НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Інформаційна система обліку медіаконтенту на основі баз даних»

Виконала:
студентка IV курсу, групи КА-15
Матюхіна Софія Юріївна _____

Керівник:
зав. каф. СП НН ІПСА, д.т.н., професор
Мухін Вадим Євгенійович _____

Консультант з економічного розділу:
доцент каф. ЕК ФММ, к.е.н.
Рощина Надія Василівна _____

Консультант з нормоконтролю:
Канцедал Георгій Олегович _____

Рецензент:
доцент кафедри НН ІАТЕ, к.т.н., доцент
Шаповалова Світлана Ігорівна _____

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

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувача кафедри
_____ Оксана ТИМОЩУК
«___»_____ 2025 р.

ЗАВДАННЯ

на дипломну роботу студентці
Матюхіной Софії Юріївні

1. Тема роботи «**Інформаційна система обліку медіаконтенту на основі баз даних**», керівник роботи Мухін Вадим Євгенович, д.т.н, професор, затверджені наказом по університету від «26 » травня 2025 р. № 1759-с

2. Термін подання студентом роботи: 11.06.2025

3. Вихідні дані до роботи: Метадані відеоконтенту (назва, опис, дата публікації, URL-адреса, тривалість), категорії контенту (наприклад, Gaming, Education, Music, Vlogging), статистичні показники відео (кількість переглядів, лайків, коментарів), інформація про користувачів та їх взаємодії (категорії перегляду, дати додавання контенту), а також структура реляційної бази даних для зберігання та обліку медіаконтенту, включаючи таблиці медіафайлів, категорій, метрик популярності та звітів.

4. Зміст роботи: 1. Аналіз баз даних у контексті побудови інформаційних систем медіаконтенту 2. Загальні принципи розробки інформаційної системи медіаобліку 3. Проектування та реалізація системи обліку медіаконтенту 4. Економічне обґрунтування розробки системи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент ЕК ФММ		

7. Дата видачі завдання: 16.04.2025

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання та затвердження теми БДР	16.04.2025	виконано
2	Аналіз предметної області та постановка задачі	24.04.2025	виконано
3	Пошук та опрацювання літературних джерел, аналогів	28.04.2025	виконано
4	Формування вимог до інформаційної системи	03.05.2025	виконано
5	Розробка структури бази даних та проектування моделі даних	07.05.2025	виконано
6	Реалізація інформаційної системи	12.05.2025	виконано
7	Тестування системи, виявлення та виправлення помилок	20.05.2025	виконано
8	Оформлення дипломної роботи та презентації	29.05.2025	виконано

Студент Софія МАТЮХІНА

Керівник Вадим МУХІН

РЕФЕРАТ

Дипломна робота містить 108 с., 38 рис., 13 табл., 15 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК МЕДІАКОНТЕНТУ, БАЗИ ДАНИХ, МЕДІАФАЙЛИ, УПРАВЛІННЯ КОНТЕНТОМ, АНАЛІЗ ВІДЕОКОНТЕНТУ, ЗБІР ДАНИХ, YOUTUBE DATA API, РЕЛЯЦІЙНІ БАЗИ ДАНИХ, MYSQL, FLASK, PYTHON, ВЕБ-ІНТЕРФЕЙС.

Об’єктом дослідження є процес обліку та аналізу медіаконтенту з платформ потокового відео, таких як YouTube.

Предмет дослідження – методи та засоби автоматизації збору, зберігання, обробки й аналізу медіаданих.

Метою дипломної роботи є розробка інформаційної системи для обліку та аналізу відеоконтенту, яка дозволяє користувачам завантажувати дані з YouTube, зберігати їх у базі даних, проводити аналіз за категоріями, порівнювати відео, а також генерувати аналітичні звіти.

У процесі виконання роботи розроблено вебзастосунок на основі клієнт-серверної архітектури з використанням Flask, MySQL та Python. Створено реляційну модель бази даних для зберігання метаданих відео, спроектовано інтерфейс користувача на основі HTML, CSS і JavaScript. Реалізовано функціонал автоматичного збору даних через YouTube API, їх обробку за допомогою бібліотек Pandas і Matplotlib, а також створення звітів.

Розроблена система дозволяє отримувати ключові показники популярності відео, аналізувати дані за категоріями, створювати графіки та формувати звіти для підтримки рішень у медіасфері. Модель прогнозування оцінює кількість переглядів і пропонує рекомендації щодо оптимальної категорії та тривалості відео. Результати роботи мають практичне значення для організацій, що працюють із медіаконтентом, і можуть бути використані для подальшого розвитку систем аналітики.

ABSTRACT

The thesis consists of 108 pages, 38 figures, 13 tables, and 15 references.

INFORMATION SYSTEM, MEDIA CONTENT ACCOUNTING, DATABASES, MEDIA FILES, CONTENT MANAGEMENT, VIDEO CONTENT ANALYSIS, DATA COLLECTION, YOUTUBE DATA API, RELATIONAL DATABASES, MYSQL, FLASK, PYTHON, WEB INTERFACE.

The object of the study is the process of accounting and analyzing media content from video streaming platforms such as YouTube.

The subject of the study is the methods and tools for automating the collection, storage, processing, and analysis of media data.

The purpose of this thesis is to develop an information system for accounting and analyzing video content, allowing users to collect data from YouTube, store it in a database, analyze by categories, compare videos, and generate analytical reports.

During the project, a web application was developed based on a client-server architecture using Flask, MySQL, and Python. A relational database model for storing video metadata was created, and a user interface was designed using HTML, CSS, and JavaScript. The system provides functionality for automated data collection via the YouTube Data API, data processing using Pandas and Matplotlib libraries, and report generation.

The developed system enables the retrieval of key video popularity metrics, analysis of data by categories, creation of charts, and generation of reports to support decision-making in the media sphere. The predictive model estimates view counts and provides recommendations on the optimal category and video duration. The results are practically significant for organizations working with media content and can be used for further development of analytics systems.

ЗМІСТ

ВСТУП

У сучасному цифровому світі медіаконтент став невід'ємною частиною повсякденного життя. Відеоплатформи, соціальні мережі, стрімінгові сервіси та інші цифрові ресурси щодня генерують величезні обсяги даних: за даними YouTube, щосекунди на платформу завантажується понад 500 годин відеоконтенту. Таке стрімке зростання обсягів інформації створює нові виклики для медіаіндустрії, зокрема потребу в ефективних інформаційних системах, які здатні забезпечити облік, аналіз і управління медіаконтентом.

Актуальність теми дослідження зумовлена стрімким розвитком цифрових технологій і зростанням попиту на інтелектуальні рішення для управління медіаконтентом. Сьогодні медіаіндустрія потребує не лише інструментів для зберігання контенту, але й систем, які дозволяють аналізувати його популярність, виявляти тренди та надавати цінні інсайти для творців, маркетологів і бізнесу. Бази даних відіграють ключову роль у таких системах, забезпечуючи надійне зберігання метаданих, а також підтримуючи аналітичні операції для оцінки ефективності контенту. У цьому контексті розробка вебдодатку для аналізу YouTube-відео, представлена в цій роботі, є прикладом практичного вирішення зазначених проблем, демонструючи можливості баз даних у медіааналітиці.

Розробка гнучких і масштабованих рішень для медіаобліку є важливим завданням, адже від цього залежить здатність компаній адаптуватися до швидкозмінних ринкових умов і потреб аудиторії.

Метою цієї дипломної роботи є дослідження теоретичних і практичних аспектів побудови інформаційних систем управління медіаконтентом на основі сучасних баз даних, а також створення прототипу такої системи для аналізу відеоконтенту.

Ця робота має як теоретичне, так і практичне значення. Теоретичний аналіз допомагає систематизувати знання про сучасні підходи до управління

медіаконтентом, тоді як практична реалізація демонструє, як ці знання можуть бути застосовані для створення функціональних рішень. У довгостроковій перспективі результати дослідження можуть бути використані для вдосконалення інформаційних систем медіаобліку, створення рекомендаційних алгоритмів і підтримки стратегічного планування в медіаіндустрії.

РОЗДІЛ 1 АНАЛІЗ БАЗ ДАНИХ У КОНТЕКСТІ ПОБУДОВИ ІНФОРМАЦІЙНИХ СИСТЕМ МЕДІАКОНТЕНТУ

1.1. Актуальність теми

Сучасна медіаіндустрія характеризується стрімким зростанням обсягів цифрового контенту, що охоплює відео, аудіо, текстові матеріали та інтерактивні формати. Наприклад, платформа YouTube щоденно обробляє мільйони годин нового відеоконтенту, а її аудиторія налічує мільярди користувачів по всьому світу. Цей глобальний феномен підкреслює ключову роль медіаконтенту в інформаційному суспільстві та створює нагальну потребу в інформаційних системах, які забезпечують ефективне управління даними. У центрі таких систем перебувають бази даних, які дозволяють структурувати, зберігати та аналізувати величезні масиви інформації

Бази даних є основою для роботи з метаданими медіаконтенту. Вони забезпечують швидке виконання запитів, підтримують аналітичні операції та дозволяють генерувати звіти для оцінки ефективності контенту. У контексті медіаобліку актуальність баз даних зумовлена необхідністю обробки великих обсягів даних у реальному часі, інтеграції з API платформ, таких як YouTube Data API, і забезпечення масштабованості для роботи з постійно зростаючими масивами інформації.

Аналіз медіаконтенту має важливе значення для різних груп стейкхолдерів: творці контенту використовують його для оптимізації своєї роботи, маркетологи – для розробки таргетованої реклами, а бізнес – для стратегічного планування. априклад, розуміння популярності категорій відео (Gaming, Education, Vlogging) допомагає адаптувати контент до аудиторії. Розроблений у рамках дослідження вебдодаток для аналізу YouTube-відео

ілюструє ці можливості, отримуючи дані через API та аналізуючи метрики в базі даних.

Технологічні виклики, пов’язані з медіаобліком, також підкреслюють актуальність теми. Інтеграція з API, обробка неструктурованих даних, таких як коментарі чи теги, забезпечення безпеки та масштабованості систем – це лише деякі з проблем, які потребують сучасних рішень у сфері баз даних. Крім того, ефективне управління медіаконтентом має соціально-економічне значення: воно сприяє розвитку медіаіндустрії, створенню нових бізнес-можливостей і підвищенню конкурентоспроможності компаній у цифровій економіці.

1.2. Порівняння типів баз даних для медіаобліку

Ефективне управління медіаконтентом вимагає ретельного вибору типу бази даних, оскільки різні системи мають свої особливості, які впливають на швидкість, масштабованість і гнучкість обробки даних. У контексті медіаобліку основними типами баз даних є реляційні (SQL) та нереляційні (NoSQL)

1.2.1. Реляційні бази даних (SQL)

Реляційні бази даних (SQL) базуються на чітко визначеній табличній структурі, де дані організовані у вигляді таблиць із фіксованими схемами, пов’язаних між собою ключами. До популярних реляційних СУБД належать MySQL, PostgreSQL і Oracle Database. У контексті медіаобліку реляційні бази

даних ефективно застосовуються для зберігання структурованих метаданих, таких як назви файлів, категорії, дати створення чи теги.

1.2.2. Нереляційні бази даних (NoSQL)

Нереляційні бази даних (NoSQL) пропонують гнучкіший підхід до управління даними, використовуючи різні моделі, такі як документо-орієнтовані (MongoDB), ключ-значення (Redis) або графові (Neo4j). У медіаобліку найпоширенішими є документо-орієнтовані бази, які дозволяють зберігати дані у вигляді JSON- або BSON-документів із динамічною структурою. Наведемо таблицю переваг і недоліків нереляційних баз даних

1.2.3. Порівняння в контексті медіаобліку

У контексті медіаобліку порівняння реляційних (SQL) та нереляційних (NoSQL) баз даних дозволяє виявити їхні сильні та слабкі сторони залежно від специфіки обробки медіаконтенту. Реляційні бази даних демонструють переваги в роботі з чітко структурованими даними, де потрібен швидкий доступ до інформації за конкретними параметрами, такими як теги, категорії чи назви медіафайлів. Завдяки фіксованій схемі та оптимізованим механізмам індексації SQL-запити забезпечують високу швидкість виконання, що робить ці бази ідеальними для систем із частими пошуковими операціями. Проте реляційні бази менш ефективні для швидкого запису великих обсягів даних або роботи з динамічними метаданими, де структура може змінюватися залежно від типу контенту.

Нереляційні бази даних, навпаки, вирізняються здатністю швидко обробляти великі обсяги даних і працювати з неструктурованими чи напівструктурованими метаданими, що є важливим для сучасних медіаплатформ, таких як стрімінгові сервіси. NoSQL забезпечує гнучкість структури, дозволяючи легко адаптувати базу до різноманітних типів контенту, де атрибути можуть значно варіюватися, наприклад, у контенті, створеному користувачами. Крім того, NoSQL-бази краще підходять для горизонтального масштабування, що є критично важливим для систем із мільйонами користувачів, адже дозволяє розподіляти навантаження між кількома серверами без значних ускладнень. Водночас реляційні бази потребують складніших рішень, таких як шардування чи реплікація, для досягнення подібної масштабованості.

Таким чином, вибір між SQL і NoSQL у медіаобліку залежить від потреб конкретної системи. Реляційні бази є оптимальними для проєктів із передбачуваною структурою даних і високими вимогами до швидкості пошуку, тоді як нереляційні бази краще відповідають потребам систем із динамічними даними та необхідністю масштабування для обробки великих обсягів контенту.

1.3. Огляд сучасних СУБД і їх застосування у сфері медіа

Сучасні інформаційні системи управління медіаконтентом значною мірою залежать від ефективних систем управління базами даних (СУБД), які забезпечують надійне зберігання, швидкий доступ і обробку великих обсягів даних. Медіаконтент, такий як відео, аудіо, зображення та їх метадані, потребує спеціалізованих підходів до організації баз даних, враховуючи їх різноманітність, обсяг і вимоги до швидкості обробки.

Розглянемо популярні СУБД, зокрема MySQL, PostgreSQL і MongoDB, та проаналізовано їх застосування в контексті управління медіаконтентом.

1.3.1. MySQL

MySQL є однією з найпоширеніших реляційних СУБД з відкритим кодом, яка використовується у багатьох галузях, включаючи медіаіндустрію. Її популярність зумовлена простотою використання, високою продуктивністю та широкою підтримкою спільноти. MySQL працює на основі реляційної моделі, де дані зберігаються у вигляді таблиць із чітко визначеною структурою. У сфері медіа MySQL ефективно застосовується для зберігання метаданих, таких як назви файлів, описи, теги, дати створення чи категорії контенту. Наприклад, медіаплатформи, які потребують швидкого доступу до структурованих даних, таких як каталоги зображень або списки відеофайлів, часто використовують MySQL для організації бібліотек контенту. Завдяки підтримці індексації та швидких запитів, ця СУБД забезпечує ефективну роботу з метаданими, хоча для зберігання великих бінарних файлів (наприклад, самих відеофайлів) зазвичай потрібні додаткові файлові системи або хмарні сховища

1.3.2. PostgreSQL

PostgreSQL, ще одна потужна реляційна СУБД, вирізняється розширеними функціональними можливостями та високою надійністю. На відміну від MySQL, PostgreSQL підтримує складніші типи даних, такі як JSON, XML або масиви, що робить її гнучкою для роботи з неструктурованими або

напівструктурованими даними, характерними для медіаконтенту. У медіаіндустрії PostgreSQL використовується для управління складними каталогами контенту, де метадані можуть містити різноманітні атрибути, наприклад, роздільну здатність відео, кодеки, геолокаційні дані чи коментарі користувачів. Крім того, PostgreSQL підтримує розширені функції, такі як повнотекстовий пошук, що дозволяє швидко знаходити контент за ключовими словами або тегами. Ця СУБД ідеально підходить для систем управління цифровими активами (Digital Asset Management, DAM), де потрібна інтеграція з різними джерелами даних і підтримка транзакцій для забезпечення цілісності.

1.3.3. MongoDB

MongoDB, як представник нереляційних (NoSQL) баз даних, пропонує принципово інший підхід до управління даними. Вона базується на документо-орієнтованій моделі, де дані зберігаються у вигляді JSON-подібних документів, що забезпечує високу гнучкість у роботі з неструктурованими даними. У медіаіндустрії MongoDB особливо ефективна для зберігання метаданих із динамічною структурою, наприклад, для контенту, створеного користувачами (User-Generated Content, UGC), де атрибути можуть варіюватися залежно від типу файлу чи платформи. Наприклад, MongoDB може використовуватися для зберігання інформації про відео, де кожен документ містить унікальний набір метаданих, таких як тривалість, формат, автор чи коментарі. Завдяки горизонтальній масштабованості, MongoDB підходить для обробки великих обсягів даних у реальному часі, що є критично важливим для стрімінгових платформ або соціальних мереж.

1.4. Вимоги до баз даних у системах управління медіаконтентом

Системи управління медіаконтентом (Content Management Systems, CMS) та цифровими активами (Digital Asset Management, DAM) покладаються на бази даних як на ключовий компонент для організації, зберігання та обробки великих обсягів даних, що включають як сам медіаконтент, так і його метадані. Для забезпечення ефективної роботи таких систем бази даних повинні відповідати низці вимог, які враховують специфіку медіаіндустрії. До основних вимог належать підтримка великих обсягів даних, швидкий пошук, гнучкість метаданих, інтеграція з різними медіаформатами, забезпечення безпеки та високий рівень доступності. Ці аспекти є критично важливими для створення надійних і продуктивних інформаційних систем.

Однією з ключових вимог є здатність бази даних обробляти великі обсяги даних. Медіаіндустрія характеризується стрімким зростанням обсягів контенту, включаючи відеофайли високої роздільної здатності, аудіозаписи, зображення та їхні метадані. База даних має ефективно зберігати та управляти цими даними, забезпечуючи швидкий доступ навіть при обробці мільйонів записів. Наприклад, стрімінгові платформи, такі як Netflix чи YouTube, потребують баз, здатних працювати з терабайтами даних, зберігаючи при цьому стабільну продуктивність.

Швидкий пошук є ще однією критично важливою вимогою, оскільки користувачі медіасистем часто виконують запити для знаходження контенту за різними критеріями, такими як назви, теги, категорії чи дати створення. База даних повинна підтримувати ефективні механізми індексації та повнотекстовий пошук, щоб забезпечити миттєвий доступ до потрібної інформації. Наприклад, у системах управління цифровими активами швидкий пошук за метаданими, такими як роздільна здатність відео чи автор зображення, значно підвищує зручність роботи для користувачів.

Гнучкість метаданих є не менш важливою вимогою, оскільки медіаконтент часто має різноманітні та динамічні атрибути. Наприклад, метадані для відео можуть включати тривалість, формат, кодек, геолокацію чи коментарі користувачів, тоді як для зображень – розміри, колірний профіль чи теги. База даних має дозволяти легко додавати нові атрибути без необхідності суттєвих змін у структурі, що особливо актуально для систем із контентом, створеним користувачами, де структура метаданих може бути непередбачуваною.

Інтеграція з різними медіаформатами є ще одним важливим аспектом. База даних повинна забезпечувати можливість роботи з різними типами файлів, такими як MP4, JPEG, WAV, та їхніми метаданими, а також підтримувати інтеграцію з хмарними сховищами чи файловими системами, де зазвичай зберігаються великі бінарні файли. Наприклад, інформаційна система може зберігати метадані у базі даних, а самі медіафайли – у хмарних сховищах, таких як Amazon S3, що вимагає безшовної взаємодії між компонентами.

Безпека даних є критично важливою вимогою, оскільки медіаконтент часто містить конфіденційну інформацію, таку як авторські права, персональні дані користувачів чи комерційні активи. База даних повинна підтримувати механізми шифрування, контролю доступу та автентифікації, щоб запобігти несанкціонованому доступу. Наприклад, у корпоративних DAM-системах доступ до контенту може бути обмежений за ролями користувачів, що вимагає надійних засобів захисту даних.

Нарешті, доступність бази даних є ключовим фактором для забезпечення безперервної роботи медіасистем. Системи управління медіаконтентом часто працюють у режимі 24/7, особливо у випадку глобальних платформ із користувачами з різних часових поясів. База даних має бути стійкою до збоїв, підтримувати реплікацію та резервне копіювання, щоб гарантувати доступ до даних навіть у разі апаратних чи програмних проблем.

1.5. Приклади існуючих інформаційних систем обліку медіаконтенту

Інформаційні системи для обліку медіаконтенту відіграють важливу роль у медіаіндустрії, забезпечуючи ефективне управління цифровими активами, такими як відео, аудіо, зображення та їхні метадані. Ці системи варіюються від спеціалізованих платформ управління цифровими активами (Digital Asset Management, DAM) до універсальних систем управління контентом (Content Management Systems, CMS), які дозволяють організаціям зберігати, організовувати, шукати та розповсюджувати медіаконтент, забезпечуючи швидкий доступ і захист даних. Нижче наведено приклади таких систем, їхній функціонал і використані технології.

Adobe Experience Manager (AEM) є однією з провідних DAM-систем, яка широко застосовується великими організаціями для управління медіаконтентом. AEM забезпечує створення, редагування, зберігання та розповсюдження цифрових активів, включаючи зображення, відео, документи й аудіофайли. Система дозволяє централізовано зберігати метадані, що полегшує пошук контенту за ключовими словами, тегами чи іншими атрибутами. AEM підтримує автоматизацію робочих процесів, таких як транскодування відео чи адаптація зображень для різних пристроїв, а також інтеграцію з хмарними сервісами, наприклад, Adobe Cloud. Технологічно система базується на мові програмування Java та використовує реляційні бази даних, такі як Oracle Database або MySQL, для структурованих метаданих, а також NoSQL-системи для обробки великих обсягів неструктурованих даних. Це забезпечує високу продуктивність і гнучкість у роботі з різноманітним контентом.

WordPress є популярною CMS, яка використовується для управління медіаконтентом на блогах, новинних сайтах і медіаплатформах. Хоча WordPress насамперед асоціюється з веб-контентом, вона ефективно

підтримує облік медіафайлів, таких як зображення, відео та аудіо, завдяки вбудованій бібліотеці медіа. Користувачі можуть завантажувати файли, додавати метадані, такі як назви, описи чи теги, та організовувати їх у категорії. WordPress підтримує плагіни для розширення функціоналу, наприклад, для оптимізації зображень або інтеграції з відеохостингами. У технічному плані система базується на PHP і використовує реляційну базу даних MySQL для зберігання контенту та метаданих, що забезпечує простоту управління та швидкий доступ до даних.

Widen Collective – це потужна DAM-система, призначена для управління великими обсягами медіаконтенту в корпоративному середовищі. Вона забезпечує централізоване зберігання цифрових активів і підтримує гнучке управління метаданими, що дозволяє швидко знаходити потрібний контент. Widen Collective пропонує кастомні робочі процеси, наприклад, для затвердження контенту чи автоматичного тегування, а також інтеграцію з хмарними сховищами, такими як Amazon S3. Система використовує гібридний підхід, комбінуючи реляційні бази даних, такі як PostgreSQL, для структурованих даних і NoSQL-бази, наприклад, MongoDB, для обробки неструктурованих даних, що забезпечує високу масштабованість і гнучкість.

Bynder – це хмарна DAM-система, орієнтована на управління брендовим контентом. Вона дозволяє зберігати медіафайли, такі як відео, зображення та документи, а також управляти їхніми метаданими для швидкого пошуку та повторного використання. Bynder підтримує інтеграцію з соціальними мережами та платформами, такими як YouTube чи Vimeo, що спрощує імпорт метаданих і контенту. Система використовує NoSQL-бази даних для забезпечення гнучкості метаданих і масштабованості, а також хмарні технології для високої доступності. Bynder також пропонує інструменти для створення аналітичних звітів, що допомагають оцінювати ефективність контенту.

Ці системи ілюструють різноманітність підходів до управління медіаконтентом, поєднуючи реляційні та нереляційні бази даних для забезпечення швидкості, гнучкості та масштабованості.

1.6. Висновки до розділу 1

У розділі проаналізовано роль баз даних у системах управління медіаконтентом. Досліджено актуальність теми, підкреслено необхідність ефективного управління великими обсягами даних у медіаіндустрії. Порівняно реляційні (SQL) та нереляційні (NoSQL) бази даних, оцінено їхні переваги й недоліки в контексті медіаобліку. Розглянуто сучасні СУБД (MySQL, PostgreSQL, MongoDB) та їх застосування для обробки структурованих і неструктурованих метаданих. Визначено ключові вимоги до баз даних, такі як швидкий пошук, гнучкість, безпека та масштабованість. Проаналізовано приклади DAM- і CMS-систем.

РОЗДІЛ 2 ЗАГАЛЬНІ ПРИНЦИПИ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ МЕДІАОБЛІКУ

2.1. Формулювання цілей та завдань системи

Метою розробки інформаційної системи обліку медіаконтенту є створення ефективного інструменту для автоматизованого збору, зберігання, аналізу та управління метаданими відеоконтенту з платформи YouTube. Система призначена для підтримки прийняття рішень у медіаіндустрії шляхом надання аналітичних даних про популярність контенту, його категоризацію та порівняльні характеристики. Вона орієнтована на різні групи користувачів, зокрема творців контенту, маркетологів і аналітиків, які прагнуть оптимізувати свою роботу, виявляти тренди та адаптувати контент до потреб аудиторії.

Для досягнення цієї мети визначено такі завдання системи.

1. Збір даних. Автоматизовано отримувати метадані та статистику відеоконтенту через YouTube Data API для подальшого аналізу.
2. Зберігання даних. Організувати метадані в реляційній базі даних MySQL, забезпечуючи чітку структуру, швидкий доступ і надійність інформації.
3. Аналіз за категоріями. Групувати відео за категоріями і вираховувати середні показники популярності для виявлення трендів.
4. Прогнозування популярності. Прогнозування популярності відео на основі категорій і тривалості для надання рекомендацій користувачам
5. Порівняння відео. Дати змогу порівнювати відео в одній категорії за метриками, відображаючи результати в таблицях і графіках.
6. Створення звітів. Генерувати звіти із узагальненою статистикою та візуалізаціями для зручного використання.

7. Зручний інтерфейс. Створити веб-інтерфейс, який спрощує введення URL відео, вибір категорій, перегляд даних і отримання звітів.

Ці завдання спрямовані на автоматизацію аналізу контенту, ефективне управління цифровими активами та підтримку стратегічного планування, забезпечуючи гнучкість і масштабованість системи.

2.2. Опис принципів проектування бази даних для медіаобліку

Для проектування бази даних інформаційної системи обліку медіаконтенту використовуються принципи, які забезпечують її ефективність і надійність. Дані організуються в чітко структуровані таблиці, що дозволяє уникнути надлишковості та забезпечити їх цілісність. Наприклад, інформація про відео, їхні категорії та статистику зберігається окремо, що полегшує оновлення даних і швидкий пошук. Водночас структура бази даних залишається гнучкою, даючи змогу легко додавати нові атрибути, такі як теги чи параметри відео, без потреби суттєво змінювати схему – це особливо важливо для роботи з динамічним контентом YouTube. Щоб прискорити обробку пошукових і аналітичних запитів, застосовується індексація ключових полів, таких як ідентифікатори відео чи категорії, що критично для аналізу великих обсягів даних. База даних також спроектована з урахуванням масштабованості, щоб ефективно справлятися зі зростанням обсягів контенту, використовуючи реляційну СУБД MySQL, яка ідеально підходить для структурованих метаданих. Нарешті, для захисту даних, зокрема інформації про авторські права чи статистику, застосовуються механізми шифрування та контролю доступу, що гарантує їхню конфіденційність. Такий підхід створює надійну основу для управління медіаконтентом, поєднуючи швидкість, гнучкість і безпеку.

2.3. Опис технологій і засобів реалізації бази даних для медіаобліку

Для реалізації інформаційної системи обліку медіаконтенту обрано технологічний стек, який забезпечує ефективне зберігання, обробку та представлення даних. Основою системи є реляційна СУБД MySQL, вибрана за її простоту, надійність і підтримку структурованих метаданих, що ідеально підходить для зберігання інформації про відеоконтент із YouTube. Логіку обробки даних і взаємодію з базою реалізовано за допомогою мови програмування Python, зокрема фреймворку Flask, який забезпечує швидке створення серверної частини системи. Для аналізу даних і побудови графіків використовуються бібліотеки Pandas і Matplotlib, що дозволяють обробляти статистичні показники та створювати наочні візуалізації. Отримання метаданих із YouTube здійснюється через YouTube Data API, який забезпечує доступ до актуальної інформації про відео, такої як перегляди, лайки чи коментарі.

Для зручної взаємодії користувачів із системою розроблено веб-інтерфейс на основі HTML, CSS і JavaScript із використанням шаблонізатора Jinja2, що дозволяє динамічно відображати дані. Генерація звітів реалізується через бібліотеку FPDF для створення документів у форматі PDF та Pandas для експорту даних у XLSX, що забезпечує зручність аналізу результатів. Технології інтегруються в архітектуру MVC: MySQL виступає моделлю для зберігання даних, Flask керує логікою як контролер, а HTML-шаблони відповідають за представлення інформації користувачу.

Такий вибір технологій зумовлений їхньою поширеністю, підтримкою спільнотою та сумісністю, що забезпечує стабільність, гнучкість і легкість розширення системи для потреб медіаобліку.

2.4. Висновки до розділу 2

У цьому розділі ми розглянули ключові принципи розробки інформаційної системи обліку медіаконтенту, які ґрунтуються на чітко сформульованих цілях і завданнях, спрямованих на автоматизацію збору, зберігання й аналізу даних відеоконтенту. Принципи проектування бази даних, такі як нормалізація, гнучкість структури, оптимізація запитів, масштабованість і безпека, створюють надійну основу для ефективного управління метаданими. Використання технологічного стека, що включає MySQL для зберігання даних, Python із фреймворком Flask для обробки логіки, YouTube Data API для отримання інформації та веб-інтерфейс на основі HTML і Jinja2, забезпечує стабільність і зручність системи.

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ОБЛІКУ МЕДІАКОНТЕНТУ

3 .1. Розробка структури бази даних для медіаконтенту

3.1.1. Визначення основних сутностей

Для системи обліку медіаконтенту виділено такі основні сутності.

1. Медіаконтент (MediaContent): інформація про контент (назва, дата публікації, тип контенту, URL, тривалість).
2. Категорії (Directions): категорії або напрямки відео (наприклад, Gaming, Music), які дозволяють групувати контент.
3. Метрики популярності (PopularityMetrics): статистичні дані про відео (лайки, коментарі, перегляди, тривалість, дата збору даних).
4. Зв’язок між медіаконтентом і категоріями (MediaContentDirections): таблиця для реалізації зв’язку "багато до багатьох" між відео та категоріями.
5. Звіти (Reports): дані про згенеровані звіти (назва, тип, шлях до файлу, дата створення).
6. Прогнози популярності (PopularityPredictions): відображає результати прогнозування популярності відео, отримані за допомогою методів машинного навчання.

3.1.2. Опис структури таблиць

База даних реалізована у реляційній СУБД MySQL, що забезпечує надійність і ефективність роботи з даними. Нижче наведено опис таблиць, їх полів, типів даних і зв’язків.

Перша таблиця, Directions (таблиця 3.1.), зберігає категорії (напрями) медіаконтенту, наприклад, Gaming, Music тощо.

Таблиця 3.1 – Directions

Поля	**Тип даних**	**Обмеження**	**Призначення**
direction_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор напряму.
direction_name	VARCHAR (100)	NOT NULL, UNIQUE	Назва напряму.
description	TEXT		Опис напряму.

Друга таблиця, названа MediaContent (таблиця 3.2), зберігає основну інформацію про медіаконтент, зокрема відео.

Таблиця 3.2 – MediaContent

Поля	**Тип даних**	**Обмеження**	**Призначення**
content_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор контенту.
title	VARCHAR (255)	NOT NULL	Назва відео.
publication_date	DATE	NOT NULL	Дата публікації відео.
content_type	VARCHAR (50)	NOT NULL	Тип контенту.
content_type	VARCHAR (50)	NOT NULL	Тип контенту.
url	VARCHAR (255)	NOT NULL	URL-адреса відео.
duration	VARCHAR (8)	DEFAULT '00:00:00'	Тривалість відео у форматі HH:MM:SS.

Таблиця MediaContentDirections (таблиця 3.3) — проміжна таблиця для зв’язку між MediaContent і Directions.

Таблиця 3.3 – MediaContentDirections

Поля	**Тип даних**	**Обмеження**	**Призначення**
content_id	BIGINT UNSIGNED	NOT NULL, PRIMARY KEY, FOREIGN KEY	Ідентифікатор контенту (посилання на MediaContent.content_id).
direction_id	BIGINT UNSIGNED	NOT NULL, PRIMARY KEY, FOREIGN KEY	Ідентифікатор напряму (посилання на Directions.direction_id).

Таблиця PopularityMetrics призначена для зберігання метрик популярності контенту. Вона містить дані про лайки, перегляди та коментарі, пов’язані з медіаконтентом. Структура таблиці, описана в таблиці 3.4, дозволяє ефективно відстежувати та аналізувати показники взаємодії користувачів із контентом.

Таблиця 3.4 – PopularityMetrics

Поля	**Тип даних**	**Обмеження**	**Призначення**
metric_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор метрики.

Продовження таблиці 3.4

content_id	BIGINT UNSIGNED	NOT NULL, FOREIGN KEY	Ідентифікатор контенту (посилання на MediaContent.content_id).
platform	VARCHAR (50)	NOT NULL	Платформа, з якої зібрано дані.
likes	INT	DEFAULT 0	Кількість лайків.
views	INT	DEFAULT 0	Кількість переглядів.
comments	INT	DEFAULT 0	Кількість коментарів.
duration	VARCHAR (8)	DEFAULT '00:00:00'	Тривалість відео у форматі HH:MM:SS.
fetch_date	DATE	NOT NULL	Дата збору метрик.

Таблиця Reports (табл. 3.5) призначена для зберігання інформації про згенеровані звіти. Вона містить дані про файли у форматах PDF або XLSX, що створюються системою.

Таблиця 3.5 – Reports

Поля	**Тип даних**	**Обмеження**	**Призначення**
report_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор.
report_name	VARCHAR (255)	NOT NULL	Назва звіту.
report_type	VARCHAR (10)	NOT NULL	Тип звіту.
file_path	VARCHAR (255)	NOT NULL	Шлях до файлу звіту.
generated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Дата та час створення звіту.

Таблиця PopularityPredictions (табл. 3.6) призначена для зберігання результатів прогнозування популярності відео. Дані, отримані за допомогою методів машинного навчання, фіксуються для подальшого аналізу.

Таблиця 3.6 – PopularityPredictions

Поля	**Тип даних**	**Обмеження**	**Призначення**
id	INT	AUTO_INCREMENT, PRIMARY KEY	Унікальний ідентифікатор.
direction_name	VARCHAR(100)	NOT NULL	Назва категорії відео
duration_range	VARCHAR(100)	NOT NULL	Тривалість відео

Продовження таблиці 3.6

predicted_views	FLOAT	NOT NULL	Прогнозована кількість переглядів відео.
prediction_date	DATETIME	NOT NULL	Дата та час створення прогнозу

3.1.3. ER діаграма

На рисунку 3.1 зображена ER діаграма сутностей відеоконтенту

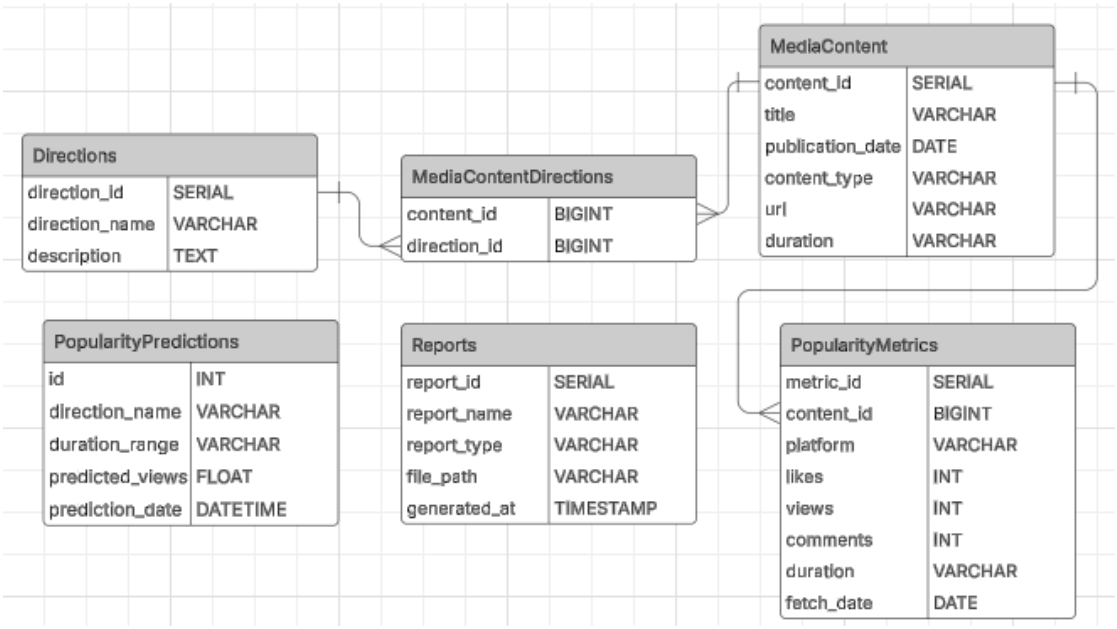


Рисунок 3.1 ER діаграма сутностей відеоконтенту

3.2. Розробка архітектури інформаційної системи

Застосунок розроблений із використанням Python, Flask, MySQL, HTML, CSS, JavaScript та YouTube API, що забезпечує низку переваг.

3.2.1. Огляд інструментів програмування та їх функціональні можливості

У процесі створення системи для обліку медіаконтенту використовувалося широке розмаиття програмних інструментів. Кожний з них мав важливе значення для функціональності системи.

Python є основною мовою програмування для серверної логіки, обробки даних і взаємодії з зовнішніми API, забезпечуючи високу читабельність коду, що спрощує розробку, підтримку та командну роботу. Завдяки підтримці широкого спектру бібліотек, таких як Pandas, Matplotlib і FPDF, Python оптимізує обробку даних, візуалізацію та генерацію звітів, а його гнучкість дозволяє швидке прототипування, що робить мову ідеальною для аналітичних систем, подібних до даної.

Flask – це веб-фреймворк, який використовується для створення серверної частини та обробки HTTP-запитів, пропонуючи легку та модульну структуру для розробки веб-додатків із мінімальною конфігурацією. Він підтримує рендеринг HTML-шаблонів через Jinja2, що дозволяє динамічно відображати дані, а також дає змогу визначати маршрути для обробки запитів користувачів. Завдяки простоті розгортання та інтеграції з базами даних і зовнішніми API, Flask ідеально підходить для невеликої аналітичної системи.

MySQL – це реляційна база даних, призначена для зберігання структурованих даних про медіаконтент, категорії, метрики та звіти, яка

підтримує складні SQL-запити, реляційні зв’язки та індексацію для підвищення продуктивності. Вона забезпечує високу надійність і цілісність даних завдяки транзакціям і зовнішнім ключам, а також має широкий набір інструментів для адміністрування, таких як phpMyAdmin або MySQL Workbench. MySQL використовується для зберігання даних у таблицях і забезпечує ефективний доступ до інформації для аналізу.

HTML – це мова розмітки, яка формує базову структуру веб-інтерфейсу, дозволяючи створювати сторінки, такі як таблиці та форми, у шаблонах на кшталт index.html чи analysis.html, а також забезпечує семантичну розмітку для покращення доступності та індексації, відображаючи дані та форми введення URL як основу користувацького інтерфейсу.

CSS, мова стилізації, відповідає за дизайн веб-інтерфейсу, налаштовуючи кольори, шрифти та розташування елементів, підтримуючи адаптивність, хоча в цій реалізації використовується мінімально, але забезпечує базову візуальну привабливість для зручного сприйняття даних.

JavaScript додає інтерактивність до веб-інтерфейсу, обробляючи події на стороні клієнта, наприклад відправку форм, і підтримує асинхронні операції для потенційного розширення функціональності. Pandas, бібліотека Python, слугує для обробки та аналізу даних, створюючи структуровані DataFrame, що дозволяють ефективно працювати з великими наборами даних і обчислювати статистичні показники.

Matplotlib, інша бібліотека Python, генерує графіки та візуалізації, такі як гістограми чи діаграми розсіювання, з можливістю експорту в PNG для відображення в HTML, покращуючи сприйняття аналітичних даних.

FPDF, бібліотека Python, створює PDF-документи, генеруючи звіти на основі даних із бази даних із можливістю кастомізації макету, наприклад таблиць.

YouTube API, зовнішній сервіс, надає доступ до метаданих відео, таких як назва, дата публікації, тривалість, а також статистичних даних, включаючи лайки, перегляди та коментарі.

3.2.2. Опис розробленого застосунку

Створений програмний комплекс призначений для аналізу медіаконтенту з YouTube, зокрема для збору, обробки та візуалізації статистичних даних про відео.

Функціонал системи дозволяє користувачам додавати відео за URL-адресою, аналізувати їх за категоріями, прогнозувати популярність відео на основі категорії та тривалості, а також генерувати звіти у форматах PDF і XLSX. Прогнозування реалізовано через модель, яка оцінює кількість переглядів і надає рекомендації щодо вибору оптимальної категорії та тривалості відео для підвищення його популярності.

Розробка додатка базується на клієнт-серверній архітектурі з використанням шаблону MVC.

3.2.3. Вибір архітектурного підходу

При розробці інформаційної системи для обліку медіаконтенту ключовим завданням було створення архітектури, яка б відповідала вимогам до ефективності, масштабованості, зручності використання та подальшого розвитку. Після аналізу можливих архітектурних підходів було обрано клієнт-серверну архітектуру, яка забезпечує оптимальну організацію взаємодії між компонентами системи та дозволяє досягти поставлених цілей. Додатково на серверній стороні використано шаблон MVC (Model-View-Controller) через фреймворк Flask, що сприяє структурованому підходу до розробки та полегшує підтримку коду.

3.2.4. Обґрунтування вибору клієнт-серверної архітектури

Клієнт-серверна архітектура є однією з найбільш поширених і перевірених часом моделей для веб-додатків, особливо тих, що потребують централізованого управління даними та доступу з різних пристроїв. У контексті системи обліку медіаконтенту ця архітектура була обрана з таких причин.

1. Централізована обробка даних на сервері. Усі обчислення, обробка даних і взаємодія з базою даних виконуються на сервері, що дозволяє зменшити навантаження на клієнтські пристрої. Наприклад, аналіз статистики YouTube-відео (лайки, коментарі, перегляди) та генерація графіків відбуваються на сервері, а клієнт отримує лише готовий результат у вигляді HTML-сторінки з таблицями та зображеннями. Централізація даних забезпечує єдине джерело правди, що спрощує управління інформацією та унеможливлює її дублювання чи неузгодженість.
2. Доступ до системи через веб-інтерфейс із будь-якого пристрою. Клієнтська частина системи реалізована як веб-додаток, що дозволяє користувачам отримувати доступ до функціоналу через браузер із будь-якого пристрою – комп’ютера, планшета чи смартфона – за наявності підключення до Інтернету. Це усуває необхідність встановлення додаткового програмного забезпечення на стороні користувача. Такий підхід відповідає сучасним тенденціям у розробці програмного забезпечення, де пріоритет віддається кросплатформним рішенням. Наприклад, користувач може ввести URL-адресу YouTube-відео та переглянути аналітичні дані прямо у браузері, незалежно від операційної системи чи типу пристрою.
3. Спрощення оновлення та масштабування системи. У клієнт-серверній архітектурі оновлення системи відбувається на сервері, що не

потребує змін на стороні клієнта. Користувачам достатньо просто оновити сторінку у браузері, щоб отримати доступ до нової версії додатку.

4. Гнучкість і модульність. Клієнт-серверна модель дозволяє чітко розділити відповідальність між фронтендом і бекендом. Фронтенд відповідає за відображення даних і взаємодію з користувачем, тоді як бекенд займається обробкою запитів, бізнес-логікою та інтеграцією з зовнішніми сервісами. Такий розподіл сприяє модульності системи: кожен компонент може розроблятися, тестуватися та масштабуватися незалежно від інших.

3.2.5. Використання шаблону MVC на серверній стороні

Для організації серверної частини системи було обрано шаблон MVC (Model-View-Controller), реалізований через фреймворк Flask. Цей шаблон забезпечує структурованість коду, розділяючи його на три основні компоненти, що полегшує розробку, тестування та підтримку системи.

Model (Модель). Модель відповідає за управління даними та їх логікою, включаючи взаємодію з базою даних MySQL. Усі операції з даними – збереження, оновлення, видалення, вибірка – реалізуються через модель.

У системі модель представлена функціями та SQL-запитами, які взаємодіють із базою даних через модуль "mysql-connector-python". Наприклад, функція "get_db_connection()" у "main.py" встановлює з’єднання з MySQL, а запити на кшталт "INSERT INTO MediaContent" або "SELECT" для аналізу даних виконуються у відповідних маршрутах Flask.

Таблиці бази даних ("MediaContent", "Directions", "PopularityMetrics" тощо) є частиною моделі, а їх структура визначається під час ініціалізації БД у функції "init_db()".

View (Представлення). Представлення відповідає за відображення даних користувачу. У даній системі це HTML-шаблони, які генеруються Flask і передаються клієнту у вигляді веб-сторінок.

Шаблони ("index.html", "analysis.html", "compare_videos.html", "show_databases.html", "popularity_prediction.html") створені з використанням Jinja2 – шаблонізатора Flask. Вони відображають таблиці з даними (наприклад, середні лайки, перегляди), графіки (у форматі base64-зображень) та форми для введення даних (наприклад, URL-адреси відео).

Controller (Контролер). Контролер обробляє запити користувача, взаємодіє з моделлю для отримання або збереження даних і передає результати у представлення. У системі контролер реалізовано через маршрути Flask.

Кожен маршрут у "main.py" (наприклад, "@app.route('/submit_url', methods=['POST'])", "@app.route('/analyze_directions')") є частиною контролера. Ці маршрути обробляють запити, викликають функції моделі для роботи з БД, обробляють дані (наприклад, за допомогою Pandas) і повертають результат у вигляді HTML-шаблону через "render_template".

Наприклад, маршрут "/submit_url" приймає URL-адресу відео, викликає YouTube API для отримання метаданих, зберігає дані в БД і повертає повідомлення про успішну обробку.

Переваги використання MVC у Flask

Наведемо таблицю 3.7 для відображення переваг використання MVC у Flask.

Таблиця 3.7 – Переваги використання MVC у Flask

Переваги	Опис
Структурованість коду	Розділення на модель, представлення та контролер робить код більш організованим і легким для розуміння.
Легкість масштабування	Нові функції можна додати, створивши нові маршрути та шаблони, не змінюючи основну структуру.
Підтримка та модифікація	Завдяки чіткому розділенню компонентів розробник може швидко знайти потрібний код для зміни
Повторне використання коду	Функції моделі використовуються в різних маршрутах, що зменшує дублювання коду.

3.3. Розробка інтерфейсу користувача: вимоги та реалізація

Інтерфейс системи спроєктовано таким чином, щоб забезпечити зручність для користувача, легку та інтуїтивно зрозумілу навігацію, а також наочне й ефективне представлення аналітичної інформації.

3.3.1. Вимоги до інтерфейсу користувача

Визначимо основні вимоги.

1. Зручність і простота використання. Інтерфейс має бути інтуїтивно зрозумілим для користувачів із різним рівнем технічних навичок, дозволяючи легко додавати URL-адреси відео, вибирати категорії та переглядати аналітичні дані. Форми введення та кнопки повинні бути чітко позначені й доступні без зайвих кроків.
2. Адаптивність. UI має коректно відображатися на різних пристроях (комп’ютери, планшети, смартфони).
3. Доступність даних. Система повинна надавати користувачу доступ до ключових даних: таблиць із статистикою (лайки, коментарі, перегляди), графіків для візуалізації (гістограми, діаграми розсіювання) та вмісту бази даних. Інформація має оновлюватися динамічно на основі даних із бази даних MySQL.
4. Функціональність. Інтерфейс має підтримувати:

- введення URL-адреси YouTube-відео та вибір категорій для їх класифікації;
- аналіз даних за категоріями (середні значення, графіки);
- порівняння відео в межах однієї категорії з додатковими графіками;
- перегляд структури бази даних;
- генерацію звітів.

5. Надійність і зворотний зв’язок. Система має повідомляти про помилки і успішне виконання дій.Навігація має включати можливість повернення на головну сторінку.

3.3.2. Реалізація інтерфейсу

3.3.2.1 index.html

Файл "index.html" виступає основним шаблоном головної сторінки, який є ключовим елементом взаємодії користувача з додатком. На малюнку 3.2 відібдражається основний інтерфейс.

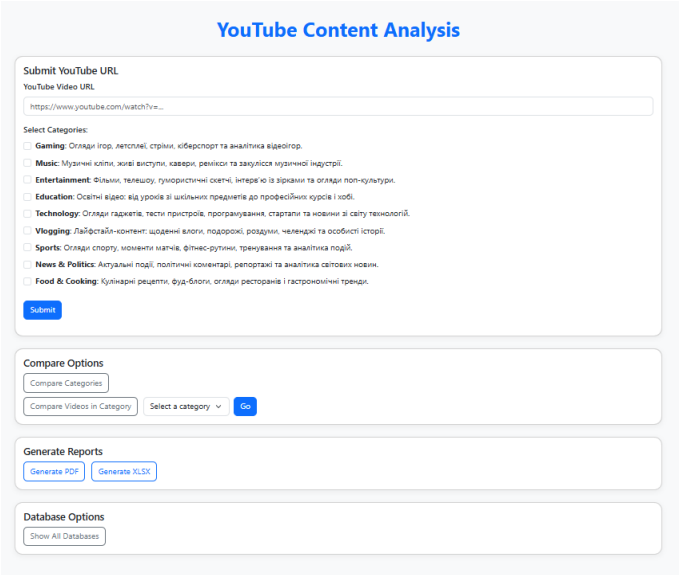


Рисунок 3.2 Основний інтерфейс програми

Ця сторінка створена за допомогою HTML із застосуванням шаблонізатора Jinja2 для динамічного рендерингу вмісту, що дозволяє адаптувати інтерфейс до даних, отриманих із серверної частини. Тут користувач може вводити URL-адреси YouTube-відео у спеціальну текстову форму, позначену як "<input type="text" name="youtube_url">", а також обирати відповідні категорії через набір чекбоксів, які генеруються динамічно на основі списку, отриманого з таблиці "Directions" у базі даних. Цей список формуються через цикл "{% for category in categories %}", що забезпечує гнучкість у разі додавання нових категорій. Окрім того, сторінка включає додаткові секції: одна з них призначена для порівняння відео за категоріями,

де користувач може обрати категорію з випадаючого списку чи чекбоксів і перейти до детального аналізу через кнопку "Go", що активує маршрут "/compare_videos_in_category", а інша секція пропонує посилання "Show All Databases" для переходу до перегляду вмісту бази даних, доступного за маршрутом "/show_databases". Уся структура сторінки чітко розподілена на блоки, включаючи заголовок "YouTube Content Analysis" і кнопку "Back to Home" для зручної навігації, що робить її центральним хабом для всіх операцій користувача. У разі помилок, таких як відсутність доступних категорій, відображається повідомлення "No categories available...", що забезпечує зворотний зв’язок.

3.3.2.2 analysis.html

Сторінка "analysis.html" призначена для відображення аналітичних даних, отриманих у результаті обробки сервером, і є логічним продовженням аналізу, виконаного через маршрут "/analyze_directions". На малюнку 3.3 представлено сторінку "Direction Analysis".

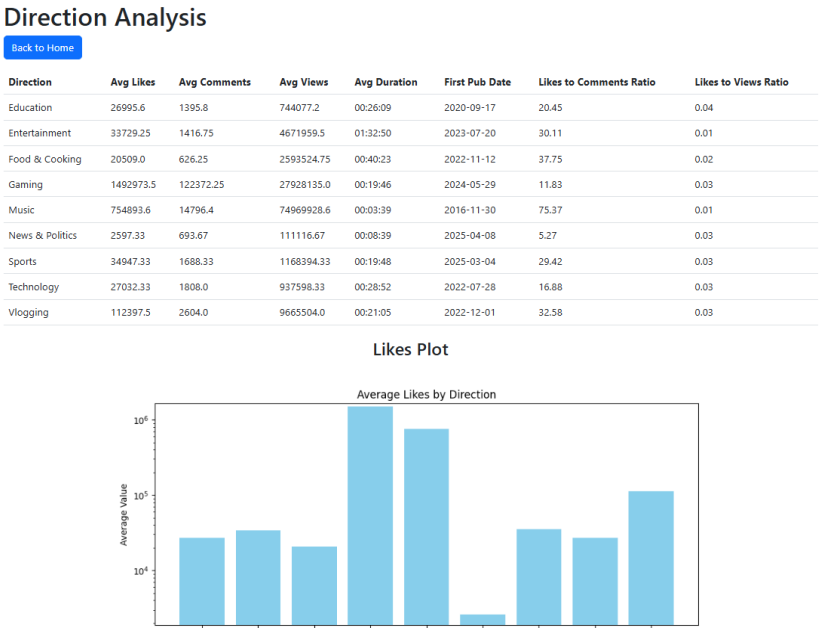
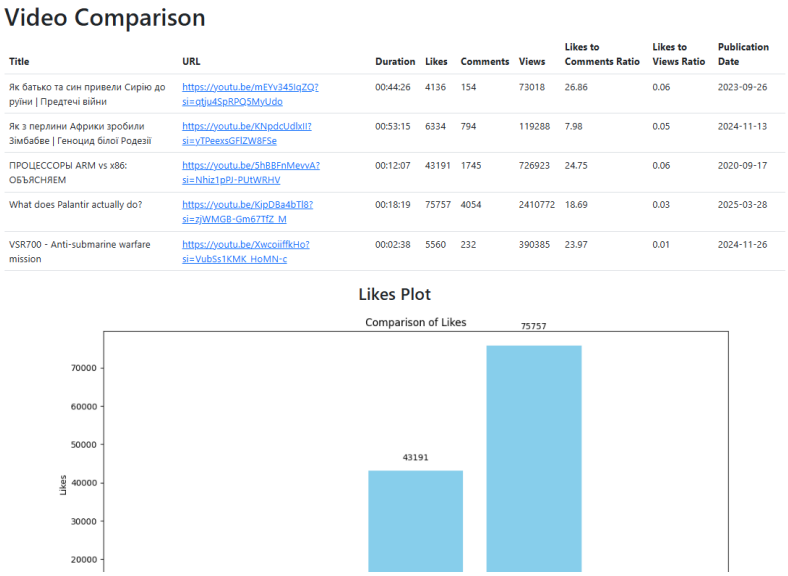


Рисунок 3.3 Сторінка "analysis.html"

Вона побудована на HTML із використанням Jinja2, що дозволяє рендерити таблицю з колонками, такими як Direction, Avg Likes, Avg Comments, Avg Views, Avg Duration, First Pub Date, Likes to Comments Ratio і Likes to Views Ratio, де дані виводяться через цикл "{% for item in data %}" із округленням значень до двох знаків після коми ("|round(2)"). Ця таблиця надає користувачу огляд середніх показників за кожною категорією, що базується на даних із таблиць "MediaContent" і "PopularityMetrics". Окрім таблиці, сторінка відображає набір графіків, таких як Likes Plot, Comments Plot і Views Plot, які генеруються на сервері за допомогою бібліотеки Matplotlib. Графіки представлені у вигляді гістограм, що візуалізують розподіл середніх значень, що робить аналіз більш наочним. Якщо дані відсутні, наприклад, через порожню базу даних, відображається повідомлення про помилку "{{ error }}", таке як "No data available", а присутність кнопки "Back to Home" забезпечує можливість повернутися до головного екрану, що підвищує зручність використання.

3.3.2.3 compare_videos.html

Файл "compare_videos.html" фокусується на детальному порівнянні відео в межах однієї обраної категорії й також виконаний на HTML із Jinja2. На малюнку 3.4 представлено сторінку "Video Comparison".



Video Comparison

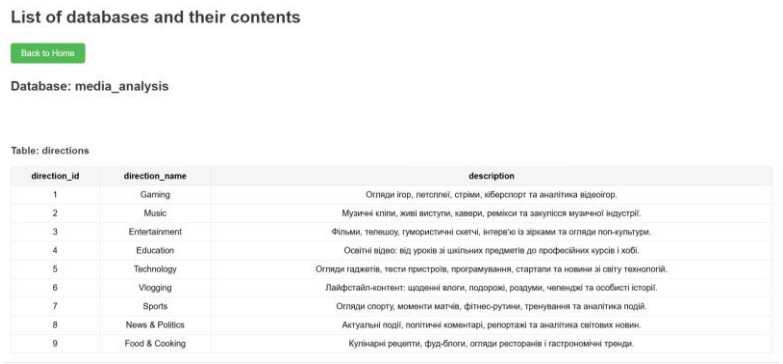
Title	URL	Duration	Likes	Comments	Views	Likes to Comments Ratio	Likes to Views Ratio	Publication Date
Як батько та син привели Сирію до руїни | Предтечі війни	https://youtu.be/mEYv345IqZQ?si=qtju4SpRPQ5MyUdo	00:44:26	4136	154	73018	26.86	0.06	2023-09-26
Як з перлини Африки зробили Зімбабве | Геноцид білої Родезії	https://youtu.be/KNpdcUdlxII?si=vTPeexsGFlZW8FSe	00:53:15	6334	794	119288	7.98	0.05	2024-11-13
ПРОЦЕССОРЫ ARM vs x86: ОБЪЯСНЯЕМ	https://youtu.be/5hB8FnMevvA?si=Nhiz1pPJ-PUtWRHV	00:12:07	43191	1745	726923	24.75	0.06	2020-09-17
What does Palantir actually do?	https://youtu.be/KipDBa4bTI8?si=zjWMGB-Gm67TfZ_M	00:18:19	75757	4054	2410772	18.69	0.03	2025-03-28
VSR700 - Anti-submarine warfare mission	https://youtu.be/XwcoiiffkHo?si=VubSs1KMK_HoMN-c	00:02:38	5560	232	390385	23.97	0.01	2024-11-26

Рисунок 3.4 Сторінка "compare_videos.html"

Ця сторінка відображає таблицю з детальною статистикою для кожного відео, включаючи колонки Title, URL, Duration, Likes, Comments, Views, Likes to Comments Ratio, Likes to Views Ratio і Publication Date, де дані рендеряться через цикл "{% for item in data %}" із відповідним форматуванням співвідношень. Цей модуль активується через маршрут "/compare_videos_in_category" і дозволяє користувачу глибше зануритися в аналіз контенту, отримавши доступ до індивідуальних показників кожного відео в категорії. Крім таблиці, сторінка включає набір графіків: гістограми для Likes, Comments і Views, а також діаграми розсіювання, такі як Duration vs Likes і Duration vs Comments, які генеруються сервером і вставляються як зображення через умовні блоки ("{% if duration_likes_plot %}"). Ці графіки допомагають виявити кореляції між тривалістю й популярністю, додаючи аналітичну глибину. У разі відсутності відео в категорії відображається відповідне повідомлення про помилку, а кнопка "Back to Home" забезпечує зручний вихід.

3.3.2.4 show_databases.html

Файл "show_databases.html" дозволяє переглядати вміст бази даних у зрозумілому форматі. На малюнку 3.5 представлено сторінку для відображення баз даних .


List of databases and their contents

Back to Home

Database: media_analysis

Table: directions

direction_id	direction_name	description
1	Gaming	Огляди ігор, летсплеї, стріми, кіберспорт та аналітика відеоігор.
2	Music	Музичні кліпи, живі виступи, кавери, ремікси та закуліса музичної індустрії.
3	Entertainment	Фільми, телешоу, гумористичні скетчі, інтерв'ю із зірками та огляди поп-культури.
4	Education	Освітні відео: від уроків зі шкільних предметів до професійних курсів і хобі.
5	Technology	Огляди гаджетів, тести пристроїв, програмування, стартапи та новини зі світу технологій.
6	Vlogging	Лайфстайл-контент: щоденні влоги, подорожі, роздуми, челенджі та особисті історії.
7	Sports	Огляди спорту, моменти матчів, фітнес-рутини, тренування та аналітика подій.
8	News & Politics	Актуальні події, політичні коментарі, репортажі та аналітика світових новин.
9	Food & Cooking	Кулінарні рецепти, фуд-блоги, огляди ресторанів і гастрономічні тренди.

Рисунок 3.5 show_databases.html

Написаний на HTML із Jinja2, він відображає ієрархічну структуру даних, де назва бази даних "media_analysis" розкривається в список таблиць, а кожна таблиця показує свої колонки й рядки через вкладені цикли "{% for db, tables in db_data.items() %}" і "{% for row in data.rows %}". Цей підхід дозволяє детально розібрати структуру таблиць, таких як "MediaContent" чи "PopularityMetrics", і переглянути їхній вміст, що є корисним для перевірки цілісності даних. Якщо якась таблиця порожня, відображається повідомлення "No data in the table". Сторінка також включає кнопку "Back to Home" для повернення до початкового екрану, що завершує її функціональність і робить її зручним інструментом для моніторингу.

3.3.2.5 popularity_prediction.html

Файл popularity_prediction.html відповідає за відображення прогнозу популярності відео на основі категорії та тривалості. На малюнку 3.6 представлено сторінку "Popularity Prediction".

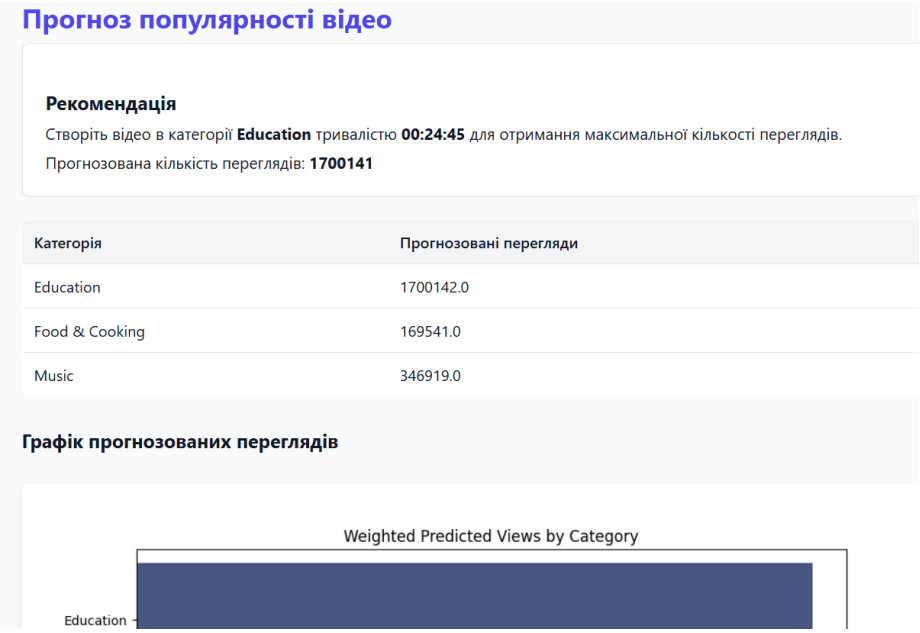


Рисунок 3.6 popularity_prediction.html

Розроблений на HTML із використанням Jinja2, цей шаблон відображає результати прогнозування, отримані від серверної частини через маршрут /predict_popularity. Сторінка містить рекомендацію щодо оптимальної категорії (top_category_name) та тривалості (top_duration_range) для максимізації переглядів, а також прогнозовану кількість переглядів (top_predicted_views). Дані презентуються в таблиці з двома колонками: "Категорія" та "Прогнозовані перегляди", що генерується через цикл {% for item in data %}. У разі відсутності даних відображається повідомлення про помилку.

Додатково сторінка включає графік прогнозованих переглядів, який генерується сервером за допомогою Matplotlib і вставляється як base64-зображення через умовний блок {% if plot_img %}. Цей графік допомагає

візуально оцінити популярність різних категорій. Кнопка "Back to Home" забезпечує повернення на головну сторінку, підвищуючи зручність навігації.

Шаблон інтегрується з моделлю машинного навчання (RandomForestRegressor), яка аналізує метадані відео (лайки, коментарі, тривалість) із таблиці PopularityPredictions у базі даних MySQL. Використання бібліотеки unidecode гарантує коректну обробку текстових даних. Цей модуль є ефективним інструментом для творців контенту, надаючи аналітичні прогнози та візуалізації для оптимізації популярності відео.

3.4. Опис основних функціональних модулів системи

3.4.1. Модуль збору даних із YouTube

Відповідає за отримання метаданих і статистики YouTube-відео за введеним URL-адресою та збереження їх у базі даних.

Користувач вводить URL-адресу відео через форму на головній сторінці (index.html).

Маршрут /submit_url у main.py обробляє POST-запит, витягує video_id із URL за допомогою urlparse і parse_qs.

Клас YouTubeDataAPI з модуля youtube_api.py викликається для роботи з даними YouTube: метод get_video_metadata(video_id) отримує метадані, такі як назва, дата публікації та тривалість, причому тривалість конвертується з ISO-формату в "hh:mm:ss" за допомогою бібліотеки isodate, а метод get_video_stats(video_id) повертає статистику, включаючи лайки, коментарі та перегляди

Отримані дані (назва, дата, тривалість, URL, лайки, коментарі, перегляди) зберігаються в таблицях MediaContent, MediaContentDirections (для зв’язку з категоріями) і PopularityMetrics.

Алгоритм:

1) отримуємо video_id із URL;
2) викликаємо YouTube API для метаданих і статистики;
3) конвертуємо дату публікації в формат DATE і тривалість у "hh:mm:ss";
4) виконуємо SQL-запити для збереження даних у відповідні таблиці;
5) повертаємо JSON-відповідь із повідомленням про успіх.

3.4.2. Модуль управління контентом

Забезпечує управління медіаконтентом, зокрема збереження, оновлення та асоціацію відео з категоріями.

Реалізовано в рамках маршруту /submit_url у main.py.

Після отримання даних із YouTube модуль зберігає відео в таблиці MediaContent, фіксуючи назву, дату публікації, тип контенту, URL і тривалість, а також асоціює відео з обраними категоріями через проміжну таблицю MediaContentDirections. Статистику, таку як лайки, перегляди, коментарі, платформа та дата збору, зберігає в таблиці PopularityMetrics. Для ініціалізації бази даних із попередньо визначеними категоріями, такими як Gaming, Music тощо, використовується функція init_db().

Алгоритм:

1) ініціалізуємо БД із таблицями та категоріями (init_db());
2) отримуємо дані відео через модуль збору даних;
3) виконуємо SQL-запит INSERT INTO MediaContent для збереження відео;

4) для кожної обраної категорії додаємо запис у MediaContentDirections;
5) зберігаємо статистику в PopularityMetrics.

3.4.3. Модуль аналізу даних за категоріями

Аналізує статистику відео за категоріями та генерує середні значення й графіки.

Реалізовано в маршруті /analyze_directions у main.py.

Виконує SQL-запит для об’єднання таблиць MediaContent, MediaContentDirections, Directions і PopularityMetrics, щоб отримати дані про відео та їхні категорії.

Використовує Pandas для обчислення середніх значень (лайки, коментарі, перегляди, тривалість) і співвідношень (Likes to Comments, Likes to Views).

Генерує графіки за допомогою Matplotlib (гістограми для середніх значень і співвідношень).

Рендерить результати в шаблон analysis.html.

Алгоритм:

1) виконуємо SQL-запит для вибірки даних;
2) групуємо дані за категоріями (direction_name) у словник direction_data;
3) обчислюємо середні значення та співвідношення за допомогою Pandas;
4) конвертуємо тривалість із "hh:mm:ss" у секунди для аналізу (time_to_seconds);
5) генеруємо гістограми для кожної метрики (Likes, Comments, Views тощо);
6) передаємо дані та графіки (як base64-зображення) у шаблон.

3.4.4. Модуль порівняння відео

Дозволяє порівнювати відео в межах однієї категорії, показуючи детальну статистику та графіки.

Реалізовано в маршруті /compare_videos_in_category у main.py.

Отримує параметр direction_id (ID категорії) із запиту.

Виконує SQL-запит для вибірки всіх відео обраної категорії з їхньою статистикою.

Обчислює співвідношення (Likes to Comments, Likes to Views) для кожного відео.

Генерує гістограми для кожної метрики та діаграми розсіювання (Duration vs Likes, Duration vs Comments).

Рендерить результати в шаблон compare_videos.html.

Алгоритм:

1) отримуємо direction_id із параметрів запиту;
2) виконуємо SQL-запит для вибірки відео за категорією;
3) створюємо DataFrame (Pandas) для обробки даних;
4) обчислюємо співвідношення для кожного відео;
5) генеруємо гістограми та діаграми розсіювання (Matplotlib);
6) передаємо дані та графіки в шаблон.

3.4.5. Модуль генерації звітів

Генерує звіти у форматах PDF і XLSX на основі даних із бази даних.

Реалізовано в маршруті /generate_report/<report_type> у main.py.

Виконує SQL-запит для об’єднання таблиць і вибірки даних (назва відео, категорії, тривалість, лайки, коментарі, перегляди).

Формує звіт.

1. Для PDF використовує FPDF для створення таблиці.

2. Для XLSX використовує Pandas (to_excel).

Зберігає файл у папці reports і додає запис про звіт у таблицю Reports.

Алгоритм:

1) виконуємо SQL-запит для вибірки даних;
2) створюємо DataFrame із результатами;
3) генеруємо унікальний ID звіту та назву файлу (report_<timestamp>);
4) для PDF створюємо таблицю через FPDF. Для XLSX експортуємо DataFrame у файл;
5) зберігаємо шлях до файлу в таблиці Reports;
6) відправляємо файл користувачу через send_file.

3.4.6. Модуль перегляду бази даних

Дає доступ до вмісту бази даних (таблиці, рядки, колонки).

Реалізовано в маршруті /show_databases у main.py.

Виконує запит SHOW TABLES для отримання списку таблиць у БД media_analysis.

Для кожної таблиці виконує SELECT * для вибірки даних і SHOW COLUMNS для назв колонок.

Рендерить результати в шаблон show_databases.html.

Алгоритм:

1) отримуємо список таблиць із БД;
2) для кожної таблиці отримуємо колонки та рядки;
3) формуємо словник із даними (db_data);
4) передаємо дані в шаблон для відображення.

3.4.7. Модуль прогнозування популярності відео

Прогнозує популярність відео на основі категорії та тривалості, використовуючи модель машинного навчання.

Реалізовано в маршруті /predict_popularity у main.py.

Виконує обробку даних із таблиці MediaContent і PopularityMetrics, отримуючи метадані відео (лайки, коментарі, тривалість, категорія). Для прогнозування переглядів використовується модель RandomForestRegressor. Результати формуються у вигляді таблиці з прогнозованими переглядами для кожної категорії, рекомендації оптимальної категорії та тривалості для максимізації переглядів, а також графіка прогнозованих переглядів, згенерованого за допомогою Matplotlib і Seaborn. Отримані результати прогнозування зберігаються в таблиці PopularityPredictions.

Алгоритм:

1) виконуємо SQL-запит для вибірки метаданих відео з таблиць MediaContent і PopularityMetrics;
2) вбробляємо дані за допомогою Pandas, створюючи DataFrame;
3) використовуємо unidecode для нормалізації текстових даних (назви категорій);
4) навчаємо або використовуємо збережену модель RandomForestRegressor для прогнозування переглядів;
5) формуємо список категорій із прогнозованими переглядами та визначаємо оптимальну категорію і тривалість;
6) генеруємо графік прогнозованих переглядів і конвертуємо його в зображення;
7) зберігаємо результати в таблиці PopularityPredictions;
8) передаємо дані (таблиця, рекомендація, графік) у шаблон popularity_prediction.html для відображення.

3.5. Реалізація та тестування програмної системи

Розроблений програмний комплекс для обліку медіаконтенту, який базується на аналізі YouTube-відео, являє собою зручний інструмент для збору, обробки та візуалізації даних, що дозволяє користувачам легко додавати відео за URL-адресами, аналізувати їх за категоріями та отримувати звіти у форматах PDF і XLSX. Система побудована на клієнт-серверній архітектурі з використанням шаблону MVC, де серверна частина реалізується через Python і фреймворк Flask, а взаємодія з користувачем забезпечується через веб-інтерфейс, створений за допомогою HTML, CSS і JavaScript із динамічним рендерингом через Jinja2. Для зберігання даних використовується реляційна база даних MySQL, а інтеграція з YouTube відбувається через YouTube Data API, що дає змогу отримувати актуальні метадани та статистику відео. Кожен файл коду відіграє свою унікальну роль у функціонуванні системи.

3.5.1. main.py

Почнемо з main.py, який є серцем серверної логіки, написаним на Python із використанням Flask. Цей файл обробляє всі HTTP-запити, взаємодіє з базою даних, інтегрується з YouTube API, аналізує дані та генерує графіки й звіти.

Функція get_db_connection, яка встановлює з’єднання з MySQL через конфігурацію db_config, забезпечуючи доступ до даних для всіх операцій.

Функція time_to_seconds конвертує тривалість відео з формату "hh:mm:ss" у секунди, обробляючи порожні значення або помилки, що

корисно для подальшого аналізу, тоді як seconds_to_time робить зворотне перетворення, повертаючи зручний для користувача формат.

Ініціалізація бази даних відбувається через init_db, яка створює таблиці, такі як Directions, MediaContent, PopularityMetrics тощо, і заповнює категорії на старті.

Функція index рендерить головну сторінку, отримуючи список категорій із Directions, а get_categories повертає їх у JSON-форматі для динамічного оновлення.

Обробка введеного URL здійснюється через submit_url, яка витягує video_id, викликає YouTube API для метаданих і статистики, а потім зберігає дані в БД, повертаючи повідомлення про успіх.

Аналіз даних за категоріями виконує analyze_directions, обчислюючи середні значення й генеруючи гістограми через Pandas і Matplotlib, тоді як compare_videos_in_category порівнює відео в категорії з додатковими графіками.

Генерація звітів реалізована в generate_report, яка створює PDF через FPDF або XLSX через Pandas, а show_databases відображає вміст БД для адміністраторів.

3.5.2. youtube_api.py

Модуль для інтеграції з YouTube Data API, реалізований у файлі youtube_api.py, є важливим компонентом системи, що забезпечує отримання актуальних даних про відео з YouTube для подальшого аналізу та збереження. Написаний на мові Python, цей модуль використовує бібліотеки googleapiclient для роботи з API YouTube і isodate для обробки тривалості відео. Його функціонал дозволяє системі отримувати як мета-дані, так і статистику відео, що є основою для всіх аналітичних операцій у системі.

Клас YouTubeDataAPI, який створюється для зручної інкапсуляції роботи з API. Конструктор __init__ цього класу приймає API-ключ як параметр і ініціалізує клієнтський об’єкт для взаємодії з YouTube Data API v3. Для цього використовується функція build із бібліотеки googleapiclient.discovery, яка створює об’єкт youtube із параметрами serviceName="youtube", version="v3" і переданим ключем. Цей об’єкт зберігається в змінній self.youtube і використовується для всіх подальших запитів до API. Такий підхід дозволяє централізовано налаштувати автентифікацію та повторно використовувати клієнт у всіх методах класу, що робить код більш структурованим і ефективним.

Перша ключова функція модуля - get_video_metadata - відповідає за отримання метаданих відео за його ідентифікатором (video_id). Вона формує запит до API через метод videos().list, передаючи параметри part="snippet,contentDetails" і id=video_id. Параметр snippet дозволяє отримати основну інформацію, таку як назва відео (title), дата публікації (publishedAt) і опис, а contentDetails - тривалість відео у форматі ISO 8601 (наприклад, "PT1H30M"). Отримавши відповідь, функція обробляє дані: витягує словник snippet із інформацією про відео, а тривалість конвертує з ISO-формату у зручний для користувача формат "hh:mm:ss" за допомогою бібліотеки isodate. Наприклад, тривалість "PT1H30M" перетворюється на "01:30:00". Якщо відео не знайдено або виникає помилка, функція повертає None, що дозволяє обробляти такі випадки на вищому рівні в main.py. Ця функція є основою для збереження базової інформації про відео в таблиці MediaContent, адже без неї система не могла б отримати ключові дані, такі як назва чи дата публікації.

Друга функція, get_video_stats, відповідає за отримання статистичних даних про відео, що є критично важливим для аналітичних функцій системи. Вона також використовує метод videos().list, але з параметром part="statistics", щоб отримати інформацію про кількість лайків (likeCount), коментарів (commentCount) і переглядів (viewCount). Функція повертає словник із цими значеннями, причому якщо певна статистика недоступна (наприклад, автор

відео заборонив показ лайків), вона повертає значення за замовчуванням ("0"), що запобігає помилкам під час збереження даних у таблиці PopularityMetrics. Ці статистичні дані є основою для аналізу популярності відео, наприклад, для обчислення співвідношень лайків до коментарів чи переглядів, які відображаються в шаблонах analysis.html і compare_videos.html.

3.5.3 HTML-шаблони

Інтерфейс користувача реалізовано за допомогою веб-технологій (HTML, CSS, JavaScript) із використанням шаблонів Flask (Jinja2) для динамічного рендерингу даних.

index.html є шаблоном головної сторінки, де користувач вводить URL і вибирає категорії через форму, а також має секції для порівняння відео й перегляду бази даних, усе це рендериться динамічно через Jinja2. Сторінка analysis.html відображає аналітичні дані в таблиці з середніми значеннями й графіками, генерованими сервером, а compare_videos.html показує детальну статистику відео в категорії з гістограмами й діаграмами розсіювання. Нарешті, show_databases.html дозволяє переглядати вміст бази даних у зрозумілому форматі, відображаючи таблиці й рядки для адміністраторів, а popularity_prediction.html відповідає за відображення прогнозу популярності відео, презентуючи таблицю з прогнозованими переглядами для кожної категорії, рекомендацію оптимальної категорії та тривалості, а також графік, що наочно ілюструє прогнозовані перегляди.

3.5.4. Тестування розробленого застосунку

Буде протестовано такі речі:

1) додавання YouTube-відео через URL;
2) аналіз даних за категоріями з відображенням середніх значень і графіків;
3) порівняння відео в категорії з таблицею і графіками;
4) генерацію звітів у форматах PDF і XLSX із правильним збереженням у Reports;
5) перегляд вмісту бази даних із коректним відображенням таблиць і даних;
6) обробку помилок із відповідними повідомленнями;
7) прогнозування популярності відео з відображенням результатів.

3.5.4.1. Додавання YouTube-відео через URL

Одним із ключових аспектів функціонування розробленої інформаційної системи є додавання YouTube-відео через введення URL-адреси, що забезпечує інтеграцію контенту в систему з подальшим коректним збереженням його метаданих і статистичних даних у відповідних таблицях бази даних.

Наприклад, розглянемо процес додавання відео за посиланням https://youtu.be/jLfyRW6x-BM?si=BImF-k1MQGlunmMi, яке є прикладом музичного міксу "Deep & Groovy Smooth Jazz House DJ Mix - Cozy Sundown Vibes" від каналу RE-TAPE Studios. На малюнку 3.7 приклад відео на платформі YouTube.

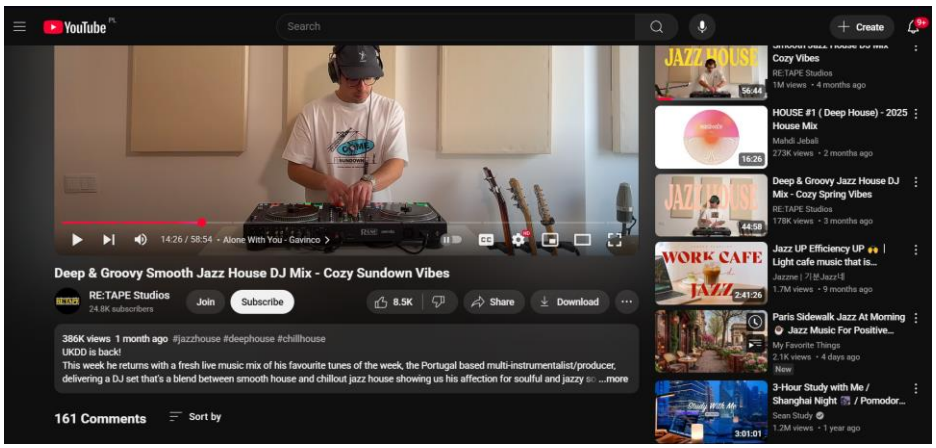


Рисунок 3.7 Приклад відео на платформі YouTube

Користувач вводить цей URL на головній сторінці системи через спеціальну форму (рисунок 3.8), обираючи відповідні категорії, після чого система обробляє запит.

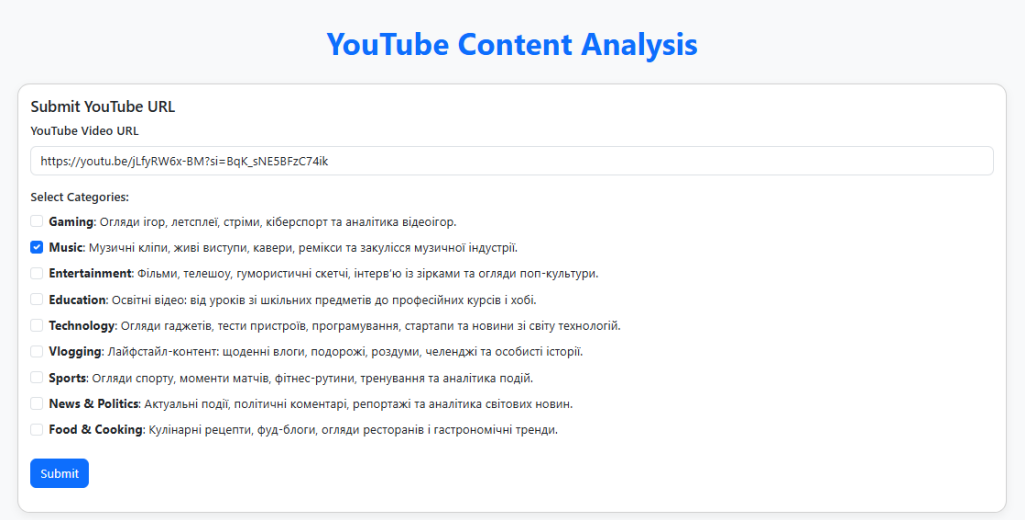


Рисунок 3.8 Приклад введення посилання на головній сторінці

Спочатку сервер витягує унікальний ідентифікатор відео ("video_id = jLfyRW6x-BM") з URL за допомогою бібліотеки "urlparse", а потім звертається до YouTube Data API для отримання метаданих, включаючи назву відео, дату публікації, тривалість (14:26) і опис контенту. Паралельно API повертає статистику, таку як кількість переглядів , лайків і коментарів, які відображаються на сторінці відео. Ці дані зберігаються в таблиці "MediaContent" (назва, дата, URL, тривалість), таблиці

"MediaContentDirections" (зв’язок із обраними категоріями), і таблиці "PopularityMetrics". Такий процес забезпечує повне відображення контенту в системі, дозволяючи подальший аналіз і візуалізацію, а коректність збереження даних підтверджується відповідними SQL-запитами та транзакціями в базі даних MySQL.

3.5.4.2. Аналіз даних за категоріями з відображенням середніх значень і графіків

Функціонал аналізу даних за категоріями дозволяє користувачу отримати узагальнену статистику та візуалізацію популярності відео. Обираючи опцію "Compare Categories", приклад на малюнку 3.9, користувач потрапляє на нову сторінку.

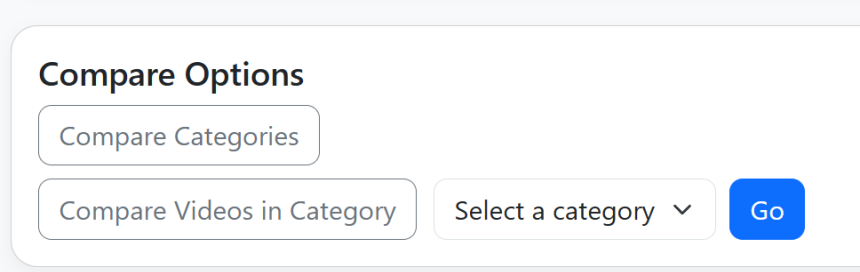
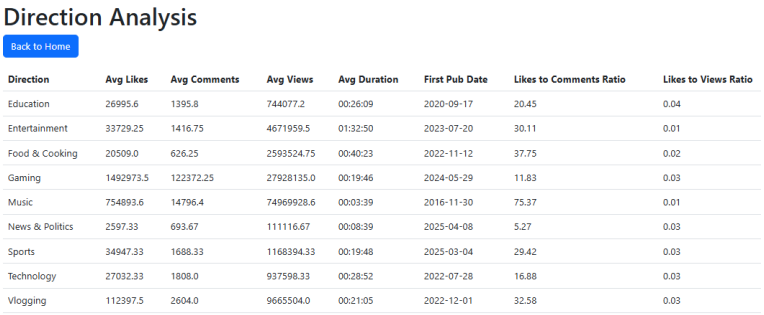


Рисунок 3.9 Вибір опції "Compare Categories"

Система об’єднує дані з таблиць "MediaContent", "MediaContentDirections", "Directions" і "PopularityMetrics", групує їх за категоріями за допомогою Pandas і обчислює середні значення лайків, коментарів, переглядів, тривалості, а також співвідношення, наприклад, лайків до коментарів. Для візуалізації генеруються гістограми через Matplotlib, які відображаються в шаблоні "analysis.html" разом із таблицею, що містить середні показники, допомагаючи швидко оцінити тенденції популярності контенту в кожній категорії. Приклади на малюнках 3.10 та 3.11.



Direction Analysis

Back to Home

Direction	Avg Likes	Avg Comments	Avg Views	Avg Duration	First Pub Date	Likes to Comments Ratio	Likes to Views Ratio
Education	26995.6	1395.8	744077.2	00:26:09	2020-09-17	20.45	0.04
Entertainment	33729.25	1416.75	4671959.5	01:32:50	2023-07-20	30.11	0.01
Food & Cooking	20509.0	626.25	2593524.75	00:40:23	2022-11-12	37.75	0.02
Gaming	1492973.5	122372.25	27928135.0	00:19:46	2024-05-29	11.83	0.03
Music	754893.6	14796.4	74969928.6	00:03:39	2016-11-30	75.37	0.01
News & Politics	2597.33	693.67	111116.67	00:08:39	2025-04-08	5.27	0.03
Sports	34947.33	1688.33	1168394.33	00:19:48	2025-03-04	29.42	0.03
Technology	27032.33	1808.0	937598.33	00:28:52	2022-07-28	16.88	0.03
Vlogging	112397.5	2604.0	9665504.0	00:21:05	2022-12-01	32.58	0.03

Рисунок 3.10 Приклад візуалізації даних

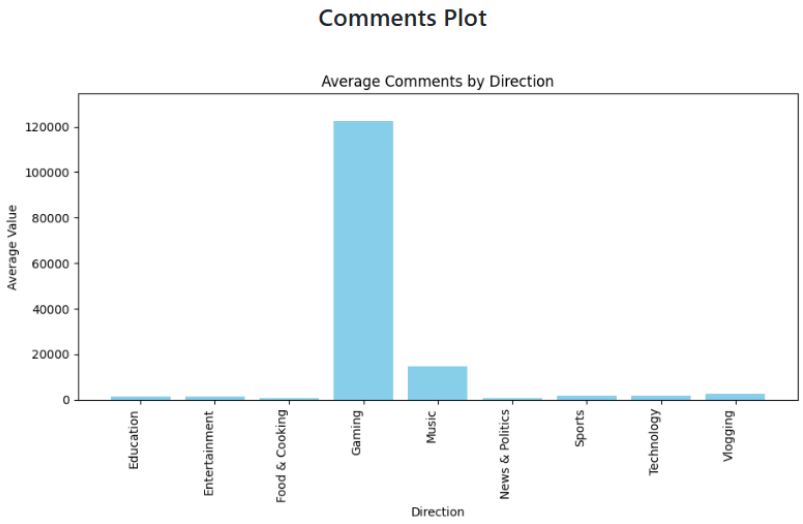


Рисунок 3.11 Приклад візуалізації даних

3.5.4.3. Порівняння відео в категорії з таблицею і графіками

Користувач обирає категорію на головній сторінці в секції "Compare Options" із випадаючого списку та натискає "Go", після чого система виконує SQL-запит до таблиць MediaContent, MediaContentDirections і PopularityMetrics, щоб зібрати дані про всі відео в цій категорії. Функціонал представлений на малюнку 3.12.

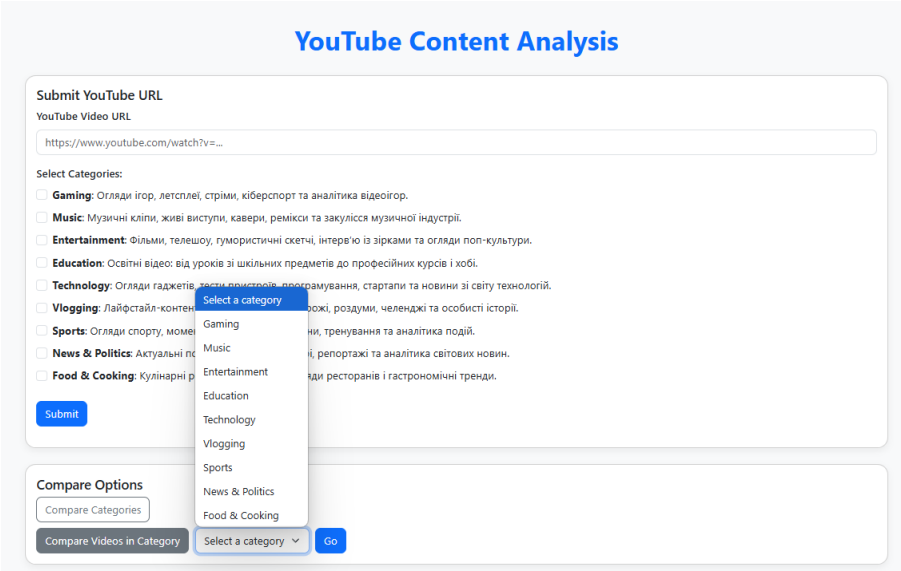


Рисунок 3.12 Приклад вибору категорії в секції "Compare Options"

Результати обробляються за допомогою Pandas, де для кожного відео обчислюються показники, такі як лайки, коментарі, перегляди, тривалість і співвідношення (наприклад, лайки до коментарів), а потім відображаються в таблиці на сторінці compare_videos.html, наприклад, як на малюнку 3.13.

Video Comparison

Title	URL	Duration	Likes	Comments	Views	Likes to Comments Ratio	Likes to Views Ratio	Publication Date
Jain - Makeba (Official Video)	https://youtu.be/59Q_IhgGANc?si=C2zsXCj8RlZXU5dC	00:03:43	3205758	60631	332625023	52.87	0.01	2016-11-30
HOTEL	https://youtu.be/GWhGT66FteU?si=55kklsWO1ndbCmyA	00:03:18	282544	2372	22057288	119.12	0.01	2022-07-14
THE HARDKISS - Сестра (official video)	https://youtu.be/hP-6Gjt-A3w?si=289KZzCxe7Cz7w4E	00:03:43	75920	6583	6537978	11.53	0.01	2021-12-09
Earth, Wind & Fire - Let's Groove || Sub Español	https://youtu.be/yHLTV-ZC4lM?si=0vnQkiyIj54iqRAt	00:04:13	143811	823	10998917	174.74	0.01	2022-01-17
Tommy Cash - Espresso Macchiato (LIVE) | Estonia ee | First Semi-Final | Eurovision 2025	https://youtu.be/F3wsy8bywXQ?si=deELOBfN57tGrDN5	00:03:21	66435	3573	2630437	18.59	0.03	2025-05-13
Deep & Groovy Smooth Jazz House DJ Mix - Cozy Sundown Vibes	https://youtu.be/jLfyRW6x-BM?si=BImF-k1MQGlunmMi	00:58:55	8522	161	386295	52.93	0.02	2025-03-26

Рисунок 3.13 Приклад відображення даних

Ця таблиця включає колонки, такі як Title, URL, Duration, Likes, Comments, Views, і співвідношення, рендерені через Jinja2.

Крім того, система генерує графіки за допомогою Matplotlib, зокрема гістограми для лайків, коментарів і переглядів, а також діаграми розсіювання,

що показують залежності, наприклад, тривалості від лайків. У разі відсутності даних відображається відповідне повідомлення, а кнопка "Back to Home" полегшує навігацію.

3.5.4.4. Генерацію звітів у форматах PDF і XLSX із правильним збереженням у Reports.

Для прикладу візьмемо генерацію звіту у вигляді XLSX-файлу.

Функціонал генерації звітів у форматах PDF і XLSX дозволяє користувачам створювати структуровані звіти на основі даних із бази даних “media_analysis” та зберігати їх із відповідним записом у таблиці "Reports". Користувач на головній сторінці в секції "Generate Reports" натискає одну з кнопок – "Generate PDF" або "Generate XLSX", після чого система виконує SQL-запит до таблиць. На малюнку 3.14 відображається вікно з автоматично створеним файлом.

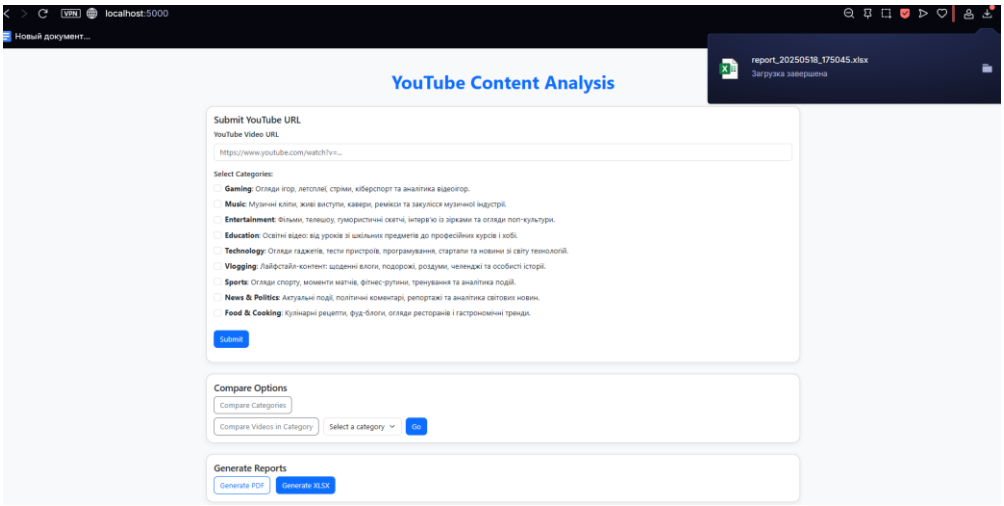


Рисунок 3.14 Приклад відображення створення excel-файлу

Після генерації файл зберігається в папці "reports", а в таблиці "Reports" створюється запис із полями: "report_id" , "report_name" , "report_type", "file_path" і "generated_at". Приклад на малюнку 3.15. Файл повертається

користувачу для завантаження через "send_file(file_path, as_attachment=True)". У разі відсутності даних відображається повідомлення про помилку у форматі JSON.

Table: reports

report_id	report_name	report_type	file_path	generated_at
1	Report_20250518_175004	XLSX	reports/report_20250518_175004.xlsx	2025-05-18 17:50:04
2	Report_20250518_175012	XLSX	reports/report_20250518_175012.xlsx	2025-05-18 17:50:12
3	Report_20250518_175045	XLSX	reports/report_20250518_175045.xlsx	2025-05-18 17:50:45
4	Report_20250518_180603	XLSX	reports/report_20250518_180603.xlsx	2025-05-18 18:06:04
5	Report_20250518_180729	PDF	reports/report_20250518_180729.pdf	2025-05-18 18:07:29
6	Report_20250518_181250	PDF	reports/report_20250518_181250.pdf	2025-05-18 18:12:50
7	Report_20250518_181336	PDF	reports/report_20250518_181336.pdf	2025-05-18 18:13:36

Рисунок 3.15 Приклад створених файлів і їх відображення в базі даних

3.5.4.5. Перегляд вмісту бази даних із коректним відображенням таблиць і даних.

Функціонал перегляду вмісту бази даних дозволяє користувачу детально ознайомитися з усіма таблицями та їхніми даними в базі даних. Користувач на головній сторінці в секції "Database Options" натискає "Show All Databases", після чого система виконує SQL-запит SHOW TABLES для отримання списку таблиць, а потім для кожної таблиці виконується SELECT * FROM {table} для витягнення всіх рядків і SHOW COLUMNS FROM {table} для отримання назв стовпців. Наприклад, на малюнку 3.16 подивимося відображення відео "Deep & Groovy Smooth Jazz House DJ Mix - Cozy Sundown Vibes" в таблиці MediaContent.

30	what I EAT in a busy week in NYC | high protein & balanced recipes	2024-09-07	video	https://youtu.be/HR61Hwt2qUA?si=VcChICCoKmtwM_UI	00:28:40
31	What I Learned in Restaurants That Made My Home Cooking Better	2025-03-29	video	https://youtu.be/akWA8-6s6us?si=xp3LWnNEHT05EDJ1	00:26:23
32	Juicy Beef Steaks on Salt Block! Cooking in the Snowy Mountains of Azerbaijan!	2025-03-14	video	https://youtu.be/KSAkH8Lts_w?si=fAhmO1-rwV7mp6WG	00:42:17
33	'Political malpractice': Republicans hide as Trump's tariff policies start to impact consumers	2025-05-05	video	https://youtu.be/HOGm8jP2JNE?si=14TftzvNV3x4Kf0I	00:07:38
34	Tommy Cash - Espresso Macchiato (LIVE) | Estonia 🇪🇪 | First Semi-Final | Eurovision 2025	2025-05-13	video	https://youtu.be/F3wsy8bywXQ?si=deELOBfN57tGrDN5	00:03:21
35	The Stress Expert: Your Brain is Like a Phone Battery! (9 Ways to Instantly Recharge)	2024-12-13	video	https://youtu.be/9EqrUK7ghho?si=dQoPAs49k_Bsc3Qm	01:17:27
36	Deep & Groovy Smooth Jazz House DJ Mix - Cozy Sundown Vibes	2025-03-26	video	https://youtu.be/jLfyRW6x-BM?si=BImF-k1MQGlunmMi	00:58:55

Table: mediacontentdirections

content_id	direction_id
11	1
13	1

Рисунок 3.16 Відображення відео в таблиці MediaContent

3.5.4.6. Обробка помилок із відповідними повідомленнями.

Функціонал обробки помилок забезпечує коректне реагування на помилкові дії користувача, відображаючи чіткі повідомлення на сторінці. Наприклад, на зображенні головної сторінки "YouTube Content Analysis" видно, що при спробі надіслати URL без вибору категорій з’являється повідомлення "Please select at least one category." у червоному тексті під кнопкою "Submit", як на малюнку 3.17.

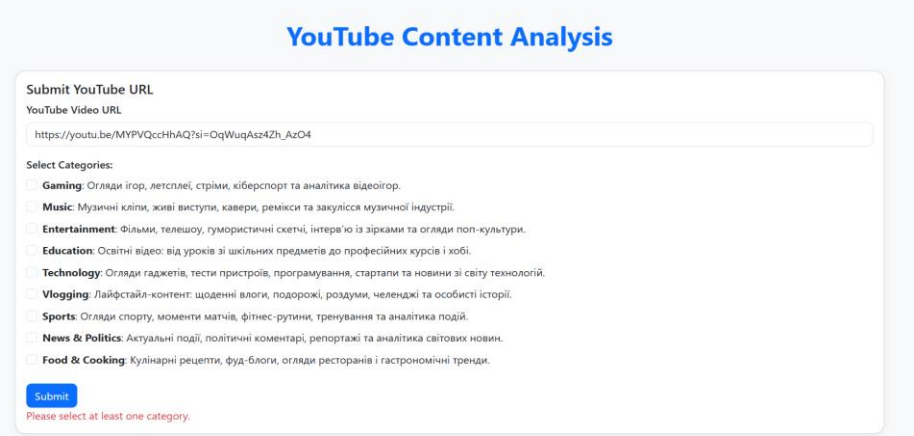


Рисунок 3.17 Попередження: "Please select at least one category."

3.5.4.7. Прогнозування популярності відео з відображенням результатів

Функціонал прогнозування популярності відео дозволяє користувачу оцінити потенційну кількість переглядів для відео на основі категорії та тривалості. Користувач на головній сторінці в секції "Analysis Options" натискає "Predict Video Popularity", після чого система виконує SQL-запит для вибірки метаданих відео (лайки, коментарі, тривалість, категорія) з таблиць MediaContent і PopularityMetrics. Модель машинного навчання RandomForestRegressor обробляє ці дані, генеруючи прогнози переглядів для

кожної категорії. Результати відображаються в шаблоні popularity_prediction.html, який включає таблицю з прогнозованими переглядами, рекомендацію оптимальної категорії та тривалості, а також графік у форматі base64-зображення, створений за допомогою Matplotlib і Seaborn. Наприклад, на малюнку 3.18 зображено прогноз популярності для категорії "Education" з рекомендацією тривалості 25 хвилин.

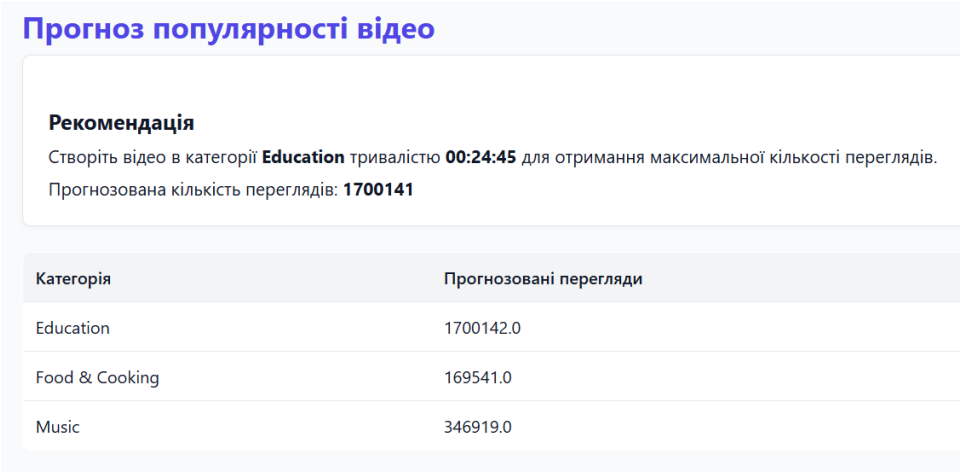


Рисунок 3.18 Відображення прогнозу популярності відео

3.6. Висновки до розділу 3

Розроблена система обліку медіаконтенту на базі Python, Flask, MySQL і YouTube API успішно реалізує ключові функції: додавання відео через URL, аналіз даних за категоріями з графіками, порівняння відео, генерацію звітів, перегляд бази даних, прогнозування популярності відео на основі категорії та тривалості, обробку помилок. Тестування підтвердило стабільність і зручність інтерфейсу, що відповідає вимогам до адаптивності, доступності даних і надійності. Використання MVC-шаблону та клієнт-серверної архітектури забезпечило структурованість, гнучкість і масштабованість системи, що робить її ефективним інструментом для аналізу YouTube-контенту.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього вбудованого програмного продукту.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1. Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації

компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

1) функціонування на персональних комп'ютерах із стандартним набором компонентів;
2) зручність та зрозумілість для користувача;
3) швидкість обробки даних та доступ до інформації в реальному часі;
4) можливість зручного масштабування та обслуговування;
5) мінімальні витрати на впровадження програмного продукту.

4.2. Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка інформаційної системи для обліку та аналізу медіаконтенту на основі баз даних, що дозволяє оцінювати різні характеристики, які впливають на ефективність управління медіаресурсами. Беручи за основу цю функцію, можна виділити наступні:

1) F_1 – вибір бази даних для зберігання даних;
2) F_2 – вибір фреймворку для серверної частини;
3) F_3 – вибір API для інтеграції з зовнішніми платформами.

Кожна з цих функцій має декілька варіантів реалізації.

Функція F_1.

1. MySQL.
2. MongoDB.
3. PostgreSQL.

Функція F_2.

1. Flask.

2. Django.

Функція F_3.

1. YouTube Data API.

2. Vimeo API.

3. Twitter API.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

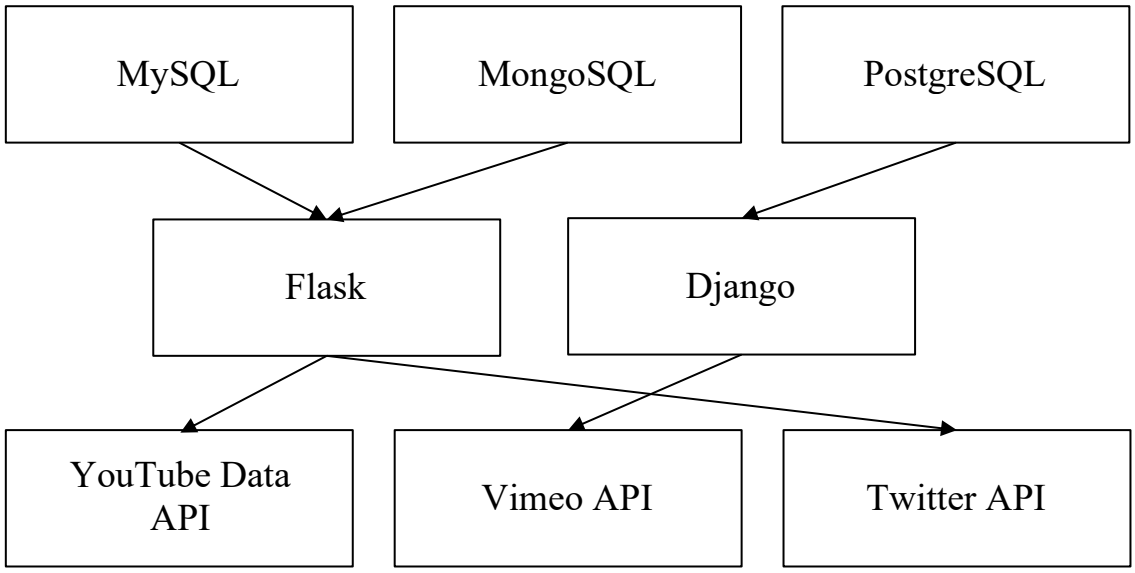


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Висока доступність, підтримка реляційних баз даних, простота інтеграції	Обмежена гнучкість для неструктурованих даних

Продовження таблиці 4.1

F_1	Б	Підтримка складних типів даних (JSON, масиви), висока надійність	Вищі вимоги до ресурсів
	В	Гнучкість для неструктурованих даних, горизонтальне масштабування	Складніша підтримка транзакцій
F_2	А	Легкість використання, швидка розробка, мінімальні ресурси	Обмежений функціонал для великих проєктів
	Б	Висока гнучкість, вбудовані інструменти, підтримка великих проєктів	Вищі вимоги до ресурсів, складніша настройка
F_3	А	Доступність, широкий спектр медіаданих, інтеграція з YouTube	Обмеження за квотами API
	Б	Доступність, підтримка професійного контенту	Менший обсяг даних порівняно з YouTube
	В	Інтеграція з соціальними мережами, аналіз трендів	Обмежений доступ до медіафайлів

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1. Перевагу надаємо стандартному варіанту для систем такого типу, тобто MySQL. Незважаючи на те, що MySQL має обмежену гнучкість для роботи з неструктурованими даними, що може бути важливим у медіааналітиці, це компенсується її високою доступністю, простотою інтеграції та широкою підтримкою спільноти. Наприклад, у проєкті MySQL забезпечила швидкий доступ до структурованих метаданих відео, таких як назви, категорії та статистичні показники. Якби використовувалася MongoDB, то обробка транзакцій була б складнішою, а PostgreSQL вимагала б більше ресурсів для розгортання, що не було виправданим для даного проєкту.

Обраний варіант А.

Функція F_2. Вибір фреймворку для серверної частини зумовлений необхідністю швидкої розробки та легкості використання. Проєкт розроблявся з урахуванням обмежених ресурсів і потреби у швидкому запуску. Flask є легким і простим у використанні фреймворком, що ідеально підходить для створення серверної частини системи з базовою функціональністю, такою як обробка запитів, інтеграція з API та рендеринг шаблонів. Django, хоча і має ширший функціонал, виявився надмірним для цього проєкту через вищі вимоги до ресурсів та складнішу настройку, що могло уповільнити розробку.

Обраний варіант А.

Функція F_3. Для інтеграції з зовнішніми платформами доступні різні API, такі як YouTube Data API, Vimeo API та Twitter API. Оскільки проєкт орієнтований на аналіз відеоконтенту, вибір пав на YouTube Data API завдяки його широкій доступності та можливості отримувати детальні метадані та статистику відео, що є ключовими для системи. Альтернативи, такі як Vimeo API (спрямоване на професійний контент з меншим обсягом даних) та Twitter API (фокус на соціальні мережі без пріоритету на медіафайли), не відповідали основним задачам проєкту. Використання YouTube Data API забезпечило ефективне збирання даних, хоча потребувало управління квотами API.

Обраний варіант А.

Таким чином, будемо розглядати такий варіант реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{а}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3. Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) X1 – потенційний об’єм програмного коду;
2) X2 – обсяг пам’яті, необхідний для зберігання бази даних;
3) X3 – час генерації звіту;
4) X4 – швидкість обробки запитів до бази даних.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Потенційний об’єм програмного коду	X1	кількість рядків коду	500	400	300
Обсяг пам’яті, необхідний для зберігання бази даних	X2	Мб	800	500	300
Час генерації звіту	X3	с	15	10	5

Продовження таблиці 4.2

Швидкість обробки запитів до бази даних	X4	запитів/с	50	100	200

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – Потенційний об’єм програмного коду.

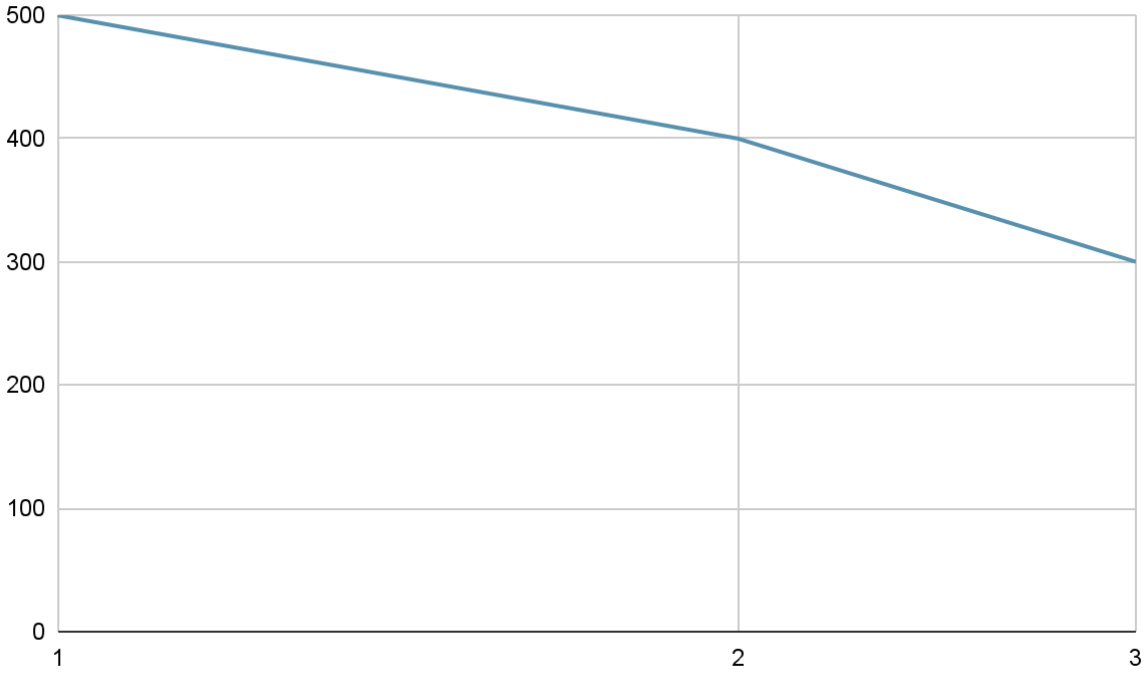


Рисунок 4.2 – X1, Потенційний об’єм програмного коду

Рис. 4.3 – Обсяг пам’яті, необхідний для зберігання бази даних, Рис. 4.4 – Час генерації звіту

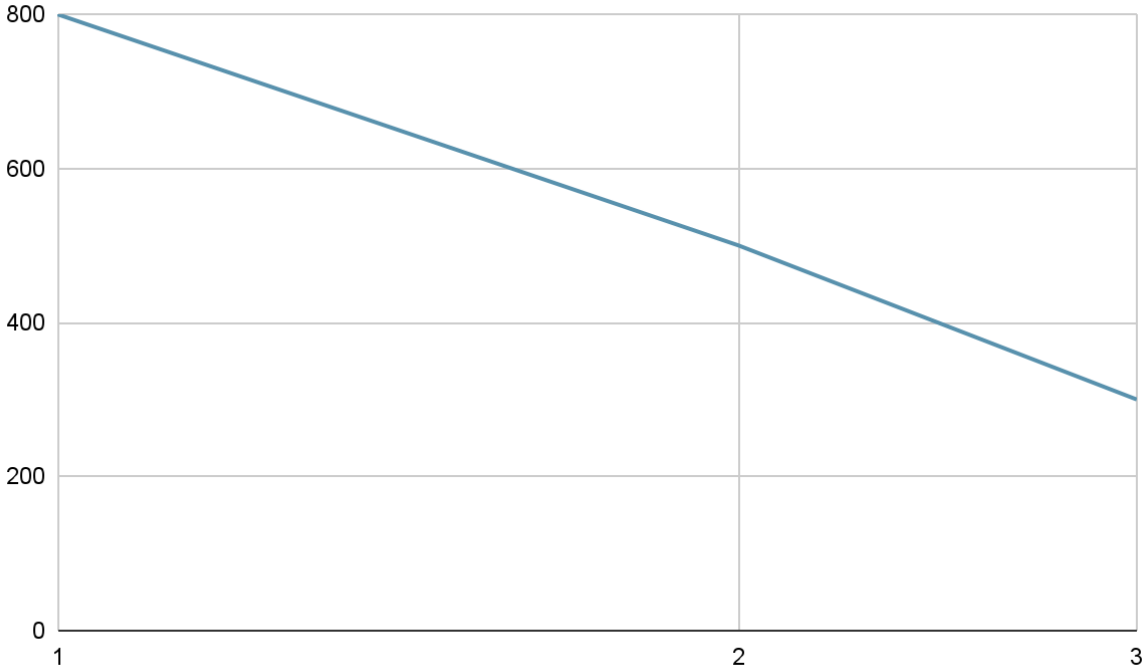


Рисунок 4.3 – X2, Обсяг пам’яті, необхідний для зберігання бази даних

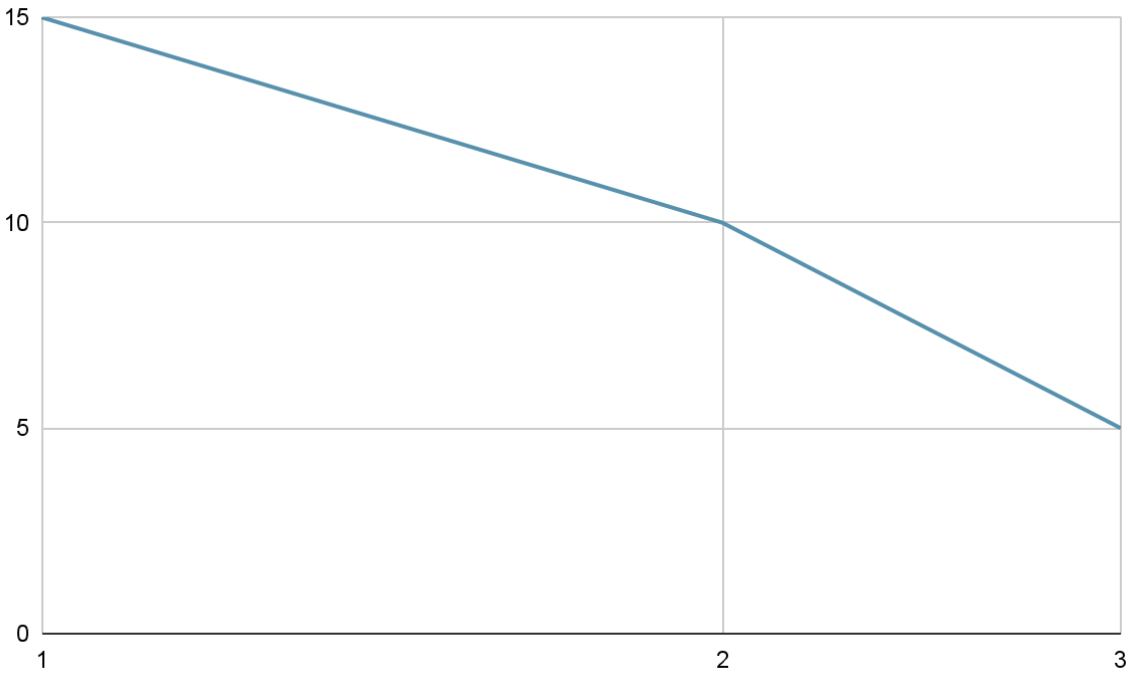


Рисунок 4.4 – X3, Час генерації звіту

На малюнку 4.5 відображається параметр “швидкість обробки запитів до бази даних”.

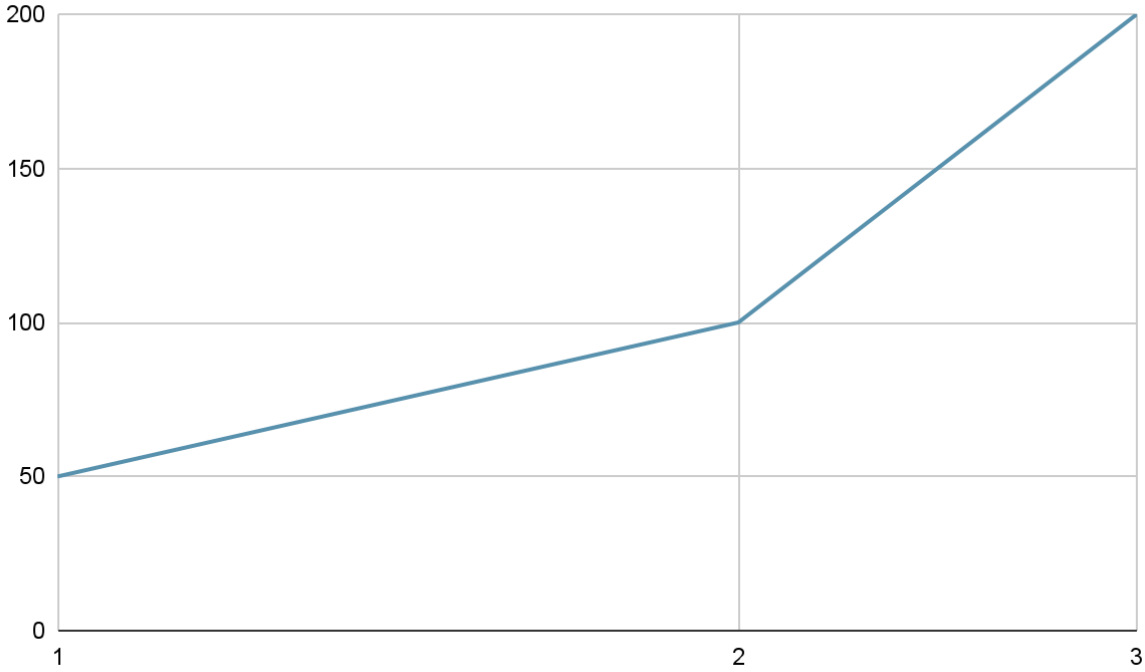


Рисунок 4.5 – X4, Швидкість обробки запитів до бази даних

4.4. Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;
2) перевірку придатності експертних оцінок для подальшого використання;

3) визначення оцінки попарного пріоритету параметрів;

4) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Потенційний об’єм програмного коду	кількість рядків коду	1	1	1	1	1	1	2	8	-9.5	90,25
X2	Обсяг пам’яті, необхідний для зберігання бази даних	Мб	4	4	3	3	4	3	4	25	7.5	56,25
X3	Час генерації звіту	с	1	1	3	3	2	2	1	13	-4.5	20,25
X4	Швидкість обробки запитів до бази даних	запитів/с	4	4	3	3	3	4	3	24	6.5	42,25
	Разом		10	10	10	10	10	10	10	70	0	209

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

1) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N – число експертів, n – кількість параметрів.

2) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5, \tag{4.2}$$

3) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

4) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 209. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 209}{7^2(4^3 - 4)} = 0{,}853 > W_k = 0{,}67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	=	=	<	<	<	<	>	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	=	=	>	>	>	>	1,5
X2 і X4	=	=	=	=	>	<	>	=	1,0
X3 і X4	<	<	=	=	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги і–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{при} X_i > X_j 1.0 \text{при} X_i = X_j 0.5 \text{при} X_i < X_j. \tag{4.6}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{вi} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{вi} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X1	1	1,5	1,0	1,5	5	0,31	19	0,32	70,25	0,32
X2	0,5	1	0,5	0,5	2,5	0,16	9,25	0,16	34,38	0,16
X3	1,0	1,5	1,0	1,5	5	0,31	19	0,32	70,25	0,32
X4	0,5	1,5	0,5	1	3,5	0,22	12,25	0,2	35,12	0,2
Всього:					16	1	59,5	1	220	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (Обсяг пам’яті, необхідний для зберігання бази даних), X3 (Час генерації звіту) та X4 (Швидкість обробки запитів до бази даних) відповідають технічним вимогам умов функціонування даного ПП. Абсолютне значення параметра X1 (Потенційний об’єм програмного коду) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \qquad (4.11)$$

де n – кількість параметрів;

$K_{\text{в}i}$– коефіцієнт вагомості і–го параметра;

B_i – оцінка і–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	400	25	0,32	8
F2	A	X2	500	22	0,16	3,52
F3	A	X3	10	23	0,32	7,36
F4	A	X4	100	24	0,2	4,8

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$К_{К1} = 8 + 3{,}52 + 4{,}8 = 16{,}32 ;$$

$$К_{К2} = 4 + 7{,}36 + 4{,}8 = 16{,}16 .$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання.

1. Розробка вбудованого програмного забезпечення.
2. Розробка допоміжної програми для візуалізації.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 3; а в завданні 2 – до групи 1.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ =40 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.3. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 40 \cdot 1.3 \cdot 0.8 = 41{,}6 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =29 людино-днів, $К_П$ =1, $К_{СК}$ = 1, $К_{СТ}$ =0,8:

$$Т_2 = 29 \cdot 0.8 = 23.2 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (41{,}6 + 23.2 + 3.7 + 21.2) \cdot 8 = 717{,}6 \text{ людино-годин.}$$

$$Т_{II} = (41{,}6 + 23.2 + 7.81 + 21.2) \cdot 8 = 750{,}48 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 30000 грн. Визначаємо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} \text{ грн.,} \qquad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{30000}{21 \cdot 8} = 178{,}57 \text{ грн.} \quad (4.15)$$

Тоді, розраховуємо заробітну плату за формулою:

$$С_{зп} = С_ч \cdot Т_i \cdot КД, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 178{,}57 \cdot 717{,}6 \cdot 1.2 = 153771{,}42857$ грн.

II. $С_{ЗП} = 178{,}57 \cdot 750{,}48 \cdot 1.2 = 160817{,}142857$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 153771{,}42857 \cdot 0.22 = 33829{,}714285$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 160817{,}142857 \cdot 0.22 = 35379{,}771429$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0{,}2 = 72000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 72000 \cdot (1 + 0.2) = 86400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 115200 \cdot 0{,}22 = 19008 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 15000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 15000 = 5250 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 15000 \cdot 0.08 = 1680 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.40 =$$

$$= 745,6 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 745,6 \cdot 0,4 \cdot 0,2 \cdot 9,43 = 562,48 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 15000 \cdot 0,67 = 10050 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$$С_{ЕКС} = 86400 + 19008 + 5250 + 1680 + 562,48 + 10050 = 122950,48 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 122950,48 / 745,6 = 164,9 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T, \quad (4.18)$$

I. $С_М = 164,9 \cdot 717,6 = 118333,241$ грн.

II. $С_М = 164,9 \cdot 750,48 = 123754,152$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 153771,42857 \cdot 0,67 = 103026.857142$ грн.

II. $С_Н = 160817,142857 \cdot 0,67 = 107747.485714$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 153771,42857 + 33829,714285 + 118333,241 + 103026,857142 = 408961,241$ грн.

II. $С_{ПП} = 5160817,142857 + 35379,771429 + 123754,152 + 107747,485714 = 427698,552$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j}, \quad (4.21)$$

$К_{ТЕР1} = 16,32 / 408961,241 = 0,00003991,$

$К_{ТЕР2} = 16,16 / 427698,552 = 0,00003778.$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 0,00003991$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0,00003991$.

Цей варіант реалізації програмного продукту має такі параметри:

1) F_1 – вибір бази даних для зберігання даних – MySQL;
2) F_2 – вибір фреймворку для серверної частини – Flask;
3) F_3 – вибір API для інтеграції з зовнішніми платформами – YouTube Data API.

Даний варіант виконання програмного комплексу гарантує швидку та не ресурсомістку реалізацію вбудованого програмного забезпечення для проекту.

4.8 Висновки до розділу 4

У цій частині дипломної роботи було проведено докладний аналіз функціональних та вартісних аспектів програмного продукту. Також було оцінено основні функції цього програмного продукту.

В результаті аналізу функціональності та вартості програмного комплексу, який розробляється, було визначено та проаналізовано його ключові функції, а також ідентифіковано параметри, що характеризують його. На основі проведеного аналізу був обраний варіант реалізації програмного продукту.

ВИСНОВКИ

У даній дипломній роботі була розглянута тема "Інформаційна система обліку медіаконтенту на основі баз даних". У процесі дослідження проведено аналіз теоретичних основ побудови інформаційних систем для обліку та аналізу медіаконтенту, розглянуто особливості реляційних (SQL) та нереляційних (NoSQL) баз даних, їхні переваги та недоліки у контексті роботи з великими обсягами даних, характерними для медіаіндустрії. Вивчено сучасні підходи до архітектурного проектування інформаційних систем, включаючи клієнт-серверну модель та шаблон MVC, що було використано у розробці системи.

Було визначено основні вимоги до систем обліку медіаконтенту, включаючи зберігання даних, швидкий пошук та інтеграцію з API платформ, таких як YouTube Data API. Проведено проектування бази даних MySQL для зберігання структурованих метаданих, таких як інформація про відео, категорії, статистичні показники та звіти. Розроблено архітектуру системи, що включає серверну частину на основі Flask, клієнтський веб-інтерфейс із використанням HTML, CSS та JavaScript, а також модулі для збору, аналізу, порівняння, прогнозування та візуалізації даних.

В рамках роботи було реалізовано програмний застосунок, що дозволяє користувачам додавати відео з YouTube за URL-адресами, зберігати їх у базі даних, здійснювати аналіз за категоріями, порівнювати популярність відео за основними метриками, створювати звіти у форматах PDF та XLSX переглядати вміст бази даних, а також прогнозувати популярність відео на основі категорії та тривалості. Проведено тестування системи, що підтвердило коректність її роботи, а також відповідність функціональності поставленим завданням.

Загалом розроблена інформаційна система обліку медіаконтенту є ефективним інструментом для управління даними, що забезпечує зручний

доступ до медіаінформації та підтримує прийняття рішень у сфері медіа. Результати дипломної роботи можуть бути використані для подальшого розвитку подібних систем, удосконалення їх функціоналу та інтеграції з іншими сервісами. Розроблена система має перспективи для масштабування, розширення функціональності та впровадження в реальні проекти у сфері цифрових медіа.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston: Pearson, 2016. URL: https://www.auhd.edu.ye/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf (last accessed: 2.04.2025).
2. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. Boston: McGraw-Hill Education, 2019. URL: https://mrce.in/ebooks/Database%20System%20Concepts%207th%20Ed.pdf (last accessed: 4.04.2025).
3. Stonebraker M. SQL Databases v. NoSQL Databases. *Communications of the ACM.* 2010. Vol. 53, No. 4. P. 10–11. URL: https://dl.acm.org/doi/pdf/10.1145/1721654.1721659 (last accessed: 15.05.2025).
4. YouTube Data API Documentation. Google Developers. URL: https://developers.google.com/youtube/v3 (last accessed: 11.05.2025).
5. MySQL Documentation. Oracle. URL: https://dev.mysql.com/doc/ (last accessed: 13.05.2025).
6. Flask Documentation. Pallets Projects. URL: https://flask.palletsprojects.com/ (last accessed: 25.05.2025).
7. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Pearson Education, 2003. URL: https://www.mbit.edu.in/wp-content/uploads/2020/05/An-Introduction-to-Database-Systems-8e-By-C-J-Date-CodeBlah.com_.pdf (last accessed: 20.05.2025).
8. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Sample pages. Boston: Pearson, 2016. URL: https://ptgmedia.pearsoncmg.com/images/9780137843107/samplepages/9780137843107_Sample.pdf (last accessed: 20.05.2025).
9. Pandas Documentation. Pandas Development Team. URL: https://pandas.pydata.org/docs/ (last accessed: 5.06.2025).

10. Matplotlib Documentation. Matplotlib Development Team. URL: https://matplotlib.org/stable/contents.html (last accessed: 6.06.2025).
11. Khan W., Kumar T., Zhang C., Raj K., Roy A. M., Luo B. SQL and NoSQL database software architecture performance analysis and assessments - a systematic literature review. Big Data and Cognitive Computing. 2023. Vol. 7, No. 2. URL: https://doi.org/10.3390/bdcc7020097.
12. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd ed. Sebastopol, CA: O'Reilly Media, 2019. 819 p. URL: http://14.139.161.31/OddSem-0822-1122/Hands-On_Machine_Learning_with_Scikit-Learn-Keras-and-TensorFlow-2nd-Edition-Aurelien-Geron.pdf (дата звернення: 28.05.2025).
13. Provost F., Fawcett T. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. O'Reilly Media, 2013. URL: https://github.com/bsharvey/bsharvey.github.io/blob/master/assets/books/data-science-for-business.pdf (last accessed: 5.04.2025).
14. Hastie, T., Tibshirani, R., Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer, 2009. URL: https://web.stanford.edu/~hastie/ElemStatLearn/ (дата звернення: 05.05.2025).
15. Breiman L. Random Forests. Machine Learning. 2001. Vol. 45, № 1. P. 5–32. URL: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf (last accessed: 11.05.2025).

ДОДАТОК А

```
Файл main.py
from flask import Flask, request, render_template, jsonify, send_file, redirect, url_for
import mysql.connector
import re
import datetime
import pandas as pd
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import io
import base64
from youtube_api import YouTubeDataAPI
import uuid
from fpdf import FPDF
import os
from unidecode import unidecode
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime
from flask import render_template
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import io
import base64
from datetime import datetime
app = Flask(__name__)
# Database configuration
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '2804',
    'database': 'media_analysis'
}
def get_db_connection():
    return mysql.connector.connect(**db_config)
# Функция для преобразования hh:mm:ss в секунды
def time_to_seconds(time_str):
    if not time_str or time_str == "00:00:00":
        return 0
    try:
        hours, minutes, seconds = map(int, time_str.split(':'))
        return hours * 3600 + minutes * 60 + seconds
    except:
        return 0
# Функция для преобразования секунд в hh:mm:ss
def seconds_to_time(seconds):
    hours = seconds // 3600
    minutes = (seconds % 3600) // 60
    seconds = seconds % 60
    return f"{hours:02d}:{minutes:02d}:{seconds:02d}"
# Initialize database
def init_db():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DROP TABLE IF EXISTS  PopularityPredictions")
    # Drop existing tables (uncomment if needed)
    '''
    cursor.execute("DROP TABLE IF EXISTS MediaContentDirections")
    cursor.execute("DROP TABLE IF EXISTS PopularityMetrics")
    cursor.execute("DROP TABLE IF EXISTS Reports")
    cursor.execute("DROP TABLE IF EXISTS APIKeys")
    cursor.execute("DROP TABLE IF EXISTS MediaContent")
    cursor.execute("DROP TABLE IF EXISTS Directions")
    '''
    # Create Directions table
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Directions (
            direction_id SERIAL PRIMARY KEY,
            direction_name VARCHAR(100) NOT NULL UNIQUE,
            description TEXT
        )
    """)
    # Create MediaContent table with duration as VARCHAR
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS MediaContent (
            content_id SERIAL PRIMARY KEY,
            title VARCHAR(255) NOT NULL,
            publication_date DATE NOT NULL,
            content_type VARCHAR(50) NOT NULL,
            url VARCHAR(255) NOT NULL,
            duration VARCHAR(8) DEFAULT '00:00:00'
        )
    """)
    # Create MediaContentDirections junction table
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS MediaContentDirections (
            content_id BIGINT UNSIGNED NOT NULL,
            direction_id BIGINT UNSIGNED NOT NULL,
            PRIMARY KEY (content_id, direction_id),
```

```
            FOREIGN KEY (content_id) REFERENCES MediaContent(content_id),
            FOREIGN KEY (direction_id) REFERENCES Directions(direction_id)
        )
    """)
    # Create PopularityMetrics table with duration as VARCHAR
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS PopularityMetrics (
            metric_id SERIAL PRIMARY KEY,
            content_id BIGINT UNSIGNED NOT NULL,
            platform VARCHAR(50) NOT NULL,
            likes INT DEFAULT 0,
            views INT DEFAULT 0,
            comments INT DEFAULT 0,
            duration VARCHAR(8) DEFAULT '00:00:00',
            fetch_date DATE NOT NULL,
            FOREIGN KEY (content_id) REFERENCES MediaContent(content_id)
        )
    """)
    # Create Reports table
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Reports (
            report_id SERIAL PRIMARY KEY,
            report_name VARCHAR(255) NOT NULL,
            report_type VARCHAR(10) NOT NULL,
            file_path VARCHAR(255) NOT NULL,
            generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    """)
    # Create PopularityPredictions table
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS PopularityPredictions (
            id INT AUTO_INCREMENT PRIMARY KEY,
            direction_name VARCHAR(100),
            duration_range VARCHAR(20),
            predicted_views FLOAT,
            prediction_date DATETIME
        )
    """)
    # Populate Directions with 7 popular YouTube categories
    popular_categories = [
        ('Gaming', 'Огляди ігор, летсплеї, стріми, кіберспорт та аналітика відеоігор.'),
        ('Music', 'Музичні кліпи, живі виступи, кавери, ремікси та закулісся музичної індустрії.'),
        ('Entertainment', 'Фільми, телешоу, гумористичні скетчі, інтерв’ю із зірками та огляди поп-культури.'),
        ('Education', 'Освітні відео: від уроків зі шкільних предметів до професійних курсів і хобі.'),
        ('Technology', 'Огляди гаджетів, тести пристроїв, програмування, стартапи та новини зі світу технологій.'),
        ('Vlogging', 'Лайфстайл-контент: щоденні влоги, подорожі, роздуми, челенджі та особисті історії.'),
        ('Sports', 'Огляди спорту, моменти матчів, фітнес-рутини, тренування та аналітика подій.'),
        ('News & Politics', 'Актуальні події, політичні коментарі, репортажі та аналітика світових новин.'),
        ('Food & Cooking', 'Кулінарні рецепти, фуд-блоги, огляди ресторанів і гастрономічні тренди.'),
    ]
    cursor.executemany(
        "INSERT IGNORE INTO Directions (direction_name, description) VALUES (%s, %s)",
        popular_categories
    )
    conn.commit()
    cursor.close()
    conn.close()
@app.route('/')
def index():
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT direction_id, direction_name, description FROM Directions")
        categories = cursor.fetchall()
        cursor.close()
        conn.close()
        return render_template('index.html', categories=categories)
    except Exception as e:
        return render_template('index.html', categories=[], error=str(e))
@app.route('/get_categories')
def get_categories():
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT direction_id, direction_name, description FROM Directions")
        categories = cursor.fetchall()
        cursor.close()
        conn.close()
        return jsonify(categories)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
@app.route('/submit_url', methods=['POST'])
def submit_url():
    youtube_url = request.form.get('youtube_url')
    categories = [cat for cat in request.form.getlist('categories') if cat]
    if not youtube_url:
        return jsonify({'error': 'URL is required'}), 400
    if not categories:
        return jsonify({'error': 'At least one category must be selected'}), 400
    from urllib.parse import urlparse, parse_qs
    parsed_url = urlparse(youtube_url)
    query_params = parse_qs(parsed_url.query)
    if 'v' in query_params:
        video_id = query_params['v'][0]
    else:
        video_id = parsed_url.path.lstrip('/').split('?')[0]
    if not video_id:
        return jsonify({'error': 'Invalid YouTube URL'}), 400
    try:
        youtube = YouTubeDataAPI('AIzaSyDz-DghDJHoSRU_vN_UmDoJBuHpoVjjWXk')
        video_data = youtube.get_video_metadata(video_id)
        if not video_data:
            return jsonify({'error': 'Video not found'}), 404
        title = video_data['snippet'].get('title', 'No title')
        published_at = video_data['snippet'].get('publishedAt')
        duration = video_data.get('duration', '00:00:00')
        if not published_at:
```

```
            return jsonify({'error': 'Publication date not available'}), 400
        from datetime import datetime
        publication_date = datetime.strptime(published_at, '%Y-%m-%dT%H:%M:%SZ').date()
        stats = youtube.get_video_stats(video_id)
        likes = int(stats.get('like_count', 0))
        comments = int(stats.get('comment_count', 0))
        views = int(stats.get('view_count', 0))
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO MediaContent (title, publication_date, content_type, url, duration)
            VALUES (%s, %s, %s, %s, %s)
        """, (title, publication_date, 'video', youtube_url, duration))
        content_id = cursor.lastrowid
        for category_id in categories:
            cursor.execute("""
                INSERT INTO MediaContentDirections (content_id, direction_id)
                VALUES (%s, %s)
            """, (content_id, int(category_id)))
        cursor.execute("""
            INSERT INTO PopularityMetrics (content_id, platform, likes, views, comments, duration, fetch_date)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (content_id, 'YouTube', likes, views, comments, duration, publication_date))
        conn.commit()
        cursor.close()
        conn.close()
        return jsonify({'message': f'Video "{title}" processed successfully'})
    except Exception as e:
        print(f"Error: {str(e)}")
        return jsonify({'error': str(e)}), 500
@app.route('/analyze_directions')
def analyze_directions():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT d.direction_name,
               pm.likes,
               pm.comments,
               pm.views,
               pm.duration,
               mc.publication_date
        FROM MediaContent mc
        JOIN MediaContentDirections mcd ON mc.content_id = mcd.content_id
        JOIN Directions d ON mcd.direction_id = d.direction_id
        JOIN PopularityMetrics pm ON mc.content_id = pm.content_id
    """)
    rows = cursor.fetchall()
    if not rows:
        cursor.close()
        conn.close()
        return render_template('analysis.html', likes_plot=None, comments_plot=None, views_plot=None, duration_plot=None, data=[],
error="No data available for analysis.")
    direction_data = {}
    for row in rows:
        direction = row['direction_name']
        if direction not in direction_data:
            direction_data[direction] = {
                'likes': [],
                'comments': [],
                'views': [],
                'durations': [],
                'first_pub_date': row['publication_date']
            }
        direction_data[direction]['likes'].append(row['likes'] or 0)
        direction_data[direction]['comments'].append(row['comments'] or 0)
        direction_data[direction]['views'].append(row['views'] or 0)
        direction_data[direction]['durations'].append(time_to_seconds(row['duration'] or '00:00:00'))
        if  row['publication_date']  and  (direction_data[direction]['first_pub_date']  is  None  or  row['publication_date']  <
direction_data[direction]['first_pub_date']):
            direction_data[direction]['first_pub_date'] = row['publication_date']
    results = []
    for direction, data in direction_data.items():
        avg_likes = sum(data['likes']) / len(data['likes']) if data['likes'] else 0
        avg_comments = sum(data['comments']) / len(data['comments']) if data['comments'] else 0
        avg_views = sum(data['views']) / len(data['views']) if data['views'] else 0
        avg_duration_seconds = sum(data['durations']) / len(data['durations']) if data['durations'] else 0
        avg_duration = seconds_to_time(int(avg_duration_seconds))
        # Debug raw data
        print(f"Direction: {direction}, Likes: {data['likes']}, Comments: {data['comments']}, Views: {data['views']}")
        # Calculate ratios
        likes_to_comments_ratios = [l / c if c else 0 for l, c in zip(data['likes'], data['comments'])]
        likes_to_views_ratios = [l / v if v else 0 for l, v in zip(data['likes'], data['views'])]
        avg_likes_to_comments = sum(likes_to_comments_ratios) / len(likes_to_comments_ratios) if likes_to_comments_ratios else 0
        avg_likes_to_views = sum(likes_to_views_ratios) / len(likes_to_views_ratios) if likes_to_views_ratios else 0
        print(f"Direction: {direction}, Likes to Comments Ratios: {likes_to_comments_ratios}, Avg: {avg_likes_to_comments}")
        print(f"Direction: {direction}, Likes to Views Ratios: {likes_to_views_ratios}, Avg: {avg_likes_to_views}")
        results.append({
            'direction_name': direction,
            'avg_likes': avg_likes,
            'avg_comments': avg_comments,
            'avg_views': avg_views,
            'avg_duration': avg_duration,
            'first_pub_date': data['first_pub_date'],
            'likes_to_comments_ratio': avg_likes_to_comments,
            'likes_to_views_ratio': avg_likes_to_views
        })
    df = pd.DataFrame(results)
    df['avg_duration_seconds'] = df['avg_duration'].apply(time_to_seconds)
    print("DataFrame before plotting:", df.to_dict())
    plots = {}
    metrics = ['avg_likes', 'avg_comments', 'avg_views', 'avg_duration_seconds', 'likes_to_comments_ratio', 'likes_to_views_ratio']
    for metric in metrics:
        plt.figure(figsize=(10, 6))
        values = df[metric]
        plt.bar(df['direction_name'], values, color='skyblue')
        plt.title(f'Average {metric.replace("avg_", "").replace("_seconds", "").replace("_", " ").title()} by Direction')
        plt.xlabel('Direction')
```

```
        plt.ylabel('Average Value' if metric not in ['likes_to_comments_ratio', 'likes_to_views_ratio'] else 'Ratio')
        plt.xticks(rotation=90, ha='right')
        plt.tight_layout(pad=3.0)
        if values.max() == 0:
            plt.text(0.5, 0.5, 'No significant data', ha='center', va='center', transform=plt.gca().transAxes)
        elif metric in ['likes_to_comments_ratio', 'likes_to_views_ratio']:
            plt.ylim(0, max(values.max() * 1.1, 10))  # Reasonable max for ratios
        elif metric in ['avg_likes', 'avg_views']:
            plt.yscale('log')
            plt.ylim(0, max(values.max() * 1.1, 1))
        else:
            plt.ylim(0, max(values.max() * 1.1, 1))
        buffer = io.BytesIO()
        plt.savefig(buffer, format='png')
        buffer.seek(0)
        plots[metric.replace('_seconds',         '').replace('_ratio',         '_ratio_plot')         +         '_plot']         =
base64.b64encode(buffer.getvalue()).decode()
        plt.close()
    print("Generated plots:", plots.keys())
    cursor.close()
    conn.close()
    return render_template('analysis.html',
                           likes_plot=plots.get('avg_likes_plot'),
                           comments_plot=plots.get('avg_comments_plot'),
                           views_plot=plots.get('avg_views_plot'),
                           duration_plot=plots.get('avg_duration_plot'),
                           likes_to_comments_plot=plots.get('likes_to_comments_ratio_plot'),
                           likes_to_views_plot=plots.get('likes_to_views_ratio_plot'),
                           data=results)
@app.route('/compare_videos_in_category')
def compare_videos_in_category():
    direction_id = request.args.get('direction_id', type=int)
    if not direction_id:
        return redirect(url_for('index'))  # Redirect to index if no direction_id
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT mc.title, mc.url, mc.duration, pm.likes, pm.comments, pm.views, mc.publication_date
        FROM MediaContent mc
        JOIN MediaContentDirections mcd ON mc.content_id = mcd.content_id
        JOIN PopularityMetrics pm ON mc.content_id = pm.content_id
        JOIN Directions d ON mcd.direction_id = d.direction_id
        WHERE d.direction_id = %s
    """, (direction_id,))
    videos = cursor.fetchall()
    if not videos:
        cursor.close()
        conn.close()
        return render_template('compare_videos.html', data=[], error=f"No videos found for direction ID {direction_id}")
    # Prepare data for plots
    df = pd.DataFrame(videos)
    df['duration_seconds'] = df['duration'].apply(time_to_seconds)
    # Calculate ratios for each video
    df['likes_to_comments_ratio'] = df.apply(lambda row: row['likes'] / row['comments'] if row['comments'] else 0, axis=1)
    df['likes_to_views_ratio'] = df.apply(lambda row: row['likes'] / row['views'] if row['views'] else 0, axis=1)
    plots = {}
    metrics = ['likes', 'comments', 'views', 'duration_seconds', 'likes_to_comments_ratio', 'likes_to_views_ratio']
    for metric in metrics:
        plt.figure(figsize=(12, 6))
        bars = plt.bar(df['title'], df[metric], color='skyblue')
        # Add values above bars
        for bar in bars:
            yval = bar.get_height()
            plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05 * yval, f'{int(yval)}' if metric != 'likes_to_comments_ratio' and
metric != 'likes_to_views_ratio' else f'{yval:.2f}',
                     ha='center', va='bottom', fontsize=10)
        plt.title(f'Comparison of {metric.replace("_seconds", "").replace("_", " ").title()}')
        plt.xlabel('Video Title')
        plt.ylabel(metric.replace('_seconds', '').replace('_', ' ').title())
        plt.xticks(wrap=True, rotation=0, ha='center')
        plt.tight_layout()
        buffer = io.BytesIO()
        plt.savefig(buffer, format='png')
        buffer.seek(0)
        plots[metric.replace('_seconds', '') + '_plot'] = base64.b64encode(buffer.getvalue()).decode()
        plt.close()
    # Add scatter plots for duration influence on likes and comments
    plt.figure(figsize=(12, 6))
    plt.scatter(df['duration_seconds'], df['likes'], color='blue', label='Likes')
    plt.title('Influence of Video Duration on Likes')
    plt.xlabel('Duration (seconds)')
    plt.ylabel('Likes')
    plt.legend()
    plt.tight_layout()
    buffer = io.BytesIO()
    plt.savefig(buffer, format='png')
    buffer.seek(0)
    duration_likes_plot = base64.b64encode(buffer.getvalue()).decode()
    plt.close()
    plt.figure(figsize=(12, 6))
    plt.scatter(df['duration_seconds'], df['comments'], color='green', label='Comments')
    plt.title('Influence of Video Duration on Comments')
    plt.xlabel('Duration (seconds)')
    plt.ylabel('Comments')
    plt.legend()
    plt.tight_layout()
    buffer = io.BytesIO()
    plt.savefig(buffer, format='png')
    buffer.seek(0)
    duration_comments_plot = base64.b64encode(buffer.getvalue()).decode()
    plt.close()
    cursor.close()
    conn.close()
    return render_template('compare_videos.html',
                           data=df.to_dict('records'),
                           likes_plot=plots.get('likes_plot'),
                           comments_plot=plots.get('comments_plot'),
```

```
                        views_plot=plots.get('views_plot'),
                        duration_plot=plots.get('duration_plot'),
                        likes_to_comments_plot=plots.get('likes_to_comments_ratio_plot'),
                        likes_to_views_plot=plots.get('likes_to_views_ratio_plot'),
                        duration_likes_plot=duration_likes_plot,
                        duration_comments_plot=duration_comments_plot)
@app.route('/generate_report/<report_type>')
def generate_report(report_type):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT GROUP_CONCAT(d.direction_name) as direction_names,
               mc.title,
               mc.duration as mc_duration,
               pm.duration as pm_duration,
               pm.likes,
               pm.comments,
               pm.views
        FROM MediaContent mc
        JOIN MediaContentDirections mcd ON mc.content_id = mcd.content_id
        JOIN Directions d ON mcd.direction_id = d.direction_id
        JOIN PopularityMetrics pm ON mc.content_id = pm.content_id
        GROUP BY mc.content_id, mc.title, mc.duration, pm.duration, pm.likes, pm.comments, pm.views
    """)
    data = cursor.fetchall()
    if not data:
        cursor.close()
        conn.close()
        return jsonify({'error': 'No data available for report'}), 404
    df = pd.DataFrame(data)
    # Ensure all text is converted to ASCII to avoid UnicodeEncodeError with latin-1
    for column in df.columns:
        df[column] = df[column].apply(lambda x: unidecode(str(x)) if isinstance(x, str) else x)
    report_id = str(uuid.uuid4())
    timestamp = datetime.datetime.now().strftime('%Y%m%d_%H%M%S')
    def dataframe_to_pdf(df, filename):
        pdf = FPDF()
        pdf.add_page()
        # Use default Arial font
        pdf.set_font("Arial", size=10)
        col_width = pdf.w / (len(df.columns) + 1)
        row_height = pdf.font_size + 2
        # Write header
        for col in df.columns:
            pdf.cell(col_width, row_height, txt=str(col), border=1)
        pdf.ln(row_height)
        # Write data
        for _, row in df.iterrows():
            for item in row:
                pdf.cell(col_width, row_height, txt=str(item), border=1)
            pdf.ln(row_height)
        pdf.output(filename)
    if report_type == 'pdf':
        file_path = f'reports/report_{timestamp}.pdf'
        dataframe_to_pdf(df, file_path)
    else:
        file_path = f'reports/report_{timestamp}.xlsx'
        df.to_excel(file_path, index=False)
    cursor.execute("""
        INSERT INTO Reports (report_name, report_type, file_path)
        VALUES (%s, %s, %s)
    """, (f'Report_{timestamp}', report_type.upper(), file_path))
    conn.commit()
    cursor.close()
    conn.close()
    return send_file(file_path, as_attachment=True)
@app.route('/show_databases')
def show_databases():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SHOW TABLES")
        tables = [table[0] for table in cursor.fetchall()]
        db_data = {}
        db_data['media_analysis'] = {}
        for table in tables:
            cursor.execute(f"SELECT * FROM {table}")
            rows = cursor.fetchall()
            cursor.execute(f"SHOW COLUMNS FROM {table}")
            columns = [col[0] for col in cursor.fetchall()]
            db_data['media_analysis'][table] = {'columns': columns, 'rows': rows}
        cursor.close()
        conn.close()
        return render_template('show_databases.html', db_data=db_data)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
@app.route('/predict_popularity')
def predict_popularity():
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        query = """
        SELECT d.direction_name,
               pm.likes,
               pm.comments,
               pm.views,
               pm.duration
        FROM MediaContent mc
        JOIN MediaContentDirections mcd ON mc.content_id = mcd.content_id
        JOIN Directions d ON mcd.direction_id = d.direction_id
        JOIN PopularityMetrics pm ON mc.content_id = pm.content_id
        """
        cursor.execute(query)
        data = cursor.fetchall()
        if not data:
            cursor.close()
            conn.close()
```

```
            return render_template('popularity_prediction.html', data=[], error="No data available for prediction.")
        df = pd.DataFrame(data)
        df['duration_seconds'] = df['duration'].apply(time_to_seconds)
        df = df[df['views'] > 0].copy()
        df['log_views'] = np.log(df['views'])
        X_numeric = df[['likes', 'comments', 'duration_seconds']]
        ohe = OneHotEncoder(sparse_output=False)
        X_cat = ohe.fit_transform(df[['direction_name']])
        X = np.hstack([X_numeric.values, X_cat])
        y = df['log_views'].values
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        model = RandomForestRegressor(n_estimators=100, random_state=42)
        model.fit(X_train, y_train)
        results = []
        categories = df['direction_name'].unique()
        for category in categories:
            subset = df[df['direction_name'] == category]
            if subset.empty:
                continue
            group_cols = ['likes', 'comments', 'duration_seconds']
            grouped = subset.groupby(group_cols).agg(
                count_videos=('views', 'size'),
                avg_views=('views', 'mean')
            ).reset_index()
            cat_vector = ohe.transform([[category]])[0]
            weighted_log_views_sum = 0
            total_weight = 0
            for _, row in grouped.iterrows():
                x_pred = np.hstack([[row['likes'], row['comments'], row['duration_seconds']], cat_vector])
                pred_log_view = model.predict([x_pred])[0]
                weight = row['count_videos']
                weighted_log_views_sum += pred_log_view * weight
                total_weight += weight
            if total_weight > 0:
                weighted_avg_log_views = weighted_log_views_sum / total_weight
                weighted_avg_views = np.exp(weighted_avg_log_views)
            else:
                weighted_avg_views = np.nan
            results.append({
                'direction_name': category,
                'predicted_views': weighted_avg_views
            })
        results_df = pd.DataFrame(results).dropna()
        top_category_row = results_df.sort_values('predicted_views', ascending=False).iloc[0]
        top_category_name = top_category_row['direction_name']
        top_category_views = top_category_row['predicted_views']
        # 🧠 Получаем оптимальную длительность
        cursor.execute("""
            SELECT likes, comments, duration
            FROM PopularityMetrics
            WHERE likes IS NOT NULL AND comments IS NOT NULL AND duration IS NOT NULL
        """)
        duration_data = cursor.fetchall()
        df_dur = pd.DataFrame(duration_data)
        df_dur['duration_seconds'] = df_dur['duration'].apply(time_to_seconds)
        df_dur['engagement'] = df_dur['likes'] + df_dur['comments']
        df_dur = df_dur[(df_dur['duration_seconds'] > 30) & (df_dur['duration_seconds'] < 1800)]
        q99 = df_dur['engagement'].quantile(0.99)
        df_dur = df_dur[df_dur['engagement'] <= q99]
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.linear_model import LinearRegression
        X_dur = df_dur[['duration_seconds']]
        y_dur = df_dur['engagement']
        poly = PolynomialFeatures(degree=2)
        X_poly = poly.fit_transform(X_dur)
        model_dur = LinearRegression()
        model_dur.fit(X_poly, y_dur)
        duration_range = np.arange(30, 1800, 5).reshape(-1, 1)
        duration_poly = poly.transform(duration_range)
        engagement_preds = model_dur.predict(duration_poly)
        best_idx = np.argmax(engagement_preds)
        best_duration_seconds = duration_range[best_idx][0]
        best_duration_formatted = seconds_to_time(int(best_duration_seconds))
        # Сохраняем в БД (по желанию)
        cursor.execute("DELETE FROM PopularityPredictions")
        for _, row in results_df.iterrows():
            cursor.execute(
                "INSERT INTO PopularityPredictions (direction_name, duration_range, predicted_views, prediction_date) "
                "VALUES (%s, %s, %s, %s)",
                (row['direction_name'], f"{best_duration_formatted}", row['predicted_views'], datetime.now())
            )
        conn.commit()
        plt.figure(figsize=(10, 6))
        sns.barplot(data=results_df.sort_values('predicted_views', ascending=False),
                    x='predicted_views', y='direction_name', palette='viridis')
        plt.title('Weighted Predicted Views by Category')
        plt.xlabel('Predicted Views')
        plt.ylabel('Category')
        buffer = io.BytesIO()
        plt.savefig(buffer, format='png')
        buffer.seek(0)
        plot_img = base64.b64encode(buffer.getvalue()).decode()
        plt.close()
        cursor.close()
        conn.close()
        return render_template('popularity_prediction.html',
                               data=results_df.to_dict('records'),
                               plot_img=plot_img,
                               top_category_name=top_category_name,
                               top_predicted_views=int(top_category_views),
                               top_duration_range=best_duration_formatted)
    except Exception as e:
        return render_template('popularity_prediction.html', data=[], error=str(e))
@app.route('/predict_optimal_duration')
def predict_optimal_duration():
    try:
```

```
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            SELECT likes, comments, duration
            FROM PopularityMetrics
            WHERE likes IS NOT NULL AND comments IS NOT NULL AND duration IS NOT NULL
        """)
        data = cursor.fetchall()
        cursor.close()
        conn.close()
        df = pd.DataFrame(data)
        df['duration_seconds'] = df['duration'].apply(time_to_seconds)
        df['engagement'] = df['likes'] + df['comments']
        # 🔎 Фильтрация выбросов
        df = df[(df['duration_seconds'] > 30) & (df['duration_seconds'] < 1800)]
        q99 = df['engagement'].quantile(0.99)
        df = df[df['engagement'] <= q99]
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.linear_model import LinearRegression
        X = df[['duration_seconds']]
        y = df['engagement']
        poly = PolynomialFeatures(degree=2)
        X_poly = poly.fit_transform(X)
        model = LinearRegression()
        model.fit(X_poly, y)
        durations = np.arange(30, 1800, 5).reshape(-1, 1)
        durations_poly = poly.transform(durations)
        predicted_engagements = model.predict(durations_poly)
        best_idx = np.argmax(predicted_engagements)
        best_duration = durations[best_idx][0]
        best_engagement = predicted_engagements[best_idx]
        return jsonify({
            'optimal_duration_seconds': int(best_duration),
            'optimal_duration_hms': seconds_to_time(int(best_duration)),
            'predicted_engagement': int(best_engagement)
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    os.makedirs('reports', exist_ok=True)
    init_db()
    app.run(debug=True)
Файл youtube_api.py
from googleapiclient.discovery import build
import isodate
class YouTubeDataAPI:
    def __init__(self, api_key):
        self.youtube = build('youtube', 'v3', developerKey=api_key)
    def get_video_metadata(self, video_id):
        request = self.youtube.videos().list(part="snippet,contentDetails", id=video_id)
        response = request.execute()
        if response['items']:
            item = response['items'][0]
            snippet = item['snippet']
            duration = item['contentDetails']['duration']
            duration_seconds = int(isodate.parse_duration(duration).total_seconds())
            # Преобразуем секунды в hh:mm:ss
            hours = duration_seconds // 3600
            minutes = (duration_seconds % 3600) // 60
            seconds = duration_seconds % 60
            duration_formatted = f"{hours:02d}:{minutes:02d}:{seconds:02d}"
            return {
                'snippet': snippet,
                'duration': duration_formatted  # Теперь duration в формате "hh:mm:ss"
            }
        return None
    def get_video_stats(self, video_id):
        request = self.youtube.videos().list(part="statistics", id=video_id)
        response = request.execute()
        stats = response['items'][0]['statistics'] if response['items'] else {}
        return {
            'like_count': stats.get('likeCount', '0'),
            'comment_count': stats.get('commentCount', '0'),
            'view_count': stats.get('viewCount', '0')
        }
Файл analysis.html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Direction Analysis</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
   <style>
       .plot-container {
           display: grid;
           gap: 20px;
           margin-top: 20px;
       }
       .plot-item {
           width: 100%;
           text-align: center;
       }
       .plot-item img {
           max-width: 100%;
           height: auto;
       }
       .summary-section {
           margin-bottom: 20px;
           padding: 15px;
           background-color: #f8f9fa;
           border-radius: 5px;
       }
   </style>
</head>
<body>
```

```
    <div class="container mt-5">
        <h1>Direction Analysis</h1>
        <!-- Back to Home Button -->
        <a href="{{ url_for('index') }}" class="btn btn-primary mb-3">Back to Home</a>

        {% if error %}
            <p class="text-danger">{{ error }}</p>
        {% else %}

            <table class="table">
                <thead>
                    <tr>
                        <th>Direction</th>
                        <th>Avg Likes</th>
                        <th>Avg Comments</th>
                        <th>Avg Views</th>
                        <th>Avg Duration</th>
                        <th>First Pub Date</th>
                        <th>Likes to Comments Ratio</th>
                        <th>Likes to Views Ratio</th>
                    </tr>
                </thead>
                <tbody>
                    {% for item in data %}
                        <tr>
                            <td>{{ item.direction_name }}</td>
                            <td>{{ item.avg_likes|round(2) }}</td>
                            <td>{{ item.avg_comments|round(2) }}</td>
                            <td>{{ item.avg_views|round(2) }}</td>
                            <td>{{ item.avg_duration }}</td>
                            <td>{{ item.first_pub_date }}</td>
                            <td>{{ item.likes_to_comments_ratio|round(2) }}</td>
                            <td>{{ item.likes_to_views_ratio|round(2) }}</td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
            <div class="plot-container">
                {% if likes_plot %}
                <div class="plot-item">
                    <h3>Likes Plot</h3>
                    <img src="{{ likes_plot }}" alt="Likes Plot">
                </div>
                {% endif %}
                {% if comments_plot %}
                <div class="plot-item">
                    <h3>Comments Plot</h3>
                    <img src="{{ comments_plot }}" alt="Comments Plot">
                </div>
                {% endif %}
                {% if views_plot %}
                <div class="plot-item">
                    <h3>Views Plot</h3>
                    <img src="{{ views_plot }}" alt="Views Plot">
                </div>
                {% endif %}
                {% if duration_plot %}
                <div class="plot-item">
                    <h3>Duration Plot</h3>
                    <img src="{{ duration_plot }}" alt="Duration Plot">
                </div>
                {% endif %}
                {% if likes_to_comments_plot %}
                <div class="plot-item">
                    <h3>Likes to Comments Ratio</h3>
                    <img src="{{ likes_to_comments_plot }}" alt="Likes to Comments Ratio">
                </div>
                {% endif %}
                {% if likes_to_views_plot %}
                <div class="plot-item">
                    <h3>Likes to Views Ratio</h3>
                    <img src="{{ likes_to_views_plot }}" alt="Likes to Views Ratio">
                </div>
                {% endif %}
            </div>
        {% endif %}
    </div>
</body>
</html>
Файл compare_videos.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video Comparison</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        .plot-container {
            display: grid;
            gap: 20px;
            margin-top: 20px;
        }
        .plot-item {
            width: 100%;
            text-align: center;
        }
        .plot-item img {
            max-width: 100%;
            height: auto;
        }
    </style>
</head>
<body>
    <div class="container mt-5">
        <h1>Video Comparison</h1>
        {% if error %}
```

```
            <p class="text-danger">{{ error }}</p>
        {% else %}
            <table class="table">
                <thead>
                    <tr>
                        <th>Title</th>
                        <th>URL</th>
                        <th>Duration</th>
                        <th>Likes</th>
                        <th>Comments</th>
                        <th>Views</th>
                        <th>Likes to Comments Ratio</th>
                        <th>Likes to Views Ratio</th>
                        <th>Publication Date</th>
                    </tr>
                </thead>
                <tbody>
                    {% for item in data %}
                        <tr>
                            <td>{{ item.title }}</td>
                            <td><a href="{{ item.url }}" target="_blank">{{ item.url }}</a></td>
                            <td>{{ item.duration }}</td>
                            <td>{{ item.likes }}</td>
                            <td>{{ item.comments }}</td>
                            <td>{{ item.views }}</td>
                            <td>{{ item.likes_to_comments_ratio|round(2) }}</td>
                            <td>{{ item.likes_to_views_ratio|round(2) }}</td>
                            <td>{{ item.publication_date }}</td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
            <div class="plot-container">
                {% if likes_plot %}
                <div class="plot-item">
                    <h3>Likes Plot</h3>
                    <img src="{{ likes_plot }}" alt="Likes Plot">
                </div>
                {% endif %}
                {% if comments_plot %}
                <div class="plot-item">
                    <h3>Comments Plot</h3>
                    <img src="{{ comments_plot }}" alt="Comments Plot">
                </div>
                {% endif %}
                {% if views_plot %}
                <div class="plot-item">
                    <h3>Views Plot</h3>
                    <img src="{{ views_plot }}" alt="Views Plot">
                </div>
                {% endif %}
                {% if duration_plot %}
                <div class="plot-item">
                    <h3>Duration Plot</h3>
                    <img src="{{ duration_plot }}" alt="Duration Plot">
                </div>
                {% endif %}
                {% if likes_to_comments_plot %}
                <div class="plot-item">
                    <h3>Likes to Comments Ratio</h3>
                    <img src="{{ likes_to_comments_plot }}" alt="Likes to Comments Ratio">
                </div>
                {% endif %}
                {% if likes_to_views_plot %}
                <div class="plot-item">
                    <h3>Likes to Views Ratio</h3>
                    <img src="{{ likes_to_views_plot }}" alt="Likes to Views Ratio">
                </div>
                {% endif %}
                {% if duration_likes_plot %}
                <div class="plot-item">
                    <h3>Duration vs Likes</h3>
                    <img src="{{ duration_likes_plot }}" alt="Duration vs Likes">
                </div>
                {% endif %}
                {% if duration_comments_plot %}
                <div class="plot-item">
                    <h3>Duration vs Comments</h3>
                    <img src="{{ duration_comments_plot }}" alt="Duration vs Comments">
                </div>
                {% endif %}
            </div>
        {% endif %}
    </div>
</body>
</html>
Файл index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>YouTube Content Analysis</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
        }
        h1 {
            font-weight: bold;
            text-align: center;
            margin-bottom: 30px;
        }
        .card {
            border-radius: 1rem;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.05);
        }
```

```
        .form-label {
            font-weight: 500;
        }
        .btn {
            border-radius: 0.5rem;
        }
        .form-control, .form-select {
            border-radius: 0.5rem;
        }
        .form-check-inline {
            margin-bottom: 10px;
        }
        .compare-form {
            flex-wrap: wrap;
            gap: 10px;
        }
        #message.error {
            color: #dc3545;
        }
        #message.success {
            color: #28a745;
        }
    </style>
</head>
<body>
    <div class="container py-5">
        <h1 class="text-primary">YouTube Content Analysis</h1>

        <!-- Submit YouTube URL -->
        <div class="card mb-4">
            <div class="card-body">
                <h5 class="card-title">Submit YouTube URL</h5>
                <form id="videoForm">
                    <div class="mb-3">
                        <label for="youtube_url" class="form-label">YouTube Video URL</label>
                        <input         type="text"         class="form-control"         id="youtube_url"         name="youtube_url"
placeholder="https://www.youtube.com/watch?v=..." required>
                    </div>
                    <div class="mb-3">
                        <label class="form-label">Select Categories:</label>
                        <div id="categories" class="d-flex flex-wrap"></div>
                    </div>
                    <button type="submit" class="btn btn-primary">Submit</button>
                </form>
                <div id="message" class="mt-3"></div>
            </div>
        </div>

        <!-- Compare Options -->
        <div class="card mb-4">
            <div class="card-body">
                <h5 class="card-title">Compare Options</h5>
                {% if categories %}
                    <a href="/analyze_directions" class="btn btn-outline-secondary mb-2">Compare Categories</a>
                    <div class="d-flex align-items-center compare-form">
                        <a href="#" id="compare-videos-btn" class="btn btn-outline-secondary mb-2">Compare Videos in Category</a>
                        <form     method="GET"     action="/compare_videos_in_category"     class="d-flex     align-items-center"
id="compareVideosForm">
                            <select class="form-select me-2" id="direction_id" name="direction_id" required>
                                <option value="">Select a category</option>
                                {% for category in categories %}
                                    <option value="{{ category.direction_id }}">{{ category.direction_name }}</option>
                                {% endfor %}
                            </select>
                            <button type="submit" class="btn btn-primary">Go</button>
                        </form>
                    </div>
                <a href="{{ url_for('predict_popularity') }}" class= "btn btn-outline-secondary mb-2" >Predict Popularity by Category
and Duration</a>
                {% else %}
                    <p class="text-danger">No categories available. Please add categories to the database.</p>
                {% endif %}
            </div>
        </div>

<!-- Generate Reports -->
<div class="card mb-4">
    <div class="card-body">
        <h5 class="card-title">Generate Reports</h5>
        <form method="GET" action="{{ url_for('generate_report', report_type='pdf') }}" style="display:inline;">
            <button type="submit" class="btn btn-outline-primary me-2">Generate PDF</button>
        </form>
        <form method="GET" action="{{ url_for('generate_report', report_type='xlsx') }}" style="display:inline;">
            <button type="submit" class="btn btn-outline-primary">Generate XLSX</button>
        </form>
    </div>
</div>



        <!-- Database Options -->
        <div class="card mb-4">
            <div class="card-body">
                <h5 class="card-title">Database Options</h5>
                <a href="/show_databases" class="btn btn-outline-secondary">Show All Databases</a>
            </div>
        </div>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

    <!-- Script for categories and form submission -->
    <script>
        // Load categories
        fetch('/get_categories')
            .then(response => response.json())
```

```
            .then(data => {
                const categoriesDiv = document.getElementById('categories');
                data.forEach(category => {
                    const checkbox = document.createElement('input');
                    checkbox.type = 'checkbox';
                    checkbox.name = 'categories';
                    checkbox.value = category.direction_id;
                    checkbox.className = 'form-check-input me-2';

                    const label = document.createElement('label');
                    label.className = 'form-check-label me-4';
                    label.innerHTML = `<strong>${category.direction_name}</strong>: ${category.description}`;

                    const wrapper = document.createElement('div');
                    wrapper.className = 'form-check form-check-inline';
                    wrapper.appendChild(checkbox);
                    wrapper.appendChild(label);

                    categoriesDiv.appendChild(wrapper);
                });
            })
            .catch(error => {
                console.error('Error loading categories:', error);
            });

        // Form submission
        document.getElementById('videoForm').addEventListener('submit', async (e) => {
            e.preventDefault();
            const messageDiv = document.getElementById('message');
            const selectedCategories = document.querySelectorAll('input[name="categories"]:checked');
            if (selectedCategories.length === 0) {
                messageDiv.className = 'error';
                messageDiv.textContent = 'Please select at least one category.';
                return;
            }

            const formData = new FormData(e.target);
            try {
                const res = await fetch('/submit_url', {
                    method: 'POST',
                    body: formData
                });
                const result = await res.json();
                messageDiv.className = result.error ? 'error' : 'success';
                messageDiv.textContent = result.error || result.message;
                if (!result.error) {
                    document.getElementById('youtube_url').value = '';
                }
            } catch (err) {
                messageDiv.className = 'error';
                messageDiv.textContent = 'Submission failed.';
            }
        });

        // Compare form check
        document.getElementById('compareVideosForm').addEventListener('submit', (e) => {
            const directionId = document.getElementById('direction_id').value;
            if (!directionId) {
                e.preventDefault();
                alert('Please select a category before proceeding.');
            }
        });
    </script>
</body>
</html>
Файл popularity_prediction.html
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Прогноз популярності відео</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        :root {
            --primary: #4F46E5;
            --background: #F9FAFB;
            --card-bg: #FFFFFF;
            --text-dark: #111827;
            --text-light: #6B7280;
            --border: #E5E7EB;
        }

        body {
            margin: 0;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: var(--background);
            color: var(--text-dark);
        }

        .container {
            max-width: 960px;
            margin: 0 auto;
            padding: 2rem;
        }

        a {
            text-decoration: none;
            color: var(--primary);
            font-weight: 500;
        }

        h2 {
            font-size: 1.75rem;
            margin-bottom: 0.5rem;
            color: var(--primary);
        }
```

```
        h3 {
            font-size: 1.25rem;
            margin-top: 2rem;
            margin-bottom: 0.5rem;
        }

        .recommendation {
            background-color: var(--card-bg);
            border: 1px solid var(--border);
            border-radius: 8px;
            padding: 1rem 1.5rem;
            box-shadow: 0 2px 6px rgba(0, 0, 0, 0.03);
        }

        .recommendation p {
            margin: 0.5rem 0;
            font-size: 1.05rem;
        }

        table {
            width: 100%;
            border-collapse: collapse;
            background-color: var(--card-bg);
            border: 1px solid var(--border);
            border-radius: 8px;
            overflow: hidden;
            margin-top: 1.5rem;
        }

        th, td {
            padding: 0.75rem;
            text-align: left;
            border-bottom: 1px solid var(--border);
        }

        th {
            background-color: #F3F4F6;
            font-weight: 600;
        }

        img {
            max-width: 100%;
            border-radius: 6px;
            margin-top: 1.5rem;
            box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);
        }

        .error {
            color: red;
            font-weight: bold;
        }

        @media (max-width: 640px) {
            table, thead, tbody, th, td, tr {
                display: block;
            }

            th {
                position: absolute;
                top: -9999px;
                left: -9999px;
            }

            tr {
                border: 1px solid var(--border);
                margin-bottom: 0.5rem;
                border-radius: 6px;
                padding: 0.5rem;
                background-color: white;
            }

            td {
                border: none;
                position: relative;
                padding-left: 50%;
                text-align: right;
            }

            td::before {
                content: attr(data-label);
                position: absolute;
                left: 0;
                width: 50%;
                padding-left: 1rem;
                font-weight: bold;
                text-align: left;
                color: var(--text-light);
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <a href="{{ url_for('index') }}">← Назад на головну</a>

        {% if error %}
            <p class="error">{{ error }}</p>
        {% else %}
            <h2>Прогноз популярності відео</h2>

            <div class="recommendation">
                <h3>Рекомендація</h3>
                <p>Створіть відео в категорії <strong>{{ top_category_name }}</strong> тривалістю <strong>{{ top_duration_range
}}</strong> для отримання максимальної кількості переглядів.</p>
                <p>Прогнозована кількість переглядів: <strong>{{ top_predicted_views }}</strong></p>
            </div>
```

```
            <table>
                <thead>
                    <tr>
                        <th>Категорія</th>
                        <th>Прогнозовані перегляди</th>
                    </tr>
                </thead>
                <tbody>
                    {% for item in data %}
                    <tr>
                        <td data-label="Категорія">{{ item.direction_name }}</td>
                        <td data-label="Прогноз">{{ item.predicted_views | round(0) }}</td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>

            {% if plot_img %}
                <h3>Графік прогнозованих переглядів</h3>
                <img src="{{ plot_img }}" alt="Графік прогнозу переглядів">
            {% endif %}
        {% endif %}
    </div>
</body>
</html>
Файл show_databases.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Show Databases</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        h1, h2, h3 {
            color: #333;
        }
        table {
            border-collapse: collapse;
            width: 100%;
            margin-bottom: 20px;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: center;
        }
        th {
            background-color: #f2f2f2;
        }
        .button {
            display: inline-block;
            padding: 10px 20px;
            background-color: #4CAF50;
            color: white;
            text-decoration: none;
            border-radius: 5px;
            margin: 10px 0;
        }
        .button:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <h1>List of databases and their contents</h1>
    <a href="/" class="button">Back to Home</a>

    {% for db, tables in db_data.items() %}
        <h2>Database: {{ db }}</h2>
        {% for table, data in tables.items() %}
            <h3>Table: {{ table }}</h3>
            {% if data.rows %}
                <table>
                    <thead>
                        <tr>
                            {% for column in data.columns %}
                                <th>{{ column }}</th>
                            {% endfor %}
                        </tr>
                    </thead>
                    <tbody>
                        {% for row in data.rows %}
                            <tr>
                                {% for cell in row %}
                                    <td>{{ cell }}</td>
                                {% endfor %}
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            {% else %}
                <p>There is no data in the table.</p>
            {% endif %}
        {% endfor %}
    {% endfor %}
</body>
</html>
```

ДОДАТОК Б

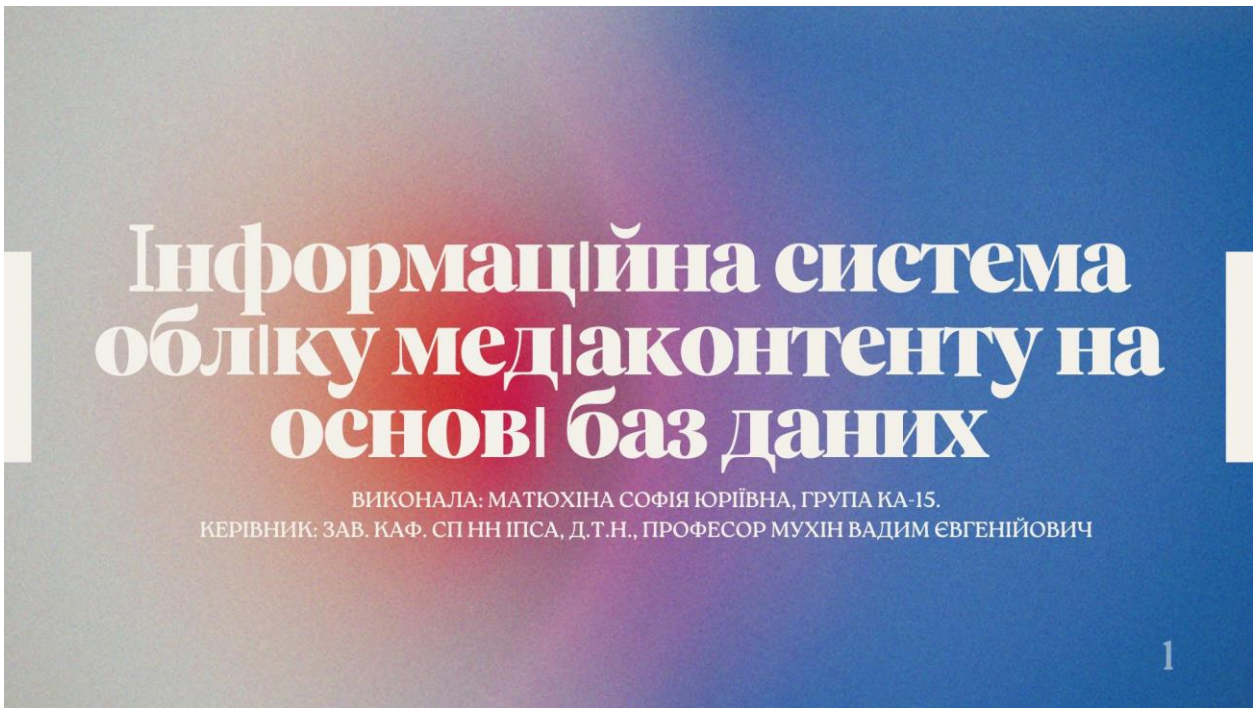


Рисунок Б.1 – Слайд 1

Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

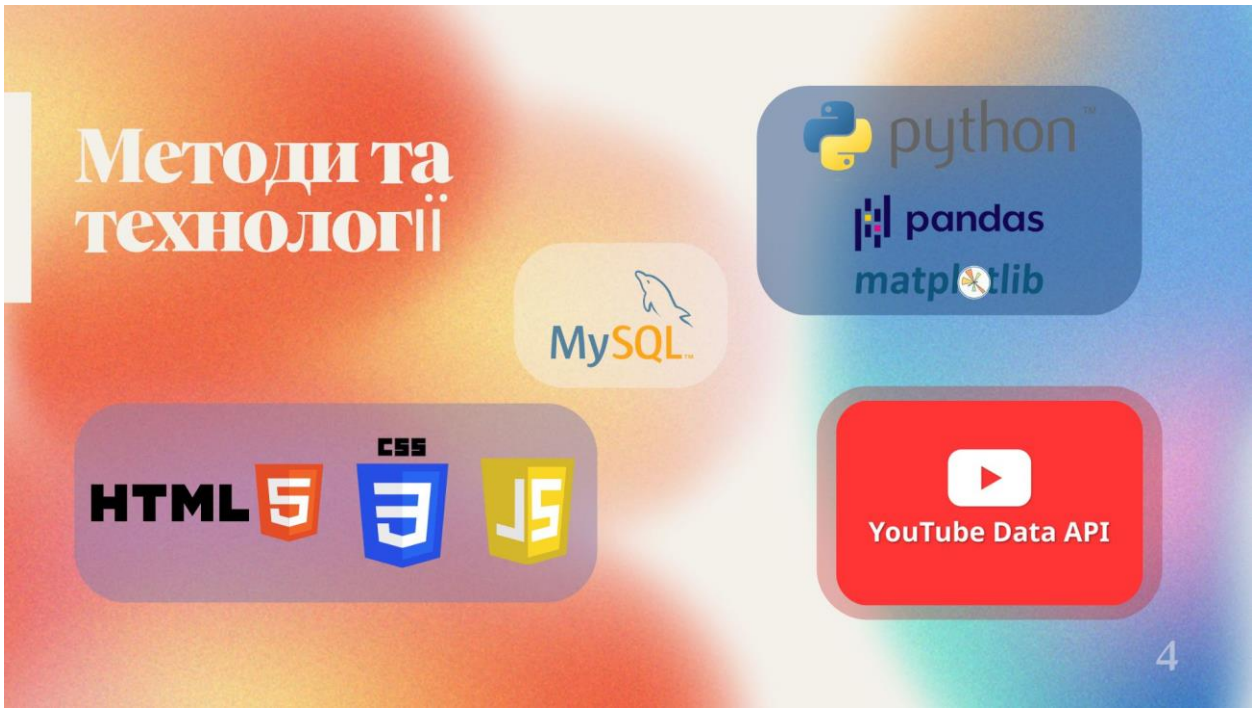


Рисунок Б.4 – Слайд 4

Рисунок Б.5 – Слайд 5

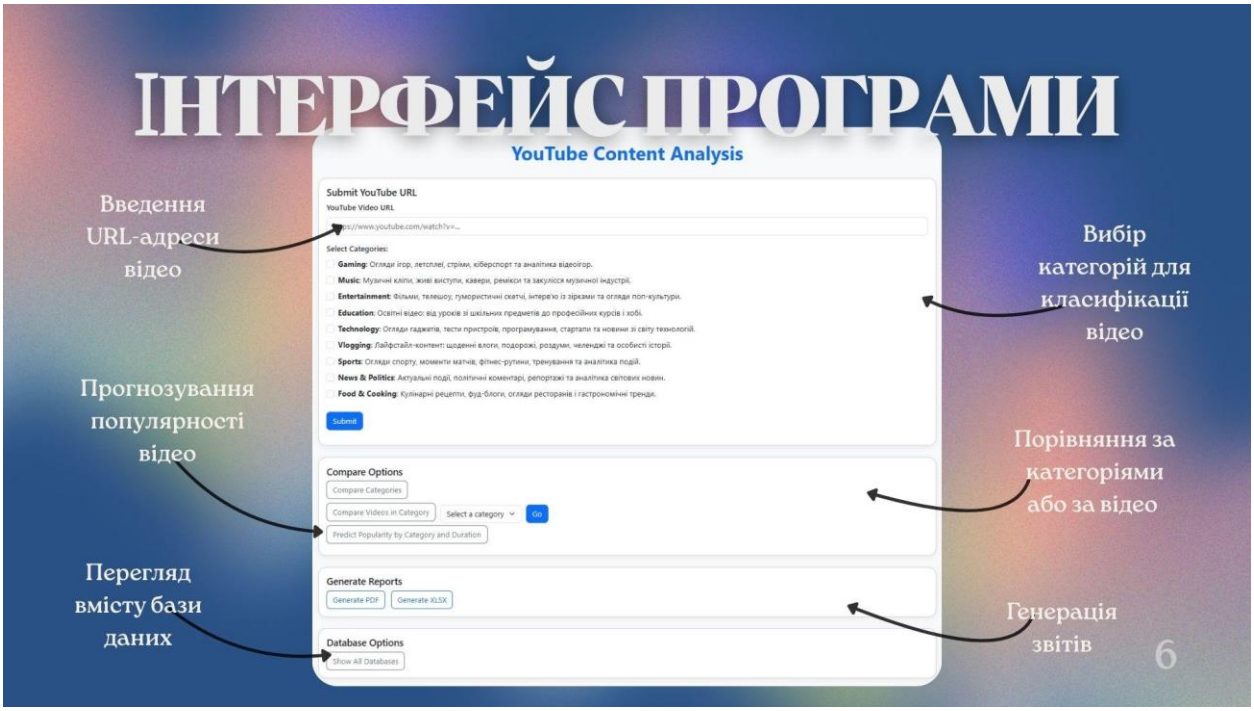


Рисунок Б.6 – Слайд 6

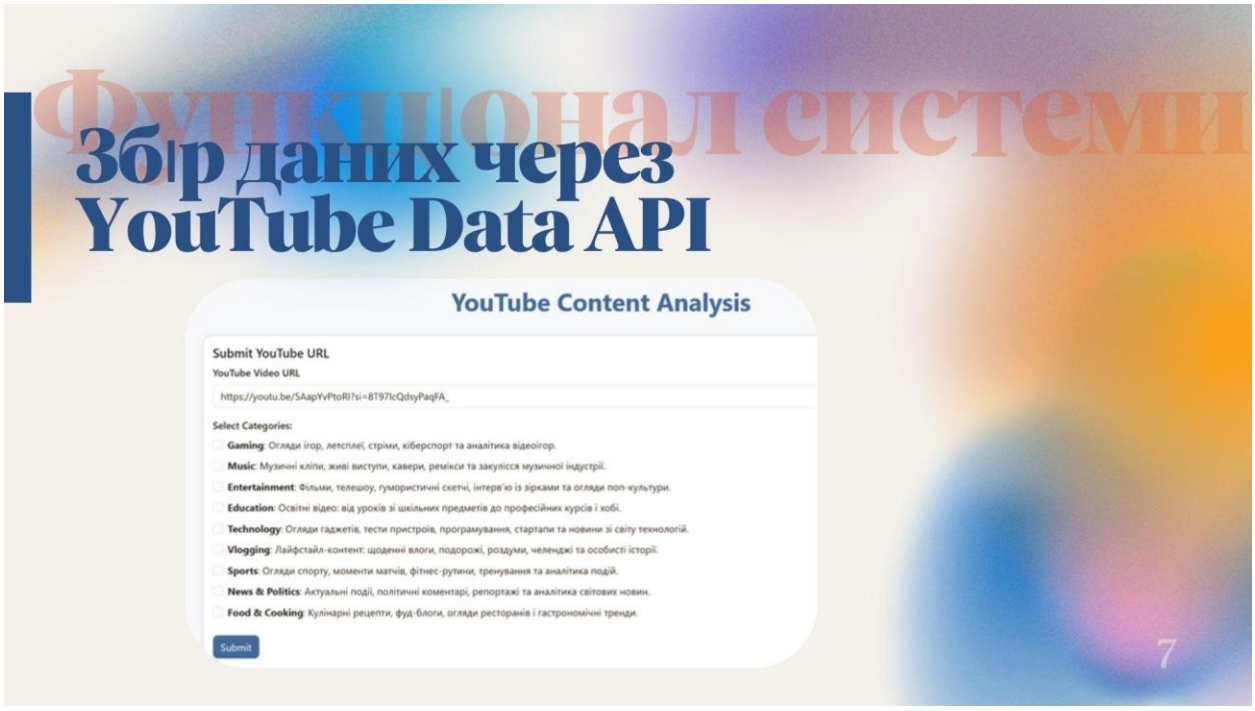


Рисунок Б.7 – Слайд 7

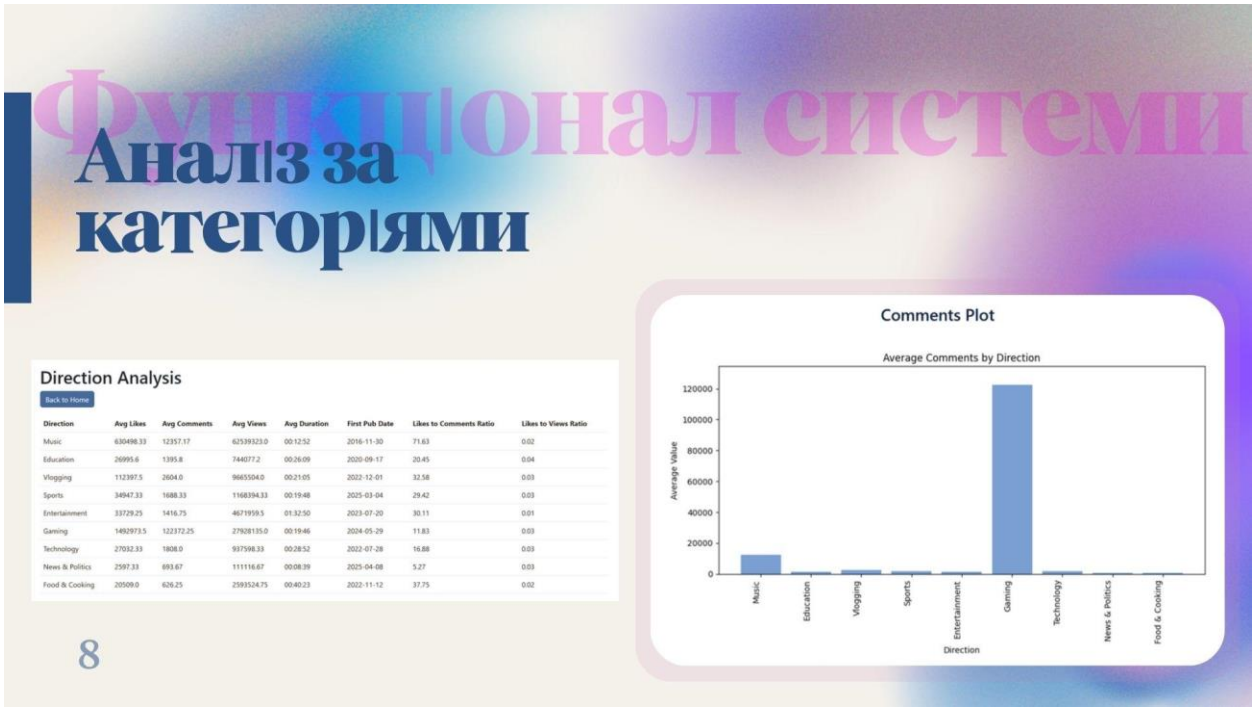


Рисунок Б.8 – Слайд 8

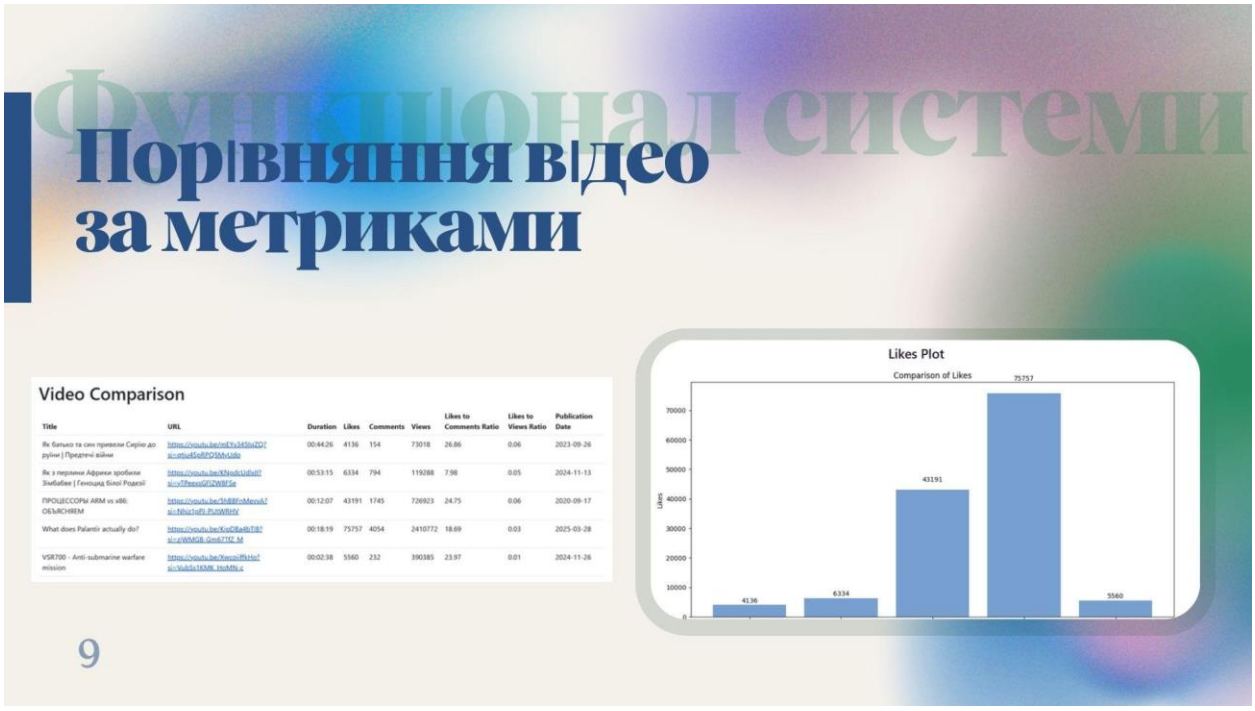


Рисунок Б.9 – Слайд 9

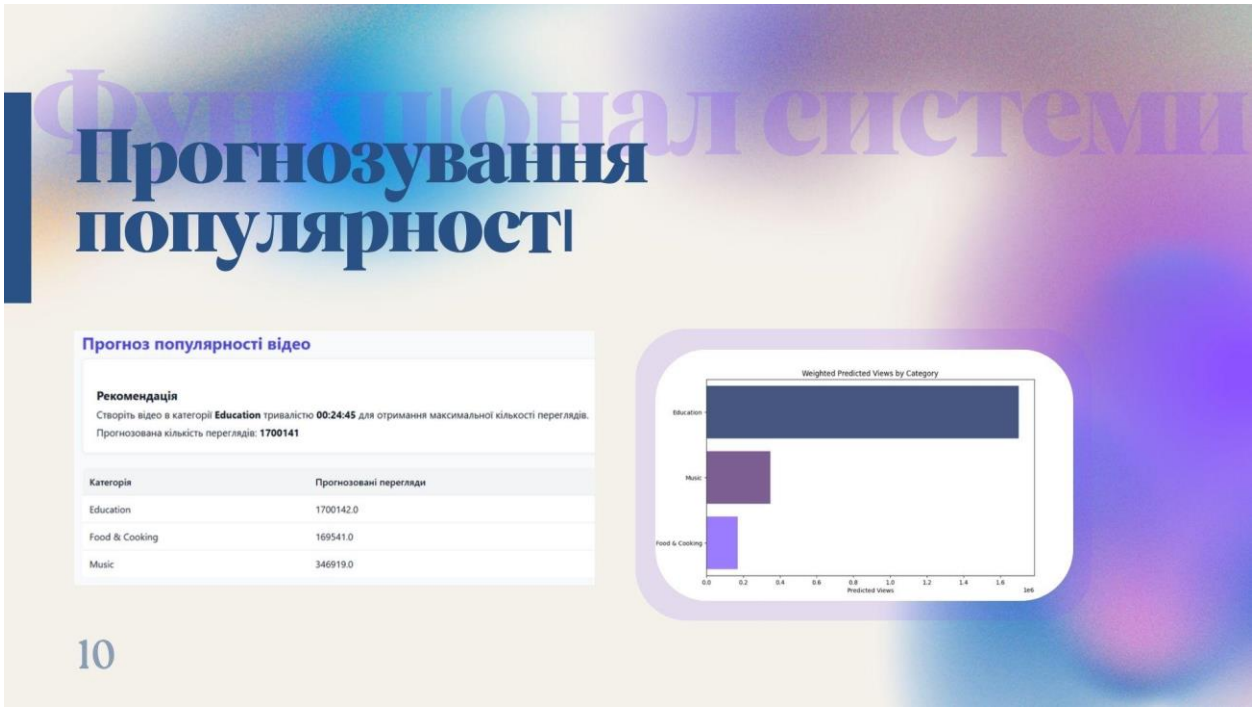


Рисунок Б.10 – Слайд 10

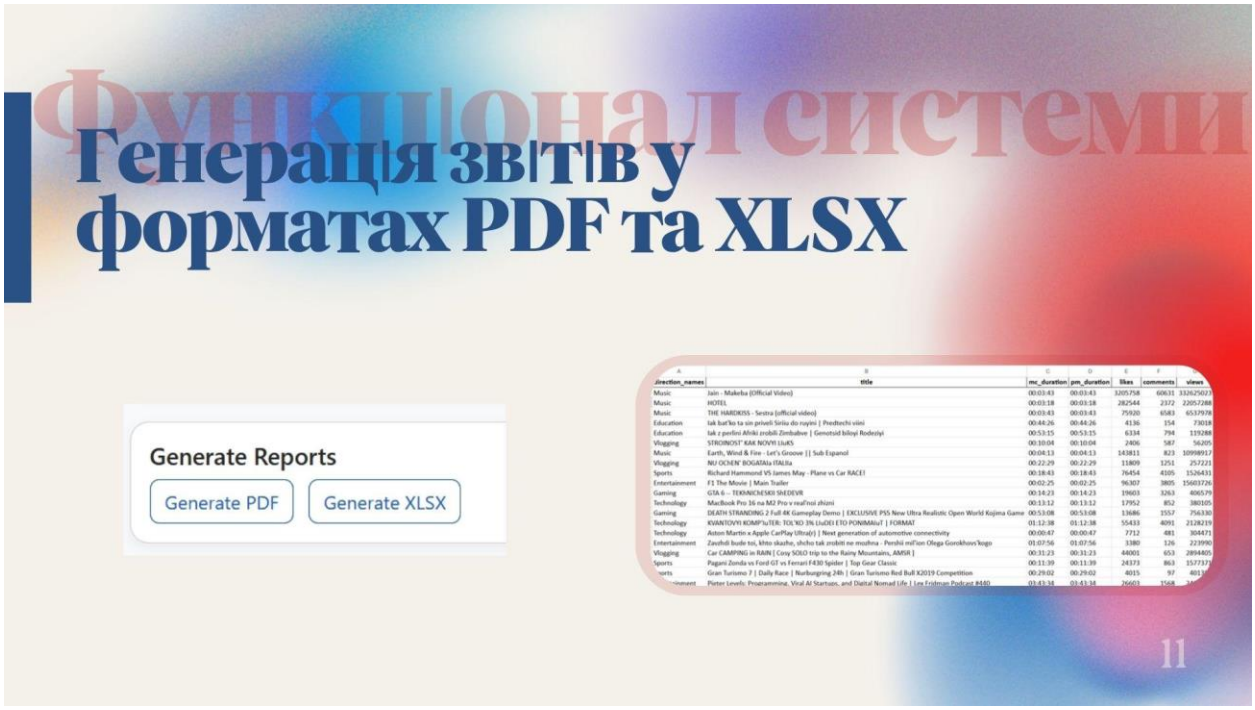


Рисунок Б.11 – Слайд 11

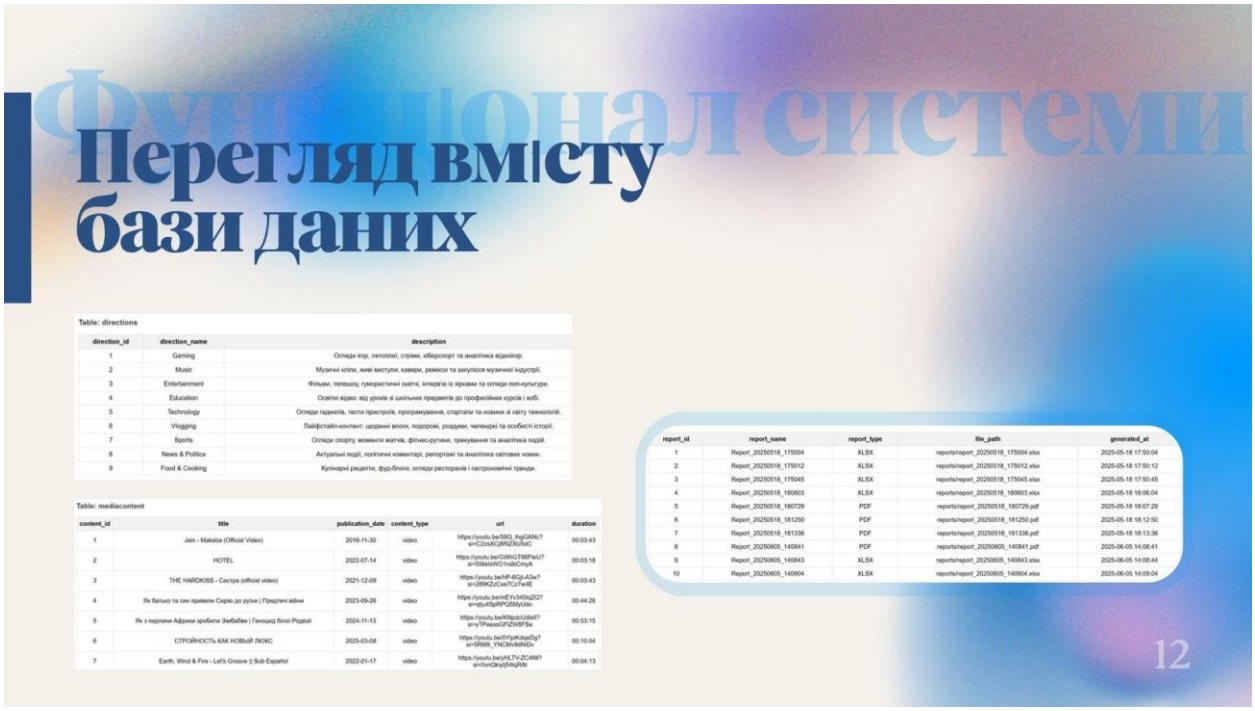


Рисунок Б.12 – Слайд 12

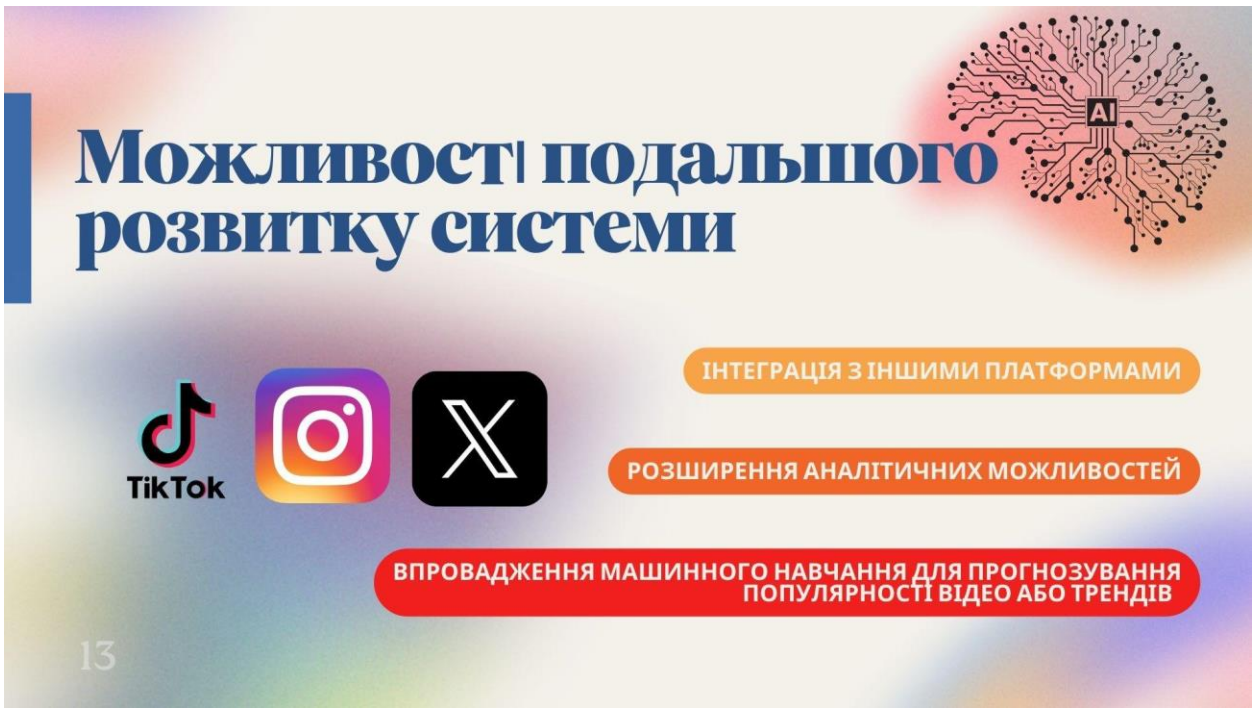


Рисунок Б.13 – Слайд 13

Рисунок Б.14 – Слайд 14