файли окремого пакету Python;

−admins.py - файл в якому налаштовується адміністративна сторінка для управління об'єктами бази даних цього модуля;

−apps.py – це головний файл модуля, через який він підключається до проєкту;

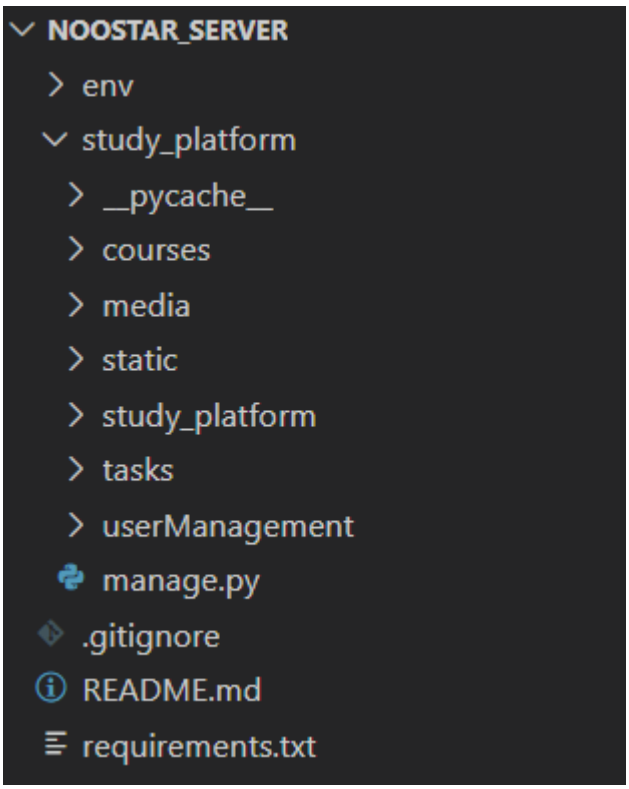
−models.py – в цьому файлі знаходиться код, який відповідає за взаємодію із базою даних;

−permissions.py – в цьому файлі описуються права доступу певних користувачів, відносно моделей модуля;

−serializers.py – файл з бізнес-логікою проєкту;

−test.py – ursl.py – файл з тестами проєкту;

−views.py – в цьому файлі описуються обробка запитів користувача.



– Рис. 2.4. Структура проєкту

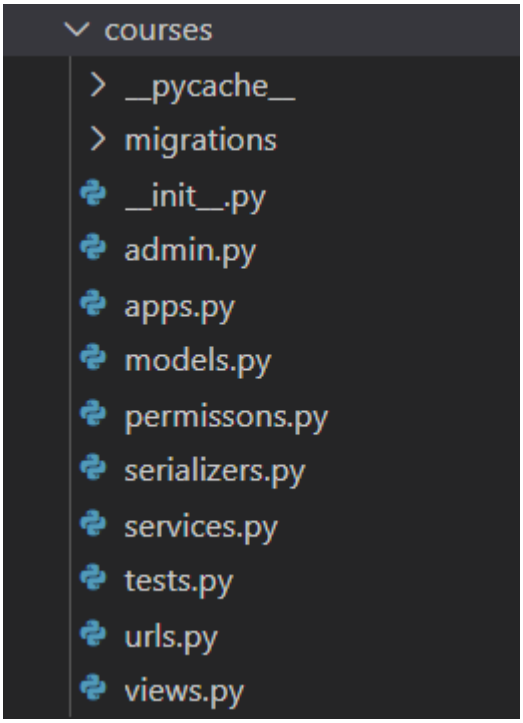


Рис. 2.5. Структура модуля courses

2.5.1. Структура сховища даних

Як було вказано раніше, в Django, управління базою даних проходить завдяки класу Model і реалізація розноситься по різним частинам програми – модулям. Також слід зазначити, що завдяки реалізації взаємодії із базою даних в Django, не вважаючи на те, яка СУБД використовується, взаємодія із базою даних буде універсальною.

2.5.1.1 Структура сховища даних модуля «userManagment»

В основі розробленої системи лежать користувачі, які мають різні права доступу до ресурсів. Через це, дуже важливо гарно спроєктувати структуру моделі, що відповідає за користувачів. Користувача в системі проєкту описує клас Person, що наслідує наступні класи Django: AbstractBaseUser та PermissionsMixin. Завдяки наслідуванню від AbstractBaseUser, новий клас можливо підставити до базової системи авторизації Django, з якою також працюють такі технології як JWT та OAuth2. Це дозволяє зменшити кількість коду в програмі, та уникнути повторної реалізації вже існуючих налагоджених систем. Клас Person реалізований в модулі userManagement і має наступні поля:

−id – це поле є унікальним ідентифікатором об'єкта(це поле, як і всі інші унікальні ідентифікатори в проєкті відповідають стандарту UUID);

−email – це поле є логіном для користувача. В системі не може бути двох об'єктів з однаковим значенням поля email;

−password – це поле є паролем користувача. В базі даних воно зберігається в кешованому виді для того, щоб неможливо було отримати пароль користувачів, навіть маючи доступ напряму до бази даних;

−is_active – це поле має значення Так або Ні й показує чи є користувач активним. При створенні нового користувача в систему, для завершення процесу реєстрації, треба ввести код підтвердження, відправлений на вказану при реєстрації почту. В проміжок часу, поки користувач ще не активував свій

обліковий запис, це поле має значення Ні. Окрім цього адміністратор системи може заблокувати активного користувача. В цьому випадку значення цього поля також зміниться на Ні. Користувачі у яких це поле має негативне значення, не можуть авторизуватися й отримати доступ до системи;

−is_supperuser – це поле показує чи являється користувач адміном. За замовчуванням це поле має негативне значення, тобто користувач не має прав адміністратора. Єдиний спосіб створення користувача адміна – це створити його завдяки консолі Django, для чого потрібно мати напряму доступ до сервера;

−first_name – це допоміжне поле, яке зберігає в собі ім'я користувача;

−is_coach – у випадку позитивного значення, це поле показує, що користувача є вчителем і має доступ до відповідного функціоналу. Це поле не може бути активним, якщо вже активне поле is_supperuser;

−photo - допоміжне поле, яке можливо заповнити вже після реєстрації. Воно зберігає в собі шлях до зображення, яку користувач завантажив на сервер для свого аватару в профілі.

Всі вище перераховані поля формують клас Person, але крім них в багатьох інших класах бази даних є зв'язок з цим класом через первинний ключ.

Серед допоміжних моделей в модулі userManagement слід зазначити AuthCode. Ця модель пов'язана із класом Person. Під час реєстрації в цій моделі створюється код, який відправляється на почту для підтвердження облікового запису.

2.5.1.1 Структура сховища даних модуля «courses»

Головним класом модуля courses, навколо якого проходить більшість взаємодій в системі є модель Course. Ця модель являє собою курс, на який може записатися учень. Курси створюють вчителі. До редагування конкретного курсу має доступ вчитель, що його створив та адміністратор системи. На рис 2.6. можливо побачити структуру моделі Course.

Course	
UUID	uuid
char	title
char	course_type
key	owner
char	description
char	duration
double	price
char	start_skill
char	final_skill
double	rating

Рис. 2.6. Структура моделі Course

Розглянемо головні поля курсу:

−course_type – це поле має два значення: Online та Offline. Всі курси можливо поділити на два типи: ті для проходження яких потрібні регулярні лекційні матеріали та взаємодія із вчителем, та ті які можливо вивчити самостійно, без постійних консультацій вчителів. Таким чином, це поле показує до якого типу відноситься курс. Завдяки йому, також, можливо відсортувати курси за типом;

−owner – це поле зв'язується через первинний ключ із моделлю користувача із модуля userManagement. Цей користувач повинен бути вчителем. Через це поле, можливо отримати інформацію об авторі курсу;

−duration – це велике текстове поле, в якому зберігається текстова інформація, що описує курс: переваги курсу, теми які будуть вивчатися,

напрямок курсу тощо;

−duration – це поле в якому описується час, який займає проходження курсу;

−prive – це поле в якому вказується ціна курсу. У випадку, якщо курс є платним, для отримання доступу до нього, потрібно разово сплатити суму, вказану в цьому полі;

−rating – це поле показує середню оцінку, яку поставили користувачі, що проходили цей курс. Це поле перераховується автоматично, після встановлення нової оцінки курсу. Ні вчитель, ні адміністратор не можуть редагувати це поле.

Для того, щоб легше було шукати курс за напрямком, що цікавить, реалізована модель Genre та User_genre. Клас Genre має поле name – назву жанру і пов'язується із курсом за принципом багато до багатьох. Створювати нові об'єкти цієї моделі може лише адміністратор проєкту. Своєю чергою, клас Genre відповідає за обрані жанри конкретного користувача. Ця модель дозволяє повертати учню курси, що його цікавлять в залежності від його підписок.

Як можливо було побачити в структурі класу Course, в ньому знаходиться інформація, яка описує курс, але там немає його наповнення. За наповнення курсу відповідає модель Topic. Структура курсу побудована наступним чином – курс поділяється на теми, в яких є певні матеріали, в темах можуть бути підтеми. Розглянемо поля моделі Topic:

−title – це поле є назвою теми;

−sub_topic – це поле є первинним ключем на саму себе, тобто на модель Topic. Це дозволяє створити деревоподібну структуру тем, в якій, у будь-якої теми, може бути скільки завгодно підтем.

До кожної темі прикріплені матеріали, тобто наповнення курсу. Вони реалізовані в моделі Material і мають наступні поля:

−position – це поле дозволяє структурувати матеріали в темі. Воно має цілочисельні позивні значення, що не повторюються і показують в якому порядку повинні розташовуватися матеріали в темі;

−description – це велике текстове поле, в якому знаходиться головна інформація матеріалу. Вона може бути лише із тексту, або можливо прикріпити

графічні матеріали до тексту завдяки моделі MediaFile;

−topic – це поле є первинним ключем, що вказує на модель Topic.

Модель MediaFile є допоміжною. Вона являє собою будь-який графічний файл, який можливо додати до матеріалу за принципом багато до багатьох. Структурно він схожий із моделлю Material. В ньому так само є поле position, що дозволяє віддавати файли в певній послідовності, але замість поля description в ньому є поле file, в якому зберігається шлях до графічного файлу.

Щоб краще зрозуміти структуру курсу, опишу структуру курсу Алгоритми на C++(Course). В першій темі(Topic) в курсі буде Алгоритми сортування, до цієї темі буде прикріплений один матеріал(Material), в якому буде ознайомлювальна інформація про алгоритми сортування. В цій темі буде декілька підтем(Topic), кожна з яких буде відповідати за сімейство алгоритмів сортування, чи за конкретний алгоритм, до яких крім тексту будуть прикріплені графічні файли(MediaFile) зі схематичним зображенням алгоритмів. На рисунку 2.7 зображена схема курсу в базі даних.

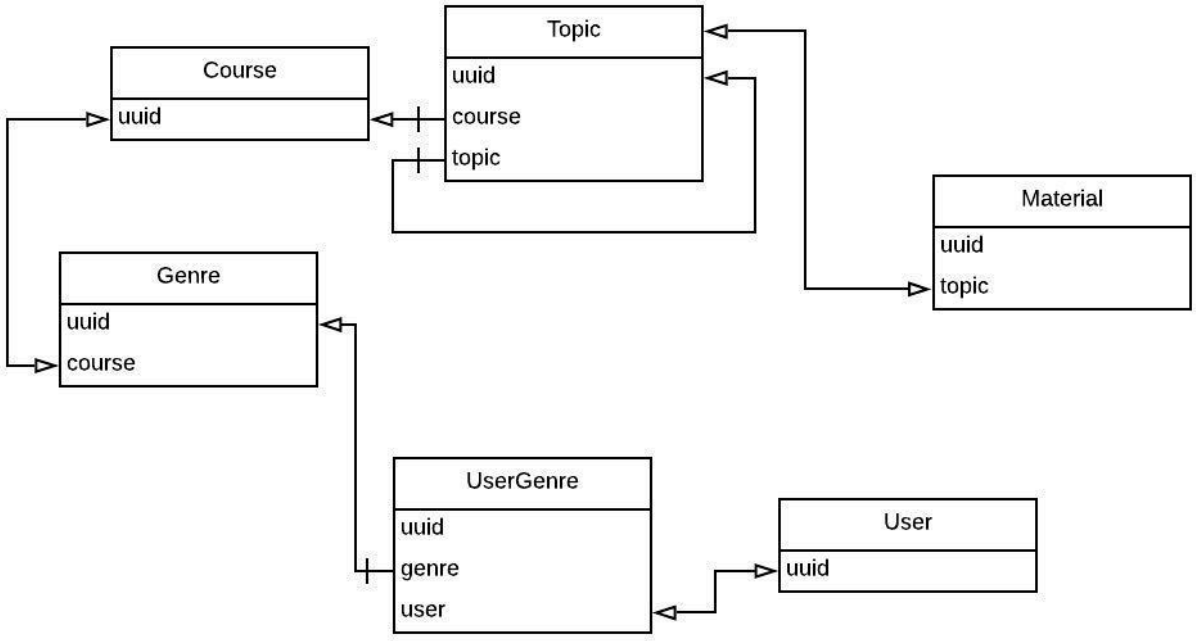


Рис. 2.7. Повна схема курсу в базі даних

Для взаємодії між курсом та учнем, була створена модель UserCourse.

Більшість дій учня, таких як оцінювання курсу, отримання розкладу, виконання завдання тощо, прив'язуються до цієї моделі. Об'єкт моделі створюється при записі учня на курс. Можливо виділити наступні важливі поля:

−rating – це цілочисельне поле з діапазоном від 1 до 5. Це оцінка курсу, яку користувач може поставити після запису на курс. Середній рейтинг курсу рахується завдяки цьому полю;

−person – це первинний ключ, що вказує на самого користувача;

−course – це первинний ключ, що вказує на курс;

−group – це первинний ключ, що вказує на групу, якщо вона є.

Для вчителя є аналогічна модель, але без поля з рейтингом – TeacherCourse.

На рис 2.8. зображена взаємодія курсу із користувачами системи.

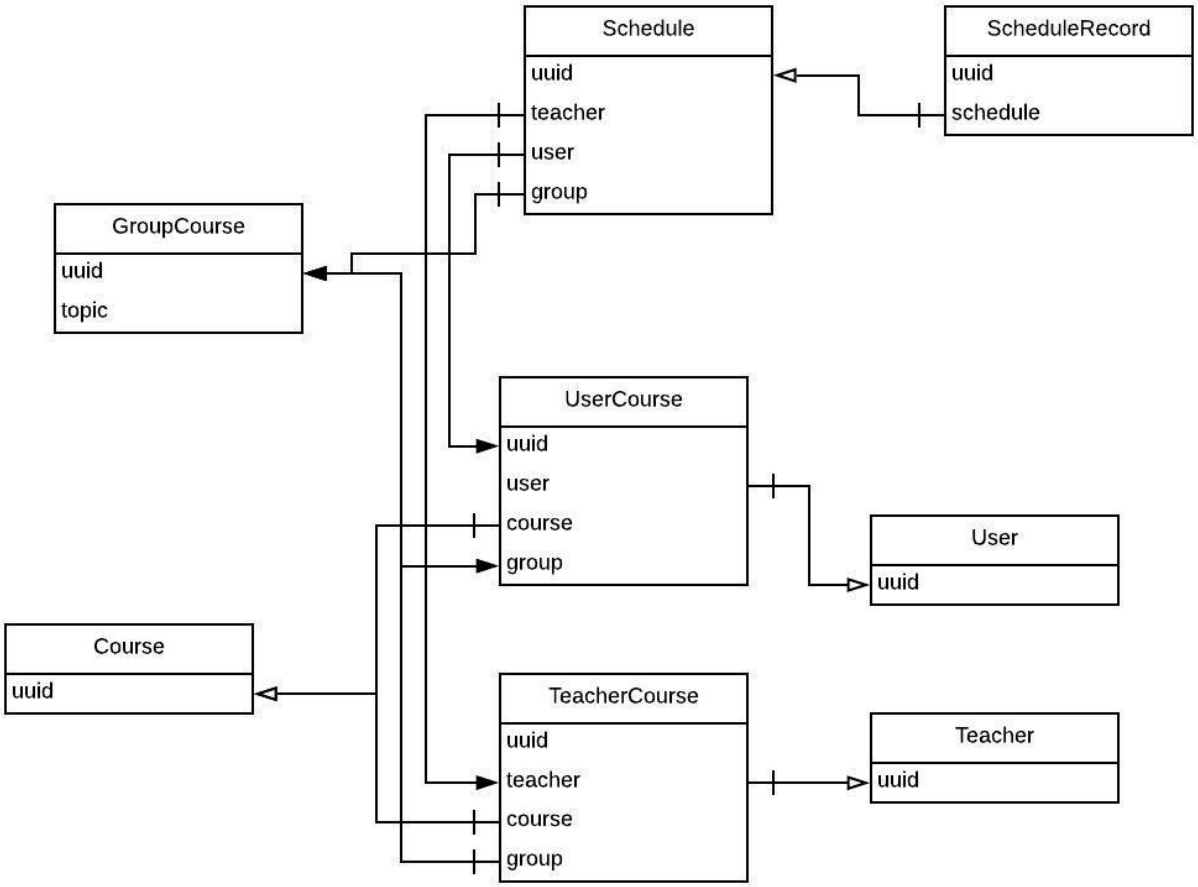


Рис. 2.8. Структура взаємодії курсу із користувачами

Система також підтримує заняття в групі. Це реалізована завдяки моделі GroupCourse. У випадку, якщо заняття групові, моделі TeacherCourse та UserCourse мають первинні ключ, що вказує на групу, до якої вони відносяться.

Для того, щоб можливо було налагодити онлайн курси, які потребують постійних лекційних завдань є модель Schedule. Цей клас прив'язаний до TeacherCourse та до UserCourse. Ця модель – сукупність записів у розкладі. Кожен запис у розкладі – об'єкт класу ScheduleRecord. В класі ScheduleRecord можливо налаштувати день неділі, час та чи буде заняття повторятися кожну неділю або лише разове.

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними застосунку, що розробляється є дані, які відправляються завдяки HTTP запитам.

При отриманні запиту на стороні сервера проходить декілька перевірок. З самого початку перевіряється наявність вказаного шляху. Після цього проходить перевірка на наявність реалізації команді запиту. Якщо все це вказано правильно, то проходить перевірка правильності переданих даних, якщо це потрібно.

Вхідними даними системи для учня є:

- унікальний ідентифікатор авторизованої сесії;
- ідентифікатор курсу для вивчення;
- ідентифікатор завдання та його рішення;
- ідентифікатор курсу для перегляду;
- ідентифікатор теми для вивчення;
- ідентифікатор матеріалу для ознайомлення.

Вхідними даними системи для вчителя є:

- унікальний ідентифікатор авторизованої сесії;
- JSON файл з інформацією для створення курсів;
- JSON файл для створення розкладу;
- JSON файл для редагування розкладу;

− JSON файл для зміни курсів;

− ідентифікатор завдання учня для перевірки.

Вхідними даними системи для адміністратора є:

− унікальний ідентифікатор авторизованої сесії;

− ідентифікатор курсу для його видалення;

− ідентифікатор користувача для його блокування.

Вихідними даними застосунку для учня є:

− авторизаційний токен для подальшої роботи з додатком;

− переляк своїх курсів;

− переляк виконанних завдань;

− інформація профілю;

− розклад занять;

− результат виконання завдання;

− інформація з обраного курсу;

− інформація з обраної теми;

− обраний матеріал.

Вихідними даними застосунку для вчителя є:

− авторизаційний токен для подальшої роботи з додатком;

− перелік створених курсів;

− розклад занять з курсів;

− виконані завдання учнів;

− список учнів;

− створений курс;

− створенні теми.

Обмін між клієнтом і сервером відбувається у форматі JSON.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Для запуску розробленої програми можна використовувати будь-який пристрій, на якому є доступ до мережі інтернет та є HTTP клієнт, зокрема, веббраузер.

2.7.2. Використані програмні засоби

Для функціонування додатку необхідна будь-яка платформа, що підтримує мову Python.

2.7.3. Виклик та завантаження програми

Для того, щоб запустити проєкт, потрібен налаштований вебсервер.

Для доступу до програми потрібне будь-який пристрій з доступом до інтернету.

2.7.4. Опис інтерфейсу користувача

2.7.4.1. Опис програмного інтерфейсу

Більшість запитів можуть бути виконані будь-яким користувачем і для цього не потрібно бути зареєстрованим в системі. Такі запити, зазвичай, повертають загальну інформацію, таку як список всіх курсів. Не зареєстровані користувачі мають доступ до наступних викликів:

−api/courses/ – в цьому виклику команда GET не потребує авторизації. В запиті повинен бути JSON з параметром «type», «course» або «genre». Параметр «type» використовується для отримання списку усіх наявних курсів. Якщо передати значення «all», то повернуться всі курси у порядку добавлення. Для того, щоб упорядити список, замість «all» потрібно вказати «ratingDec». Такий

запит поверне список курсів відсортованих по їх рейтингу. Параметр «course» використовується для отримання інформації щодо конкретного курсу. Значенням цього параметру є назва курсу. У випадку якщо вказаного в запиті курсу не існує, повернеться відповідна помилка. Параметр «genre» дозволяє отримати усі курси за заданим жанром.

−api/genre/ – завдяки цьому виклику можливо отримати список всіх жанрів курсів, що доступні. Для цього потрібно відправити запит типу GET з пустим тілом.

−api/register/ – цей ендпоінт дозволяє зареєструвати нового користувача в системі(учня чи вчителя) через систему серверу. Для цього в запиті повинен бути параметр «person» з полями «email», «password» та «type». На сервері проходить валідація цих полів, та в випадку успіху створює нового користувача і повертає токен для подальшої авторизації.

−api/token/ – виклик для отримання токену для авторизації. Потребує поля «email» і «password».

−api/token/refresh/ – ендпоінт для оновлення jwt токену.

−api/user/google/ – виклик дозволяє авторизуватися завдяки обліковому запис google.

2.7.4.2. Опис програмного інтерфейсу учня

Усі запроси учня потребують токена для авторизації. У випадку відсутності токена, повертається помилка авторизації.

Учень має доступ до наступних викликів:

−api/userGenre/ – виклик повертає жанри на які підписаний користувач.

−api/schedule/ – виклик приймає JSON типу GET з полем «course», в якому потрібно вказати унікальний ідентифікатор курсу і повертає його розклад.

−api/userCourses/ – виклик типу GET, який повертає всі курси на які підписаний учень.

−api/courses/topics/ – виклик приймає два обов'язкових поля: «course» та

«topics». їх значення - назви курсу та теми відповідно. У випадку правильності полів, повертає JSON із вказаною темою курсу.

−api/courses/topics/materials/ – виклик приймає поля «course», «topics» та «material» та повертає файл матеріалу.

−api/tasks/ – виклик приймає ідентифікатор завдання, та повертає JSON із завданням на рішення.

2.7.4.2. Опис інтерфейсу для вчителя

Усі запроси вчителя потребують токен для авторизації. У випадку відсутності токена, повертається помилка авторизації.

Вчитель має доступ до наступних викликів:

−api/courses/ – при використанні команд запиту DELETE та POST, потрібно бути авторизованим, як вчитель. Команда POST дозволяє створити новий курс. Для цього потрібно передати в запиті усю необхідну інформацію для створення курсу. На верхньому рівні JSON знаходиться поле «course». Якщо це поле відсутнє, запит поверне помилку. В полі знаходиться інформація о курсі, а саме: «title», «type», «genres», «description», «duration» та «topics». В полі «title» вказується назва курсу. Для поля «type» вказується одне з двох значень – «online» чи «offline». Якщо значення цього поля буде невірним. повернеться помилка. Поле «genre» – це список значень string, кожне значення якого це ідентифікатор вже існуючого об'єкта з сервера. Список усіх доступних жанрів можливо отримати через запит «api/genres/». В полі «description» пишеться опис курсу. Опис повинен мати як мінімум 40 символів і в випадку, коли символів менше, повернеться помилка. Поле «topics» повинне містити список об'єктів тем. Кожна тема має два поля: «title» та «subTopic». Поле «title» обов'язкове і містить в собі назву теми. Своєю чергою поле «subTopic» не обов'язкове. Це поле – список підтем, кожна з яких є об'єктом «topic». У випадку, коли підтем немає, як значення подається пустий список(рис 2.9). Команда DELETE дозволяє видалити існуючий курс, у випадку, якщо вчитель є його власником. В запиті передається

ідентифікатор курсу, повертається статус операції.

−api/courses/topics/ – запит дозволяє взаємодіяти із темами. Вчителю доступні наступні команди: GET, POST, PUT і DELETE. Команда GET працює так само як і учня. Команда POST дозволяє додавати нові теми до курсу, який був створений вчителем. Для цього в запиті передається обов'язкове поле «course», в якому вказується ідентифікатор курсу, та поле «topic». Це поле не обов'язкове. Воно слугує для того, щоб додати нову підтему для вже існуючої теми. Якщо це поле відсутнє, тоді тема додається до зовнішнього шару тем. Команда PUT дозволяє змінити назву теми або її місце знаходження в курсі. Для цього, крім ідентифікатора курсу та теми, передається два не обов'язкових поля: «title» та «topic». Для функціонування команди, потрібна наявність як мінімум одного поля з обов'язкових. Команда змінює вказані поля в темі. Команда DELETE приймає ідентифікатор теми, та в випадку наявності теми не сервері та при наявності прав доступу на курс, тема видаляється. ЇЇ підтеми переносяться на шар вище.

−api/course/topics/materials/ – запит дозволяє взаємодіяти із матеріалами теми. Вчителю доступні наступні команди: POST, PUT і DELETE. Команда POST дозволяє додати новий візуальний матеріал та пояснення до нього. В цій команді повинні бути указані наступні параметри: «topic», «position» та від нуля до декількох полів «mesia». Поле «topic» містить ідентифікатор теми до якої додається матеріал. Поле «position» вказує порядковий номер матеріалу, тобто який по черзі він розташований в темі. Цей номер повинен бути позитивним. У випадку, якщо вже існує матеріал з таким порядковим номером, переданий об'єкт ставиться на вказане місце, при цьому порядкові номера наступних матеріалів збільшується на одиницю. Команда PUT приймає такі самі поля як і команда POST, але, крім ідентифікаторів, інші поля не обов'язкові. Команда дозволяє редагувати матеріали. Команда DELETE дозволяє видалити матеріал. У випадку успіху також видаляються медіафайли матеріалу.

−api/course/topics/materials/mediaFile/ – запит має команди POST та DELETE. На відміну, від інших запитів, формат в командах POST та PUT дані

відправляються не в форматі JSON, а в форматі Multipart Form. Цей формат дозволяє відправляти не лише текстові поля, а й медіафайли. Завдяки команді POST вчитель може завантажити файл до конкретної теми. Для цього потрібно добавити поле material в якому вказується ідентифікатор матеріалу, до якого буде прикріплений файл, поле position, яке вказує послідовність файлів у випадку коли матеріал містить кілька файлів та поле file, до якого прикріплюється файл.

–api/teacherCourses/ – цей запит має команду GET, яка повертає список усіх курсів.

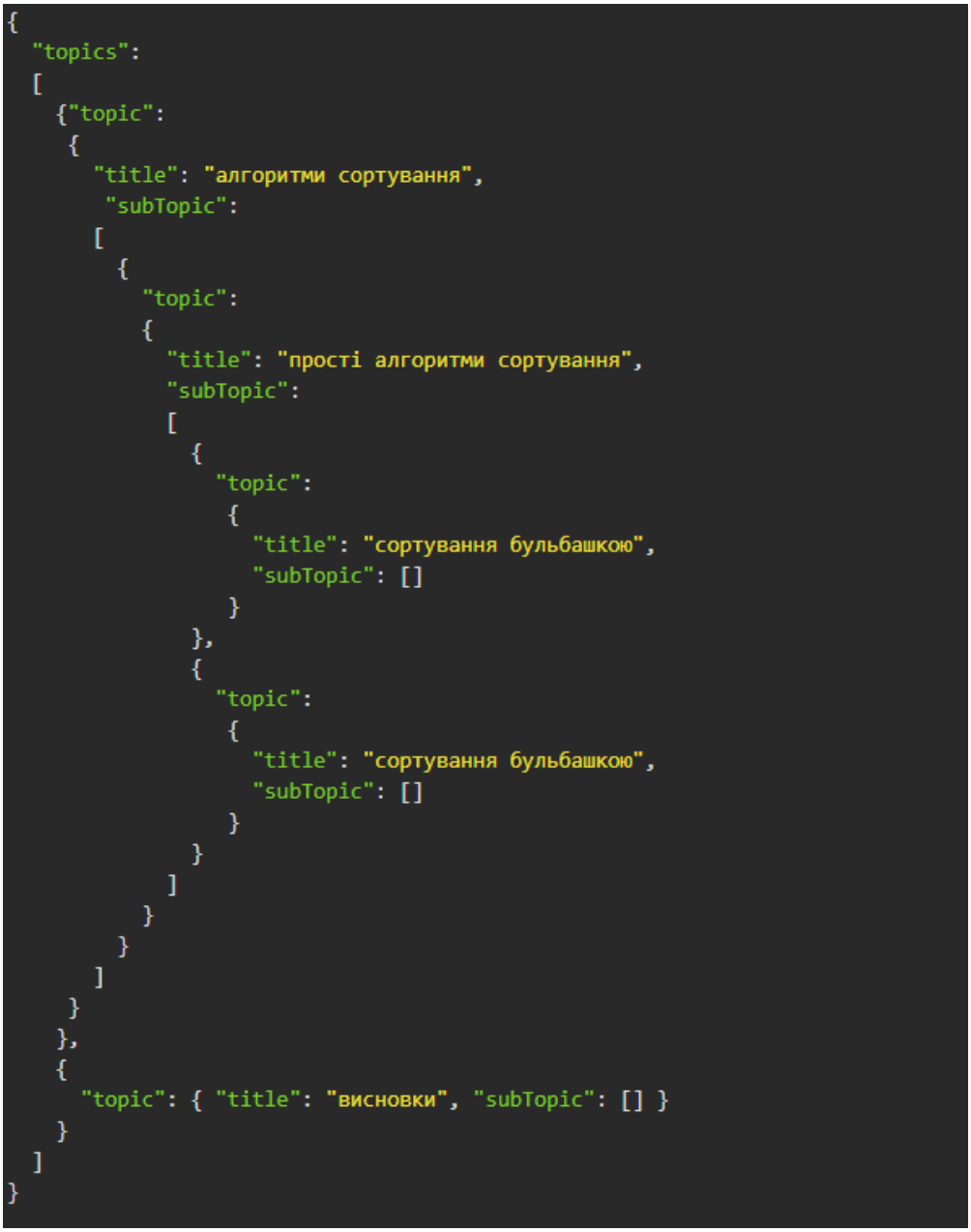


Рис 2.9. JSON для створення тем курсу

2.7.4.3. Опис інтерфейсу для адміністратора

На відміну від інших користувачів системи, адміністратор має доступ до адміністративної панелі із UI. Основна частина панелі створюється автоматично, завдяки Django. Сама панель створюється на основі бази даних, та прав доступу до об'єктів, що були налаштовані в системі.

Для того, щоб отримати доступ до можливостей адміністратора, потрібно авторизуватись, використовуючи обліковий запис адміністратора, для створення якого потрібен доступ до системи рис. 2.10.

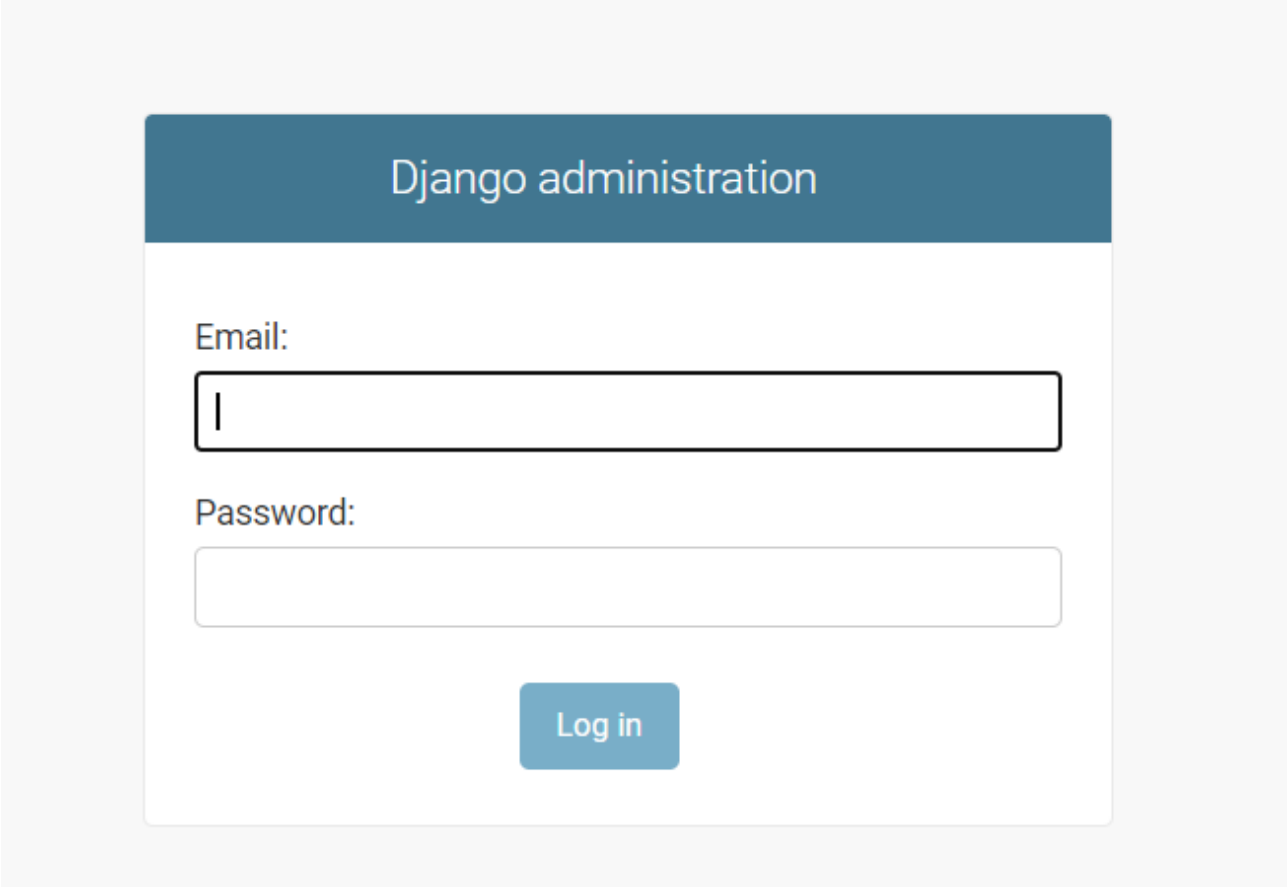


Рис 2.10. Авторизація в адміністративній панелі

На головній сторінці адміністративної панелі знаходиться перелік усіх доступних моделей рис.. При вибиранні конкретної моделі, проходить перехід на сторінку зі списком всіх існуючих об'єктів цієї моделі (рис. 2.11). На цій сторінці можливо додавати нові об'єкти, натиснувши на відповідну кнопку,

видалити обрані об'єкти, та редагувати конкретний об'єкт рис.

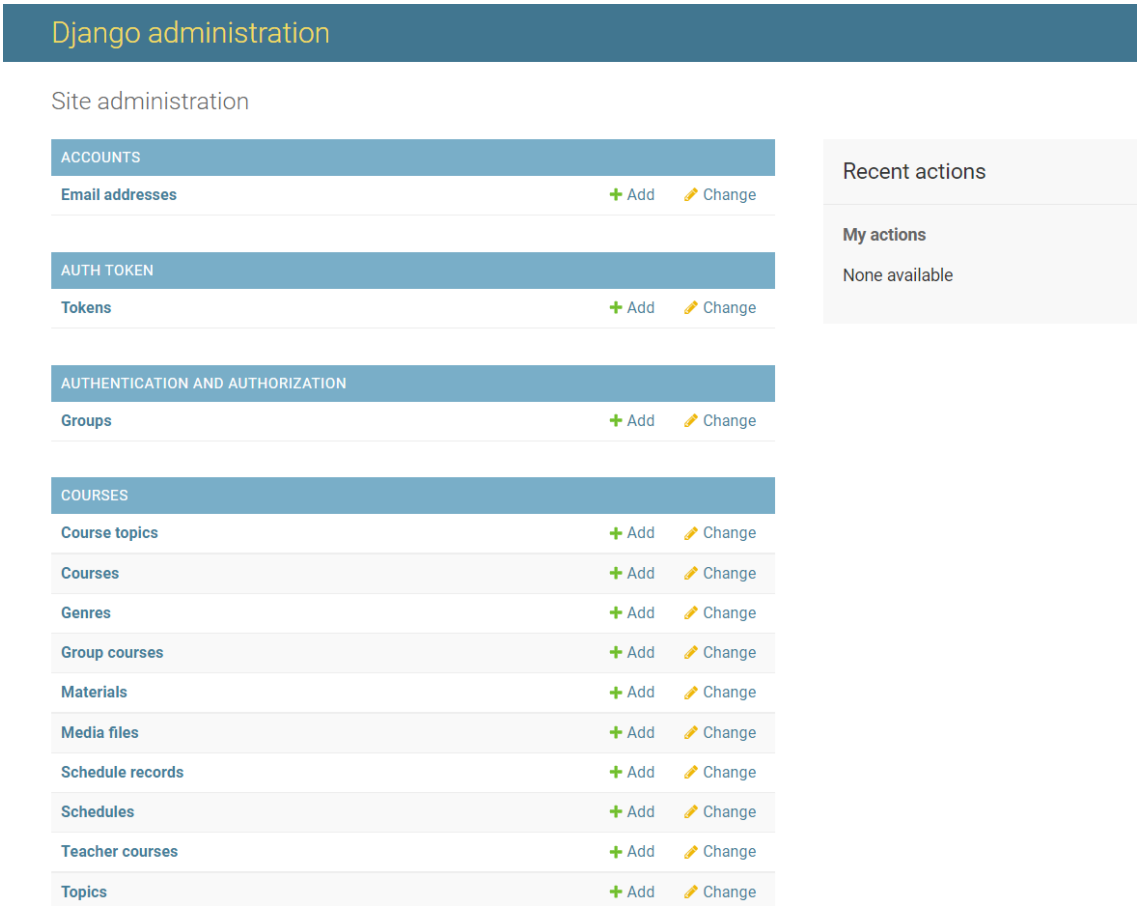


Рис 2.11. Перелік доступних моделей на сайті.

Перелік в будь-якій з категорій виглядає як показано на рисунку 2.12

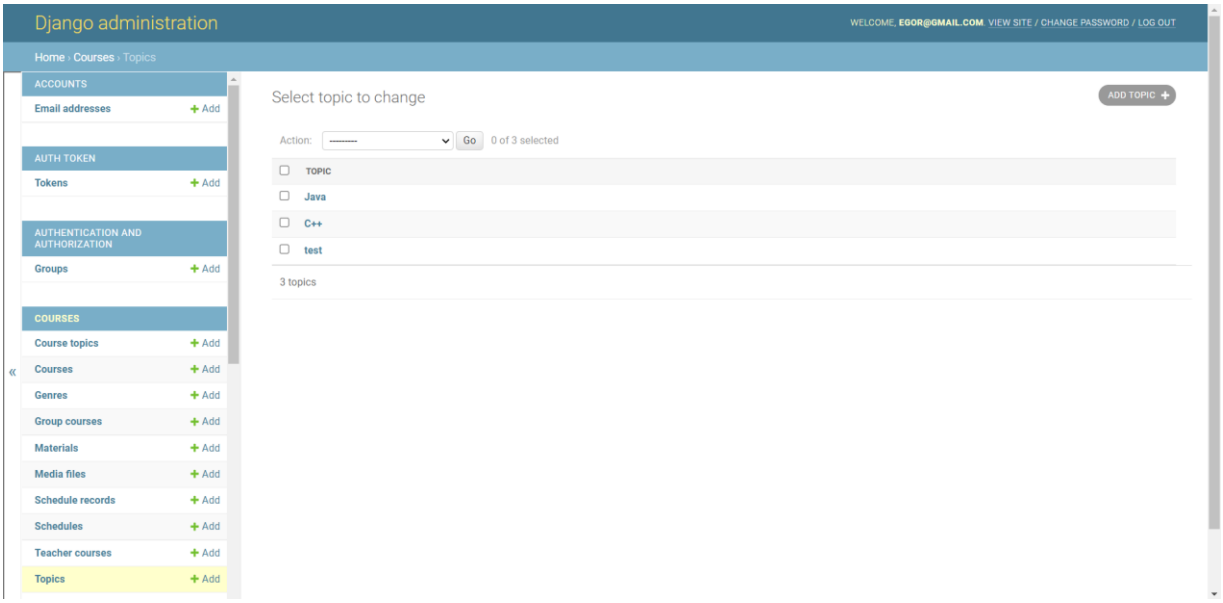


Рис 2.12. Перелік об'єктів в адміністративній панелі

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані для розробки програмного забезпечення:

1. передбачуване число операторів програми – 1027;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста– 65 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ –12 грн/год.

Нормування праці в процесі створення ПЗ э достатньо складним через творчий характер праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \qquad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n -витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ -витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (тегів):

$$Q = q \cdot C \cdot (1+ p), \tag{3.2}$$

де q - передбачуване число операторів (1027);

C - коефіцієнт складності програми (1,4);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Підставивши у формулу (3.2) початкові дані, отримуємо умовне число операторів в програмі:

$$Q = 1027 \cdot 1,3 \cdot (1+ 0,05) = 1401,86.$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,} \tag{3.3}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

З урахуванням початкових даних, використавши формулу (3.3) отримуємо витрати праці на вивчення опису завдання:

$$t_u = (1401,86 \cdot 1,3) / (75 \cdot 1,2)=20,25, \text{ людино-години.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}, \text{ людино-годин,} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), отримуємо:

$$t_a = 1401{,}86 / (20 \cdot 1{,}2) = 58{,}41 \text{ людино-годин.}$$

Витрати на складання програми по готовій схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}, \text{ людино-годин.} \qquad (3.5)$$

Підставивши відповідні значення в формулу (3.5), отримуємо:

$$t_n = 1401{,}86 / (25 \cdot 1{,}2) = 46{,}73, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}, \text{ людино-годин,} \qquad (3.6)$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 \cdot t_{отл}\text{, людино-години,} \qquad (3.7)$$

З формули (3.6) виходить:

$$t_{отл} = 1401{,}86 / (5 \cdot 1{,}2) = 233{,}64\text{, людино-години,}$$

Підставивши значення в формулу (3.7), отримуємо:

$$t^{k}_{отл} = 1{,}5 \cdot 250{,}25 = 350{,}47\text{, людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до}\text{, людино-годин,} \qquad (3.8)$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k}\text{, людино-годин,} \qquad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}\text{, людино-годин,} \qquad (3.10)$$

Підставляючи відповідні значення в формули (3.8-3.10), отримаємо:

$$t_{др} = 1401{,}86 / (18 \cdot 1{,}2) = 64{,}9\text{, людино-годин,}$$

$$t_{до} = 0{,}75 \cdot 69{,}5 = 48{,}68\text{, людино-годин,}$$

$$t_д = 69,5 + 52,13 = 113,58,$$ людино-годин,

З формули (3.1), отримаємо повну оцінку трудомісткості розробки ПЗ:

$$t_д = 50 + 20,25 + 58,41 + 46,73 + 350,47 + 113,58 \approx 639,$$ людино-годин,

3.2. Розрахунок витрат на бібліотеки WEB API

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ},$$ грн. (3.11)

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР},$$ грн. (3.12)

де t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

Підставивши дані у формулу (3.12) отримуємо:

$$З_{ЗП} = 639,44 \cdot 65 = 41563,6$$ грн.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot С_{мч},$$ грн, (3.13)

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (12 грн/год).

Вартість необхідного для налагодження машинного часу за (3.13):

$$З_{МВ} = 350,47 \cdot 12 = 4205,64, \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 41563,6 + 4205,64 = 45769 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = t / (B_k \cdot F_p), \text{ міс.} \qquad (3.14)$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

З (3.14) визначимо витрати на створення програмного продукту:

$$T = 639 / (1 \cdot 176) \approx 3,6, \text{ міс.}$$

Висновок: У економічному розділі були визначені витрати на створення даного програмного забезпечення. Вартість даного додатку склала 45769 грн. Очікуваний час створення – 3,6 місяця. Цей термін пов'язаний зі значною кількістю операторів і включає в себе час для дослідження та розробки алгоритму розв'язання задачі, розробку архітектури, створення WEB API та підготовку документації.

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи була досягнута мета, поставлена на початку роботи – була створена гнучка бібліотека WEB API для онлайн навчальної платформи, що реалізує можливість для учнів отримувати актуальні знання в зручному для нього форматі, та можливість для вчителів публікувати навчальний контент.

Актуальність розробленого продукту обумовлена зростанням ринку освітніх онлайн послуг та потребі у створені нових більш потужних та зручних засобів його публікації.

Розроблена бібліотека WEB API може використовуватися різними клієнтськими програмами на будь-якій платформі за наявності доступу до інтернету.

Структура програми являє собою серверний додаток, написаний на основі фреймворку Django, що використовує мову програмування Python 3. Для реалізації серверу було використано цілий ряд допоміжних бібліотек, зокрема для налагодження REST системи було використано бібліотеку Django REST Framework.

В порівнянні з існуючими аналогами, розроблене програмне забезпечення має ряд переваг:

- може бути використана, як навчальними сайтами, так й навчальними мобільними додатками;

- надає можливість безкоштовного і платного доступу до контенту, з можливістю гнучкого налаштування моделі монетизації платного доступу.

В майбутньому планується покращити та збільшити функціональність даного продукту, покращити систему рекомендацій, шляхом впровадження штучного інтелекту, перенести лекційні матеріали та відеоматеріали на свою платформу. Також, хоча Django надає можливість використання вбудованої адміністративної панелі, для більш гнучкої взаємодії, краще переробити адміністративну панель у вигляді окремого WEB додатку.

Також у даній кваліфікаційній роботі в «Економічному розділі» було визначено трудомісткість розробки програмного забезпечення були підраховані витрати на створення програмного забезпечення і очікуваний період розробки, що складає приблизно 3 місяці (3,6 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Петькова Ю.Р. Журнал Успехи современного естествознания. ООО «Научно-издательский центр Академия Естествознания», 2015. – 199-204 с.

2. Django Documentation. URL: https://docs.djangoproject.com/en/3.2/ (дата звернення: 09.04.2021).

3. Django REST Framework Documentation URL: https://www.django-rest-framework.org/ (дата звернення: 15.04.2021).

4. Sebastian Peurott The JWT Handbook: AuthO Inc, 2016. – 6 с.

5. Ryan Boyd. Getting Started with OAuth 2.0: Programming Clients for Secure Web API Authorization and Authentication. O'Reilly Media, 2012 – 62 с.

6. Рыбальченко М.В. Архитектура информационных систем: учебное пособие. Ч. 1. Таганрог: Изд-во ЮФУ, 2015. – 92 с.

7. Васкевич Д. Стратегии клиент/сервер. Киев : Диалектика,1997. – 45 с.

8. Куроуз Дж., Росс К. Компьютерные сети. Многоуровневая архитектура Интернета. СПб.: Питер, 2004. – 765 с.

9. Вагонова О.Г., Волотковська Ю.А., Романюк Н.Н. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Програмне забезпечення”. Дніпропетровськ: Національний гірничий університет, 2013. – 11 с.

10. Шевцова О.С., Удовик І.М., Коротенко Л.М. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 121 «Програмна інженерія». Дніпропетровськ: НТУ «Дніпровська політехніка», 2019. – 65 с.

11. Пьюривал С. Основы разработки веб-приложений. – СПб.: Питер, 2015. – 272 с.

12. Schwaber K. Agile Project Management with Scrum Developer Best Practices. Microsoft Press, 2004. – 192 p.

13. O'Reilly Media. PostgreSQL: Up and Running. O'Reilly Media, Inc. 2012 – 172 с.

14. Adrian Holovaty. The Definitive Guide to Django: Web Development Done Right (Expert's Voice in Web Development). Apress, 2009 – 240c.

15. Luciano Ramalho. Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media, 2015 – 418 c.

16. Alex Soojung-Kim Pang. Rest: Why You Get More Done When You Work Less. Basic Books, 2016 – 16 c.

17. Coursera. URL: https://www.coursera.org/ (дата звернення: 02.04.2021).

18. edX. URL: https://www.edx.org/ (дата звернення: 02.04.2021).

19. Skillshare. URL: https://www.skillshare.com/ (дата звернення: 02.04.2021).

20. Github. URL: https://github.com/features/ (дата звернення 01.04.2021).

ДОДАТОК А

КОД ПРОГРАМИ

view.py

```
from django.shortcuts import render
from rest_framework import generics, permissions
from rest_framework.decorators import api_view, permission_classes, action
from rest_framework.response import Response
from django.core.mail import send_mail
from rest_framework.permissions import IsAuthenticated, AllowAny
from django.contrib.sites.models import Site
from rest_framework_simplejwt.tokens import BlacklistedToken, RefreshToken
from rest_framework_simplejwt.token_blacklist.models import OutstandingToken

from django.utils.decorators import method_decorator
from django.views.decorators.csrf import csrf_protect
from django.conf import settings
from dj_rest_auth.registration.views import SocialLoginView
from allauth.socialaccount.providers.oauth2.client import OAuth2Client
from allauth.socialaccount.providers.google.views import GoogleOAuth2Adapter

from .serializers import *
from .permissions import *
from .models import Person
from .services import fileUpload
from study_platform.settings import MEDIA_URL

import socket
class RegistrationAPI(generics.GenericAPIView):  # , CsrfExemptMixin):s
    serializer_class = PersonSerializer
    queryset = Person.objects.all()
```

```
    permission_classes = [AllowAny, ]


    def get(self, request):
        return Response(status=405)


    def post(self, request, *args, **kwargs):
        if 'validate' in request.query_params:
            validationCode = request.query_params['validate']
            if validationCode == '':
                return Response({'status': 'error', 'message': 'Validation code is
absent'}, status=400)
            email = request.data.pop('email', None)
            if not email:
                return Response({'status': 'error', 'message': 'Email is absent'},
status=400)


            validationResult = AuthCodeSerializer.validateCode(
                email, validationCode)
            if validationResult == False:
                return Response({'status': 'error', 'message': 'Code is wrong'},
status=400)


            person = self.get_queryset().filter(email=email)
            if len(person) == 1:
                person[0].is_active = validationResult
                person[0].save()
                return Response({'status': 'OK', 'message': f'User {person[0].email}
was validated!'}, status=200)
            else:
                return Response({'status': 'error', 'message': f'Something went
wrong'}, status=400)
        else:
            personData = request.data.pop('person', None)
            if not personData:
                return Response({'status': 'error', 'message': 'Person data is
absent'}, status=400)
```

```
            if 'email' in personData and 'password' in personData and 'phone' in
personData and 'first_name' in personData and 'last_name' in personData:

                password = personData.pop('password')

                serializer = self.get_serializer(

                    data=personData)



                if serializer.is_valid():

                    authCode = AuthCodeSerializer(

                        data={'email': personData['email'], })

                    if authCode.is_valid():

                        code = authCode.save()

                        result = send_mail(

                            f'Код подтверждения на учебной платформе: "PlatformName"',

                            f'От Вас поступил запрос на регистрацию на
"PlatformName"\nРегистрация была произведена на сайте https://study-
platform.duckdns.org\nДля того, что бы подтвердить регистрацию, используйте ваш код
подтверждения {code.code}',

                            'from@gmail.com',

                            [f'{personData["email"]}', ],

                            fail_silently=False,

                        )

                        if result == 1:

                            serializer.save()

                            person = self.get_queryset().filter(

                                email=personData['email'])[0]

                            person.set_password(password)

                            person.is_active = False

                            person.save()

                            return Response({'status': 'OK', 'message': f'Person
{personData["email"]} was added, verify your account with code, that was sended to your
email'}, status=200)

                        return Response({'status': 'Error', 'message': f'Person
{personData["email"]} wasn\'t added, cause the verification code can\'t be send to your
email'}, status=400)

                    return Response({'status': 'Error', 'message': authCode.errors},
status=400)

                return Response({'status': 'Error', 'message': serializer.errors},
status=400)
```

```
            return Response({'status': 'Error', 'message': 'There are something wrong
with your JSON data'}, status=400)



class UserAPI(generics.GenericAPIView):  # , CsrfExemptMixin):

    queryset = Person

    serializer_class = PersonSerializer

    permission_classes = [IsAuthenticatedReadOnly, ]


    def put(self, request):

        if 'type' in request.query_params:

            if request.query_params['type'] == 'passwordChange':

                if len(Person.objects.filter(email=request.data['email'])) == 1:

                    authCode = PassChangeCodeSerializer(

                        data={'email': request.data['email'], })

                    if authCode.is_valid():

                        code = authCode.save()

                        send_mail(

                            f'Востановление пароля на учебной платформе:
"PlatformName"',

                            f'От Вас поступил запрос на востановление пароля на
"PlatformName"\nИспользуйте ваш код подтверждения для востановления пароля:
{code.code}',

                            'from@gmail.com',

                            [f'{request.data["email"]}', ],

                            fail_silently=False,

                        )

                        return Response('Validation code was sended to your email!',
status=200)

                    else:

                        return Response(authCode.errors, status=400)

            elif request.query_params['type'] == 'validate':

                if 'code' in request.query_params and 'email' in request.data and
'password' in request.data:

                    code = request.query_params['code']

                    validationStatus = AuthCodeSerializer.validateCode(
```

```
                        request.data['email'], request.query_params['code'])
                    if validationStatus:
                        users = Person.objects.filter(
                            email=request.data['email'])
                        if len(users) > 0:
                            users[0].set_password(request.data['password'])
                            users[0].save()
                            return Response('Password was successfully changed!',
status=200)
                        return Response('Email address is wrong!', status=404)
                    return Response('Something is wrong with your code', status=400)
                return Response('There are not enough data in your request',
status=404)
            elif request.query_params['type'] == 'teacherRequest':
                if request.user.is_coach:
                    return Response('You are already a teacher', status=500)
                if 'description' in request.data:
                    requests = TeacherRequest.objects.filter(
                        person=request.user)
                    if len(requests) > 0:
                        return Response('You have already done the request',
status=500)

                    TeacherRequest.objects.create(
                        person=request.user, description=request.data['description'])

                    send_mail(
                        f'Запрос на получение прав преподавателя от пользователя
{request.user.first_name} {request.user.last_name}',
                        f'Поступил запрос на получение прав преподавателя от
пользователя {request.user.first_name} {request.user.last_name} на платформе
"PlatformName"',
                        'from@gmail.com',
                        [f'{settings.EMAIL_HOST_USER}', ],
                        fail_silently=False,
                    )
```

```
                return Response('Request was sent')
        return Response('There are no such type parameter', status=404)
    else:
        if self.userDataChange(request):
            return Response('Data was updated!', status=200)
        return Response('Something went wrong!', status=400)
    return Response('Something is wrong with your request!', status=400)

def userDataChange(self, request):
    if not request.user.is_authenticated:
        return False
    print(request.data)
    user = request.user
    if len(request.FILES) != 0:
        if 'file' in request.FILES:
            url = fileUpload(request.FILES, user.id)
        else:
            url = user.photo
    else:
        url = user.photo
    if len(request.data.keys()) > 0:
        user = request.user
        serializer = self.get_serializer(
            user, data=request.data, partial=True)
        if serializer.is_valid():
            serializer.save()
            print(serializer.data)
            user = Person.objects.filter(
                email=request.user.email).update(photo=url)
            return True
    return False

def get(self, request):
```

```
        user = request.user
        role = ''
        if user.is_active:
            if user.is_coach:
                role = 'teacher'
                TeacherRequest.objects.filter(person=request.user).delete()
            else:
                role = 'student'
        else:
            role = 'blocked'
        domain = request.META['HTTP_HOST']
        obj = {
            "user": {
                "email": request.user.email,
                "phone": request.user.phone,
                "first_name": request.user.first_name,
                "last_name": request.user.last_name,
                "second_name": request.user.second_name,
                "photo": f'https://{domain}{MEDIA_URL}{request.user.photo}',
                "role": role
            }
        }
        return Response(obj)

    def post(self, request):
        OutstandingToken.objects.filter(
            user__email=request.data['email']).delete()
        return Response('Tokens deleted')


class LogoutView(generics.GenericAPIView):
    queryset = Person
    permission_classes = (IsAuthenticated,)
```

```
    serializer_class = PersonSerializer

    def get(self, request):
        return Response('Ok', status=200)

    def post(self, request):
        # tokens = OutstandingToken.objects.filter(user_id=request.user.id)
        # for token in tokens:
        #     t, _ = BlacklistedToken.objects.get_or_create(token=token)
        try:
            print(request.data)
            refresh_token = request.data["refresh_token"]
            token = RefreshToken(refresh_token)
            token.blacklist()

            return Response(status=205)
        except Exception as e:
            print(e)
            return Response(status=400)



class GoogleLogin(SocialLoginView):
    authentication_classes = []  # disable authentication
    adapter_class = GoogleOAuth2Adapter
    callback_url = "http://localhost:3000/login/"
    client_class = OAuth2Client



class GoogleLogOut(generics.GenericAPIView):
    def get(self, request):
        logout_uri = "http://localhost:3000/"

        if not ('client_id' in request.query_params and 'logout_uri' in
request.query_params):
```

```
            return Response('Error', status=500)
        else:
            if request.query_params['client_id'] == settings.OAUTH_CLIENT_ID and
request.query_params['logout_uri'] == logout_uri:
                return Response('Logged out')
            else:
                return Response('Something is wrong', status=500)



class GoogleUser(generics.GenericAPIView):


    def get(self, request):
        user = request.user
        role = ''
        if user.is_active:
            if user.is_coach:
                role = 'teacher'
                TeacherRequest.objects.filter(person=request.user).delete()
            else:
                role = 'student'
        else:
            role = 'blocked'
        domain = request.META['HTTP_HOST']
        obj = {
            "email": request.user.email,
            "first_name": request.user.first_name,
            "last_name": request.user.last_name,
            "second_name": request.user.second_name,
            "photo": f'https://{domain}{MEDIA_URL}{request.user.photo}',
            "role": role
        }
        return Response(obj)

…
```

Продовження коду приведене на диску.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Розробка веб-клієнта інформаційної системи ветеринарної клініки»

студентки групи 121-17-1 Вахрушина Єгора Валентиновича

Керівник економічного розділу,

доцент каф. ПЕП та ПУ, к.е.н. **О.Г. Вагонова**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Вахрушин.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_Вахрушин.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF.
Програма	
Програма_Вахрушин.rar	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація_Вахрушин.pptx	Презентація кваліфікаційної роботи.