МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Інформаційна система підбору пісень на основі вподобань користувача»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-12 Чорненький Микита Анатолійович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2022 р.

Науковий керівник _____ к.т.н., доц., Парфененко Ю.В.
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Суми-2022

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____С. М. Ващенко
«___» _____ 2022 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Чорненький Микита Анатолійович

(прізвище, ім'я, по батькові)

1 Тема проекту Інформаційна система підбору пісень на основі вподобань користувача

Затверджена наказом по університету від «__»_____2022 р. № _____

2 Термін здачі студентом закінченого проекту «12» ___грудня___ 2022 р.

3 Вхідні дані до проекту інформація про плейлисти, інформація про акаунт користувача, інформація від рекомендаційної системи, інформація про пісні.

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Аналіз предметної області, постановка задачі та методів її дослідження, моделювання рекомендаційної системи, реалізація інформаційної системи для рекомендації музики.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація (25 слайдів)

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	**Підпис, дата**	
		Завдання видав	**Завдання прийняв**

Дата видачі завдання _____.

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Аналіз предметної області	4 дні	
2	Постановка задачі та методи дослідження	5 днів	
3	Моделювання рекомендаційної системи	5 днів	

4	Реалізація інформаційної системи для рекомендації музики	15 днів	
5	Тестування створеної інформаційної системи	15 днів	

Магістрант _____ Чорненький М.А.

Керівник роботи _____ к.т.н., доц. Парфененко Ю.В.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра – «Інформаційна система підбору пісень на основі вподобань користувача».

Пояснювальна записка складається зі вступу, основної частини, що складається з 4 розділів, висновку, списку використаних джерел із 32 джерелами та 2 додатками. Загальний обсяг роботи 63 сторінки, 34 сторінки основного тексту, 4 сторінки використаних джерел та 18 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці інформаційної системи для рекомендації музики на основі вподобань користувача.

Перший розділ являє собою аналіз актуальності проблеми, проводиться аналіз галузі рекомендаційних систем, проаналізоване поняття того, що таке рекомендаційна система, був проведений аналіз сучасних аналогів для рекомендації музики, була створена порівняльна таблиця, котра показує переваги та недоліки між системами, та були обрані інструменти для реалізації такої інформаційної системи. У другому розділі формується мета та задачі дослідження, проходить вибір методів дослідження, описуються методи за якими проходить реалізація інформаційної системи. У третьому розділі будується структура проекту, будуються діаграми принципів роботи інформаційної системи, діаграми декомпозиції та діаграма використання. У четвертому розділі приведено процес реалізації додатку та приклади роботи веб-додатку.

Результатом роботи став створений веб-додаток з можливістю рекомендації музики за вподобаннями користувача та можливістю перегляду та пошуку плейлистів.

Ключові слова: веб-додаток, рекомендаційна система, Spotify, Django, Python, косинус подібності.

ЗМІСТ

ВСТУП

У світі, в якому велику кількість контенту, що переглядається, почав займати Youtube, Spotify, Netflix, люди все більше почали надавати перевагу візуальному або аудіоконтенту замість читання книжок, перегляду телебачення. Такі сервіси, окрім того, що просто надають великий асортимент медіаконтенту, так у них ще й виходить зробити рекомендацію на основі переглянутого фільму, серіалу чи відео, яка спонукає переглянути увесь можливий контент, що надає сервіс. Також не потрібно забувати про рекламу, яка теж є своїм родом рекомендацією, коли на основі пошукових запитів або голосових повідомлень користувачам починають пропонувати схожі товари або послуги, хоча вони про це навіть не просили.

Сама по собі рекомендаційна система – це алгоритм, метою якого є запропонувати користувачеві релевантні елементи або ж схожі елементи, залежно від сфери, для якої було створено таку систему, змінюються і ці елементи [1].

Такі системи дуже важливі для деяких галузей, оскільки вони можуть приносити прибуток, а також дають змогу виділитися серед інших конкурентів. Як приклад важливості такої рекомендаційної системи можна згадати Netflix, котра влаштувала конкурс на створення нової, набагато кращої системи, ніж у них є, переможець якої міг отримати приз у розмірі один мільйон доларів.

Рекомендаційна система – це підклас системи для фільтрації інформації, яка намагається передбачити "рейтинг" або "перевагу", яку користувач міг би віддати тому чи іншому елементу [2].

В інтернеті існує всього кілька систем для підбору музики, деякі з них належать великим компаніям, які не надто прагнуть поділитися розробкою, для її поліпшення. Решта ж систем, які можна знайти в інтернеті, хоча й не належать до великих компаній, що володіють великими медіасервісами, та на

жаль, не перебувають у відкритому доступі, а ті що є у відкритому доступі, були зроблені занадто давно, це означає, що зроблено їх було за допомогою вже застарілих технологій, а отже, вони не зможуть забезпечити гідного рівня конкуренції з уже наявними системами.

Тому для цієї роботи було обрано саме цю проблему. Предметом для досліджень стане система рекомендацій для підбору музики.

Метою кваліфікаційної роботи магістра є створення інформаційної системи підбору пісень на основі вподобань користувача. Інформаційна система буде реалізована у вигляді веб-додатку з відкритим вихідним кодом, що здійснюватиме рекомендацію музики на основі власних плейлистів, а також додатковий функціонал у вигляді можливості перегляду вже існуючих плейлистів, а також пошуку за ними. Відкритий вихідний код дозволить через великий проміжок часу замінити застарілу на той момент часу інформаційну систему на новішу, що дасть змогу постійно покращувати алгоритми рекомендації.

Відповідно маючи таку мету, потрібно вказати задачі дослідження:

- зробити аналізгалузі рекомендаційних систем;
- виконати проектування інформаційної системи підбору пісень на основі вподобань користувача;
- реалізувати систему, яка дасть змогу зробити рекомендацію на основі плейлисту та на основі всіх прослуханих пісень;
- провести тестування розробленої інформаційної системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Інформація завжди була невіддільною частиною життя людини, зокрема знання про те, коли настане зима, або де знайти їжу, як її приготувати, що можна з'їсти, а що їсти не варто, були важливими. По суті, стосунки між батьками та їхніми дітьми можна вважати рекомендаційною системою, оскільки досвідченіші батьки постійно передавали накопичений роками досвід і рекомендували своїм дітям те, як правильно жити й що взагалі потрібно в житті робити. Але з роками проблеми людей почали набувати іншого характеру, і зараз, наприклад, сучасні люди не знають, на що витратити свої гроші, наприклад, який смартфон купити і який одяг сьогодні одягнути, тому й рекомендаційні системи змінили свій характер, перетворившись із системи, що дає змогу виживати потомству, на систему, що дає змогу людям простіше провести свій час, не замислюючись, який фільм подивитися, або який альбом свого улюбленого гурту послухати, або ж інформацію, яку варто прочитати, щоб щось нове дізнатися.

Першою робочою системою стала комунікаційна система Usenet, яку було створено в університеті Дюка. Вона давала змогу обмінюватися текстовим контентом один з одним. Суть роботи програми полягала в тому, що користувачі були розділені на групи і підгрупи, де користувачі просто рекомендували одне одному контент який їм сам подобався, але орієнтації на вподобання користувачів на той момент ще не було. Після з'явилася автоматична система так званий "розумний бібліотекар", який спершу дізнавався вподобання користувачів, а потім рекомендував книги. Система на основі зібраної інформації створювала стереотипні групи, а потім рекомендувала книги, які були прочитані в цій групі. Зараз такий підхід вважається застарілим. Але в той момент це був зсув парадигм у можливості

автоматизації послуг. Так само цей підхід розкритикували через недосконаліть системи. На думку експертів, люди могли спеціально виділяти свої якості, щоб здатися більш унікальними, хоча насправді такими не були, що й ускладнювало створення системи, яка добре функціонує.

З часом такі системи розвинулися у два різні напрямки, де однією була "спільна фільтрація", а другою "фільтрація на основі контенту" [3].

Перша відображає вподобання користувачів і пропонує їм контент, який міг би сподобатися користувачеві зі схожими вподобаннями.

Суть "фільтрації на основі контенту" полягає в знанні параметрів об'єкта, який рекомендуватимуть. Наприклад, для музики це були б такі параметри: стиль, виконавець, епоха. А також вона запам'ятовує переваги користувачів щодо цих параметрів або характеристик [1].

Першим прикладом системи, що базується на алгоритмах спільної фільтрації стала система Tapestry, розроблена компанією Xerox PARC, що давала змогу читати документи та давати їм оцінку – подобається їм чи не подобається. Це дало змогу звузити кількість документів, які шукають, а також прискорити їхній пошук за допомогою нотаток і оцінок, які залишали користувачі.

Після цього GroupLens запустила повноцінну систему рекомендацій Usenet у відкритий доступ, де користувачеві могли порекомендувати статті після того, як він робив оцінки деяким статтям із системи.

Пізніше з'явилося багато тематичних сайтів рекомендацій, такі як Ringo, так само з'явилася перша система рекомендації музики Firefly й так само система рекомендацій фільмів BellCore.

Першою системою, яка спробувала охопити більше ніж просто документи, музику або фільми, став пошуковик Yahoo. На той момент це був тематичний каталог веб-сайтів з індексованими сторінками, що допомогло спростити пошук для мільйонів людей в інтернеті.

Першою ж контентною фільтрацією стала програма Music Genome, її було створено вже в 99 році. Причиною такої затримки стала проблема створення функціонуючої системи що б нормально працювала. Адже потрібно було спершу зрозуміти зв'язок усіх факторів і як користувач впливає на їхній вибір. Таким чином було виявлено 450 властивостей. Усі вони були пов'язані за допомогою алгоритмів. Пізніше на основі цієї системи було створено інтернет-радіо Pandora.

Першу систему, котра об'єднувала в собі спільну і контентну фільтрацію, стала програма Fab. Мета створення такої гібридної системи – це усунення недоліків обох систем. Така система мала два процеси, де спершу збирався певний контент за певними темами, а після для кожного користувача обираються елементи, які мають зацікавити його [3].

Нині існують сервіси-гіганти, такі як Spotify, Youtube, Amazon, Google та інші, кожна з них володіє власною системою рекомендацій, не всі вони досконалі та не всі вони знаходяться у відкритому доступі. Так, наприклад, під час перегляду Youtube, рекомендації будуть показуватися тільки на основі того, що було переглянуто, немає ніяких прихованих параметрів підбору, а також є рекомендація на основі популярності, яка дуже погано просуває новачків на платформі. Суть її полягає в тому, що чим менше ваш контент дивляться, тим менше його будуть рекомендувати. Такий підхід дуже часто критикують, адже нові користувачі, які роблять контент, не можуть отримати приплив нових глядачів, тому що їх просто не рекомендуватиме система. Так само присутня проблема обмеження монетизації, коли користувачів обмежують з тих чи інших причин, їх починають так само менше рекомендувати, а відео і зовсім можуть зникнути зі списку нових відео для підписників.

Spotify, хоч і вважається найпопулярнішою платформою для прослуховування музики, не є ідеальною системою. Проблеми спільної фільтрації від Spotify виникають під час появи нової музики менш популярних

виконавців, такі системи добре справляються з уже завантаженою музикою, а також тією, яку вже прослуховували користувачі. Однак використання тільки таких пісень ігнорує нових виконавців і непопулярні пісні. Так само система NLP не може підібрати параметри для рекомендації, якщо про нові пісні або виконавців нічого не пишуть у мережі. Така проблема отримала назву "проблема холодного запуску" [4-5].

Актуальність цієї роботи аргументована відсутністю таких систем у відкритому доступі для таких сервісів, що не дозволяє створювати нові системи на основі систем що існують з можливістю додати, замінити або покращити наявні інформаційні системи.

1.2 Поняття рекомендаційної системи

Система рекомендацій – це штучний інтелект або ж алгоритм для нього, зазвичай пов'язаний з машинним навчанням, що також використовує BigData, даючи змогу рекомендувати або пропонувати нові продукти споживачам.

Вони ґрунтуються на різних критеріях, можуть містити інформацію про минулі покупки, історію пошуку, демографічну інформацію тощо.

Також ці системи навчені розуміти вподобання та попередні рішення і характеристики продуктів, а також інформацію про взаємодію з ними.

Система працює на дуже персоналізованому рівні, тому вона популярна серед творців контенту. Так само дані системи діляться на різні алгоритми – це колаборативна фільтрація, фільтрація заснована на контенті, гібридна фільтрація, і контекстна фільтрація [5].

Колаборативна фільтрація або по-іншому спільна фільтрація – це один із методів побудови рекомендації в рекомендаційних системах, що використовує вже наявні вподобання або оцінки від груп користувачів для подальшого прогнозування вподобань іншого користувача. У музиці це дає

змогу проаналізувати, яка музика сподобається користувачеві на основі його вподобань [6].

Існує кілька типів колаборативної фільтрації:

- на основі пам'яті;
- заснований на моделі;
- гібридні;
- на основі Deep-Learning.

Колаборативна фільтрація має так само й недоліки, такі як:

- розрідженість даних, коли через великий набір даних створюється матриця користувацьких елементів, яка може бути великою та розрідженою, що може створити проблеми під час рекомендацій;
- проблеми з масштабованістю при великій кількості елементів і користувачів, можна зіткнутися з проблемою продуктивності, тому для таких обчислень потрібен дуже потужний комп'ютер;
- проблеми синонімів – це коли однакові або дуже схожі елементи можуть мати різні імена і записи;
- проблема "сірих овець" – це коли низка користувачів, у яких розходиться думка, не можуть отримати вигоди, оскільки система не може зробити адекватну рекомендацію;
- проблема шилінгових атак – це навмисне завищення або заниження оцінок з метою знизити шанс на рекомендацію [7].

Контентна фільтрація – це рекомендаційна система, яка на основі контенту намагається вгадати функції або поведінку користувача з урахуванням функцій елемента, на які він позитивно реагує. Вона не вимагає даних від інших користувачів під час рекомендації, а використовує ключові слова й атрибути, призначені об'єктам у базі даних, а потім порівнює їх із профілем користувача, що формується внаслідок дій самого користувача, наприклад інформація про куди ви клацаєте мишкою, відмітки «подобається» чи «не подобається», переходи за посиланнями тощо. На

основі зважування атрибутів та історії рекомендаційна система створює унікальну модель уподобань кожного користувача.

У неї є і свої недоліки:

- проблема з новизною і різноманітністю;
- проблеми з масштабуванням;
- проблеми з атрибутами – вони можуть бути неправильними або несумісними [8].

Гібридна рекомендаційна система – це система, яка об'єднує в собі кілька рекомендаційних систем в одну, щоб отримати результат. Така система дає змогу об'єднати два і більше методи для досягнення кращої продуктивності та дає змогу пом'якшити недоліки для кожного з підходів.

Більшість з атрибутів у такій системі мають гібридні характеристики. Так, наприклад, підхід на основі контенту використовує глобальні атрибути релевантності для ранжування кандидатів або використання графових методів [9-10].

Гібридна рекомендаційна система може містити [9-10]:

- зваження: об’єднання балів усіх компонентів для створення рекомендації;
- перемикання: вибір між компонентами рекомендацій та застосування обраного з них;
- змішування: рекомендації від різних експертів об'єднуються для надання рекомендації;
- комбінація ознак: ознаки, отримані з різних джерел знань, об'єднуються разом і надаються єдиному алгоритму рекомендацій;
- доповнення ознак: обчислення ознаки або набору ознак, які потім є частиною вхідних даних для наступного методу;
- каскад: рекомендаціям надається суворий пріоритет, при цьому рекомендації з нижчим пріоритетом розривають зв'язки в оцінці вищих.

1.3 Аналіз існуючих аналогів

Першою створеною програмою, яка вміла рекомендувати музику, став проект Тіма Вестергрена під назвою Music Genome (рис.1.1) [13], що була спробою аналітично підійти до організації музики у вигляді геному, подібних до ланцюжків ДНК. Програма працює за принципом пошуку “музичного сусіда”. Вона має в собі специфічні ознаки, таких ознак нараховується близько 400, після початку роботи програма аналізує пісні та намагається надати схожі пісні, котрі найближче підходять до наявних ознак. На цю мить в бібліотеці програми нараховується близько мільйону проаналізованих пісень та виконавців.

Ця система була побудована на основі колаборативної фільтрації на основі контенту.

Також було визначено кілька широких геномів, які програма використовує для визначення жанрів, сюди включено такі жанри, як рок, блюз, класика, святкова музика, а також підкатегорії, наприклад, хор.

Якщо ж програма не може визначити жанр, вона намагається приблизно зрозуміти, до якого жанру вона належить, аналізуючи атрибути пісні, наприклад, використання синкопи, розмір пісні, ритм, рівень свінгу та інші елементи. Так само в геном включені такі атрибути, як мажорні і мінорні тональності, вокальний контрапункт і дисонанс, гнівні тексти, нашарування акустичної гітари, напружений електронний хет, середньо темповий шаффл, тонкий жіночий вокал і вокальний скеттінг.

Працює ця програма в браузері у вигляді сайту або ж існує версія для смартфона. У ній користувачі створюють радіостанцію, де користувач вказує групу, пісню або музичний стиль. Опісля програма транслює аналогічну музику, ґрунтуючись на геномах, які має схожа пісня, і залежно від вашої оцінки, позитивної чи негативної, програма намагатиметься підібрати вже іншу пісню.

У 2011 році кількість користувачів становила 80 мільйонів [14-15].

Рисунок 1.1 – Інтерфейс Music Genome

Далі розглянемо проект – Gnoosic [16] та систему рекомендацій GNOD, до якої належить Gnoosic. Цей сервіс дуже простий у своєму використанні, є сайт, на якому потрібно ввести назви від однієї до трьох музичних груп, після чого сервіс запропонує деяких нових виконавців, яких вам потрібно оцінити позитивно, негативно або ж відповісти, що ви їх ніколи не слухали. Після цього система надає результат у вигляді списку запропонованих виконавців. Проблема полягає в тому, що користувачі просто отримують текст, за яким їм доводиться самим шукати цих виконавців в інтернеті.

Яку систему рекомендації використовує Gnoosic, невідомо, бо Gnoosic або ж Gnod – це система що адаптується сама, яка вивчає зовнішній світ, запитуючи своїх відвідувачів, що їм подобається, а що ні. Тобто це штучний інтелект, котрий працює за певним алгоритмом.

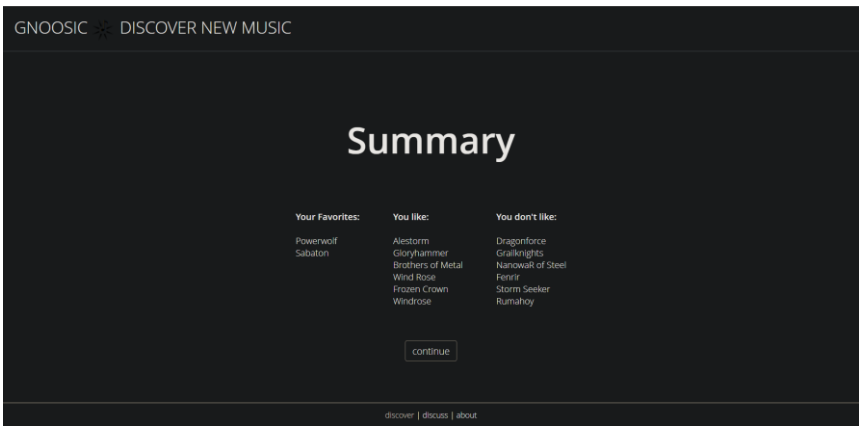

Рисунок 1.2 – Інтерфейс Gnoosic

Наступним розглянуто сервіс Last.fm(рис.1.4) [17]. Це гібридна система фільтрації, в котрій існує декілька інших систем. Перша – це їхня власна система для рекомендацій під назвою аудіоскроблер, що являється фільтрацією на основі контенту. Вона створює детальний профіль музичних смаків для кожного користувача, опісля записує деталі треку. Поки користувач слухає музику з радіостанції додатка або ж зі свого комп'ютера, програма сканує всю інформацію про трек і додає в базу даних, після чого ці дані відображаються у профілі користувача і компілюються для створення довідкових сторінок для окремих виконавців. А також є друга система, котра є колаборативною фільтрацією, це дає змогу користувачам переглядати та прослуховувати виконавців, яких немає в їхніх профілях, але які з'являються в інших користувачів зі схожими смаками [18-19].

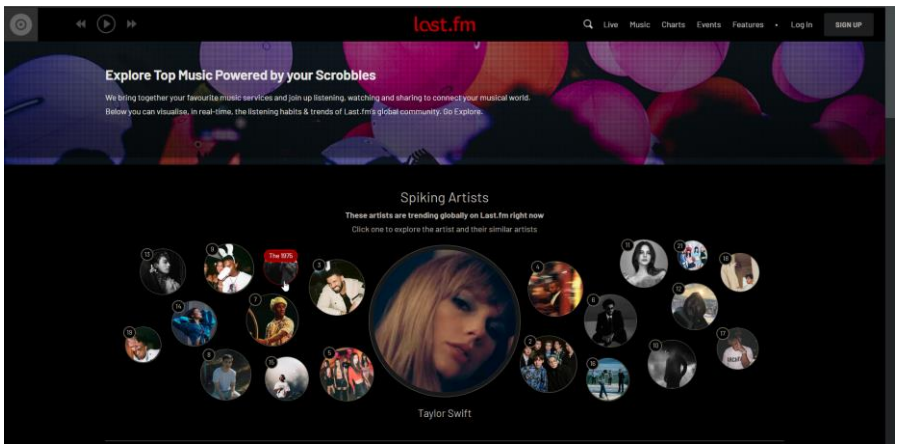

Рисунок 1.3 – Інтерфейс Last.fm

Indieshuffle (рис.1.4) [20] – це сервіс, який створений для рекомендації музики, але працює зовсім за іншим принципом, в якому роль рекомендаційної системи виконують звичайні люди. Суть полягає в тому, що люди відповідають на запити інших людей і складають їм плейлисти за своїм смаком, акцент в яких йде на незалежних виконавців. Кожен може стати учасником цього сервісу.

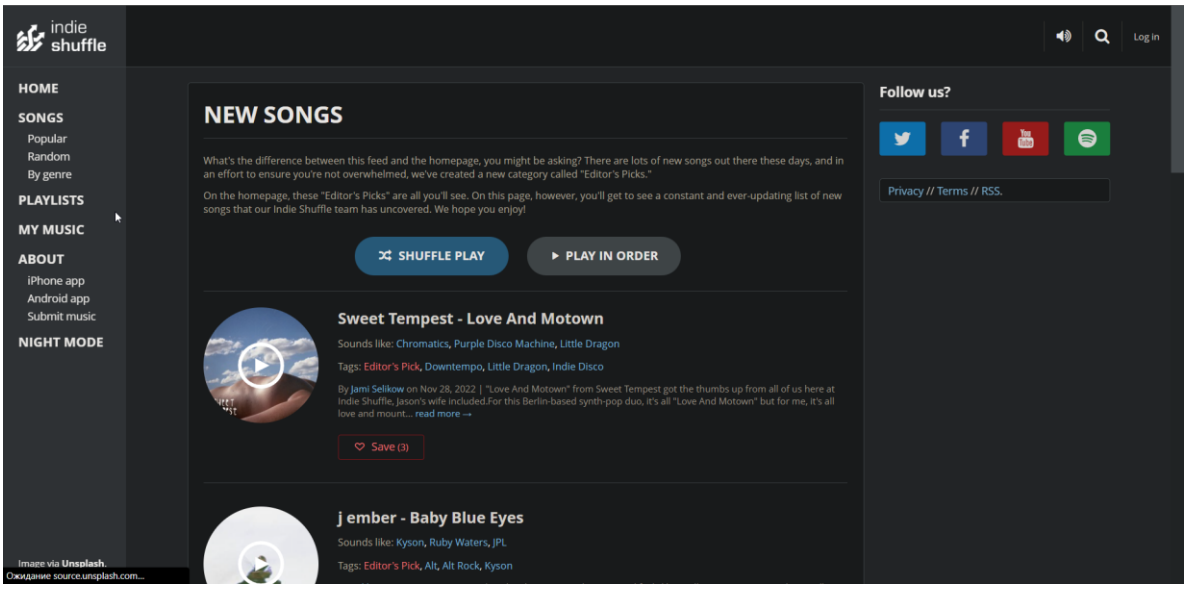

Рисунок 1.4 – Інтерфейс Indieshuffle

Останнім розглянутим сервісом, який може надати музичні рекомендації, стане Spotify (рис.1.5) [21]. Це один із найпопулярніших музичних сервісів. В її основі лежить гібридна система рекомендації, бо в неї входять декілька різних систем, котрі доповнюють одна одну, що дозволяє створити потужний сервіс.

В основі лежить гібридна система фільтрації.

Рекомендаційна система починається з системи BaRT, яка складається з двох компонентів – експлуатації та розвідки.

Модель експлуатації рекомендує контент, який був заснований на звичках людей, а також історії даних. Використовується така інформація, як історія прослуховування, дані про користувача, пропущені пісні і створені

плейлисти. Ця система заснована на спільній фільтрації. Така система фільтрує пісні з високою і низькою достовірністю на основі їхньої релевантності. Тобто, якщо пісня програвалася не часто, то і даних буде недостатньо для того, щоб довести актуальність цієї пісні, яка могла б вплинути на рекомендацію. Але така модель швидко зазнає невдачі, якщо нею почне користуватися новий користувач, бо для роботи такої системи потрібна якась історія користувача, а тому доведеться спершу прослухати якусь кількість пісень, щоб почати отримувати рекомендації.

Модель дослідження ж рекомендує контент незалежно від уподобань користувача, що і дає змогу зібрати дані про користувача, якщо даних було недостатньо для застосування моделі експлуатації. Така модель корисна для рекомендацій контенту новим користувачам, вона дає рекомендації на основі вашої країни, в якій ви зареєструвалися, а також на підставі того, що популярно у світі.

Також існує правило 30 секунд, система перевіряє, як довго користувач слухає пісню.Якщо користувач прослухав пісню довше 30 секунд, то алгоритм вважатиме це правильною рекомендацією, а якщо пісню прослухають менше ніж 30 секунд, то система вважатиме таку рекомендацію неуспішною, а алгоритм перестане рекомендувати цю пісню.

Після придбання Spotify компанією Echo Nest, вийшло впровадити систему Natural Language Processing (NLP), яка відповідальна за аналіз інформації про те, що люди пишуть про виконавців, пісні, а також переглядає інформацію в соціальних мережах, повідомленнях у блогах і статтях [22-23].

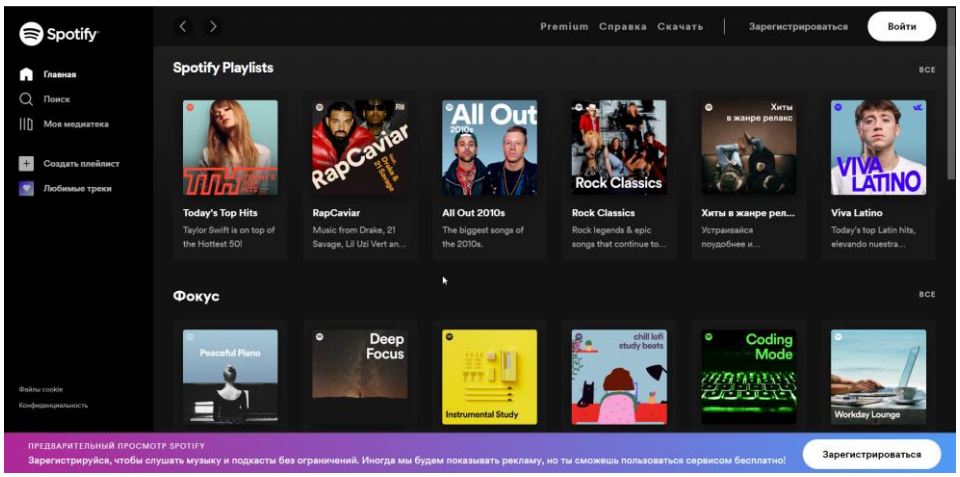

Рисунок 1.5 – Інтерфейс Spotify

Порівняння функціоналу сервісів для рекомендації музики між собою та у порівнянні з додатком котрий створюється під час роботи над магістерською роботою, зображено у таблиці 1.1:

Таблиця 1.1 – Порівняння функціоналу сервісів

Параметри	Music Genome	Gnoosic	Last.fm	Spotify	Indieshuffle	Вебдодаток власної розробки
Інтеграція зі Spotify	-	-	-	-	-	+
Рекомендація на основі плейлисту	-	-	-	+	-	+
Перегляд своїх плейлистів	-	-	+	+	-	+
Перегляд інформації про свій акаунт	-	-	+	+	+	+
Відкритий вихідний код для рекомендаційної системи	-	-	-	-	-	+
Пошук за своїми плейлистами	-	-	+	+	-	+
Перегляд найпрослуховані ших пісень	-	-	-	-	-	+
Перегляд можливих рекомендацій	+	+	+	+	+	+

1.4 Вибір інструментів для реалізації рекомендаційної інформаційної системи

Для розробки якісного додатка важливою частиною є правильний вибір інструментів розробки. Для того, щоб зробити взаємодію між користувачем та додатком простішою та ефективною, цей вебдодаток повинен повністю задовольняти користувача та його потреби й виконувати усі потрібні йому вимоги.

Інформаційна система реалізована за допомогою мови програмування Python [24]. Вибір цієї мови програмування зумовлений її простотою, сучасністю, великою кількістю бібліотек та модулів, що дозволяють робити складні обчислення, а також через її орієнтованість для наукових цілей та аналізу даних.

Для реалізації серверної частини був обраний фреймворк Django, що написаний на основі мови програмування Python, що полегшить майбутню розробку веб-додатку.

Django [25] має такі переваги:

- швидкість: Django є одним з найшвидших веб-фреймворків, через це він дуже дозволяє економити ресурси при розробці нового проекту;
- великий функціонал: через те що Django написаний на мові програмування Python, це дозволяє користуватися великою бібліотекою різних модулів, що полегшують розробку додатку [25].

Для реалізації зовнішнього вигляду сайту були використані HTML5 [26] та CSS [27].

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою цієї роботи є створення рекомендаційної системи для підбору музики. Однією з цілей стане створення можливості підключатися до цієї системи за допомогою свого облікового запису Spotify, але так само буде можливість користуватися і без попередньої авторизації.

Система повинна обробляти запит користувача і мати можливість видавати рекомендацію після аналізу цілого плейлиста з подальшим виведенням результату у вигляді сформованого нового плейлиста.

Інтеграція зі Spotify відбувається за допомогою таких бібліотек як SpotiPy [28] та Spotify Api [29]. Це дозволить зробити вивід інформації про акаунт користувача, його список плейлистів, а також зробити вивід інформацію про найпрослуховаліші пісні та виконавців.

Метод рекомендації під назвою Cosine Similarity був взятий з бібліотеки scikit-learn [30], аналіз дата сетів відбувався за допомогою бібліотеки Pandas, а інтерфейс веб-додатку було створено за допомогою фреймворку Django, що являється одним з найпопулярніших веб-фреймворків, бо має вбудовані шаблони що дозволяють пришвидшити створення такого додатку.

У результаті було окреслено такі вимоги до веб-додатку:

- вхід за допомогою наявного акаунта Spotify;
- перегляд інформації про свій акаунт;
- перегляд своїх плейлистів;
- пошук за своїми плейлистами;
- перегляд найпрослуховаліших пісень;
- перегляд можливих рекомендацій;
- можливість рекомендації на основі готового плейлисту.

2.2 Вибір методів дослідження

Для створення рекомендаційної системи потрібно обрати метод рекомендації, що ляже в її основу. Таким методом став метод контентної фільтрації або по іншому фільтрація на основі контенту. Даний метод був обраний через свою ефективність та простоту в реалізації та й тому, що саме музика в Spotify має велику кількість прихованих параметрів, що дозволяють реалізувати таку систему.

Математичною частиною методу контентної фільтрації став Косинус Подібності (Cosine Similarity) (рис.2.1).

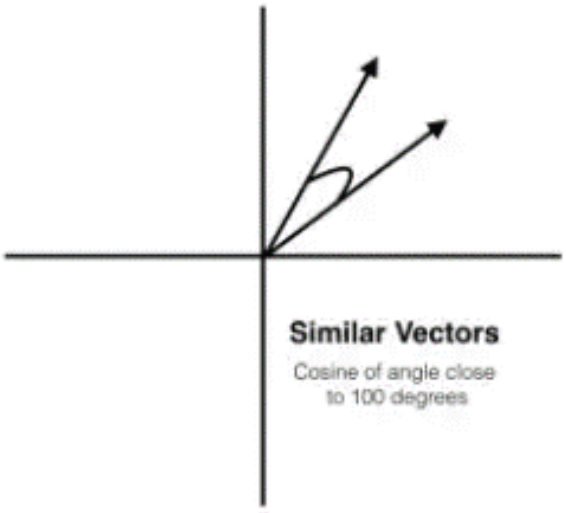

Рисунок 2.1 – Вигляд міри збіжності косинуса

Цей метод являється найпопулярнішим варіантом метрик, що використовується для інформаційного пошуку. Він дозволяє побачити схожість між даними, що отримує рекомендаційна система та порівняти, з тими даними, які присутні в системі. Метод шукає косинус кута між двома векторами. Працює цей метод в діапазоні в [-1, 1], де -1 являється повною протилежністю, 0 являється ортогональним, тобто спрямований під прямим

кутом, а 1 означає вектори що спрямовані в одну сторону, величина векторів може бути різною [31].

Математична модель пошуку подібності пісень на основі Косинуса Подібності представлена на формулах 2.1-2.3.

$$\cos\Theta = \frac{A*B}{\|A\|*\|B\|} \ , \tag{2.1}$$

Формула 2.1 являється основною математичною формулою для пошуку подібності.

У рівнянні 2.1 $\cos\Theta$ – являється косинусоїдальною подібністю. A та B це вектора частоти.

$\|A\|$ та $\|B\|$ – це евклідова норма векторів A та B що означає відстань вектора від початку координат до кінцевої точки.

Формула 2.2 – це представлення евклідової норми вектору A.

$$\|A\| = \sqrt{\sum_{i=1}^{n} a_i^2} = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \tag{2.2}$$

Формула 2.3 являється точковим добутком двох векторів [31].

$$A * B = \sum_{i=1}^{n} a_i b_i \tag{2.3}$$

Розглянемо приклад розрахунку рекомендації. Для початку потрібно витягнути дані про пісні, для цього використано бібліотеки Spotify Api, Spotipy та Pandas. Після чого витягнуті дані потрібно обробити, треба підсумувати усі дані про плейлисти (рис. 2.2).

	Дата	Жанр пісні					Популярність і дата					Гучність, енергічність і т.д.		
Пісня 1	17.07.2019	0	0.1	0.3	0	0	0	0.1	0.3	0	0	0.5	0.2	0.8
Пісня 2	07.02.2020	0	0	0	0.8	0	0	0	0	0.8	0	0.6	0.23	0.3
Пісня 3	19.04.2021	0.9	0	0	0.7	0	0.9	0	0	0.7	0	0.6	0.1	0.2
Пісня 4	25.03.2022	0.6	0	0	0.2	0	0.6	0	0	0.2	0	0	0.2	0.2
Фінальний вектор		1,2	0.1	0.3	1,7	0,0	1,5	0.1	0.3	1,7	0	1,7	0.73	1,5

Рисунок 2.2 – Приклад матриці з підсумованими параметрами

Після підсумування всіх пісень потрібно об’єднати усі пісні в один вектор, котрий буде порівнюватися з іншими треками з набору даних для подальшого знаходження подібної пісні. Тому для такого підходу важливою частиною є розмір датасету, тобто, чим більше буде датасет, тим якісніше буде рекомендація.

У рамках цієї роботи було використано бібліотеку scikit-learn, що дає доступ до бібліотеки з математичними функціями, у тому числі й функції розрахунку косинуса подібності.

На вхід до цієї функції входить датафрейм з піснями із плейлиста, вектор з підсумованими параметрами з готового плейлисту та готовий плейлист, з котрого й будуть братися пісні для створення нової рекомендації.

У результаті роботи функції розрахунку косинуса подібності отримуємо плейлист зі схожими піснями, після чого ці пісні додаються до акаунту в Spotify.

3 МОДЕЛЮВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

3.1 Структурно-функціональне моделювання процесу формування рекомендації музики

Контекстна IDEF0 діаграма для рекомендаційної інформаційної системи музики представлена на рис.3.1.

Рисунок 3.1 – Контекстна діаграма інформаційної системи рекомендації музики

Входи моделі: інформація про акаунт, інформація про пісню, інформація про плейлист користувача.

Виходи: результат у вигляді готового плейлисту.

Декомпозиція на функціональні блоки:

1. Авторизація за допомогою акаунту Spotify;
2. Перегляд існуючих, сформованих плейлистів та пошук за ними;
3. Перегляд інформації про свій акаунт;
4. Створення рекомендації на основі плейлисту;
5. Збереження створеної рекомендації на акаунт.

Розглянемо діаграму декомпозиції IDEF0 для рекомендаційної системи музики (рис.3.2).

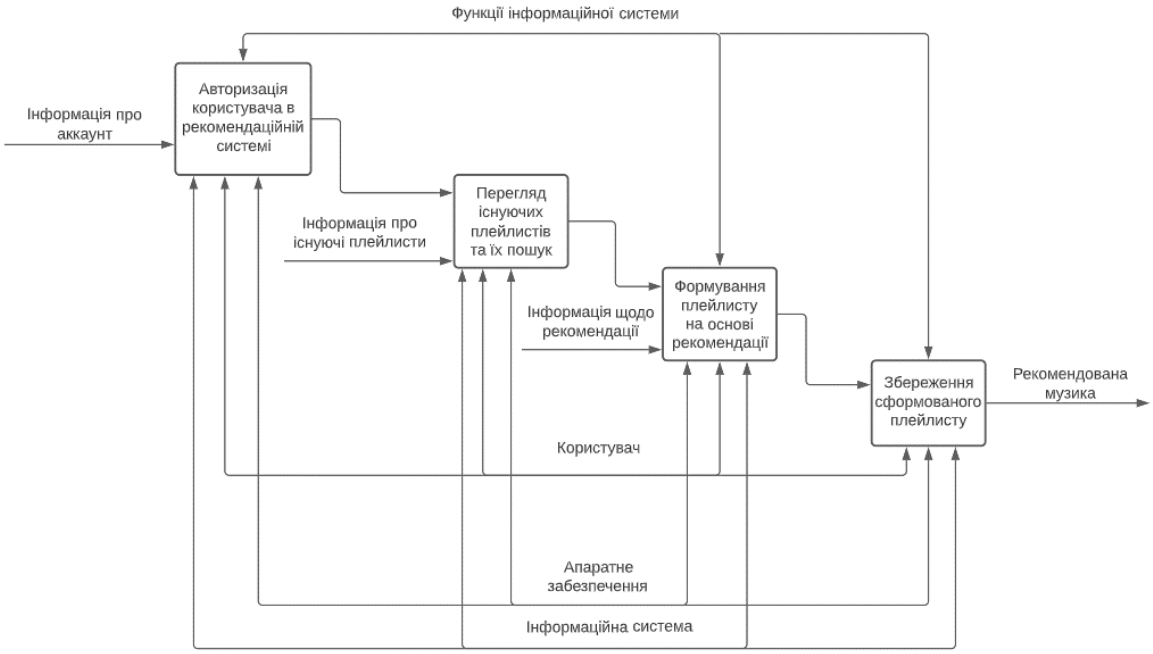

Рисунок 3.2 – Діаграма декомпозиції інформаційної системи рекомендації музики

3.2 Моделювання діаграми варіантів використання

Для представлення функцій інформаційної системи рекомендації музики було створено діаграму варіантів використання в нотації UML (рис. 3.3). Опис акторів та варіантів використання наведено в табл.3.1 та табл. 3.2 відповідно.

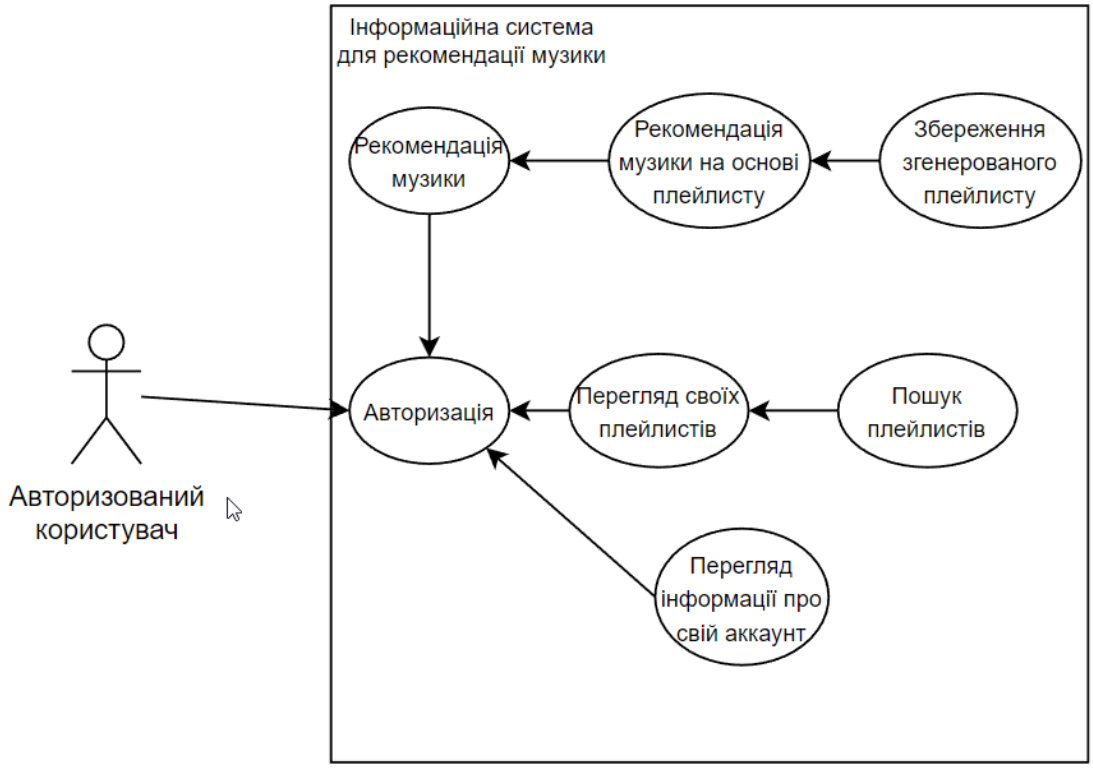

Рисунок 3.3 – Діаграма варіантів використання для користувачів системи

Таблиця 3.1. – Опис акторів

№	Іменування актора	Детальний опис
1	Авторизований користувач	Користувач рекомендаційної системи, що може використовувати увесь функціонал програми.

Таблиця 3.2. – Опис варіантів використання

№	Варіанти використання	Детальний опис
1	Авторизація	Зареєстровані користувачі в Spotify мають змогу авторизуватися в системі.
2	Перегляд інформації про свій акаунт	Авторизований користувач може переглянути інформацію про свій акаунт
3	Перегляд існуючих плейлистів	Авторизований користувач може переглянути свої існуючі плейлисти
4	Пошук за плейлистами	Авторизований користувач може зробити пошук вже існуючих плейлистів на своєму акаунті
5	Перегляд найпрослухованіших пісень та виконавців на вашому акаунті	Авторизований користувач має змогу побачити, які пісні та виконавців він найбільше слухає
6	Перегляд рекомендацій	Авторизований користувач має змогу побачити, що йому рекомендує система на основі всіх прослуховувань
7	Створення рекомендації	Авторизований користувач має змогу створити рекомендацію на основі свого існуючого плейлисту

4 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ РЕКОМЕНДАЦІЇ МУЗИКИ

4.1 Структура проекту

Дана система реалізована у вигляді веб-додатку на основі фреймворку Django. Доступ до такої системи відбувається за допомогою браузеру на локальному сервері. Система має декілька веб-сторінок де відбувається її робота, спочатку користувач повинен авторизуватися за допомогою наявного акаунту в Spotify, далі користувач отримає доступ до головної сторінки з інформацією про свої плейлисти, а також полем їх пошуку, та інформацію про найпрослухованіші пісні та виконавців. Також присутня сторінка з інформацією про профіль користувача. Останньою сторінкою буде сторінка з рекомендаціями пісень та виконавців. А нижче буде знаходитися поле для створення нового плейлисту на основі існуючого.

Реалізація даного проекту відбувалася за допомогою середовища Microsoft Visual Studio Code. Фреймворк на якому розроблялася система побудована на архітектурі MVT(Model View Template). На рис 4.1 зображено схему даної архітектури.

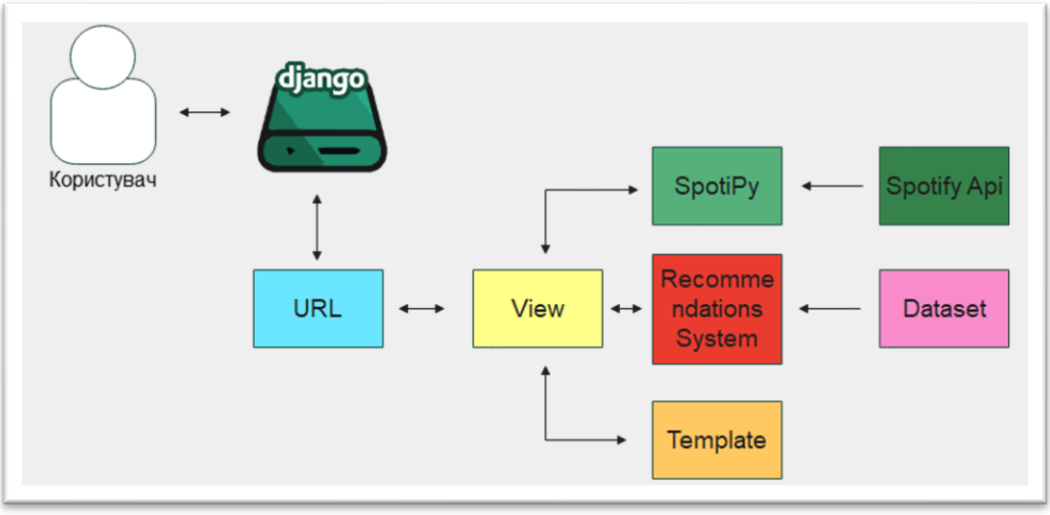

Рисунок 4.1 – Архітектура проекту на Django

Структура проекту показана на рис 4.2 та має такий вигляд:

Data 1 – папка з відсортованим та невідсортованими дата сетами

Static – папка зі статичними CSS файлами та зображеннями.

Templates – папка з HTML шаблонами веб-сторінок

Spotify_recomendation – папка з додатком

App.py – список з підключеними додатками до системи

Dataclient.py – створений клас Dataclient для збереження інформації про акаунт а також збір інформації про пісні.

Ouauth.py – реалізація авторизації системі, також зберігає в собі інформацію про Client ID та Client Secret Key для авторизації в системі.

Spotifyapiclient.py – формує списки найпрослухованіших пісень, виконавців, а також видає рекомендації щодо пісень та артистів.

Urls.py – шаблон, котрий Django використовує для того, щоб створити адресу для відображення View.py

Utils.py – файл, що має в собі декілька функцій для створення рекомендацій на основі готового плейлисту.

Views.py – файл, що використовується для відображення всього функціоналу на сайті.

Manage.py – скрипт, що дозволяє керувати сайтом.

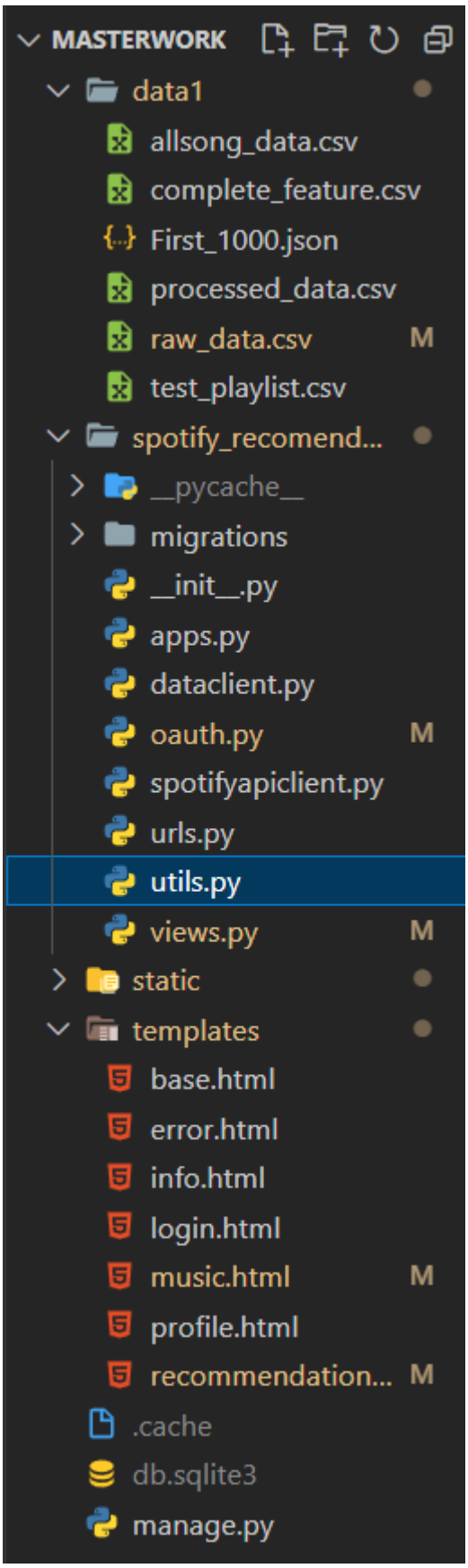

Рисунок.4.2 – Структура проекту

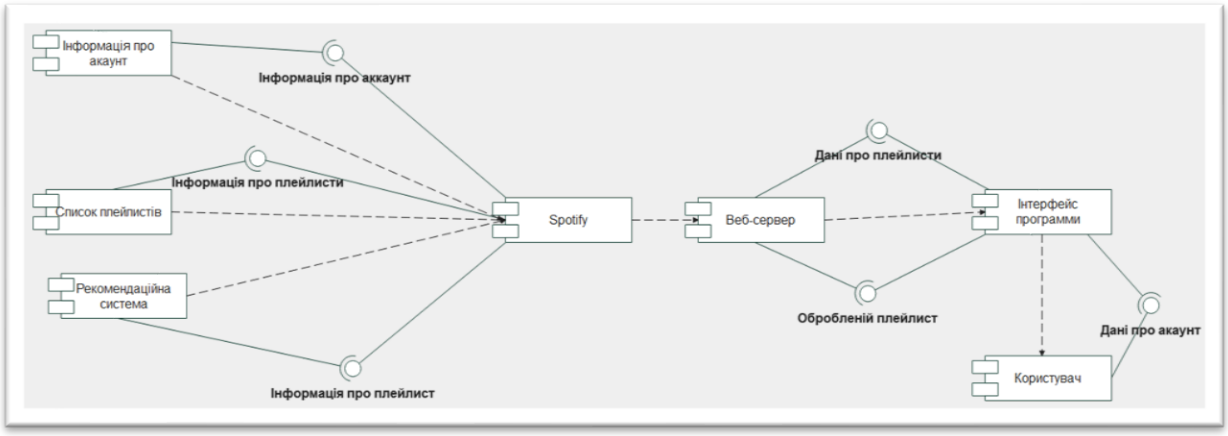

Рисунок.4.3 – UML діаграма компонентів інформаційної системи

4.2 Розробка інформаційної системи як веб-додатку

Реалізація рекомендаційної системи дозволить отримати готовий плейлист на основі рекомендаційного алгоритму.

На рис 4.4-4.13 можна побачити приклад роботи розробленої системи для рекомендації музики.

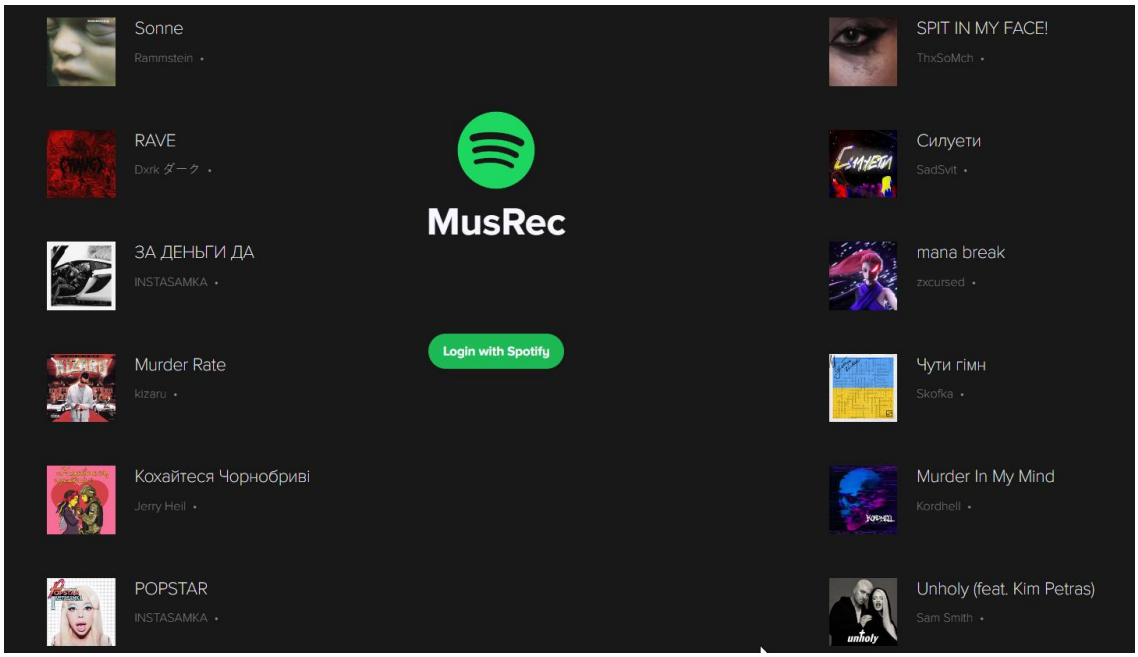

Рисунок.4.4 – Сторінка з полем для авторизації в системі.

На рис. 4.4 представлене початкове меню, котре користувач зустрічає при переході на сайт. Це поле, в якому є кнопка авторизації, а також список найпопулярніших треків в Україні.

Для авторизації у системі необхідно використовувати існуючий акаунт Spotify.

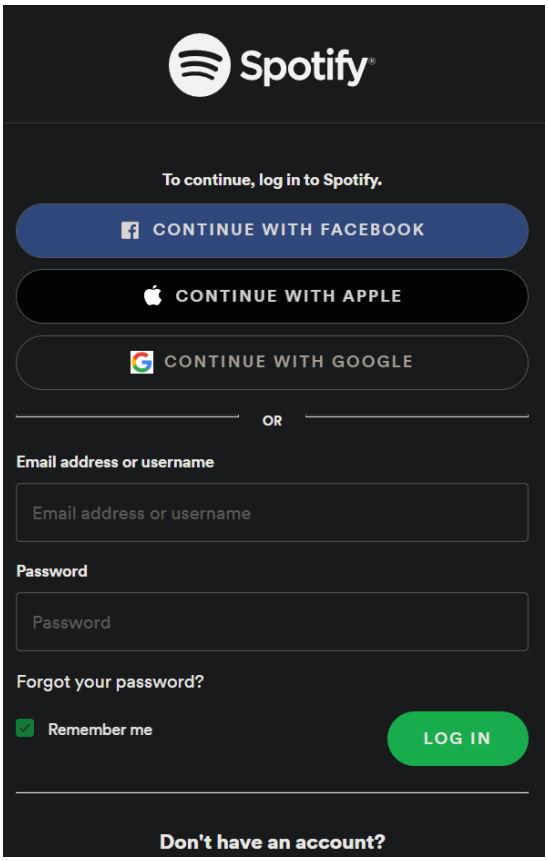

Рисунок.4.5 – Сторінка інтерфейсу авторизації користувача

Після авторизації користувач потрапляє до головної сторінки з плейлистами й полем їх пошуку та списком найпрослухованіших треків та виконавців.

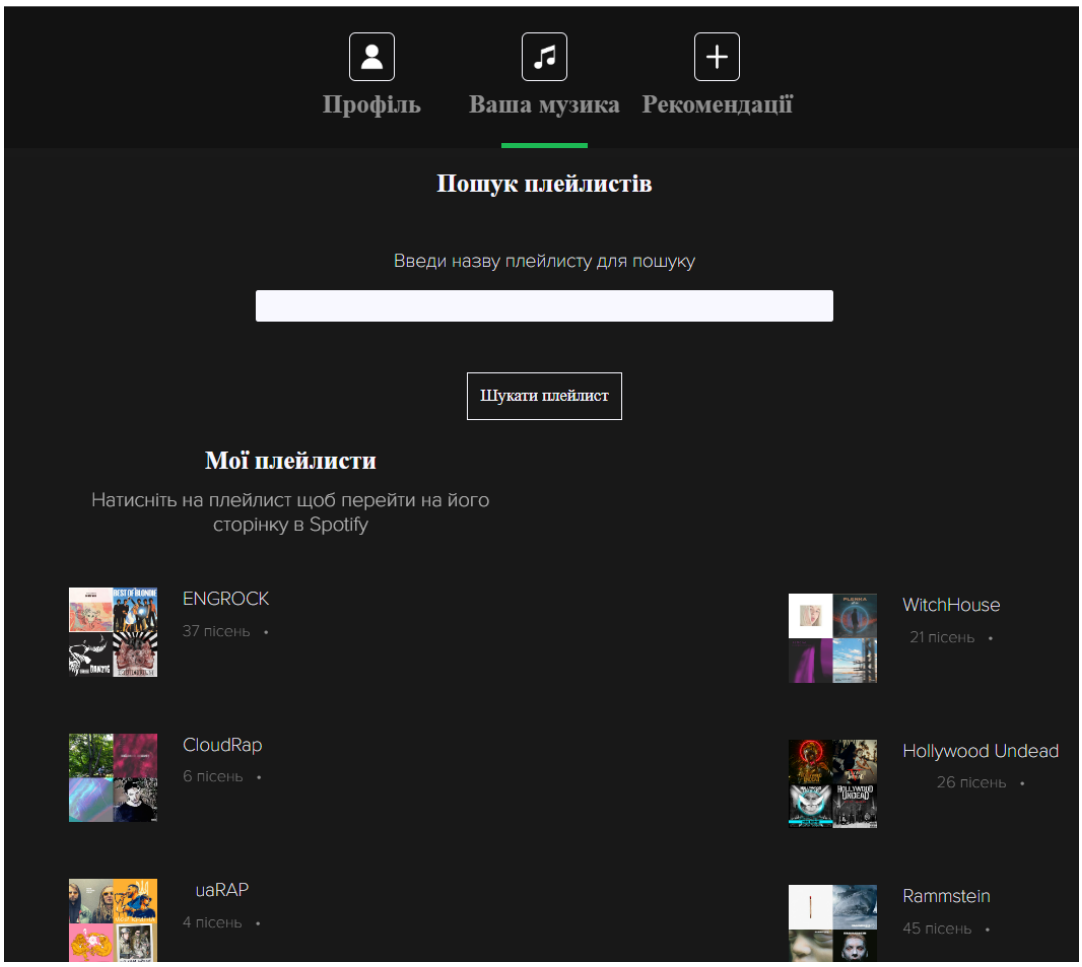

Рисунок.4.6 – Інтерфейс основної сторінки з інформацією про плейлисти та полем для їх пошуку

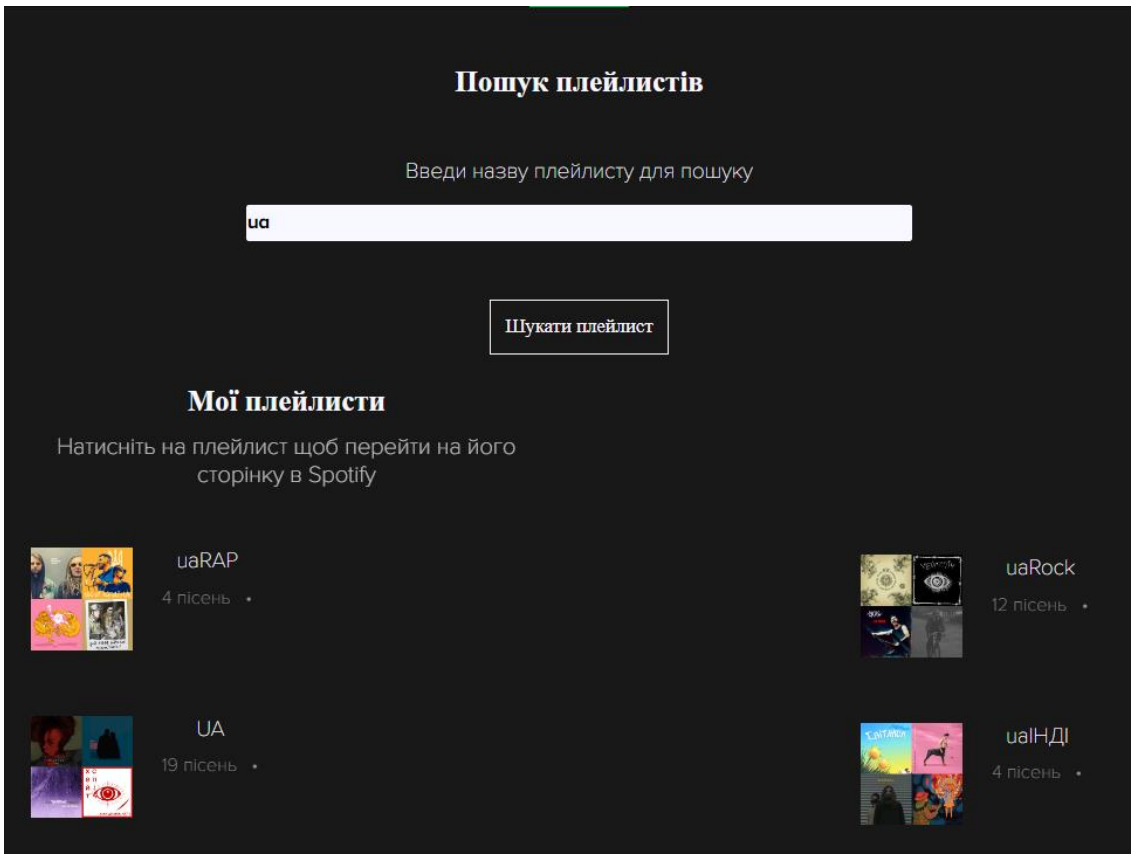

Рисунок.4.7 – Приклад пошуку плейлистів

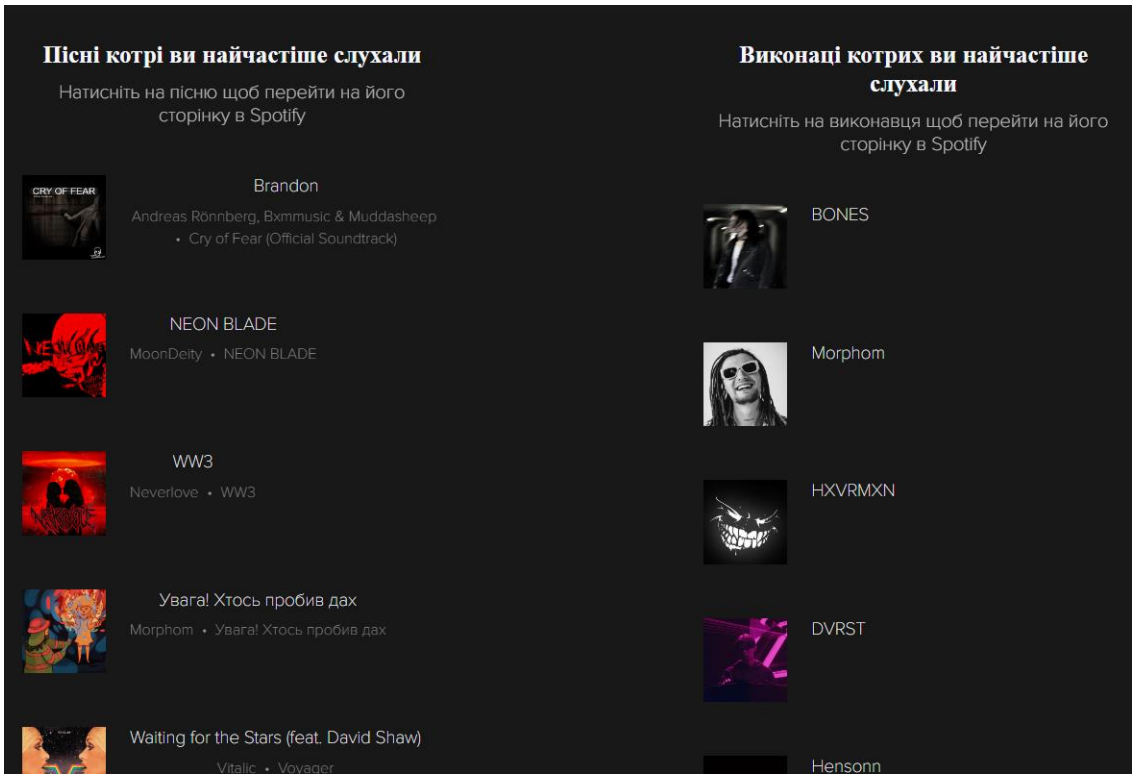

Рисунок.4.8 – Приклад інформації про пісні та виконавців, яких найчастіше слухали

Наступною сторінкою стане сторінка профілю користувача. Тут відображається інформація про акаунт, ім’я користувача, фото, кількість поціновувачів, кількість улюблених виконавців, кількість плейлистів на акаунті та кнопка для виходу з акаунту.

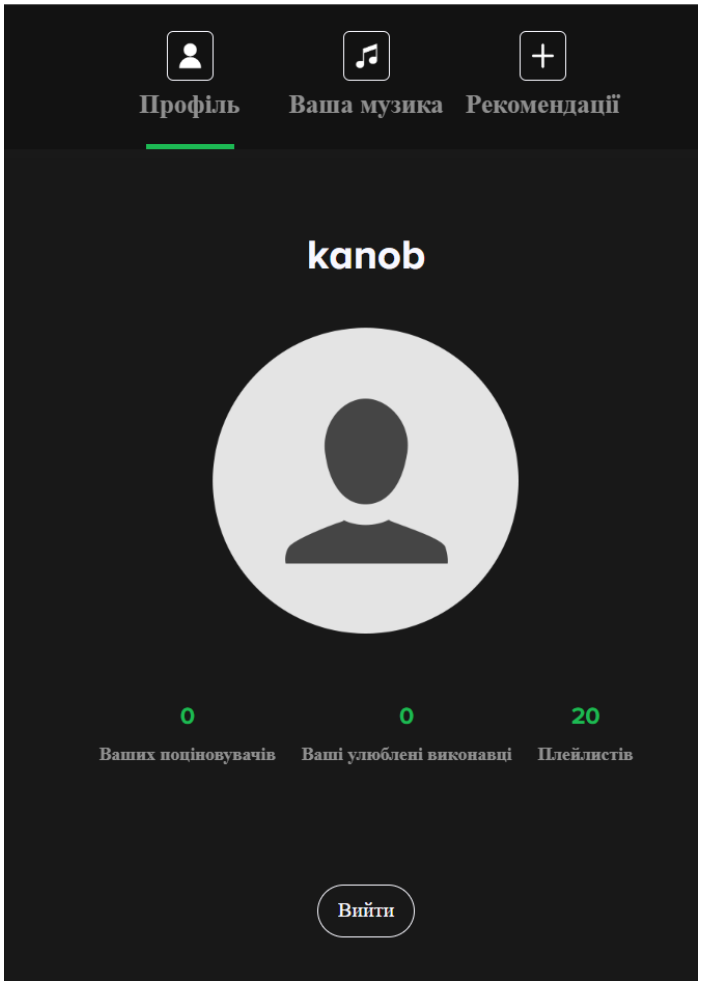

Рисунок.4.9 – Приклад інтерфейсу з інформацією про профіль

Останньою сторінкою веб-додатку є сторінка з рекомендаціями. На ній можна побачити, які пісні та виконавців система Вам рекомендує. Нижче є поле для створення нового плейлисту на основі існуючого.

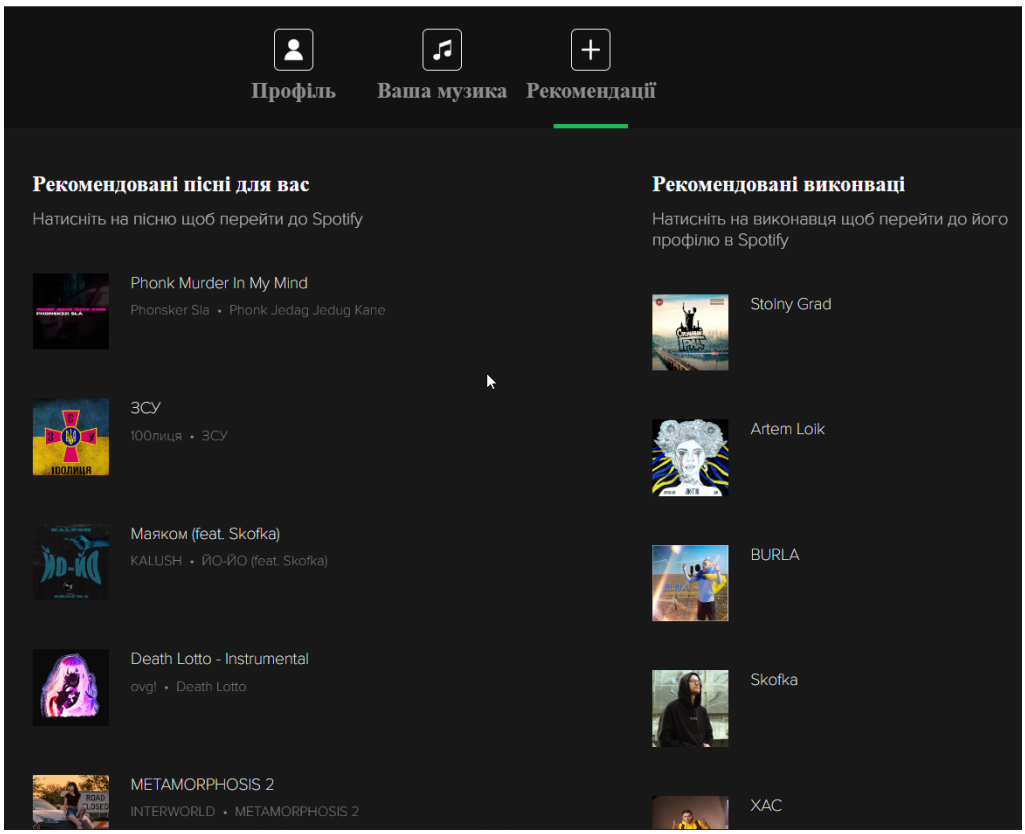

Рисунок.4.10 – Приклад інтерфейсу сторінки з рекомендаціями

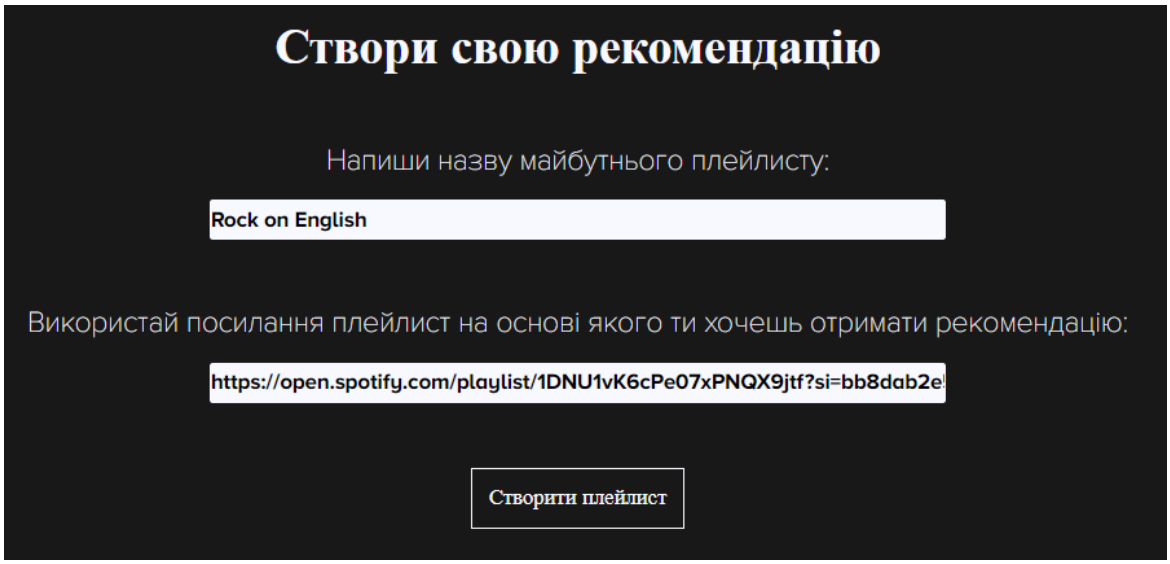

Рисунок.4.11 – Приклад інтерфейсу сторінки з рекомендаціями на основі існуючого плейлисту

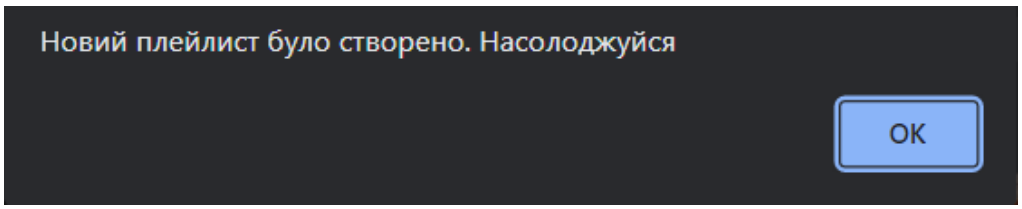

Рисунок.4.12 – Приклад звіту про успішне створення плейлисту

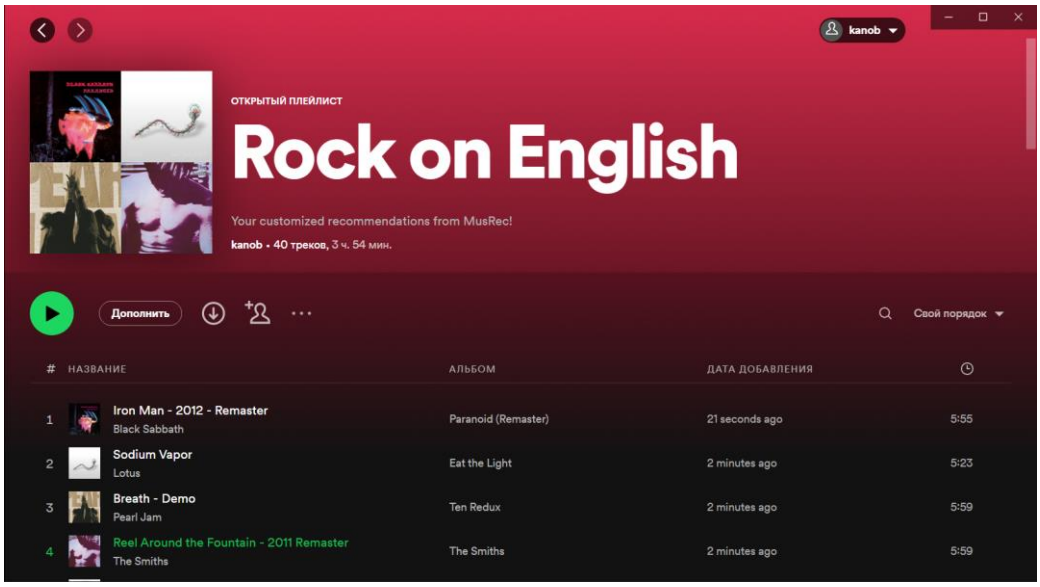

Рисунок.4.13 – Приклад створеного плейлисту

У результаті роботи програми до акаунту в Spotify був доданий новий плейлист, котрий був сформований на основі існуючих композицій в плейлисті.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи магістра було створено інформаційну систему для рекомендації музики на основі власних вподобань користувача та вже існуючого плейлисту. Реалізація проекту була зроблена на мові програмування Python з використанням фреймворку Django та таких Python бібліотек як Spotipy, Spotify Api, Skilearn.

Були проаналізовані існуючі системи для рекомендації музики, їх переваги та недоліки. Такий аналіз допоміг зрозуміти актуальність цієї проблеми та затребуваність у реалізації такої інформаційної системи.

Була проведена робота з планування розробки проекту, описані етапи ініціалізації та розробки. Створені структури WBS та OBS, й описаний принцип роботи інформаційної системи, її складова та функціонал.

Інформаційну систему для рекомендації музики розроблено у вигляді веб-додатку. Як результат рекомендації користувач отримує плейлист, який може прослухати у сервісі Spotify.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. All You Need to Know About a Music Recommendation System with a Step-By-Step Guide to Creating It [Електронний ресурс] – Режим доступу до ресурсу: https://www.eliftech.com/insights/all-you-need-to-know-about-a-music-recommendation-system-with-a-step-by-step-guide-to-creating-it/

2. Recommendation system [Електронний ресурс] – Режим доступу до ресурсу: https://research.aimultiple.com/recommendation-system/

3. History of recommender systems [Електронний ресурс] – Режим доступу до ресурсу: https://www.onespire.net/news/history-of-recommender-systems/

4. Nextone Player: A Music Recommendation System Based on User Behavior [Електронний ресурс] – Режим доступу до ресурсу: https://core.ac.uk/download/pdf/211823416.pdf

5. Recommendation System [Електронний ресурс] – Режим доступу до ресурсу: https://www.nvidia.com/en-us/glossary/data-science/recommendation-system/

6. Парфененко Ю.В. Рекомендаційна інформаційна система для пошуку відеоматеріалів [Електронний ресурс] / Ю. В. Парфененко, А. А. Ковтун, А. А. Вербицька // Вісник КрНУ імені Михайла Остроградського. – 2019. – №5 (118). С. 97-102. – Режим доступу до ресурсу: http://visnikkrnu.kdu.edu.ua/statti/2019_5_2019-5-97-102.pdf.

7. Recommendation systems and machine learning: driving personalization [Електронний ресурс] – Режим доступу до ресурсу: https://www.itransition.com/machine-learning/recommendation-systems

8. What Content-Based Filtering is and Why You Should Use It [Електронний ресурс] – Режим доступу до ресурсу: https://www.upwork.com/resources/what-is-content-based-filtering

9. A Guide to Building Hybrid Recommendation Systems for Beginners [Електронний ресурс] – Режим доступу до ресурсу: https://analyticsindiamag.com/a-guide-to-building-hybrid-recommendation-systems-for-beginners/

10. Recommendation systems: Principles, methods and evaluation [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/283180981_Recommendation_systems_Principles_methods_and_evaluation

11. Babak Joze Abbaschian, Samira Khorshidi. A review of Hybrid Recommender Systems. [Електронний ресурс] – Режим доступу до ресурсу: http://www.magnanimitas.cz/ADALTA/0702/papers/I_khorshidi.pdf

12. Pandora – Режим доступу до ресурсу: https://www.pandora.com/

13. The Music Genome Project [Електронний ресурс] – Режим доступу до ресурсу: http://worldmusicandtaxonomy.weebly.com/music-genome-project.html

14. Michael Howe Pandora’s Music Recommender [Електронний ресурс] – Режим доступу до ресурсу: https://courses.cs.washington.edu/courses/csep521/07wi/prj/michael.pdf

15. Gnoosic – Режим доступу до ресурсу: https://www.gnoosic.com/

16. last.fm – Режим доступу до ресурсу: https://www.last.fm/

17. What Is Last.fm and Should You Use It? [Електронний ресурс] – Режим доступу до ресурсу: https://www.lifewire.com/what-is-last-fm-3486395

18. Last.fm tracks all your music stats by 'scrobbling' them. Here's what that means and how it works. [Електронний ресурс] – Режим доступу

до ресурсу: https://www.businessinsider.com/guides/tech/what-is-last-fm-scrobbling

19. Indieshuffle – Режим доступу до ресурсу: https://www.indieshuffle.com/

20. Spotify – Режим доступу до ресурсу: https://www.spotify.com/

21. Spotify's Recommendation Engine [Електронний ресурс] – Режим доступу до ресурсу: https://medium.datadriveninvestor.com/behind-spotify-recommendation-engine-a9b5a27a935

22. Uncovering How the Spotify Algorithm Works [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/uncovering-how-the-spotify-algorithm-works-4d3c021ebc0

23. Inside Spotify's Recommender System: A Complete Guide to Spotify Recommendation Algorithms [Електронний ресурс] – Режим доступу до ресурсу: https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022

24. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

25. Django FrameWork [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com/

26. HTML5 [Електронний ресурс] – Режим доступу до ресурсу: https://html.spec.whatwg.org/multipage/

27. CSS5 [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/Style/CSS/

28. SpotiPy [Електронний ресурс] – Режим доступу до ресурсу: https://spotipy.readthedocs.io/en/2.21.0/

29. Spotify Api [Електронний ресурс] – Режим доступу до ресурсу: https://developer.spotify.com/documentation/web-api/

30. scikit-learn [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/

31. Shubhit K. Evaluating the Effectiveness of Music Recommendations Using Cosine Similarities of Various Combined Feature Sets Consisting of Metadata and User Generated Tags. A Paper Submitted to the Graduate Faculty of the North Dakota State University of Agriculture and Applied Science. 2022. URL: https://library.ndsu.edu/ir/bitstream/handle/10365/32461/Evaluating%20the%20Effectiveness%20of%20Music%20Recommendations%20Using%20Cosine%20Similarities%20of%20Various%20Combined%20Feature%20Sets%20Consisting%20of%20Metadata%20and%20User%20Generated%20Tags.pdf?sequence=1&isAllowed=y.

ДОДАТОК А – ПЛАНУВАННЯ РОБІТ

ПЛАНУВАННЯ РОБІТ
для розробки кваліфікаційної роботи магістра
«Інформаційна система підбору пісень на основі вподобань користувача»

А.1 Ідентифікація мети ІТ-проекту

Метою проекту є розробка інформаційної системи для рекомендації музики на основі власних вподобань.

Деталізація мети методом SMART:

S – Метою проекту є розробка інформаційної системи для рекомендації музики на основі власних вподобань.

M – Система яка зможе стати відкритою платформою, в котрій за можливості кожний зможе використати для своїх потреб.

A – Головною ціллю є створення відкритої для всіх системи для рекомендації музики.

R – Проект має можливість монетизувати додаток за рахунок пожертвувань від людей для покращення системи.

T – На розробку проекту потрібно – 12 тижнів, при роботі по 5 годин на день.

А.2 Планування змісту структури робіт інформаційної системи

WBS (рис. А.2.1) це ієрархічна структура, що може показати майбутніх цілей та майбутніх цілей – це може бути програма, проект або договір. Він спрямований на планування, оцінку та визначення і розподіл відповідальностей між виконавцями.

Рисунок А.2.1 – WBS діаграма

Після створення WBS структури проекту, потрібно створити OBS (рис.А.2.2).

Організаційна структура є графічним відображенням учасників проекту та осіб які задіяні в реалізації проекту.

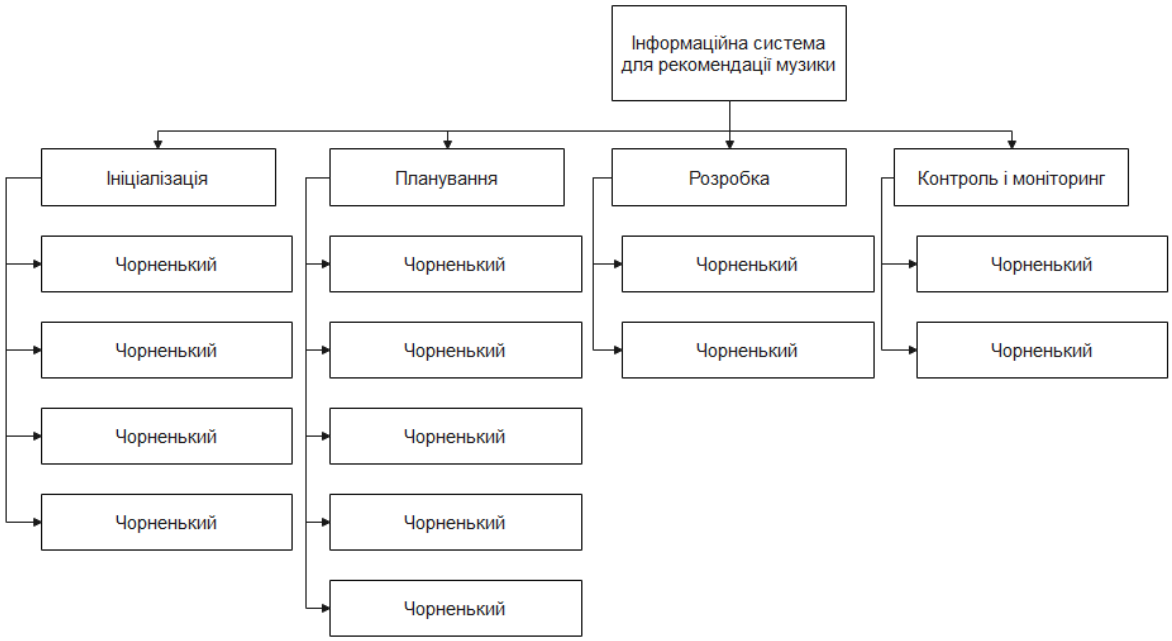

Рисунок А.2.2 – OBS діаграма

А.3 Побудова календарного графіку виконання ІТ - проекту

Діаграма Ґанта – це горизонтальна діаграма з тимчасової шкалою, яка використовується для ілюстрації плану робіт за проектом з прив'язкою до часу.

За допомогою діаграм Ґанта керівники проектів і менеджери по продукту розбивають проекти на робочі завдання для зручності управління, підтримують порядок в роботі і роблять залежності між завданнями наочними.

Діаграми Ґанта дозволяють спростити складові проекти. За допомогою цього засобу можна досить наочно і зручно для узагальнення представити велику кількість даних. Завдяки цій гістограмі велика кількість зацікавлених осіб, команд або їх учасників не стане проблемою для запису завдань, як і часті зміни обсягу роботи. Ще одна перевага використання діаграми Ґанта полягає в тому, що вона дає загальне уявлення про проект в цілому, в тому числі про всі контрольних точках і терміни виконання. Діаграму Ґанта можна уявити як ефективний засіб раннього попередження.

Розглянемо створену діаграму Ґанта до заданої інформаційної системи (рис.А.2.3-4).

Ініціалізація	01/10/2022	04/10/2022	4 d. 0 h.	100.0%
Аналіз предметної області	01/10/2022	01/10/2022	1 d. 0 h.	100.0%
Аналіз потреб користувача	02/10/2022	02/10/2022	1 d. 0 h.	100.0%
Аналіз конкурентів	03/10/2022	03/10/2022	1 d. 0 h.	100.0%
Аналіз актуальності	04/10/2022	04/10/2022	1 d. 0 h.	100.0%
Планування	04/10/2022	14/10/2022	10 d. 4 h.	100.0%
Розробка проектної частини	04/10/2022	07/10/2022	4 d. 0 h.	100.0%
Написання чек-листів	07/10/2022	09/10/2022	1 d. 4 h.	100.0%
Планування структури робіт IT-проекту	10/10/2022	12/10/2022	3 d. 0 h.	100.0%
Побудова діаграми Ганта	13/10/2022	13/10/2022	0 d. 5 h.	100.0%
Планування ризиків	14/10/2022	14/10/2022	0 d. 4 h.	100.0%
Розробка	27/10/2022	27/11/2022	32 d. 0 h.	100.0%
Верстка сторінки	11/11/2022	15/11/2022	5 d. 0 h.	100.0%
Розробка бек-енду	27/10/2022	10/11/2022	15 d. 0 h.	100.0%
Розробка локалізації	15/11/2022	16/11/2022	2 d. 0 h.	100.0%
Розробка системи рекомендацій	18/11/2022	27/11/2022	10 d. 0 h.	100.0%
Контроль і моніторинг	16/11/2022	30/11/2022	15 d. 0 h.	50.0%
Тестування додатку	16/11/2022	30/11/2022	15 d. 0 h.	50.0%
Тестування системи рекомендацій	18/11/2022	30/11/2022	13 d. 0 h.	50.0%
Фіналізація	01/12/2022	09/12/2022	9 d. 0 h.	0.0%
Підсумування ітогів	01/12/2022	09/12/2022	9 d. 0 h.	0.0%

Рисунок А.3.1 – Діаграма Ґанта

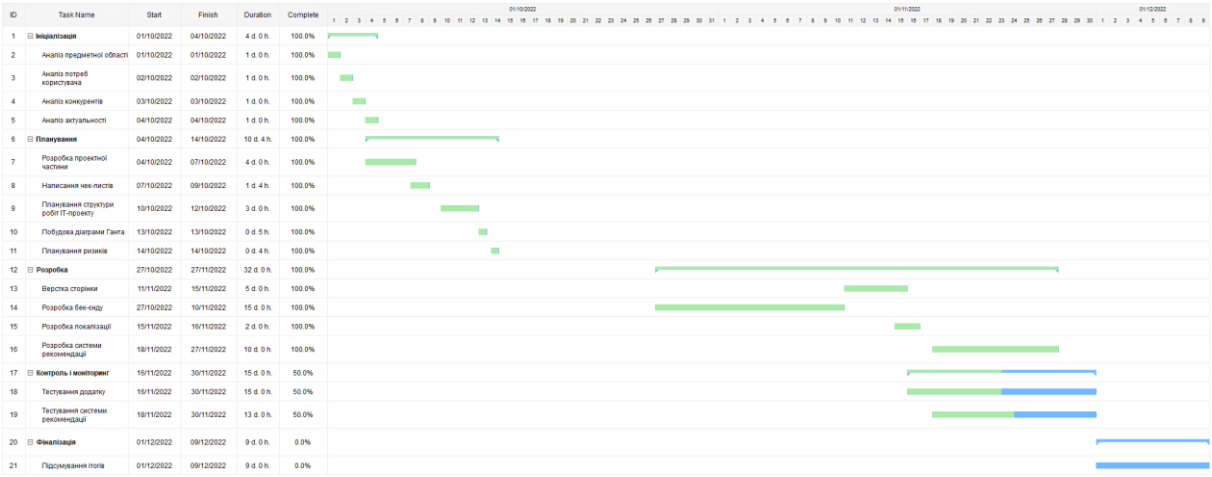

Рисунок А.3.2 – Діаграма Ґанта

Дата початку проекту – 01.10.2022. При виконанні проекту використовувалося по 6 годин на день включно з вихідними днями.

Судячи з діаграми, бачимо що робота над різними етапами йшла послідовно окрім етапу контролю та моніторингу, в ньому все відбувалося паралельно одне одному. Усього було витрачено 70 днів на його виконання.

А.4 Планування ризиків проекту

Ризиком є подія, що ймовірно може статися, котра у випадку виникнення негативно впливає на проект, тому управління ризиком – це важливий процес для реагування на події під час роботи над проектом. Важливою частиною реалізації проекту є моніторинг ризику.

Під аналізу для визначення значень можливостей появи ризику була застосована методика експертної оцінки. Виходячи з цих оцінок, можемо знайти ранг ризику:

R = P * L, де

- R – ранг ризику;
- P – ймовірність виникнення;
- L – ступінь впливу.

Шкала оцінки ризику може відповідати емпіричній шкалі оцінки ризику:

- 5 балів - критичний ризик (0,81 - 1);
- 4 бали - максимальний ризик (0,61 - 0,8);
- 3 бали - високий ризик (0,41 - 0,6);
- 2 бали - нормальний ризик (0,31 - 0,4);
- 1 бал - малий ризик (0 - 0,3).

Таблиця Б.4.1 – Розрахунок ризиків

N	Назва ризику	Опис	P	L	R	План реагування
1	Збільшення кількості роботи над завданням	У разі спроби додати новий функціонал ми можемо стикнутися з тим що ми	0,7	0,8	0,56	У разі настання ризику такого ризику бюджет нашої компанії може бути збільшений для

		маємо більше працювати над проектом а можливо й розширити кількість розробників.				найму нових робітників для пришвидшення роботи над проектом
2	Високе на вантаження на сайт	Через великий попит людей на сайт, ми можемо стикнутися з тим що сайт перестане працювати через навантаження	0,6	0,6	0,36	В цьому випадку необхідно збільшити потужність сервера на котрому знаходиться наш сайт
3	Спроба ДДОС атак	Через можливу популярність нашого сайту, люди можуть зробити ДДОС атаку на нього з ціллю відключити його	0,3	0,9	0,27	

ДОДАТОК Б – КОД ВЕБ-ДОДАТКУ

Views.py

```
from django.shortcuts import render
import time
from spotify_recomendation.dataclient import *
from spotify_recomendation.spotifyapiclient import *

from django.views.generic import TemplateView
from .oauth import SpotifyOauthClient
from django.shortcuts import redirect
from .utils import recommend_from_playlist
from django.conf import settings
import math
import pprint

oauth_client = SpotifyOauthClient()

def init_api_client(session) -> SpotifyApiClient:

    oauth_info = session.get('oauth_info')
    start_time = session.get('start_time') #gets the time at which access_token was first given

    current_time = int(time.time()) #gets the time when this function is called
    token_expiry = oauth_info.get('expires_in', time.time())  #sets the token expiry time which is 3600 seconds or 1 hour
    time_diff = current_time - start_time #how much time has passed since token was given

    if time_diff > token_expiry: #if more than an hour has passed, a new access_token will be provided
        new_token = oauth_client.refresh_token(oauth_info['refresh_token'])   #logic for refreshing access token
        start_time = int(time.time())
        return SpotifyApiClient(new_token['access_token'], oauth_client)
```

```
    else:
        return SpotifyApiClient(oauth_info['access_token'], oauth_client)

class IndexView(TemplateView):
    template_name = "login.html"

    def get_context_data(self, **kwargs):
        context = super(IndexView, self).get_context_data(**kwargs)
        context["url"] = oauth_client.get_auth_url()
        tracks = []

        playlist_infos = playlist_info(oauth_client, "37i9dQZEVXbNcoJZ65xktI")

        for i in playlist_infos:

            author_name = i.get("authors")[0].get("author_name")
            author_link = i.get("authors")[0].get("author_link")
            song_link = i.get("song_link")
            song_name = i.get("song_name")
            song_pictures = i.get("pictures")

            tracks.append([author_name, song_name, song_link, song_pictures])

        context["tracks"] = tracks[:math.ceil(len(tracks)/2)][:6]
        context["tracks2"] = tracks[math.ceil(len(tracks)/2):][:6]

        return context

def redirect_page(request):

    auth_code = request.GET.get("code")
    auth_info = oauth_client.get_token_info(auth_code)

    request.session['oauth_info'] = auth_info
```

```
    request.session['start_time'] = int(time.time())
    request.session['time_frame'] = "short_term"
    request.session['cols'] = ['Danceability', 'Energy', 'Acousticness', 'Speechiness', 'Valence', 'Instrumentalness']

    return redirect("my-music")

def profile_page(request):

    api_client = init_api_client(request.session)
    request_data = api_client.get_user_info()

    #user general info
    user_info = request_data["user_info"]
    username = user_info["display_name"]
    followers = user_info["followers"]["total"]
    profile_pic = user_info["images"][0]["url"] if len(user_info["images"]) != 0 else None

    #playlists user has
    playlist_info = request_data["playlist_info"]
    num_of_playlists = len(playlist_info.get('items', []))

    #how many people the user is following/followed by
    user_follow_info = request_data["following_info"]
    num_of_followed_artists = len(user_follow_info['artists']['items'])

    context = {
        "username":username,
        "followers":followers,
        "pic":profile_pic,
        "playlists":num_of_playlists,
        "follows":num_of_followed_artists
    }
```

```
    return render(request,"profile.html", context=context)

def my_music(request):

    return configure_user_top(request,'music.html', 10)

def configure_user_top(request, html_page, limit):

    api_client = init_api_client(request.session)
    time_frame = request.session.get('time_frame')

    user_top_tracks = api_client.get_user_top_info(limit, time_frame, "tracks")
    user_top_artists = api_client.get_user_top_info(limit, time_frame, "artists")

    if not user_top_tracks: #if the returned data is empty it will set the values to empty
        songs = ['']
        song_ids = ['']
        song_covers = ['']
        song_artists = ['']
        song_albums = ['']

    else:
        songs = user_top_tracks['name']
        song_ids = user_top_tracks['id']
        song_covers = user_top_tracks['image']
        song_artists = user_top_tracks['trackartistname']
        song_albums = user_top_tracks['trackalbumname']

    if not user_top_artists:
        artists = ['']
        artist_ids = ['']
        artist_covers = ['']

    else:
```

```
    artists = user_top_artists['name']
    artist_ids = user_top_artists['id']
    artist_covers = user_top_artists['image']

context = {
"songs": songs,
"song_ids": song_ids,
"song_covers": song_covers,
"song_artists": song_artists,
"song_albums": song_albums,
"artists": artists,
"artist_ids": artist_ids,
"artist_covers": artist_covers,
"zip": zip,
"time": time_frame,
"zip_list": zip(songs, song_covers, song_artists, song_albums, song_ids),
"zip_list2": zip(artists, artist_covers, artist_ids)
}

# sp = oauth_client.get_spotify()
sp = spotipy.client.Spotify(auth= request.session['oauth_info']["access_token"])
playlist_data = sp.user_playlists(sp.me()["id"])
playlists = []

for playlist in playlist_data["items"]:

    track_name = playlist["name"]
    playlist_id = playlist["id"]
    picture_link = playlist["images"][0]["url"]
    total_traks =  playlist["tracks"]["total"]
    playlists.append([track_name, playlist_id, picture_link, total_traks])

if request.method == "POST" and request.POST["playlist_name"]:
```

```
    playlists = [i for i in playlists if i[0].lower().strip().find(request.POST["playlist_name"].lower().strip()) >= 0]


    if playlists:
        if len(playlists) >= 2:
            context.update(playlists=playlists[:math.ceil(len(playlists)/2)])
            context.update(playlists2=playlists[math.ceil(len(playlists)/2):])
        else:
            context.update(playlists=playlists)
            context.update(playlists2=[])

    return render(request, html_page, context=context)

def new(request):

    api_client = init_api_client(request.session)
    user_top_songs = api_client.get_user_top_info(50, request.session.get('time_frame'), "tracks")
    user_top_artists = api_client.get_user_top_info(33, request.session.get('time_frame'), "artists")

    if not user_top_songs or not user_top_artists: #if the user has no data (i.e the returned dict is empty)
        return error_page(request, "Sorry, your account does not seem to have any data I can analyze. Please go back to the 'My Music' section and try switching the timeframe to see if you have any data there!")

    else:
        song_ids = user_top_songs['id']
        artist_ids = user_top_artists['id']

    data_client = DataClient(api_client, song_ids, artist_ids, request.session.get('time_frame'))
    cols = request.session.get('cols')

    #user info
```

```
    user_id = api_client.get_user_info()['user_info']['id']

    #tracks
    seeds = data_client.get_recommendation_seeds(len(user_top_songs['name']),
len(user_top_artists['name']))
    user_audio_features = data_client.get_user_top_avg_audio_features(cols)
    user_popularity = data_client.get_user_avg_popularity("tracks")

    #tracks
    try:
        get_recommended_tracks_info = api_client.get_track_recommendations(10, seeds,
user_audio_features, user_popularity, "normal")

    except:
        return error_page(request, "Can't get recommendations.")

    track_names = get_recommended_tracks_info['name']
    track_ids = get_recommended_tracks_info['id']
    track_image = get_recommended_tracks_info['image']
    track_artists = get_recommended_tracks_info['trackartistname']
    track_albums = get_recommended_tracks_info['trackalbumname']
    track_urls =  get_recommended_tracks_info['id']

    #artists
    get_recommended_artists_info = api_client.get_artist_recommendations(seeds['artist'])
    artist_names = get_recommended_artists_info['name']
    artist_ids = get_recommended_artists_info['id']
    artist_images = get_recommended_artists_info['image']


    tracks=zip(track_names, track_image, track_artists, track_albums, track_ids)
    artists=zip(artist_names, artist_images, artist_ids)

    #user created recommendations
```

```
    if request.method == "POST":

        # user_inputed_popularity = int(request.POST['pop'])
        playlist_url = request.POST['playlist_url']
        df = api_client.extract(playlist_url)
        edm_top=                                    recommend_from_playlist(settings.SONG_DF,
settings.COMPLITE_FEATURES_SET, df)
        # user_inputed_audio_features = [[f"target_{col.lower()}", float(request.POST[col])] for col
in cols if float(request.POST[col]) != 0.0]
        playlist_name = request.POST['playlistname']

        try:
            df = api_client.extract(playlist_url)
            edm_top=                                recommend_from_playlist(settings.SONG_DF,
settings.COMPLITE_FEATURES_SET, df)

        except:
            return error_page(request, "Can't get recommendations.")

        get_new_playlist_id = api_client.create_new_playlist(user_id, playlist_name)['id']
        modified_ids = ["spotify:track:" + track_id for track_id in edm_top["track_uri"]]
        csv_ids = ','.join(modified_ids)

        api_client.add_items_to_playlist(get_new_playlist_id, csv_ids)
        sp = oauth_client.get_spotify()
        playlist_items = sp.playlist_items(get_new_playlist_id)

        tracks = []
        artists = []

        for item in playlist_items["items"]:
            try:
                tracks.append([
                    item["track"]["name"],
```

```
                item["track"]["album"]["images"][0]["url"],
                item["track"]["artists"][0]["name"],
                item["track"]["album"]["name"],
                item["track"]["id"],
                ]
            )
            artists.append(
                [
                    item["track"]["artists"][0]["name"],
                    sp.artist(item["track"]["artists"][0]["id"])["images"][0]["url"],
                    item["track"]["artists"][0]["id"],
                ]
            )
        except:
            continue

    return render(
        request,
        'recommendations.html',
        context=dict(
            cols=cols,
            tracks=tracks,
            artists=artists
            )
        )

def error_page(request, text):
    return render(request,'error.html', context=dict(text=text))
```

utils.py

```
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

```
def generate_playlist(feature_set, playlist_df):

  set_playlist = feature_set[complete_feature_set['id'].isin(playlist_df['id'].values)]
  set_nonplaylist = feature_set[~complete_feature_set['id'].isin(playlist_df['id'].values)]
  set_playlist_final = feature_set_playlist.drop(columns = "id")
   return feature_set_playlist_final.sum(axis = 0), set_nonplaylist


def generate_playlist_recos(df, features, nonplaylist_features):

   non_playlist_df = df[df['id'].isin(nonplaylist_features['id'].values)]
   non_playlist_df['sim'] = cosine_similarity(nonplaylist_features.drop('id', axis = 1).values, features.values.reshape(1, -1))[:,0]
   comlpete_df = non_playlist_df.sort_values('sim',ascending = False).head(40)

   return comlpete_df

def recommend_from_playlist(songDF,complete_feature_set,playlistDF_test):

   feature_set_playlist_vector, feature_set_nonplaylist = generate_playlist_feature(complete_feature_set, playlistDF_test)
   result = generate_playlist_recos(songDF, feature_set_playlist_vector, feature_set_nonplaylist)
   return result
```