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

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Чуба Євгена Олексійовича*
	(ПІБ)
академічної групи	*122М-20-1*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«122 Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Розробка архітектури сховищ великих даних для системи дистанційної освіти*

_____*Є. О. Чуб*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Алексєєв М.О.*			
економічний	*Доц. Касьяненко Л.В.*			
Рецензент	*Проф.Коротенко Г.М.*			
Нормоконтролер	*Доц. Реута О.В.*			

Дніпро
2020

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

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

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

« » _____ 20 ____ Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122м-20-1* *Чубу Євгену Олексійовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка архітектури сховищ великих даних для системи дистанційної освіти*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. № 1036-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес накопичення великих даних в освітніх онлайн системах.

Предмет досліджень – методи проєктування, фреймворки, бази даних, модулі за допомогою яких розроблюються архітектури систем великих даних.

Мета роботи – розробка архітектури великих даних для дистанційної освіти шляхом пошуку, порівняння та поліпшення вже наявних рішень.

Вихідні дані для проведення роботи – теоретичні та експериментальні дослідження, основи побудови архітектур систем великих даних .

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає у розробці архітектури підсистеми великих даних з використанням зв'язки інструментів Hadoop, Apache Spark, MongoDB а також у створені моделі даних для системи дистанційної освіти.

Практична цінність результатів полягає у впровадженні системи яка буде збирати та аналізувати дані від користувачів за допомогою яких вона матиме змогу покращити результати навчання.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання розробленої архітектури. В результаті роботи повинна бути розроблена архітектура підсистеми великих даних для системи дистанційної освіти.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз предметної галузі та постановка завдання	12.09.2021-30.09.2021
Аналіз систем великих даних в освітніх системах	01.10.2021-31.10.2021
Розробка архітектури сховищ великих даних для системи дистанційної освіти	01.11.2021-16.12.2021

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки впровадження нових методик навчання за допомогою аналізу великих даних та поліпшення користуванням аналітичними даними.

Завдання видав _____ _____
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Чуб Є.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2021 р.*

Термін подання кваліфікаційної роботи до ЕК *19.01.2022*

РЕФЕРАТ

Пояснювальна записка: 86 стор., 36 рис., 2 таблиці, 4 додатка, 63 джерел.

Об'єкт дослідження: процес накопичення великих даних в освітніх онлайн системах.

Предмет дослідження: методи проєктування, фреймворки, бази даних, модулі за допомогою яких розроблюються архітектури систем великих даних.

Мета роботи: розробка архітектури великих даних для дистанційної освіти шляхом пошуку, порівняння та поліпшення вже наявних рішень.

Методи дослідження. Задля розв’язання поставлених задач використовувались методи порівняльного аналізу наявних архітектур інструментів та систем великих даних, а також методи моделювання об'єктно орієнтованого сховища великих даних для накопичення поточної інформації про освітні результати.

Новизна отриманих результатів полягає у розробці архітектури підсистеми великих даних з використанням зв’язки інструментів Hadoop, Apache Spark, MongoDB а також у створені моделі даних для системи дистанційної освіти.

Практична цінність результатів полягає у впровадженні системи яка буде збирати та аналізувати дані від користувачів за допомогою яких вона матиме змогу покращити результати навчання.

Область застосування. Спроєктована система може буде застосована як початкова архітектура для створення за її участі системи дистанційної освіти.

Значення роботи та висновки. В ході роботи були досліджені методи та засоби зберігання великих даних в освітніх системах. Були розглянуті різні комбінації інструментів для створення, актуалізації, збереження, аналітики та інших засобів обробки великих даних. Була обрана архітектура та комбінація інструментів за допомогою яких була розроблена підсистема великих даних системи дистанційної освіти.

Прогнози щодо розвитку досліджень. Необхідно дослідити ефективність прийнятого рішення протягом довшого проміжку часу ніж час розробки. Спланувати, розробити та налагодити інструменти аналізу великих освітніх даних.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ і тривалості його розробки, а також проведені маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: Hadoop, Spark, MongoDB, великі дані, архітектура великих даних, система дистанційної освіти.

ABSTRACT

Explanatory note: 86 pages, 36 figures, 2 tables, 4 appendices, 63 sources.

Object of research: the process of accumulation of big data in online educational systems.

Subject of research: design methods, frameworks, databases, modules with which the architectures of big data systems are developed.

Purpose of Master's thesis: develop a big data architecture for distance education by finding, comparing and improving existing solutions.

Research methods. In order to solve the set tasks, methods of comparative analysis of existing architectures of tools and systems of big data were used, as well as methods of modeling object-oriented repository of big data for accumulation of current information about educational results.

Originality of obtained results is in developing of the architecture of the big data subsystem using the connection of Hadoop, Apache Spark, MongoDB tools, as well as the created data models for the distance education system.

Practical value of the results lies in the implementation of a system that will collect and analyze data from users through which it will be able to improve learning outcomes.

Scope of application. The designed system can be used as an initial architecture to create a distance education system with its participation.

The value of the work and conclusions. In the course of the work the methods and means of storing big data in educational systems were investigated. Different combinations of tools for creating, updating, saving, analytics and other big data processing tools were considered. The architecture and combination of tools used to develop the big data subsystem of the distance education system was chosen.

Research forecast and development. It is necessary to investigate the effectiveness of the decision for a longer period of time than the time of development. Plan, develop and establish tools for analyzing large educational data.

In the Economic section were done calculations of the complexity of software development, the cost of creating software and the duration of its development, as well as marketing research of the market for the software product.

Keyword: Hadoop, Spark, MongoDB, big data, big data architecture, distance education system.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HDFS – Hadoop Distributed File System

RDBMS – Relational Database Management System

SQL – structured query language

ML – Machine learning

JSON – JavaScript Object Notation

XML – eXtensible Markup Language

OWL – Web Ontology Language

BSON – Binary JavaScript Object Notation

ЗМІСТ

ВСТУП

Розвиток світової інформаційної інфраструктури з лавиноподібним зростанням кількості інформації, що надходить від різних застосунків на кшталт соціальних мереж та месенджерів тощо, призвів до виникнення проблеми збеігання та обробки великих обсягів даних. Виявилось що наявні системи збереження даних до вирішення цієї проблеми не готові, тож перед фахівцями з інформаційних технологій постала задача розробки концепції сховищ великих даних та винайдення нових алгоритмів які б ефективно можно було застосовувати шо до даних у реальному часі.

Такі системи збереження отримали назву «сховища великих даних» і набули широкого розповсюдження у глобальних сервісах, зокрема соціальних мережах, месенжерах, державних реєстрах, онлайн енциклопедіях тощо. Одним із випадків застосувань сховищ великих даних є системи дистанційної освіти які мають не тільки зберігати освітній контент але й забезпечити зберігання та аналіз даних користувачів з метою поліпшення якості навчання кожного окремого студента. Саме створеню архітектури такої підсистеми великих даних присвячена ця кваліфікаційна робота.

Метою кваліфікаційної роботи є дослідити методи та засоби створення сховищ великих даних та розробити власну архітектуру підсистеми великих даних для системи дистанційної освіти яка б надала можливість збирати та колекціонувати інформацію про поточні результати навчання користувачів.

Кваліфікаційна робота складається з чотирьох розділів. У першому розділі будо розглянуто визначення поняття та характеристику великих даних, проаналізовані архітектури за допомогою яких проєктують системи великих даних, описані інструменти збирання та аналізу, а також сервери баз даних, які використовуються під час побудови систем великих даних, і крім, того, проведено порівняльний аналіз інструментів задля виявлення їх слабких та сильних сторін. У другому розділі розглянуто декілька прикладів реалізації

систем та архітектур великих даних, а також інструменти які використовувались під час їх створення. Третій розділ містить опис та реалізацію спроєктованої архітектури великих даних. У четвертому розділі виконано розрахунки вартості та тривалості розробки підсистеми та проведено маркетингове дослідження системи великих даних для дистанційної освіти.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні відомості з предметної галузі

Big Data (Великі дані) - це великі за об’ємом, складні набори даних, які надходять зі швидкістю пропорційній обсягу. Звичайні комп’ютери не мають потужностей для необхідної ефективної обробки та аналізу цих об’ємних наборів даних. Але ці об’ємні набори даних мають велику цінність через те, що містять дуже детальний опис процесів та явищ які їх продукують [1].

Системи великих даних використовують в різноманітних галузях нашого життя від медіа та комунікацій до страхування і торгівлі. Одним з найкращих прикладів систем великих даних є платформа YouTube, яка накопичує данні від клієнтів про їх симпатії чи антипатії, а потім, після аналізу поведінкової моделі, пропонує відео на вподобану їм тему [2].

Для опису великих даних використовують моделі, які складаються з кількох характеристик системи (рис. 1.1), які своєю чергу будуть накопичуватися у сховищі та будуть використовуватись надалі.

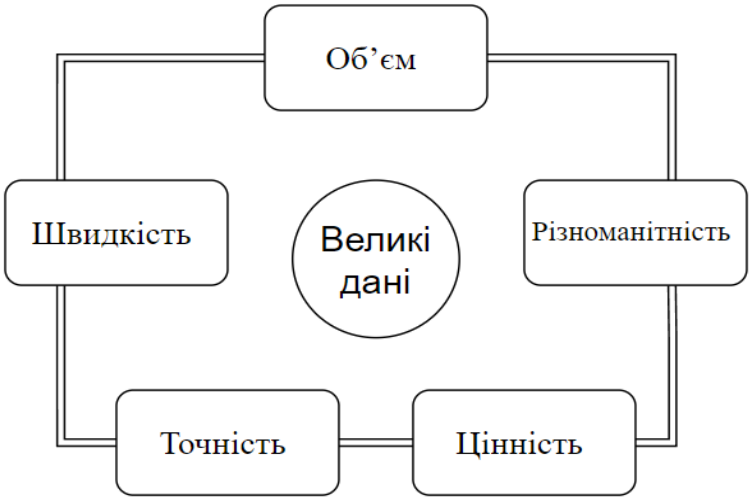

Рис. 1.1. Характеристики системи великих даних

Наведенні вище характеристики даних мають такий опис:

1. Об’єм – великі дані збираються з мільйонів різних ресурсів. Дані, які надходять, зазвичай мають низьку щільність і неструктуровані. Обсяг великих даних, що надходять від різних споживачів, залежить від характеристик їх систем.

2. Швидкість – цей аспект описує темп надходження даних і обміну ними або спільного використання. Швидкість великих даних зазвичай вимірюється в режимі реального часу.

3. Різноманітність – великі дані представлені різними типами, включаючи сирі, неструктуровані, структуровані та перевірені дані. Деякі з цих типів вимагають подальшої оцінки та аналізу, в той час, як інші готові до використання.

4. Точність – у великих даних існує аспект точності, який визначає вірогідність вхідних даних. Точність досягається шляхом концентрації на видаленні упереджень, невідповідностей, недоліків та, що більш важливо, дублювання.

5. Цінність – цей аспект описує різноманітність і послідовність способів отримання великих даних. Дані надходять періодично. Це може бути сезонно і викликано певним явищем.

1.2. Методи зберігання великих даних

У сучасному світі Великих даних виділяють два методи зберігання це локальні сховища та хмарні сховища.

Подібно до сховища для зберігання фізичних товарів, локальне сховище даних є об’єктом, основною функцією якого є зберігання та обробка даних на рівні підприємства. Ці великі сховища даних накопичують у собі різноманітні звіти, бізнес-аналітику, аналізи даних досліджень тощо. Ці сховища, як правило, оптимізовані для збереження та обробки чималих обсягів даних у будь-який

час, надаючи їх і виводячи через онлайн-сервери, де користувачі можуть отримати доступ до своїх даних без затримок [3].

Інструменти сховища даних дозволяють ефективніше керувати даними, оскільки дають змогу знаходити, отримувати доступ, візуалізувати й аналізувати дані, щоб приймати кращі бізнес-рішення та досягати бажаніших результатів. Крім того, вони створені з урахуванням експоненційного зростання даних.

Іншим методом зберігання великих обсягів даних є хмарне сховище. Завдяки хмарному сховищу дані та інформація зберігаються в електронному вигляді, до них можна отримати доступ з будь-якого місця, що не вимагає прямого доступу до жорсткого диска або комп’ютера. Завдяки такому підходу можна зберігати в Інтернеті практично безмежну кількість даних і отримувати до них доступ [4].

Хмара забезпечує не тільки легкодоступну інфраструктуру, але й можливість її швидко масштабувати, щоб керувати значним збільшенням трафіку або різким збільшенням даних.

Хмара також забезпечує легкий доступ і зручність її використання. Якщо користувач хоче отримати доступ до своїх даних у хмарі, все що йому потрібно зробити – це ввести свої облікові дані. Хмарні сховища значно покращили продуктивність і ефективність бізнесу, оскільки співробітники можуть миттєво ділитися файлами, отримувати до них доступ та редагувати їх віддалено [5].

На додачу хмарні сховища мають набагато меншу вартість при їх використанні у порівнянні з локальними сховищами, які своєю чергою потребують значних витрат ресурсів на приміщення, електроенергію та підтримку.

1.3Опис архітектур великих даних

Для прикладу розгляньмо як виглядають архітектури великих даних Lambda та Kappa.

Архітектура Lambda — це архітектура великих даних, яка розроблена для задоволення потреб у надійній системі, вона є відмовостійкою як проти збоїв обладнання, так і проти людських помилок. При цьому ця архітектура використовує переваги як пакетної, так і потокової обробки. По суті, архітектура складається з трьох рівнів, включаючи рівень пакетної обробки, рівень потокової обробки (або в режим обробки в реальному часі) і рівень обслуговування. (рис. 1.2) Рівень пакетної обробки має дві функції: керування основним набором даних (незмінний набір необроблених даних, який тільки додається) та для попереднього обчислення пакетних переглядів. Основний набір даних зберігається за допомогою розподіленої системи обробки, яка може обробляти велику кількість даних. Пакетні перегляди створюються шляхом обробки всіх доступних даних. Таким чином, будь-які помилки можна виправити шляхом повторного обчислення на основі повного набору даних і подальшого оновлення наявних представлень.

Рівень потокової обробки обробляє потоки даних в режимі реального часу і працює лише з кінцевими даними. Загалом, є дві основні функції рівня потоку: зберігання переглядів даних у реальному часі та обробка вхідного потоку даних для оновлення цих переглядів. Рівень швидкої обробки компенсує високу затримку пакетного рівня, щоб забезпечити актуальні результати для запитів. Перегляд рівня швидкої обробки не такий точний і повний у порівнянні з переглядом пакетного рівня, але він доступний майже відразу після отримання даних [6].

Сервісний рівень індексує пакетні перегляди, щоб їх можна було запитувати випадковим способом із низькою затримкою. Запити об’єднуються за результатами обробки даних пакетного та швидкого рівнів, щоб відповідати

на випадкові запити, повертаючи попередньо обчислені представлення даних або створюючи представлення з оброблених даних.

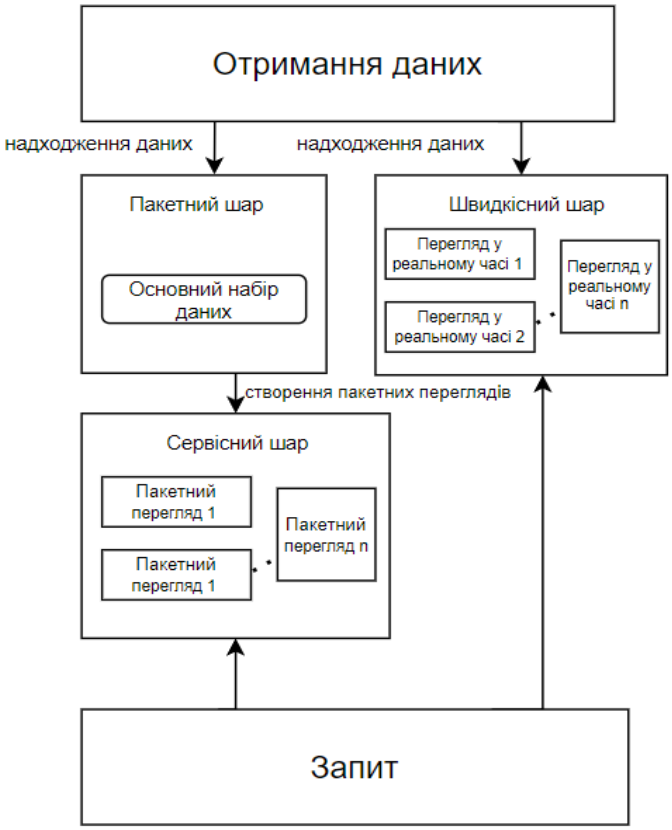

Рис. 1.2. Архітектура Lambda

Наступною з архітектур є Kappa архітектура (рис. 1.3). Причиною розробки архітектури Kappa було уникнення підтримки двох окремих кодових баз для пакетного та швидкого рівнів. Ключова ідея полягає в тому, щоб обробляти дані в режимі реального часу і повторно обробляти дані за допомогою єдиного механізму обробки потоку. Повторна обробка даних є важливою вимогою для того, щоб виявити вплив змін коду на результати. Як наслідок, архітектура Kappa складається лише з двох рівнів: потокової обробки та обслуговування. Рівень обробки потоку виконує завдання обробки потоку. Зазвичай для обробки даних в режимі реального часу виконується завдання обробки одного потоку. Повторна обробка даних виконується лише

тоді, коли потрібно змінити деякий код завдання обробки потоку. Це досягається шляхом запуску іншого модифікованого завдання обробки потоку та відповіді на всі попередні дані. Нарешті, подібно до архітектури Lambda, рівень обслуговування використовується для запиту результатів.

Рис. 1.3. Архітектура Kappa

Для полегшення проєктування архітектур великих даних виділяють такі основні компоненти:

1. Збір даних. Джерела даних не входять в архітектуру Big Data, але до неї включаються програмні та технічні засоби, здатні здійснювати збір. Різноманітність способів збору даних безпосередньо залежить від їхнього джерела. Також впливає і природа інформації, що підлягає збору та подальшому аналізу. Виділяють наступні види даних які підлягають збору:

– структуровані дані (архіви даних, внутрішні системи підприємств тощо),

– неструктуровані дані (аудіо та відеофайли, текстові файли, дані GPS тощо);

– частково структуровані дані (дані мережевих служб, дані журналів подій внутрішніх систем тощо).

2. Сховище даних. Усі зібрані дані розподіляються на зберігання та, залежно від типу даних, опиняються у розподілених/нерозподілених сховищах або фіксуються в окремих журналах запису подій.

3. Перетворення даних. Перед передачею даних на стадію обробки вони повинні бути перетворені на зрозумілий для програм формат за допомогою інструментів імпорту/експорту. Такі інструменти можуть бути як вбудованими у самі застосунки, що відповідають за зберігання даних, так і зовнішніми, тобто доданими або створеними фахівцями [7].

4. Обробка даних. На цьому етапі відбувається об'єднання всіх зібраних даних. Обробка може проходити пакетами (сегментовано обробляється встановлений обсяг даних) або в режимі реального часу (обробляються всі дані без формування пакетів). На цьому етапі виділяються корисні для подальшого аналізу відомості.

5. Аналіз даних. Інструменти, що використовуються на цьому етапі, залежать від цілей користувача. Слід зауважити, що виділені на етапі обробки дані не підготованими до подальшої обробки: з ними можна працювати, але без обробки в контексті поставленої користувачем задачі вони не мають практичного застосування. Для вирішення окремих завдань можуть бути створені самостійні алгоритми аналізу, утиліти чи використані стандартні [8].

6. Виведення даних. Результати аналізу мають бути представлені у форматі, зручному для сприйняття користувачем. Це можуть бути таблиці, діаграми тощо. Залежно від виду та складності візуалізації до системи великих даних можуть бути додані додаткові застосунки, служби або надбудови.

1.4. Типи баз даних у системах великих даних

Розвиток технологій та Інтернету спонукає появу нових вебтехнологій і дешевих сховищ, а також появу систем спроможних зберігати чималий об’єм даних. Такі тенденції створюють нові виклики щодо зберігання, обробки,

аналізу та зображення даних. Великі набори даних охоплюють різноманітні структуровані, частково структуровані та неструктуровані дані, з якими важко працювати у реляційних базах даних таких як RDBMS та SQL [9].

Для розв'язання цієї проблеми були розроблені бази даних NoSQL. NoSQL бази даних надають різні категорії, які відповідають вимогам різних випадків використання, серед них виділяють такі як файлові, графічні, стовпчасті бази даних. NoSQL – це система керування базами даних, яка підходить для розподілених систем і не вимагає реляційного зберігання даних. Варто наголосити на тому, що системи NoSQL доповнюють і не замінюють СУБД з її мовою запитів SQL. Важливо також зазначити, що рішення NoSQL не були створені для того, що і рішення на основі SQL. Реляційні бази даних в основному призначені для структурованих даних і обробки транзакцій, тоді як бази даних NoSQL були створені для розв'язання проблем зберігання масивних неструктурованих наборів даних. Поширення хмарних технологій, особливо баз даних як послуги, і нагальна потреба у швидких, масштабованих і дешевших базах даних для обробки великих обсягів даних сприяли поширенню баз даних NoSQL, а також необхідність зберігати дані у звичайній структурі для підтримки об'єктноорієнтованих принципів, уникаючи дорогих об'єктно-реляційних баз даних. Іншою тенденцією до поширення не реляційних баз даних став розвиток вебтехнологій і хмарних обчислень, які потребують низьких витрат на адміністрування та високої масштабованості [10].

Бази даних RDBMS і SQL були панівною моделлю протягом багатьох років. Але вони можуть обробляти лише один тип попередньо визначеної схеми та підтримувати лише структуровані набори даних. Вони гарантують атомарність, узгодженість, ізольованість та довговічність, які важливі для багатьох застосунків. Насправді властивості ACID є однією з ключових функцій для надійного захисту онлайн-транзакцій. З іншого боку, NoSQL більше підходить для не реляційних, неструктурованих наборів даних. NoSQL пропонує дешевший спосіб управління великими даними за допомогою

кластерів і стандартних серверів. У табл. 1.1 представлено порівняння між базами даних SQL і NoSQL [11].

Таблиця 1.1

Таблиця порівняння SQL та NoSQL баз даних

	SQL	NoSQL
Типи	Один тип (бази даних на основі таблиць)	Багато різних типів: ключ-значення, документ, графічні бази даних
Схеми	Попередньо визначена схема	Динамічна схема для неструктурованих даних
Властивості	Атомарність, послідовність, ізоляція та довговічність	Послідовність, доступність і толерантність до розподілу
Масштабування	Вертикально масштабовані	Горизонтально масштабовані
Безпека	Обмежені механізми безпеки	Авторизація та аутентифікація, відсутність шифрування, кілька інтерфейсів збільшують шанси на атаки
Складний запит	Добре підходить для інтенсивних складних запитів	Не підходять для складних запитів
Мова запитів	Стандартизована мова SQL	Спеціальні мови запитів для кожної бази даних.
Режим розвитку	Поєднання відкритого та комерційного продукту	В основному з відкритим кодом
Підтримка	Відмінна підтримка	Обмежена підтримка на основі спільнот

Є три основні проблеми, з якими RDMS стикається під час роботи з великими даними та деякими вебзастосунками. Це масштабування даних, продуктивність окремих серверів, негнучкий дизайн схеми. Своєю чергою бази NoSQL надають велику масштабованість та продуктивність, що робить їх найкращим вибором для системи великих даних.

1.5. Опис програмних засобів які використовуються при створенні систем великих даних та їх порівняння

1.5.1 Фреймворки

При розробці систем великих даних використовують багато модульних додатків та фреймворків, таких як Apache Hadoop (рис. 1.4), Apache Spark (рис. 1.5).

Рис. 1.4. Apache Hadoop

Apache Hadoop – це фреймворк з відкритим кодом, який використовується для ефективного зберігання та обробки великих наборів даних розміром від гігабайта до петабайта. Замість того, щоб використовувати один комп’ютер для зберігання та обробки даних, Hadoop дозволяє об’єднувати кілька комп’ютерів у кластери для швидшого паралельного аналізу масивних наборів даних [12].

Hadoop складається з чотирьох основних модулів: Hadoop Common, HDFS, YARN та MapReduce.

Hadoop Common є набором бібліотек програмних модулів, сценаріїв виконання завдань та допоміжних програм, які призначені для створення

програмної інфраструктури, що лежить в основі роботи всіх інших компонентів та продуктів.

Розподілена файлова система HDFS – забезпечує зберігання даних на вузлах кластера Hadoop як файлів. Завдяки дублюванню інформації в HDFS досягається висока надійність зберігання навіть дуже великих файлів: у разі недоступності або виходу з ладу одного вузла кластера Hadoop, дані будуть вилучатися з блоків на інших вузлах [13].

Система планування завдань та управління кластером YARN – це модуль, який дозволяє керувати обчисленнями на кластері, у тому числі й наданням його ресурсів окремим розподіленим програмам. Що важливо, ці обчислення можуть виконуватись одночасно на безлічі вузлів, завдяки чому досягається висока швидкість обчислень. YARN виконує функцію програмного шару між фізичними ресурсами кластера та програмами, що з ним працюють.

Hadoop MapReduce – це фреймворк на базі YARN, що реалізує відомий підхід до організації розподілених обчислень. Дані у ньому спочатку розподіляються на безліч вузлів кластера, де паралельно запускається їх попередня обробка, після чого отримані результати передаються на центральний вузол кластера, який забезпечує отримання підсумкових результатів [14].

Основними перевагами Hadoop є характеристики вартості, масштабованості, гнучкості та швидкості.

Hadoop постачається з відкритим вихідним кодом і використовує економічно вигідне обладнання, яке допомагає отримати більший економічний зиск, на відміну від традиційних реляційних баз даних, які вимагають коштовного обладнання та високоякісних процесорів для роботи з великими даними. Проблема традиційних реляційних баз даних полягає в тому, що зберігання великого обсягу даних нерентабельно, тому що компанія, яка використовує реляційні бази даних, має час від часу видаляти дані, що може призвести до втрати цінної інформації. Це означає, що Hadoop надає дві основні

переваги за вартістю. По-перше, це відкритий код, по-друге, це використання спеціалізованого обладнання для систем великих даних [15].

Hadoop — це високо масштабований фреймворк, у якому велика кількість даних поділена на кілька машин у кластері, які своєю чергою обробляють їх паралельно. Кількість машин або вузлів може бути збільшена або зменшена відповідно до вимог підприємства. Традиційні СУБД не можна масштабувати, щоб наблизитися до вимог систем великих даних.

Hadoop розроблено таким чином, що він може ефективно працювати з будь-якими наборами даних, такими як структуровані, напівструктуровані, неструктуровані дані. Це означає, що він може легко обробляти будь-які дані незалежно від їх структури, що робить його надзвичайно гнучким. Оскільки Hadoop може легко обробляти великі набори даних, то підприємства можуть використовувати його для аналізу цінних даних з таких джерел, як соціальні мережі, електронна пошта тощо [16].

Hadoop використовує розподілену файлову систему для керування своїм сховищем HDFS. У HDFS файл великого розміру подрібнюється на блоки файлів невеликого розміру, а потім розподіляється між вузлами, доступними в кластері Hadoop, оскільки ця чимала кількість файлових блоків обробляється паралельно, завдяки чому і забезпечується висока продуктивність, у порівнянні з традиційними системами управління базами даних. Коли користувач матиме справу з обширною кількістю неструктурованих даних, швидкість є важливим фактором, за допомогою Hadoop він може легко отримати доступ до великих даних всього за кілька хвилин [17].

Але своєю чергою жодна система не може існувати без недоліків тож Hadoop не може ефективно працювати з невеликою кількістю файлів великого розміру. Файли у ньому зберігаються у вигляді блоків розміром від 128 МБ до 256 МБ.

Hadoop розроблений для роботи з великими наборами даних, тому його можна ефективно використовувати для організацій, які генерують чималий

обсяг даних. Його ефективність знижується під час роботи в середовищі невеликих даних.

За замовчуванням функція безпеки в Hadoop недоступна. Що спонукає користувачів Hadoop використовувати стороні застосунки.

Apache Spark – це фреймворк для обробки великих даних, який може швидко виконувати завдання обробки наборів даних, а також може розподіляти завдання обробки даних між кількома комп'ютерами таким чином, щоб вони оброблювалися самостійно або разом з іншими розподіленими обчислювальними інструментами (рис. 1.5). Ці дві якості є ключовими для систем великих даних і машинного навчання, які вимагають упорядкування немалої обчислювальної потужності для роботи з великими сховищами даних. Spark також має простий у використанні API який знімає частину тягаря програмування з плечей розробників, та за допомогою якого абстрагує більшу частину важкої роботи розподілених обчислень і обробки великих даних [18].

Рис. 1.5. Apache Spark

Apache Spark має модулі Apache Spark Core, Spark SQL, Spark Streaming, MLlib для створення систем великих даних.

Apache Spark Core є базовим механізмом загального виконання для платформи Spark, на якому побудовані всі інші функції. Він забезпечує обчислення в пам'яті та посилання на набори даних у зовнішніх системах зберігання даних [20].

Spark SQL – це модуль Apache Spark для роботи з структурованими даними. Інтерфейси, які пропонує Spark SQL, надають йому більше інформації про структуру даних і обчислень, що виконуються.

Spark Streaming – цей компонент дозволяє обробляти потокові дані в реальному часі. Дані можна отримати з багатьох джерел, таких як MongoDB і HDFS. Потім дані можуть бути оброблені за допомогою складних алгоритмів і передані в файлові системи, бази даних.

MLlib (Бібліотека машинного навчання) – ця бібліотека містить широкий набір алгоритмів машинного навчання класифікацію, регресію, кластеризацію та спільну фільтрацію. Вона також включає інші інструменти для побудови, оцінки та налаштування конвеєрів ML [23].

При користуванні фреймворком Spark виділяють наступні позитивні сторони:

– Spark швидко обробляє дані, оскільки має технологію обчислень в пам’яті, тоді як MapReduce який використовується у Hadoop має технологію обчислень на диску. Spark зберігає всі свої дані в пам'яті для всіх обчислень, тому кількість операцій введення/виведення менше, що значно збільшує швидкість обробки за допомогою цього інструменту.

– Spark підтримує багато мов, якими користувач може створювати додатки, наприклад Java, Python.

– Spark може легко інтегруватися майже з усіма технологіями великих даних, що робить його найкращим інструментом для системи великих даних, він також забезпечує 128-бітне шифрування та підтримку SSL для своєї мережі.

Серед недоліків Spark виділяють те що в ньому немає системи керування файлами, яку можливо інтегрувати з іншими платформами. Отже, це залежить від інших платформ, таких як Hadoop або будь-яка інша хмарна платформа для системи керування файлами. Spark не підтримує пакетну обробку даних. У пакетному режимі потік даних у реальному часі поділено на пакети, відомі як Spark RDD (Resilient Distributed Database). Для їх обробки застосовуються такі

операції, як приєднання, зображення, зменшення тощо. Після обробки результату дані знову перетворюється на пакети. Таким чином, пакетна передача Spark – це просто мікропакетна обробка [24].

Ще одним недоліком є те, що під час роботи Spark потребує великої кількості оперативної пам’яті для обробки та стабільної роботи. Додаткова пам’ять, яка може знадобитися для роботи Spark, коштує дорого, що робить Spark економічно не вигідним.

1.5.2 Бази даних

Для збереження чималих обсягів інформації у системах великих даних використовують такі типи баз даних, як MongoDB (рис. 1.6), HBase (рис 1.7), CouchDB.

MongoDB – це база даних, яка забезпечує високу продуктивність, доступність і легку масштабованість. Це кросплатформна документно-орієнтована система баз даних, класифікована як база даних NoSQL. MongoDB є відносно новим продуктом у колі зберігання даних, але вона привернула чималу увагу своїм розподіленим сховищем значень ключів, можливістю обчислення MapReduce та функціями NoSQL, орієнтованими на документи. Через свої особливості MongoDB використовують у системах великих даних [25].

Рис. 1.6. MongoDB

Серед переваг MongoDB виділяють підтримку швидших запитів за допомогою індексів, що значно зменшує перевантаження вводу-виводу, яке зазвичай пов’язане з системами баз даних [26].

MongoDB також надає кілька корпоративних функцій, таких як висока доступність і горизонтальна масштабованість. Висока доступність досягається завдяки наборам реплік, які мають такі функції, як резервування даних і функцію автоматичного перемикання при збоях. Це гарантує, що система буде працювати, навіть якщо вузол у кластері виходить з ладу.

Крім того, MongoDB забезпечує підтримку кількох механізмів зберігання даних, що дає змогу користувачеві налаштувати базу даних відповідно до робочого навантаження, яку вона обслуговує. Деякі з найпоширеніших випадків використання MongoDB включають перегляд клієнтських даних, мобільних додатків та систем керування вмістом у режимі реального часу. Нарешті, MongoDB має структури вкладених об’єктів, індексовані атрибути масиву та інкрементальні операції [27].

Серед недоліків в MongoDB можна виділити те, що необхідно вручну обробляти операції з'єднання, що може призвести до зниження продуктивності. В наслідок того, що в MongoDB відсутні об’єднання, це призводить до надмірного споживання пам'яті, оскільки всі файли мають бути зчитані з диска.

Розмір документа, який надходить у базу, не може бути більше 16 МБ, а також вкладеність обмежена і не може перевищувати 100 рівнів.

HBase – це розподілена база даних із відкритим вихідним кодом. Вона була розроблена у 2008 році в рамках проекту Apache Hadoop. Побудована на основі HDFS, вона має кілька функцій, такі як операції в пам’яті, стиснення та фільтри Блума. Побудована на Java, HBase забезпечує підтримку зовнішніх API, таких як Scala, REST, тощо. Hbase пропонує окрему версію своєї бази даних, але вона в основному використовується для конфігурації розробки, а не в виробничих реаліях [28].

Рис. 1.7. HBase

Однією з сильних сторін HBase є використання HDFS як розподіленої файлової системи. Це дозволяє базі зберігати великі набори даних, навіть розміром в мільярди рядків, і надавати аналіз за короткий період часу. Ця підтримка розріджених даних разом із тим фактом, що їх можна розміщувати між стандартним серверним обладнанням, гарантує, що рішення буде економічно дуже ефективним, коли дані масштабуються до гігабайт або петабайт.

Ще однією перевагою є використання фільтрів Блума і кеша блоків, що також можна використовувати для оптимізації запитів. Їх тісна інтеграція з проектами Hadoop і MapReduce робить їх привабливим рішенням для дистрибутивів Hadoop.

Деякі з типових випадків використання HBase включають онлайн-аналітику журналів, дистрибутиви Hadoop, застосунки з чималим обсягом записів і програм, які потребують обробки великого обсягу даних.

До недоліків HBase відносять використання архітектури master-slave. Це також виявляється єдиним обмеженням, оскільки перехід від одного HMaster до іншого може зайняти певний час, що також може бути проблемою для продуктивності системи.

HBase не має мови запитів. Це означає, що для досягнення можливостей, подібних до SQL, необхідно використовувати оболонку HBase та додаткових

застосунків. Основною проблемою цього підходу є висока затримка при суміщенні цих технологій.

Іншим важливим фактором, який слід враховувати при виборі HBase, є його взаємозалежність від інших систем, таких як HDFS для зберігання та інших застосунків для керування статусом і метаданими. Отже, при розробці рішень архітектура може стати заскладною. У табл. 1.2 виконане порівняння головних функцій MongoDB та HBase.

Таблиця 1.2.

Таблиця порівняння MongoDB та HBase

	MongoDB	HBase
Архітектура	Файлова	Стопчкова
Реплікація	Master-slave	Master-slave
Мова програмування	C++	Java
Підтримувана мова програмування	C, C#, C++, Python	C, C#, C++, PHP

CouchDB (рис. 1.8) це база даних з відкритим вихідним кодом, механізм зберігання у якій заснований на документах з самостійними записами та без внутрішніх зв’язків, яка також розроблена для легкого масштабування на кількох вузлах. CouchDB забезпечує доступність даних. Таким чином, усі клієнти мають доступ до зчитування та запису. Представлена база даних базується на підході «map-reduce» та концепції перегляду. Запити у CouchDB виконуються через «перегляди». CouchDB зберігає дані на диску за допомогою файлів лише для накопичення [29].

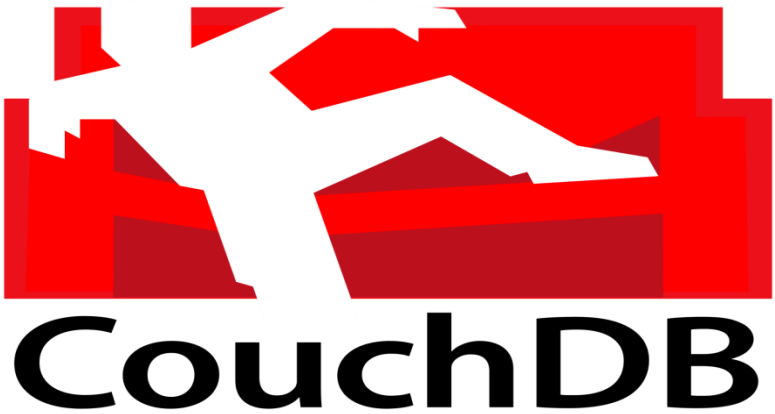

Рис. 1.8. CouchDB

Як формат обміну, CouchDB пропонує HTTP API як для доступу до даних, так і для адміністрування. Для стійкості до відмов CouchDB підтримує реплікацію master/master або master/slave. Реплікацію можна точно налаштувати за допомогою фільтрів.

Слід зазначити, що ця база даних має високу відмовостійкість, реплікацію з можливістю її налаштуваня, MapReduce та підтримку спільноти.

Однак CouchDB не адаптована до змінюваних даних, що робить її не дуже придатною до систем великих даних. У табл. 1.3 наведена порівняльна характеристика CouchDB і MongoDB.

Таблиця 1.3

Таблиця порівняння MongoDB та CouchDB

	MongoDB	CouchDB
Мова програмування	C++	Erlang
Відмовостійкість	Реплікація	Реплікація
Протокол	TCP/IP	HTTP/REST
Архітектура	Файлова	Файлова
Реплікація	Master-slave	Multi-master

1.6. Постановка задачі

Метою кваліфікаційної роботи є створення кращої архітектури великих даних для дистанційної освіти шляхом пошуку, порівняння та поліпшення вже наявних прикладів.

Для досягнення поставленої мети необхідно дослідити наявні інструменти реалізації систем великих даних, проектування архітектур, а також взаємодію інструментів між собою, розглянути приклади створених архітектур у сфері освіти та запропонувати власну розроблену архітектуру.

Після проведених досліджень необхідно проаналізувати зібрані відомості та на підставі результатів аналізу розробити власну архітектуру великих даних для дистанційної освіти.

1.7. Висновки

У цьому розділі були розглянуті та порівняні між собою інструменти розробки систем великих даних та наведені архітектури. Також наведено пояснення що таке великі дані та їх характеристики. Найкращими інструментами для створення систем виявились база даних MongoDB – як краще сховище для великих даних, фреймворк Hadoop – як набір модулів для пакетної обробки даних та фреймворк Spark, який має досить гнучкий функціонал для аналітики даних та потокової обробки даних.

РОЗДІЛ 2
СИСТЕМИ ВЕЛИКИХ ДАНИХ

2.1. Великі дані в освітніх системах

Впровадження комп’ютерних приладів у переважну більшість процесів реального світу, призвело до експоненційного збільшення кількості даних та інформації. У контексті різних явищ і факторів, аналіз та інтерпретація масивів даних, що генеруються, можуть пришвидшити прийняття рішень та покращити реалізації майбутніх пропозицій. Саме для цього і з’явилися системи, метою яких стала обробка та аналіз великих даних. Ці системи є набором технологій та інструментів, які дозволяють керувати, аналізувати, інтерпретувати та оцінювати великі обсяги даних та інформації, що генеруються різними способами. Тому цінність великих даних полягає в можливості аналізувати чималі обсяги інформації, які раніше не враховувались, через складність її збереження та аналізу. Це спонукало до відкриття нових знань, які слід враховувати при прийнятті рішень. Крім того, аналіз даних дозволяє зменшити суб’єктивність рішень, надаючи докази та дані, які можуть спростувати або підтвердити їх з більш об’єктивної точки зору. Тому мова йде не тільки про велику кількість даних, які необхідно розглянути, а й про типи зв’язків, які можна встановити між ними [30].

По-перше, тому, що можна зібрати та проаналізувати знання про те, яким чином конкретний студент набуває знання різного штибу. Крім того, можна зібрати та проаналізувати відомості про його помилки під час виконання практичних завдань, що дає можливість зробити висновки про засвоєння чи не засвоєння інформації певного типу. Також завдяки накопиченні інформації про різних користувачів можна створити та навчити різні математичні моделі, які дозволять врахувати певні особливості схожих за когнітивними параметрами учнів і, таким чином, зробити їхнє навчання ефективнішим.

2.2 Системи великих даних та опис їх архітектури

Наведемо декілька прикладів систем великих даних, проаналізуємо методи проєктування архітектур та застосунки, які використовувались при реалізації наведених систем.

Першою з розглянутих архітектур є архітектура системи рекомендацій курсів, яка розроблена для вищих навчальних закладів K. Dodouh [19].

У цій системі можна виділити наступну методологію роботи з даними: отримання, виявлення, підготовку, моделювання, обробку та візуалізацію результатів аналізу. Завдяки такому аналізу даних, система пропонує більш індивідуальний підхід у навчанні. На (рис. 2.1) схематично представлена методологія використання великих даних у наведеному середовищі дистанційного навчання.

Рис. 2.1. Методологія середовища дистанційного навчання

На першому етапі відбувається процес збору даних, створених середовищами дистанційної освіти, які є інформацією про персональні навички та дані учня, а також навчальні ресурси.

Другим етапом є підготовка даних, яка настає відразу після етапу отримання та зчитування. Це узгоджений набір операцій, які отримують, завантажують і перетворюють кілька джерел даних. Цей етап включає інтеграцію даних, які були зібрані з платформи дистанційного навчання, їх підготовку та передачу для зберігання в розподіленій файловій системі або базах даних NoSQL, MongoDB, HBase тощо. Цей етап є важливим кроком у процесі аналізу, тому що саме на цьому рівні зібрані дані повинні бути відфільтровані та збережені у релевантному вигляді.

На третьому етапі відбувається моделювання великих даних. Цей етап спрямований на визначення методу, який дозволить скористатися перевагами великих наборів даних. Саме для цього деякі фреймворки мають бібліотеки машинного навчання, зокрема класифікацію, кластеризацію, регресію та спільну фільтрацію [31].

Четвертим етапом йде обробка великих даних. Дані, які були завантаженні у сховище, розподіляються у пакетний, потоковий та інші рівні, де вони аналізуються, сортуються або агрегуються.

Останнім етапом є візуалізація оброблених даних. Цей етап також описує набір методів, програмного забезпечення та утиліт створених з метою допомогти фахівцям проаналізувати та мати чітке уявлення про дані, які були проаналізовані.

На (рис. 2.2) зображено структуру взаємодії застосунків великих даних.

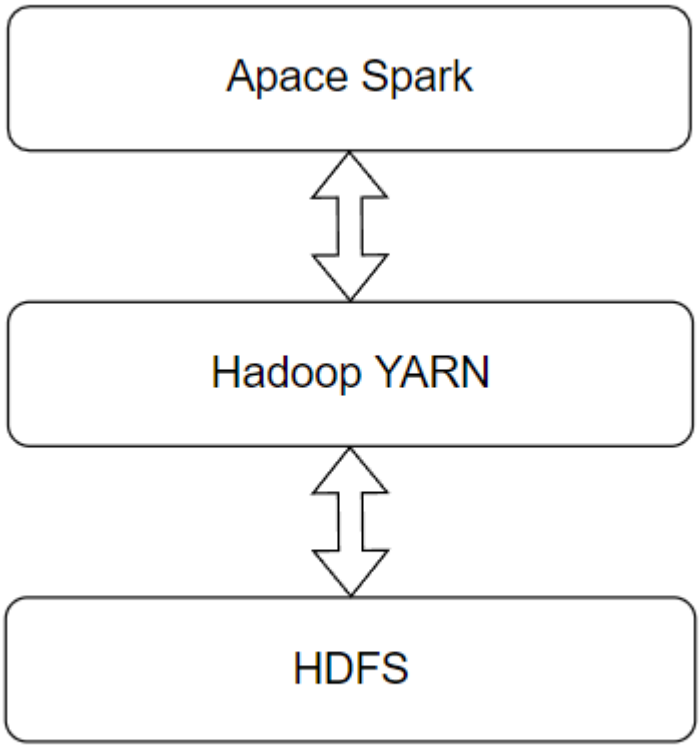

Рис. 2.2. Структура взаємодії застосунків великих даних

Реалізовані технології можна згрупувати в 3 шари. На першому кроці виконується розподілення збережених даних за допомогою файлової системи Hadoop (HDFS), приклад роботи якої наведено у першому розділі.

На другому рівні знаходиться модуль під назвою YARN, за допомогою якого здійснюється управління дисковими ресурсами, пам'яттю, центральним процесором і мережею кластерів. Основною функцією YARN є відокремлення управління ресурсами від обчислювальної моделі [38].

На третьому рівні представлений фреймворк Spark, який відповідає за обробку та аналіз даних. Він використовується для застосування методів правил асоціації до навчальних даних учнів, зібраних зі спроєктованої системи. Для системи рекомендацій курсів використовували мову Scala. Перевагою Spark є його здатність підтримувати кілька мов програмування, таких як Python

та R. На цьому ріні також використовувались модулі Spark SQL, Spark DataFrames і Spark MLlib [32].

Spark SQL виконує підключення до бази даних Moodle LMS і виконує SQL-запити. Spark DataFrames – це модуль, який займається обробкою структурованих даних. Spark MLlib реалізує кілька алгоритмів машинного навчання.

На (рис 2.3) детально описана архітектура спроєктованої системи.

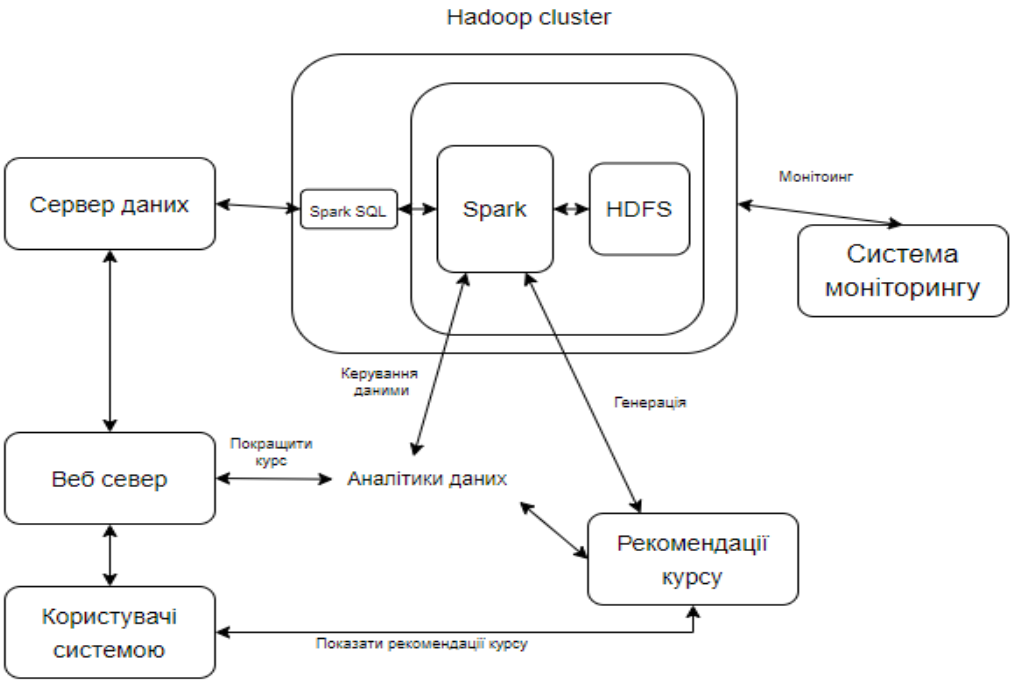

Рис. 2.3. Детально описана архітектура спроектованої системи

На першому кроці здійснюється завантаження даних, отримані в результаті взаємодії учнів з онлайн платформою. Потім дані, завантажені бібліотекою Spark SQL, обробляються розподіленим способом за допомогою Spark Framework, який працює в кластері Hadoop і використовує диспетчер ресурсів Yarn. Після обробки підготовлені дані аналізуються за допомогою бібліотеки Spark MLlib. Потім Spark підключається до Hadoop HDFS для запису

даних у кластери машин. На наступному кроці рекомендаційна система формує каталог найбільш актуальних курсів. Після аналізу отриманих даних система формує рекомендації щодо освітнього курсу, які найбільше відповідають вподобанням користувача [39].

На додачу, у представленій системі використовується інструмент з відкритим кодом Ganglia. Це застосунок для моніторингу великих середовищ даних, таких як кластери та сітки, або для вимірювання продуктивності та споживання ресурсів ЦП, пам'яті та зберігання даних кожного вузла кластера. Цей інструмент також контролює та візуалізує мережевий трафік.

У наступному прикладі реалізації системи великих даних для аналізу освітніх ресурсів буде розглянута система A. Caffai [21].

У цій системі було обрано систему LMS як джерело наборів даних. Набори даних складалися з ідентифікатора студента, екзаменаційної оцінки, загальної оцінки, щотижневої діяльності в освітній системі Moodle, середній бал, опитування, поведінкові дані. У контексті вищої освіти час прийняття рішення має вирішальне значення, оскільки дані студентів генеруються в режимі реального часу. Вимогами до вихідних даних, які висуваються у цій системі, є наявність інформації про курс і програму. Системи вебсайтів і порталів надають послуги представлення та доступу до інформації.

Для аналізу великих даних у вищій освіті потрібні дані про спостереження та взаємодії учасників навчального процесу. Функціональні можливості запропонованої аналітичної архітектури великих даних для вищої освіти показані на (рис. 2.4). Внутрішні джерела даних корисні у вирішенні низки проблем студентів, але, працюючи ізольовано та застосовуючи аналітику навчання лише до внутрішніх даних, вони можуть не дати покращень, які потребують студенти [40].

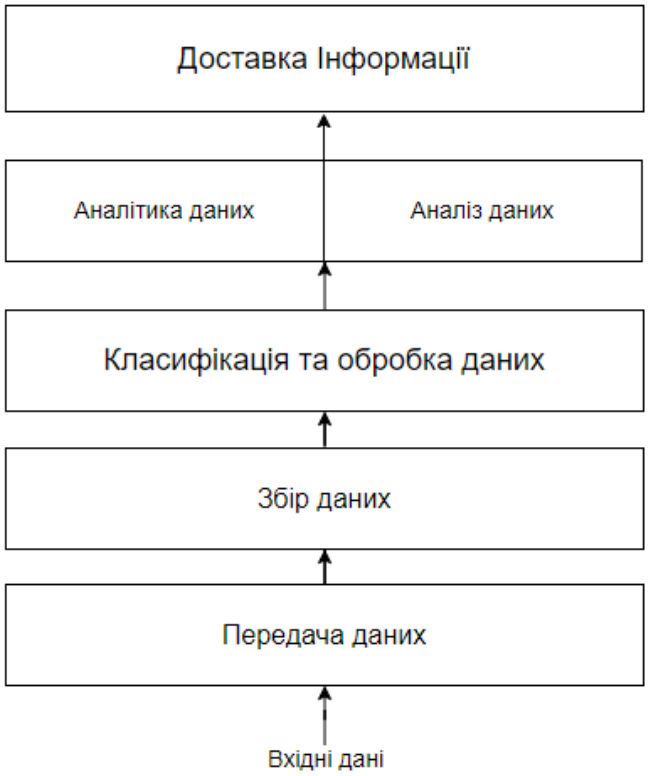

Рис. 2.4. Функціональні можливості запропонованої аналітичної архітектури

Шари, з яких складається запропонована архітектура великих даних, показана на рис. 2.5. У цій архітектурі були обрані інструменти для обробки різних форматів типів даних, що застосовуються при збереженні, обробці та аналізі даних.

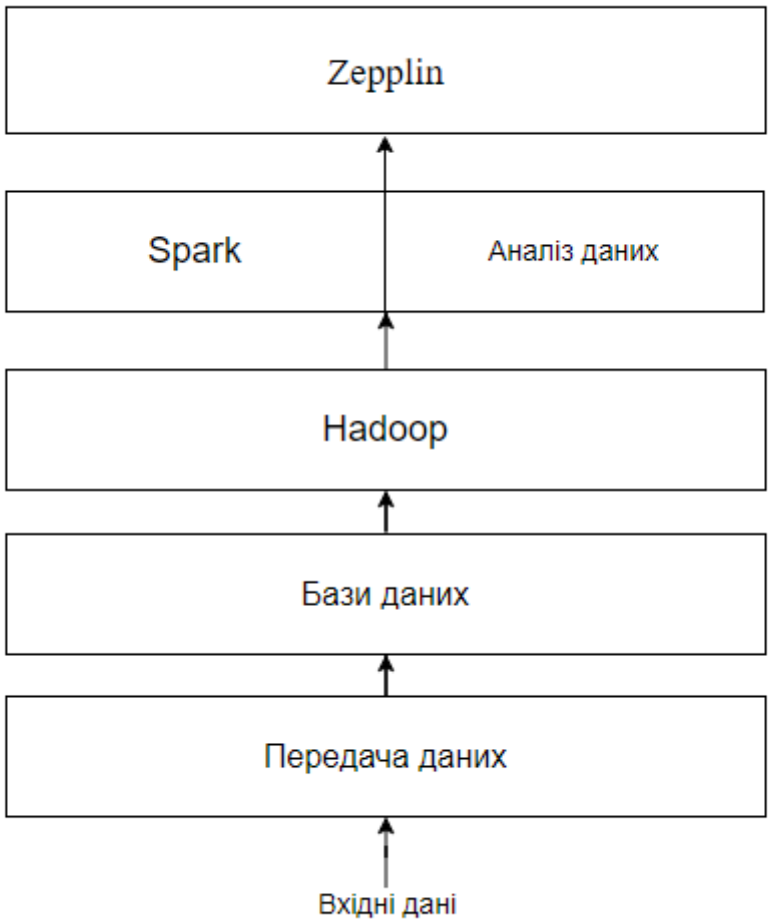

Рис. 2.5. Запропонована архітектура великих даних

Рівень збору даних у цій архітектурі містить всю інформацію, необхідну для обробки. Цей рівень містить усі необроблені дані, які накопичуються під час навчального процесу. Інструмент під назвою Apache SQOOP використовується на цьому рівні для передачі даних між реляційними базами даних та Hadoop. Рівень сховища містить усі зібрані дані в одному місці. Цей рівень служить системою для зберігання даних, а також системою для організації та категоризації їх у базах даних. На цьому рівні використовується HBase та Hadoop з його файловою системою HDFS. Для менеджменту та обробки даних використовується інструмент Spark, усі дані які надходять до нього передаються у вигляді стійких розподілених даних. Далі оброблені дані

зберігаються в HBase, яка використовується також як зовнішнє сховище. У рівні аналітики використовувались алгоритми машинного навчання.

На останньому рівні доставлення інформації використовувався інструмент візуалізації даних Apache Zepplin. Apache Zepplin – це інтерактивний браузерний програмний засіб який інтегрується з розподіленими системами обробки даних.

Розглянемо систему великих даних, яка стосується більше медичної сфери, ніж освітньої, але вона має досить цікавий підхід до проєктування архітектур великих даних. Як зазначає L.Benhlima спроєктована система може працювати з різними ситуаціями, такими як рання діагностика та екстрене виявлення захворювань [22].

Медичні дані, які збираються, надсилаються одночасно на пакетний рівень і потоковий рівень. У пакетному режимі дані зберігаються у вузлах даних, після чого передаються до семантичного модуля, який опрацьовує їх за допомогою онтологічного сховища, після цього до отриманих даних застосовуються операції очищення та фільтрації перед їх обробкою. На наступному кроці підготовлені дані аналізуються на різних етапах. Наприкінці підготовлені дані використовуються для розробки моделей прогнозування стану здоров’я пацієнтів у майбутньому. Цей режим періодично запитує дані в автономному режимі. У сценарії потоку дані надходять з кількох джерел, таких як медичні датчики, підключені до тіла пацієнта, які вимірюють кілька медичних параметрів, такі як кров’яний тиск тощо.

Медичні дані можуть включати структуровані дані, такі як традиційні електронні медичні записи, напівструктуровані дані, такі як журнали, створені деякими медичними пристроями, і неструктуровані дані, створені за допомогою біомедичних показників [37].

Першим кроком алгоритму є збір даних, мета якого полягає в тому, щоб зчитувати дані, зібрані з датчиків, у кількох форматах, а потім дані проходять через семантичний модуль перед нормалізацією. У цьому модулі створюються медичні онтології, які зберігають у собі знання, які своєю чергою допомагають

автоматизувати роботу медичного персоналу, яка у більшості випадків перенавантажена. Впровадження автоматизації в додатку охорони здоров'я покращує загальну продуктивність медичного персоналу. Для створення онтологій була використана мова вебонтології (OWL) яка являє собою стандартний формат обміну даними онтології, яка використовує синтаксис XML [33].

Наступним кроком є підготовка даних. Обробка необроблених даних без процедур підготовки може вимагати додаткових обчислювальних ресурсів, які недоступні в контексті великих даних. Підготовка даних у розглянутій системі складається з двох етапів: очищення даних і фільтрація даних.

Фільтрація даних досягається шляхом відкидання інформації, яка не є корисною. При очищенні даних виконується нормалізація, зменшення шуму та керування даними, які не потрібно оброблювати.

Для того, щоб усунути шум у даних та з'ясувати значення відсутніх даних, використовується кілька методів. Помилки при заповненні відсутніх значень можуть вплинути на якість отриманих знань і призвести до неправильних результатів.

Загалом, шум у даних оброблюється за двома основними підходами. Перший полягає у виправленні шумових значень на основі методів полірування даних. Наведені методи важко реалізувати й застосувати лише у випадку невеликої кількості шумних даних. Другий підхід базується на фільтрах шуму, які визначають і усувають шумові екземпляри в навчальних даних, і ці фільтри не вносять модифікацій у прийняті методи аналізу даних.

Коли етап підготовки даних проходить весь цикл обробки, дані зберігаються в підготовленому сховищі даних.

На наступному етапі відбувається побудова моделі, здатної давати прогнози для нових спостережень на основі попередніх даних. Якість заданої прогнозної моделі оцінюється за її точністю. Ці моделі розроблені на основі інструментів, доступних у сховищі статистики та машинного навчання, що

надається запропонованою архітектурою. Результати пакетної обробки будуть збережені в сховищі моделі.

Далі відбувається зберігання даних, яке є одним із найскладніших завдань у системах великих даних, особливо у випадку систем моніторингу які включають великі обсяги даних. Тому традиційний аналіз даних непридатний для керування цими системами. Цим компонентом може бути HDFS, NoSQL бази даних як MongoDB. У запропонованій системі дані пацієнтів, зібрані з різнорідних джерел, можна розділити на структуровані (дані про користувачів), неструктуровані (біомедичні показники) та напівструктуровані дані (документи XML і JSON). Ці дані будуть збережені в сховищі необроблених даних у цільових базах даних. Потокові дані, такі як соціальні мережі, будуть зберігатися в сховищі тимчасових даних потоку.

Рівень аналізу потокових даних складається з модуля синхронізації даних, модуля адаптивного препроцесора та модуля адаптивного предиктора.

Робота модуля синхронізації даних полягає в тому, щоб переконатися, що дані обробляються в правильному порядку щодо критерію часу. Крім того, процес синхронізації даних відхиляє вимірювання, які є непослідовними, і враховує відсутні значення. Виявлення неузгоджених значень здійснюється шляхом визначення порогових значень для вхідних параметрів.

У багатьох програмах передбачається, що завдання попередньої обробки даних буде виконується за допомогою алгоритмів навчання. На додаток, щоб залишатися надійним і підтримувати певний ступінь точності, прогнозні моделі повинні адаптуватися, коли відбуваються зміни даних. В результаті процес прогнозування можна розглядати як частину адаптивної системи, яка буде об'єднанням двох окремих частин, які об’єднують роботу препроцесора і предиктора.

Адаптивний препроцесор розбиває потік даних, що надходять у певний проміжок часу. Адаптивний препроцесор ділить потоки даних на підмножини кортежів, де кожен кортеж включений у кілька вікон. Для кожного вікна обчислюється середнє значення кожного показника та порівнюється з

попередньо визначеним порогом користувача. Якщо конкретне середнє значення перевищує тривожне порогове значення, воно буде збережено для створення екстреного сповіщення в іншому випадку просто зберігається. Коли порівняння з пороговими значеннями припиняються, фаза вилучення інформації триває шляхом вибору відповідних ознак, які будуть передані компоненту адаптивного предиктора [34].

Адаптивний предиктор повинен дотримуватися певного рівня точності, предиктори повинні оновлюватися відповідно до змін даних. Адаптивна функція вимагає встановлення зв’язку між адаптивним процесором і адаптивним предиктором. Предиктор надсилає зворотний зв’язок препроцесору чи потрібні оновлення чи ні, а потім блок попередньої обробки надасть йому необроблені дані. Результати обробки потоку будуть передані в сховище.

На наступному кроці починає свою роботу процесор запитів, який має знайти статус пацієнтів, об’єднавши відповіді на запити, надіслані як у сховище результатів потокової обробки, так і в сховище результатів пакетної обробки.

Після чого настає черга рівня візуалізації, який створює кілька вихідних даних, які включають візуалізацію звітів про моніторинг стану здоров’я пацієнта та звіт про прогнозні рішення [35].

На рис. 2.6 зображено запропоновану автором архітектуру великих даних для охорони здоров’я.

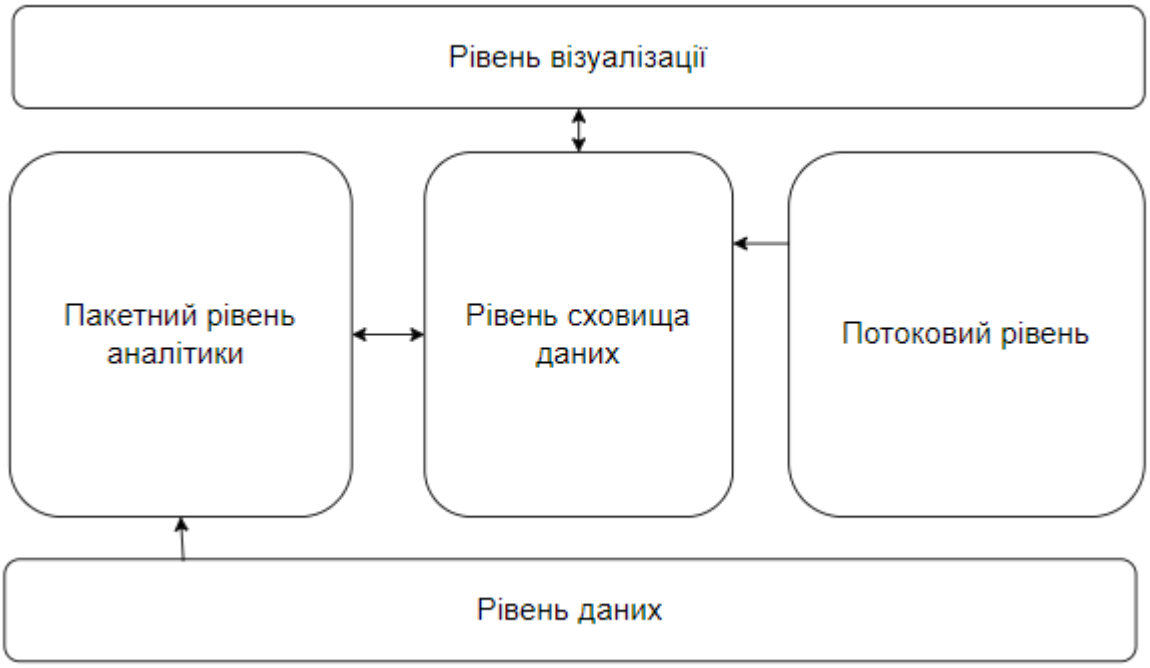

Рис. 2.6. Архітектура великих даних для охорони здоров’я

Також у цій архітектурі представлена система виявлення надзвичайних випадків яка буде виявляти потенційну небезпеку для пацієнтів. Для реалізації цієї системи автор використав Spark Stream і MongoDB. Система використовує Spark для зчитування даних з MongoDB на пакетному рівні. Spark Streaming використовується для обробки потоків даних у реальному часі, за допомогою чого він отримує дані з медичних джерел і виявляє нетипові ситуації на основі порогових значень користувача. Потім Spark Stream надсилає повідомлення до MongoDB, які будуть використовуватися для сповіщення лікарів про нетипові ситуації. Модулі Spark MLLib і Spark Stream використовуються для моніторингу в реальному часі та онлайн-навчання, щоб передбачити, чи є поточний стан пацієнтів небезпечним чи ні, що є контрольованою класифікацією. На рис 2.7 проілюстрована система виявлення надзвичайних випадків [36].

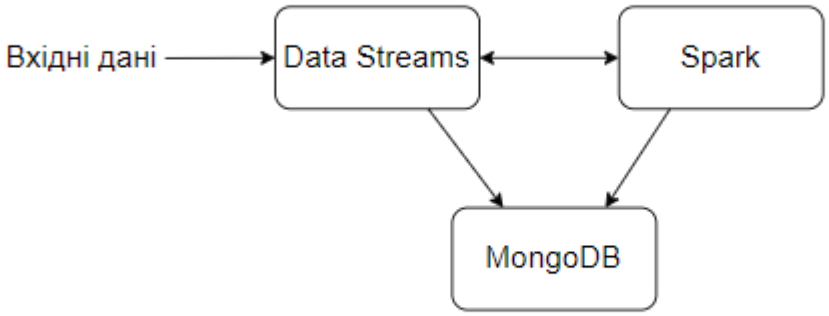

Рис. 2.7. Система виявлення надзвичайних випадків

2.3 Висновки

У цьому розділі були розглянуті приклади використання інструментів обраних у першому розділі та прийняте остаточне рішення щодо розробки архітектури великих даних за допомогою обраних інструментів. У представлених системах були описані архітектури, але жодна з них не підходить для реалізації розроблювальної системи. Але наведені архітектури мають певний недолік у використані HDFS або RDBMS як головного сховища даних, що є досить не вигідно. Якщо треба зберігати та швидко обробляти великі дані, то найкращим вибором для цього буде використання MongoDB. Отже, було прийнято рішення створити та описати власну архітектуру системи великих даних для дистанційної освіти.

РОЗДІЛ 3

РОЗРОБКА АРХІТЕКТУРИ СХОВИЩ ВЕЛИКИХ ДАНИХ ДЛЯ СИСТЕМИ ДИСТАНЦІЙНОЇ ОСВІТИ

3.1. Обґрунтування вибору архітектури та інструментів

Для розробки та знаходження оптимальної архітектури підсистеми великих даних був виконаний огляд модулів, фреймворків та типів архітектур, які були описані у першому розділі, а також вже реалізованих систем, які були описані у другому розділі. Виходячи з порівняння було прийняте рішення розробки системи на архітектурі Lambda за допомогою інструментів Hadoop, Spark та сховища даних MongoDB.

Вибір архітектури обґрунтований тим, що Lambda архітектура поєднує у собі два окремих шари, які можуть опрацьовувати дані не залежно один від одного як в реальному часі, так й в пакетному режимі, зокрема це буде необхідно для аналізу поточних результатів навчання з метою корекції навчальної методики щодо кожного окремого учня. Такий аналіз не вимагає реакції у реальному часі і дозволяє виконати корекцію у дешевий спосіб. Менш з тим система мусить мати можливість реагувати на поточні дії користувача на сторінках сайту чи у застосунку тому, в ній має бути передбачений компонент обробки в реальному часу. З огляду на це, найкраще наведеним вимогам відповідає саме Lambda архітектура.

Згідно з обраною архітектурою була створена підсистема великих даних системи дистанційної освіти та розроблені моделі даних для збереження даних, які розглядатимуться надалі.

3.2. Розгортання підсистеми великих даних

В розробленій підсистемі великих даних кожний застосований інструмент відповідає за певний функціонал та бере участь у збирані, фільтрації, аналізі та збереженні даних. Підсистема була розгорнута на п’яти комп’ютерах (рис 3.1).

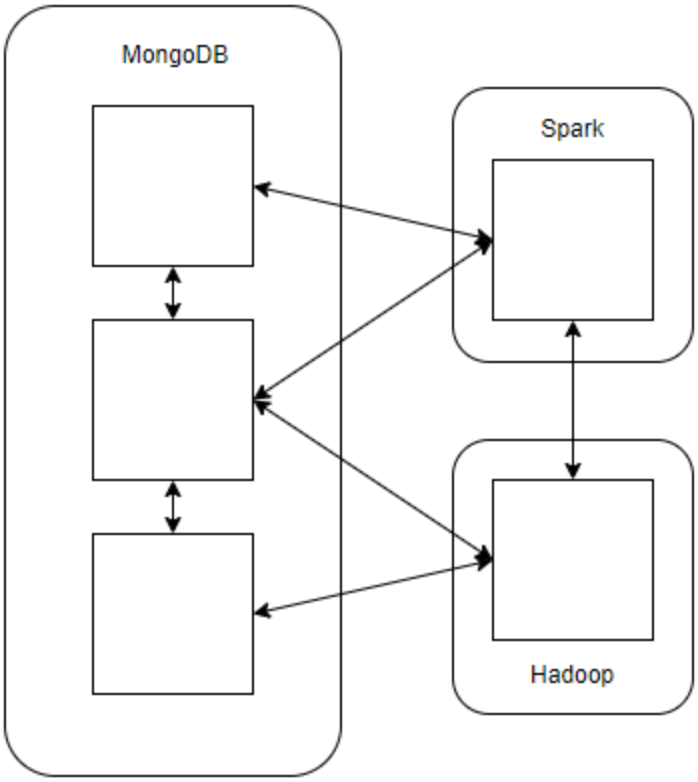

Рис. 3.1. Схема підключення комп’ютерів

Перші три комп’ютери, на яких було встановлено MongoDB, використовуються у якості сховища даних які надходять від клієнтів системи дистанційної освіти та опрацьованих даних з Hadoop та Spark. Фреймворки Hadoop та Spark були встановлені на окремих комп’ютерах. Комп’ютери у системі були з’єднані у віртуальну приватну мережу за допомогою OpenVpn. Цей інструмент був обраний тому, що забезпечує безпеку підключення за

допомогою шифрування засобами бібліотеки OpenSSL. Операційною системою для кожного комп’ютера було обрано Ubuntu 20.04. Конфігурація комп’ютерів для сховища MongoDB наведені на рис 3.2.

Назва компютера	Процесор	Оперативна пам'ять	Місткість накопичувача	Тип накопичувача	Конектор
Mongo 1	Intel Xeon E5-2680 V2	8GB	1TB	SDD	Raid
Mongo 2	Intel Xeon E5-2680 V2	8GB	1TB	SDD	Raid
Mongo 3	Intel Xeon E5-2680 V2	8GB	1TB	SDD	Raid
Резервний сервер	Intel Core i5-4590	8GB	20TB	HDD	Raid

Рис. 3.2. Конфігурація комп’ютерів для сховища

Для забезпечення відмовостійкості та продуктивності на комп’ютери сховища даних були встановленні та відповідним чином були сконфігуровані масиви дисків RAID 5. Конфігурація комп’ютерів для фреймворків Hadoop та Spark наведені на рис. 3.3.

Назва компютера	Процесор	Оперативна пам'ять	Місткість накопичувача	Тип накопичувача
Hadoop	Intel Core i5-4590	8GB	1TB	SDD
Spark	Intel Core i5-4590	32GB	1TB	SDD

Рис. 3.3. Схема підключення комп’ютерів

Великі дані мають тенденцію швидко збільшуватися, тому для їх зберігання треба з часом збільшувати обсяг сховища. На рис. 3.4 зображено діаграму яка показує прогнозоване зростання обсягу сховища залежно від кількості користувачів.

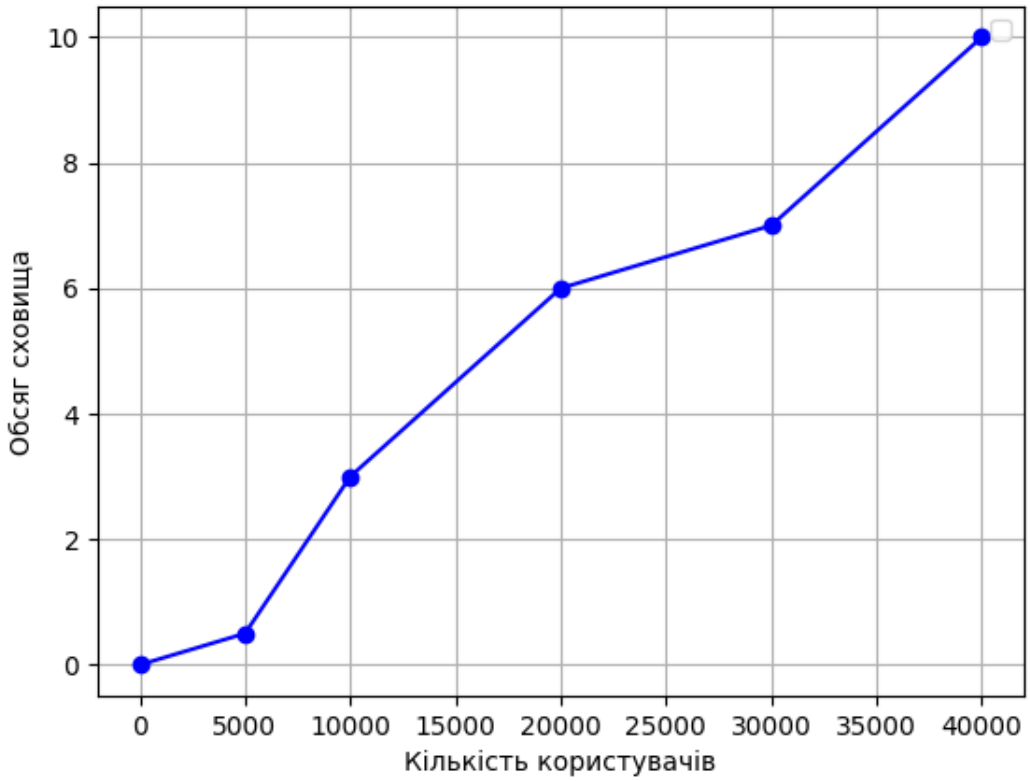

Рис. 3.4. Діаграма прогнозу зростання обсягу сховища від кількості користувачів

На комп’ютерах MongoDB була налаштована фрагментація. Це процес зберігання даних на кількох комп’ютерах, що дозволить безперешкодно горизонтально розширювати підсистему при збільшенні кількості даних що збираються. На рис. 3.5 зображено взаємодію компонентів у фрагментованому кластері.

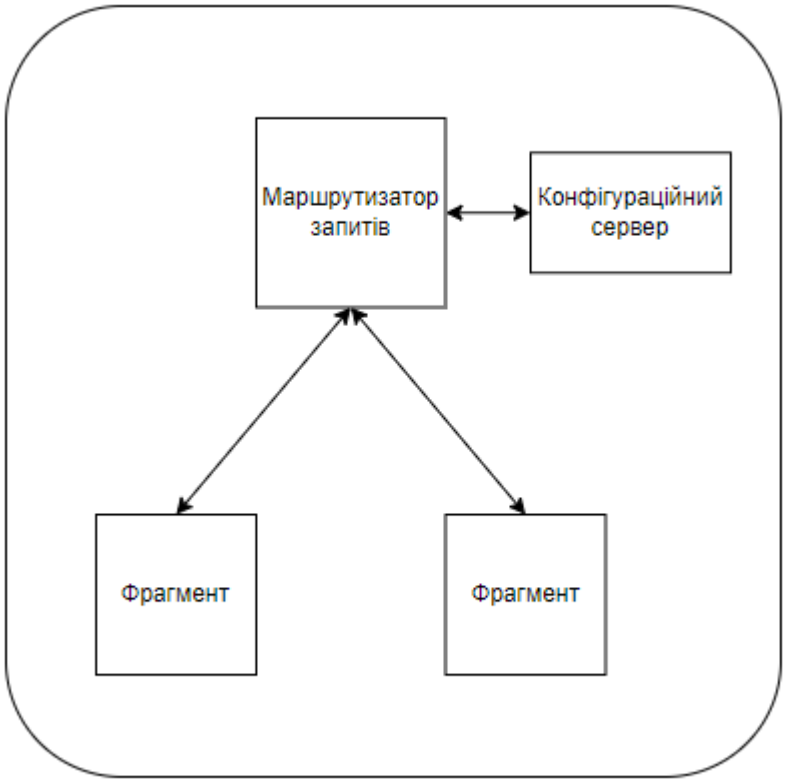

Рис. 3.5. Взаємодія компонентів у фрагментованому кластері

Фрагменти зберігають у собі дані. Конфігураційні сервери своєю чергою зберігають маршрути до фрагмента. Маршрутизатор запитів обробляє та націлює операції на сегменти. Фрагментація дозволяє значно зменшити навантаження на систему при обробці великих даних.

У кожному фрагменті та на конфігураційному сервері налаштована реплікація, яка забезпечує надлишковість та доступність даних (рис. 3.6).

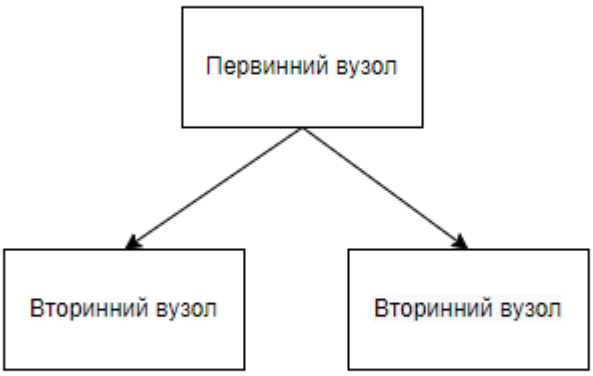

Рис. 3.6. Реплікація даних

Реплікація складається з первинного та вторинних вузлів. Первинний вузол отримує всі операції запису. Вторинні вузли копіюють журнал операцій первинного вузла та перевіряють що дані, які вони отримали, збігаються з наборами даних з первинного вузла. Якщо первинний вузол не відповідає вторинним вузлам, то один з вторинних вузлів бере на себе роль первинного та копіює свій вміст на інший вторинний вузол.

3.3. Взаємодія компонентів

Робота створеної підсистеми полягає у збережені, обробці та аналізі даних, які надходять від клієнтів системи дистанційної. На рис. 3.7 наведено архітектуру підсистеми великих даних системи дистанційної освіти.

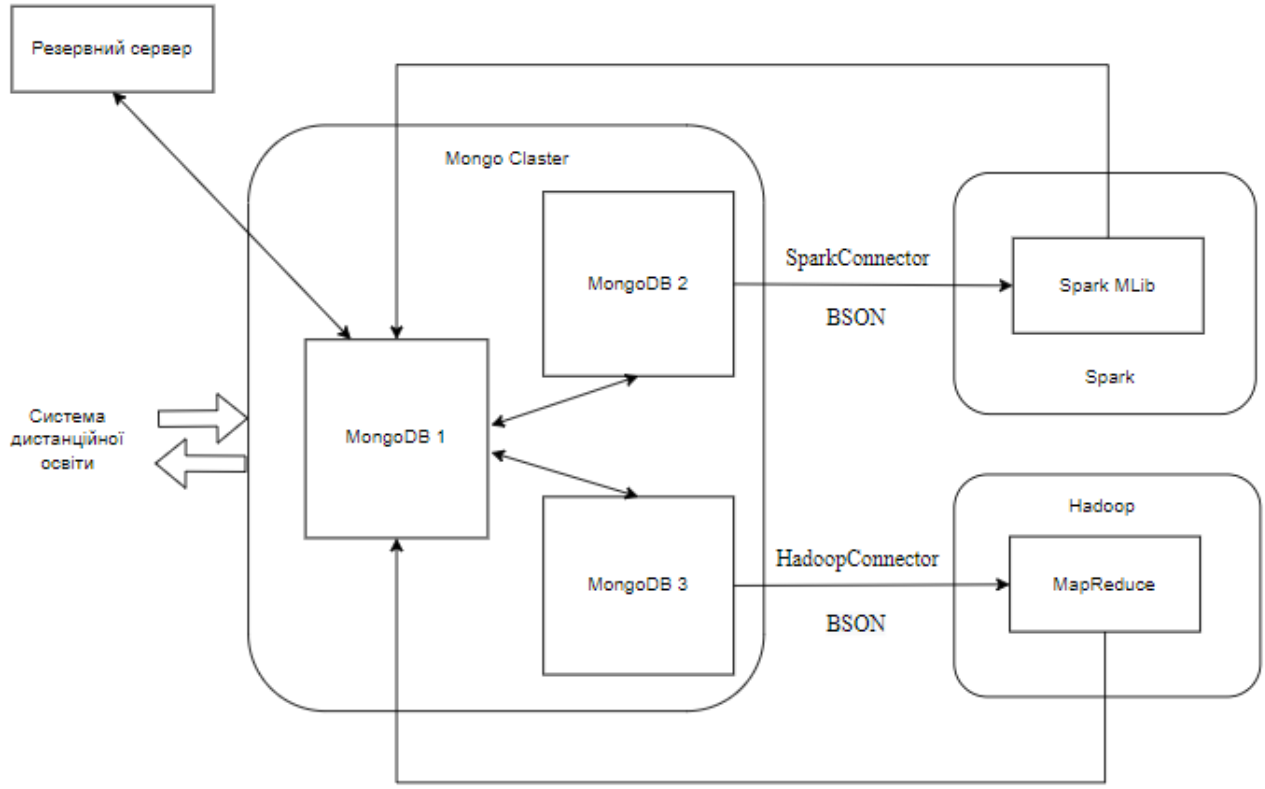

Рис. 3.7. Архітектура підсистеми великих даних

На першому кроці дані від користувачів зберігаються у головне сховище MongoDB у форматі BSON.

Після завантаження даних у MongoDB вони надсилаються у Hadoop та Spark які сполучені з MongoDB за допомогою SparkConnector та HadoopConnector. У Hadoop виконується пакетна обробка за допомогою модуля MapReduce. Дані після обробки зберігаються у сховищі MongoDB. Spark своєю чергою обробляє потокові дані за допомогою бібліотеки машинного навчання Spark MLlib. Потік даних, який надходить зі сховища MongoDB аналізується, після чого результати аналізу зберігаються в MongoDB, де вони можуть бути використані у системі дистанційної освіти.

Обрана архітектура передбачає що у Spark виконуватиметься поточна обробка поведінки користувача на сторінці з метою впливу на поточну видачу контенту користувачеві залежно від його поточних потреб. Hadoop як

інструмент пакетної обробки використовуватиметься для того, щоб уточнити уявлення системи дистанційної освіти про поточний стан когнітивної системи та онтологічної картини кожного окремого студента задля можливої зміни навчальної методики щодо нього.

3.4. Моделі даних системи дистанційного навчання

Головним джерелом даних для розробленої підсистеми виступає система дистанційної освіти. Для опису даних у фреймворку Django були створенні моделі, які своєю чергою визначають структуру даних включаючи типи полів, їх максимальний розмір тощо. У системі дистанційного навчання були описані файли моделей courses, tasks, media files, custom_auth, а також модель поведінки користувачів на сторінці та модель накопичення рішень завдань від користувачів. На рис 3.8 зображений файл моделей tasks.

```
class Task(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    is_public = models.BooleanField(default=False)
    updated_at = models.BigIntegerField()
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    topic = models.ForeignKey(Topic, null=True, on_delete=models.SET_NULL, related_name='tasks')
    difficulty = models.CharField(max_length=2, choices=settings.CHOICES_DIFFICULTY)
    number = models.CharField(max_length=15)


class TaskTitles(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(Task, on_delete=models.CASCADE, related_name='titles')
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)


class TaskTags(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(Task, on_delete=models.CASCADE, related_name='tags')
    tag = models.ForeignKey(Tags, on_delete=models.CASCADE)


class TaskContent(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(Task, on_delete=models.CASCADE, related_name='content')
    media_file = models.ForeignKey(MediaFile, on_delete=models.CASCADE)
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
```

Рис. 3.8. Модель tasks

Файл Tasks має чотири класи та зберігає у собі інформацію про задачі у навчальній системі.

У цьому файлі знаходяться клас Task у якому зберігаються дані про задачі, та він містить у собі поля uuid, is_public, updated_at, course, topic, difficulty, number. Наступний клас – TaskTitles, у якому зберігається назва задачі, він містить поля uuid, task, language, title.

Клас TaskTags зберігає дані про тег задачі та містить поля uuid, task, tag. В останньому класі, TaskContent, зберігається наповнення задачі цей клас містить поля uuid, task, media_file, language. На рис 3.9 зображений файл моделей course.

```
class Course(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    tags = models.ManyToManyField(Tags, related_name='courses')
    owner = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    base_currency = models.CharField(max_length=3, default='EUR')  # default =EUR
    difficulty = models.CharField(max_length=2, choices=settings.CHOICES_DIFFICULTY)
    price = models.DecimalField(decimal_places=2, max_digits=10, default=0)
    rating = models.FloatField(default=0)
    table_of_content = models.JSONField(null=True, blank=True)
    is_deleted = models.BooleanField(default=False)


class CourseTitle(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='titles')


class CourseDescription(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    description = models.JSONField()
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='descriptions')


class CourseTableOfContent(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    table_of_content = models.JSONField()
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='tables_of_content')


class CourseAuthors(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    authors = models.JSONField()
```

Рис. 3.9. Модель course

Наведений файл містить у собі класи Course, CourseTitle, CourseTableOfContent, CourseAuthors. Клас Course має поля uuid, tags, owner, base_currency, difficulty, price, rating, table_of_content, is_deleted. Клас CourseTitle містить поля uuid, language, title, course. Клас CourseDescription містить поля uuid, language, description, course. Клас CourseTableOfContent містить uuid, language, table_of_content, course. Клас CourseAuthors містить uuid, language, authors, course. На рис 3.10 зображений файл моделей media files.

```python
class MediaFile(models.Model):
    CHOICES_MEDIA_FILE_TYPE = (('vd', 'video'), ('im', 'image'), ('wp', 'webpage'))

    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    is_main = models.BooleanField(default=False)  # main media file for content
    is_demo = models.BooleanField(default=False)
    type = models.CharField(max_length=2, choices=CHOICES_MEDIA_FILE_TYPE, null=True)
    is_useful = models.BooleanField(default=False)


class MediaFileTitle(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    media_file = models.ForeignKey('MediaFile', on_delete=models.CASCADE, related_name='titles')


class MediaFileCover(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
    media_file = models.ForeignKey(
        'MediaFile', on_delete=models.CASCADE,
        related_name='covers'
    )
    cover = models.ForeignKey('MediaFile', on_delete=models.CASCADE)


class MediaFileUrls(models.Model):
    CHOICES_QUALITY = (('360', '360'), ('480', '480'), ('720', '720'),)

    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    media_file = models.ForeignKey('MediaFile', on_delete=models.CASCADE, related_name='urls')
    quality = models.CharField(max_length=4, choices=CHOICES_QUALITY, null=True)
    url = models.CharField(max_length=500)
```

Рис. 3.10. Модель media files

Цей файл містить у собі наступні класи: MediaFile, MediaFileTitle, MediaFileCover, MediaFileUrls. Клас MediaFile містить поля uuid, is_main,

is_demo, type, is_useful. Клас MediaFileTitle містить поля uuid, language, title, media_file. Клас MediaFileCover містить поля uuid, language, title, media_file. Клас MediaFileUrls містить поля uuid, media_file, quality, url. На рис 3.11 зображений файл моделей custom_auth.

```
class Profile(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    user = models.OneToOneField('CustomUser', on_delete=models.CASCADE, related_name="profile")
    first_name = models.CharField(max_length=50)
    photo = models.ImageField(null=True, blank=True, upload_to=get_file_path)
    specialization = models.TextField(null=True, blank=True)
    last_name = models.CharField(max_length=50)
    second_name = models.CharField(max_length=50, null=True, blank=True)
    phone = models.CharField(max_length=20, null=True, blank=True)
    skills = models.TextField(null=True, blank=True)
```

Рис. 3.11. Модель custom_auth

Цей файл містить у собі наступний клас: Profile. Клас Profile містить поля uuid, user, first_name, photo, specialization, last_name, second_name, phone, skills. На рис 3.12 зображений файл моделей user_events.

```
class UserEvents(models.Model):
    user = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    page = models.ForeignKey(Page, on_delete=models.CASCADE)
    registered_add = models.BigIntegerField()
    event_type = ForeignKey(IventTypes, on_delete=models.CASCADE)
    event_sorts = models.CharField()
```

Рис. 3.12. Модель user_events

Цей файл містить у собі наступний клас: UserEvents. Клас UserEvents містить поля user, page, registered_add, event_type, event_sorts. На рис 3.13 зображений файл моделей user_expireance.

```
class UserExpireance(models.Model):
    user = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    task = models.ForeignKey(Task, on_delete=models.CASCADE)
    attempt = models.TextField()
    updated_at = models.BigIntegerField()
```

Рис. 3.13. Модель user_expireance

Цей файл містить у собі наступний клас: UserExpireance. Клас UserExpireance містить поля user, task, attempt, updated_at.

3.5. Висновки

Запропонована архітектура підсистеми великих даних системи дистанційної освіти обрана з огляду на потреби пакетної обробки даних та обробки у реальному часі та враховує потенціал зростання кількості даних від користувачів системи освіти надає можливість збирати та аналізувати великі дані, що своєю чергою можуть значно підвищити рівень зацікавленості у створеній системі. Використання широковживаних інструментів Apache Spark, Hadoop, MongoDB дозволяє створити підтримувану систему, конфігурацією якої легко керувати. Запропонована модель даних системи дистанційної освіти надає можливість зберігати як освітній контент, так і дані про поточні освітні результати та характер використання освітнього контенту користувачами освітньої системи.

РОЗДІЛ 4

ЕКОНОМІЧНИЙ РОЗДІЛ

4.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. Передбачуване число операторів – 1150.
2. Коефіцієнт складності програми – 1,4.
3. Коефіцієнт корекції програми в ході її розробки – 0,2.
4. Годинна заробітна плата програміста, грн/год – 80.
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,1.
6. Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,5.
7. Вартість машино-години ЕОМ, грн/год – 20.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_о + t_и + t_а + t_п + t_{отл} + t_д, людино - годин, \quad (4.1)$$

де $t_о$ – витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ – витрати праці на дослідження алгоритму рішення задачі;

$t_а$ – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – витрати праці на програмування по готовій блок-схемі;

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

$t_д$ – витрати праці на підготовку документації.

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

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (4.2)$$

q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

$$Q = 1150 * 1{,}4 * (1 + 0{,}2) = 1932; \quad (4.3)$$

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

$$t_и = \frac{Q * B}{(75 \dots 85) * k}, \quad \text{людино} - \text{годин}, \quad (4.4)$$

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

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

$$t_и = \frac{1932 * 1{,}1}{75 * 1{,}5} = 18{,}9, людино - годин \quad (4.5)$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_а = \frac{Q}{(20 \dots 25) * k}, \qquad людино - годин, \quad (4.6)$$

$$t_а = \frac{1932}{23 * 1{,}5} = 56, людино - годин \quad (4.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_п = \frac{Q}{(20 \dots 25) * k}, людино - годин \quad (4.8)$$

$$t_п = \frac{1932}{24 * 1{,}5} = 53{,}6, людино - годин \quad (4.9)$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4 \dots 5) * k}, людино - годин, \quad (4.10)$$

$$t_{отл} = \frac{1932}{4 * 1.5} = 322, людино - годин \quad (4.11)$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 * t_{отл}, людино - годин. \quad (4.12)$$

$$t_{отл}^{k} = 1{,}5 * 322 = 483, людино - годин \quad (4.13)$$

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, \quad людино - годин, \quad (4.14)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{15..20 * k}, людино - годин. \quad (4.15)$$

$$t_{др} = \frac{1932}{19 * 1{,}5} = 67{,}8, людино - годин \quad (4.16)$$

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

$$t_{до} = 0{,}75 * t_{др}, людино - годин \quad (4.17)$$

$$t_{до} = 0{,}75 * 67{,}8 = 50{,}9, людино - годин \quad (4.18)$$

$$t_{д} = 67{,}8 + 50{,}9 = 118{,}7, людино - годин \quad (4.19)$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 +18,9+56+53,6 +322+118,7= 619,2, людино-годин. (4.20)

Отже дивлячись на результати ми отримуємо, що для розробки даного

програмного продукту нам знадобиться 619,2 людино-годин.

4.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{по}$ включають витрати на заробітну плату виконавця програми $З_{зп}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{по} = З_{зп} + З_{мв}, грн. \quad (4.21)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{зп} = t * С_{пр}, грн, \quad (4.22)$$

де t – загальна трудомісткість, людино-годин;

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

$$З_{зп} = 619{,}2 * 80 = 49536, грн. \quad (4.23)$$

$З_{МВ}$– Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} * С_{мч}, грн, \quad (4.24)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 322 * 20 = 6440, грн. \quad (4.25)$$

Звідси витрати на створення програмного продукту:

$$К_{по} = 49536 + 6440 = 55976 \text{ грн.} \quad (4.26)$$

Очікуваний період створення ПЗ:

$$Т = \frac{t}{B_к * F_р}, міс, \quad (4.27)$$

де $B_к$ - число виконавців;

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

$$Т = \frac{619,2}{1 * 176} = 3,5, \text{ міс} \quad (4.28)$$

4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту

Метою кваліфікаційної роботи є розробка архітектури системи великих даних для дистанційної освіти та дослідження інструментів за допомогою яких конструюються системи великих даних, аналіз всіх накопичених даних та розробки власної архітектури. Розроблена та проаналізована архітектура не є універсальною та розроблялась для обробки великих даних для конкретної системи й не передбачала продаж.

Однак, дані які будуть отримані після обробки на представленій архітектурі можуть бути використані для поліпшення аналізу великих об’ємів даних за допомогою яких буде досягнутий соціальний ефект у вигляді впровадження нових методик навчання за допомогою аналізу великих даних та поліпшення користуванням аналітичними даними.

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

Оскільки розроблена архітектура має дослідницький характер і створена з метою тестування її можливостей з застосунком дистанційного навчання. Тому чисельний розрахунок ефективності не проводився.

Висновки: Після підрахунку всіх формул в економічному розділі отримаємо наступні витрати що для створення ПЗ знадобиться приблизно 3,5 місяці що у людино-годинах буде дорівнювати 619,2, а вартість на розробку буде складати 55976 грн.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи були досліджені наявні архітектури був вивчений інструментарій таких систем такі як фреймворки, інструменти для збирання та аналізу сервери баз даних тощо та запропонована власна архітектура підсистеми великих даних для системи дистанційної освіти. Запропонована підсистема базується на зв'язці інструментів збирання та аналізу Hadoop, Spark та інструменту збереження Mongo DB. Також була розроблена модель даних для збереження інформації про освітній контент та поточні результати навчання у системі дистанційної освіти. Розробка запропонованої моделі була виконана засобами мови Python та фреймворку. У економічному розділі були виконані розрахунки часу на розробку які склали 3,5 місяці що у людиногодинах дорівнює 619,2 та вартості які склали 55976 грн. Результати отримані під час кваліфікаційної роботи можуть бути застосовані в процесі створення не тільки систем дистанційної освіти, а й будь-яких інших систем.

Для проєктування використовувались можливості фреймворків Hadoop та Spark та сховища даних MongoDB. Розроблена система здатна аналізувати, фільтрувати, впорядковувати, збирати статистичні дані. Отриманні результати роботи спроєктованої архітектури показують що розроблена система спроможна обробляти велику кількість даних та зберігати їх у сховищі.

Обрана архітектура може використовуватись як для систем дистанційної освіти, так і для систем навчальних закладів що своєю чергою дасть її користувачам дуже гнучкий та легкий інструмент для аналітики та збору великих даних та надасть можливість після аналізу оновити підходи до навчання студентів або змінити методики, покращити та пришвидшити процес набору знань студентами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aytas Y. Designing Big Data Platforms: How to Use, Deploy, and Maintain Big Data Systems / Y. Aytas – New York: Wiley, 2021. – 336 p.

2. Security and Privacy for Big Data, Cloud Computing and Applications / [W. Ren, L. Wang, K.K. Raymond Choo, F. Xhafa] – The Institution of Engineering and Technology, 2019. – 328 p.

3. Marz N., Warren A.J. Big Data: Principles and Best Practices of Scalable Real-Time Data Systems / N. Marz, A.J. Warren – New York: Manning, 2015. – P 210-220.

4. Franks B. The Analytics Revolution: How to Improve Your Business by Making Analytics Operational in the Big Data Era / B. Franks – New York: Wiley, 2014. P 105-115.

5. Viktor M.S., Kenneth C. Big Data: A Revolution That Will Transform How We Live, Work, and Think / M.S. Viktor, C. Kenneth – London: Harper Business, 2014. P 55-57.

6. Bill C., Matei Z. Spark: The Definitive Guide: Big Data Processing Made Simple / C. Bill, Z. Matei – London: O'Reilly Media, 2018. – 606 p.

7. Thomas E. Big Data Fundamentals: Concepts, Drivers & Techniques / E. Thomas, K. Wajid, B. Paul – London: Pearson, 2016. P 100-104.

8. Bernard M. Big Data in Practice: How 45 Successful Companies Used Big Data Analytics to Deliver Extraordinary Results / M. Bernard – New York: Wiley, 2016. P 205-209.

9. Armstrong H. Big Data, Big Design: Why Designers Should Care about Artificial Intelligence / H. Armstrong – New York: Princeton Architectural Press, 2021. – 176 p.

10. Pop F., Neagu G. Big Data Platforms and Applications: Case Studies, Methods, Techniques, and Performance Evaluation / F. Pop, G. Neagu – Luxembourg: Springer, 2021. P 406-409.

11. White T. Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale / T. White – London: O'Reilly Media, 2015. – 756 p.

12. Miner D., Adam S. MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems / Miner D., Adam S. – London: O'Reilly Media, 2013. – 250 p.

13. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann – London: O'Reilly Media, 2017. P 359-363.

14. Malaska T., Seidman J. Foundations for Architecting Data Solutions: Managing Successful Data Projects / T. Malaska, J. Seidman – London: O'Reilly Media, 2018. – 190 p.

15. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage / S. Bradshaw, E. Brazil, K. Chodorow – London: O'Reilly Media, 2019. – 514 p.

16. Holt B. Format: Kindle EditionScaling CouchDB: Replication, Clustering, and Administration / B. Holt – London: O'Reilly Media, 2011. – 87 p.

17. W. Ben Big Data in Education: The digital future of learning, policy and practice / Ben W. – California: SAGE Publications Ltd, 2017. – 256 p.

18. Chris A. J., Lehnardt J., Slater N. CouchDB: Replication, Clustering, and Administration / A. J. Chris, J. Lehnardt, N. Slater – London: O'Reilly Media, 2010. – 367 p.

19. Improving Online Education Using Big Data Technologies website. URL: https://www.intechopen.com/chapters/68814 (дата звернення: 15.09.2021).

20. Chaffai A., Hassouni L., Anoun H. Real-Time Analysis of Students' Activities on an E-Learning Platform based on Apache Spark / A. Chaffai, L. Hassouni, H. Anoun // International Journal of Advanced Computer Science and Applications – 2017. – Vol. 8, № 7. – P. 106-108.

21. Jha S., Jha M., O'Brien L. A Step Towards Big Data Architecture for Higher Education Analytics / S. Jha, M. Jha, L. O'Brien //Asia-Pacific World Congress on Computer Science and Engineering – 2018. – Vol. 8. – P. 181-182.

22. Big Data Management for Healthcare Systems: Architecture, Requirements, and Implementation Technologies website. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6032968/ (дата звернення: 22.08.2021).

23. Michalik P., Stofa J., ZolotovaReal I. Concept Definition for Big Data Architecture in the Education System / P. Michalik, J. Stofa, I. Zolotova // International Symposium on Applied Machine Intelligence and Informatics – 2014. – Vol. 14. – P. 333-334.

24. MongoDB & The McGraw-Hill Education Learning Analytics Platform website. URL: https://www.slideshare.net/mongodb/mongodb-the-mcgrawhill-education-learning-analytics-platform (дата звернення: 05.09.2021).

25. Ruggero R., Angela M. Big Data Performance and Comparison with Different DB Systems / R. Ruggero, M. Angela // International Journal of Computer Science and Information Technologies – 2017. – Vol. 8. – P. 60-62.

26. Lu Z.W. NoSQL Distributed Big Data Storage Technology and Application Based on Cloud Platform / International Conference on Advanced Design and Manufacturing Engineering // – 2017. – Vol. 7. – P. 335-337.

27. Big Data website. URL: https://www.it.ua/ru/knowledge-base/technology-innovation/big-data-bolshie-dannye (дата звернення: 15.08.2021).

28. Perkins L., Redmond E., Wilson J. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement / L. Perkins, E. Redmond, J. Wilson – Pragmatic Bookshelf, 2018. – P 150-160.

29. Dayley B. NoSQL with MongoDB in 24 Hours, Sams Teach Yourself / B. Dayley – Carmel : Sams Publishing, 2018. – 538 p.

30. Copeland R. MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database / R. Copeland – London: O'Reilly Media, 2013. – 176 p.

31. Barrios P. A. E. Hands-on Data Virtualization with Polybase: Administer Big Data, SQL Queries and Data Accessibility Across Hadoop, Azure, Spark,

Cassandra, MongoDB, CosmosDB, MySQL and PostgreSQL / P. A. E. Barrios – New York: BPB Publications, 2021. – 614 p.

32. Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage / K. Chodorow – London: O'Reilly Media, 2013. – 432 p.

33. Berman E. Small Wars, Big Data: The Information Revolution in Modern Conflict / E. Berman – Princeton : Princeton University Press, 2018. – 408 p.

34. Emrouznejad A., Charles V. Big Data for the Greater Good / A. Emrouznejad, V. Charles – Berlin: Springer, 2019. – 214 p.

35. Das T., Lee D. Learning Spark: Lightning-Fast Data Analytics / T. Das, D. Lee – London: O'Reilly Media, 2020. – 400 p.

36. Lane J. E., Zimpher N. L. Building a Smarter University: Big Data, Innovation, and Analytics / J. E. Lane, N. L. Zimpher – New York: SUNY Press, 2014. – 344 p.

37. Karau H., Konwinski A., Wendell P. Learning Spark: Lightning-Fast Big Data Analysis / H. Karau, A. Konwinski, P. Wendell – London: O'Reilly Media, 2015. – 276 p.

38. E. Capriolo, D. Wampler, J. Rutherglen Programming Hive: Data Warehouse and Query Language for Hadoop / E. Capriolo, D. Wampler, J. Rutherglen – London: O'Reilly Media, 2012. – 350 p.

39. Warnock D. MongoDB: Learn MongoDB in a simple way! / D. Warnock – London: O'Reilly Media, 2016. – 89 p.

40. Densmore J. Data Pipelines Pocket Reference: Moving and Processing Data for Analytics / J. Densmore – London: O'Reilly Media, 2021. – 276 p.

Додаток А

КОД ПРОГРАМИ

```
import uuid as uuid
from django.db import models

from custom_auth.models import *
from media_files.models import *
from tags.models import *
from django.conf import settings

CHOICES_LANGUAGES = (('uk', 'uk'), ('ru', 'ru'), ('en', 'en'),)
# base currency
# price -> dictionary -> {[baseCurrency]: currencyValue} JSONField, baseCurrency = EUR


class Course(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    tags = models.ManyToManyField(Tags, related_name='courses')
    owner = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    base_currency = models.CharField(max_length=3, default='EUR')  # default =EUR
    difficulty = models.CharField(max_length=2, choices=settings.CHOICES_DIFFICULTY)
    price = models.DecimalField(decimal_places=2, max_digits=10, default=0)
    rating = models.FloatField(default=0)
    table_of_content = models.JSONField(null=True, blank=True)
    is_deleted = models.BooleanField(default=False)


class CourseTitle(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='titles')


class CourseDescription(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    description = models.JSONField()
    course = models.ForeignKey('Course', on_delete=models.CASCADE,
related_name='descriptions')


class CourseTableOfContent(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    table_of_content = models.JSONField()
    course = models.ForeignKey('Course', on_delete=models.CASCADE,
related_name='tables_of_content')


class CourseAuthors(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    authors = models.JSONField()
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='authors')


class CourseCover(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    cover = models.ForeignKey(MediaFile, on_delete=models.CASCADE)
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='covers')

class Subscription(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    duration = models.BigIntegerField()
    is_deleted = models.BooleanField(default=False)
    price = models.DecimalField(decimal_places=2, max_digits=10)
```

```
class SubscriptionTitle(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    subscription = models.ForeignKey('Subscription', on_delete=models.CASCADE,
related_name='titles')


class SubscriptionDescription(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    description = models.TextField()
    subscription = models.ForeignKey('Subscription', on_delete=models.CASCADE,
related_name='descriptions')

class CourseSubscription(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='courses')
    subscription = models.ForeignKey('Subscription', null=True, on_delete=models.SET_NULL,
                                     related_name='subscription_info')


class SubscriptionHistory(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    user = models.ForeignKey(CustomUser, on_delete=models.CASCADE, )
    date_start = models.BigIntegerField()
    date_finished = models.BigIntegerField()
    course = models.ForeignKey('Course', null=True, on_delete=models.SET_NULL, )
    subscription_data = models.JSONField()


class MediaFileContent(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    content = models.ForeignKey('Content', on_delete=models.CASCADE,
related_name='media_contents')
    media_file = models.ForeignKey(MediaFile, on_delete=models.CASCADE,
related_name='media_files')


class Topic(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    course = models.ForeignKey('Course', null=True, on_delete=models.CASCADE,
related_name='topics')
    topic_level = models.IntegerField()
    parent = models.ForeignKey('self', null=True, on_delete=models.CASCADE)
    order = models.IntegerField()


class TopicTitle(models.Model):
    CHOICES_IDENTIFIERS = (('p1', 'part1'), ('p2', 'part2'), ('p3', 'part3'), ('p4',
'part4'))
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    identifier = models.CharField(max_length=2, choices=CHOICES_IDENTIFIERS)
    topic = models.ForeignKey('Topic', on_delete=models.CASCADE, related_name='titles')


class Content(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    last_changed = models.BigIntegerField()
    topic = models.ForeignKey('Topic', on_delete=models.CASCADE, related_name='contents')


class ContentTitle(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    content = models.ForeignKey('Content', on_delete=models.CASCADE,
related_name='content_titles')
```

```
import uuid as uuid
from django.db import models
from django.conf import settings
from django.dispatch import receiver
import os


class MediaFile(models.Model):
    CHOICES_MEDIA_FILE_TYPE = (('vd', 'video'), ('im', 'image'), ('wp', 'webpage'))

    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    is_main = models.BooleanField(default=False)  # main media file for content
    is_demo = models.BooleanField(default=False)
    type = models.CharField(max_length=2, choices=CHOICES_MEDIA_FILE_TYPE, null=True)
    is_useful = models.BooleanField(default=False)


class MediaFileTitle(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)
    media_file = models.ForeignKey('MediaFile', on_delete=models.CASCADE,
related_name='titles')


class MediaFileCover(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
    media_file = models.ForeignKey(
        'MediaFile', on_delete=models.CASCADE,
        related_name='covers'
    )
    cover = models.ForeignKey('MediaFile', on_delete=models.CASCADE)


class MediaFileUrls(models.Model):
    CHOICES_QUALITY = (('360', '360'), ('480', '480'), ('720', '720'),)

    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    media_file = models.ForeignKey('MediaFile', on_delete=models.CASCADE,
related_name='urls')
    quality = models.CharField(max_length=4, choices=CHOICES_QUALITY, null=True)
    url = models.CharField(max_length=500)

from django.db import models
import uuid as uuid
from django.conf import settings
from courses.models import *


class Task(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    is_public = models.BooleanField(default=False)
    updated_at = models.BigIntegerField()
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    topic = models.ForeignKey(Topic, null=True, on_delete=models.SET_NULL,
related_name='tasks')
    difficulty = models.CharField(max_length=2, choices=settings.CHOICES_DIFFICULTY)
    number = models.CharField(max_length=15)


class TaskTitles(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(Task, on_delete=models.CASCADE, related_name='titles')
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)
    title = models.CharField(max_length=100)


class TaskTags(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(Task, on_delete=models.CASCADE, related_name='tags')
    tag = models.ForeignKey(Tags, on_delete=models.CASCADE)
```

```
class TaskContent(models.Model):
    uuid = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(Task, on_delete=models.CASCADE, related_name='content')
    media_file = models.ForeignKey(MediaFile, on_delete=models.CASCADE)
    language = models.CharField(max_length=3, choices=settings.CHOICES_LANGUAGES, null=True)

from rest_framework import serializers
from .models import *
from custom_auth.serializers import *
from custom_auth.models import *
from user_space.models import *
from tasks.serializers import *
from media_files.serializers import *
from tags.serializers import *


class MediaFileContentSerializer(serializers.ModelSerializer):
    media_file = MediaFileSerializer()

    class Meta:
        model = MediaFileContent
        exclude = ['content']


class MediaFileContentDeserializer(serializers.ModelSerializer):
    class Meta:
        model = MediaFileContent
        fields = '__all__'

    def create(self, validated_data):
        return MediaFileContent.objects.create(
            language=validated_data['language'],
            content=validated_data['content'],
            media_file=validated_data['media_file']
        )


class TopicTitleSerializer(serializers.ModelSerializer):
    class Meta:
        model = TopicTitle
        exclude = ['topic']


class TopicTitleDeserializer(serializers.ModelSerializer):
    class Meta:
        model = TopicTitle
        fields = "__all__"

    def create(self, validated_data):
        return TopicTitle.objects.create(
            title=validated_data['title'],
            identifier=validated_data['identifier'],
            topic=validated_data['topic'],
            language=validated_data['language'],
        )


class ContentTitleSerializer(serializers.ModelSerializer):
    class Meta:
        model = ContentTitle
        exclude = ['content']


class ContentSerializer(serializers.ModelSerializer):
    content_titles = ContentTitleSerializer(many=True)
    media_contents = MediaFileContentSerializer(many=True)

    class Meta:
        model = Content
        exclude = ['topic']
```

```
class ContentDeserializer(serializers.ModelSerializer):
    class Meta:
        model = Content
        fields = '__all__'

    def create(self, validated_data):
        return Content.objects.create(
            topic=validated_data['topic'],
            last_changed=validated_data['last_changed']
        )


class TopicDeserializer(serializers.ModelSerializer):
    class Meta:
        model = Topic
        fields = "__all__"

    def create(self, validated_data):
        return Topic.objects.create(
            course=validated_data['course'],
            topic_level=validated_data['topic_level'],
            parent=validated_data['parent'],
            order=validated_data['order'],
        )


class TopicContentSerializer(serializers.ModelSerializer):
    titles = TopicTitleSerializer(many=True)
    # Мы не строим древовидную структуру топиков с сабтопиками, возвращая их в виде
одноуровневого списка
    # subtopic = serializers.SerializerMethodField()
    contents = ContentSerializer(many=True)
    tasks = TaskSerializer(many=True)

    class Meta:
        model = Topic
        exclude = ['course']
    # def get_subtopic(self, obj):
    #     if obj.subtopic is not None:
    #         return TopicSerializer(obj.subtopic).data
    #     else:
    #         return None


class CourseTitleSerializer(serializers.ModelSerializer):
    class Meta:
        model = CourseTitle
        exclude = ['course']


class CoursedDescriptionSerializer(serializers.ModelSerializer):
    class Meta:
        model = CourseDescription
        exclude = ['course']


class CourseAuthorsSerializer(serializers.ModelSerializer):
    class Meta:
        model = CourseAuthors
        exclude = ['course']


class CourseCoversSerializer(serializers.ModelSerializer):
    cover = MediaFileSerializer()

    class Meta:
        model = CourseCover
        exclude = ['course']


class CourseTableOfContentSerializer(serializers.ModelSerializer):
    class Meta:
        model = CourseTableOfContent
```

```
        exclude = ['course']


# deprecated, we use TopicContentSerializer to get course content
class CourseContentSerializer(serializers.ModelSerializer):
    tags = TagsSerializer(many=True)
    titles = CourseTitleSerializer(many=True)
    covers = CourseCoversSerializer(many=True)
    authors = CourseAuthorsSerializer(many=True)
    descriptions = CoursedDescriptionSerializer(many=True)
    tables_of_content = CourseTableOfContentSerializer(many=True)
    topics = TopicContentSerializer(many=True)

    class Meta:
        model = Course
        exclude = ['owner']


class CourseSerializer(serializers.ModelSerializer):
    tags = TagsSerializer(many=True)
    titles = CourseTitleSerializer(many=True)
    covers = CourseCoversSerializer(many=True)
    authors = CourseAuthorsSerializer(many=True)
    descriptions = CoursedDescriptionSerializer(many=True)

    class Meta:
        model = Course
        exclude = ['owner']


class CourseDeserializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = '__all__'

    def create(self, validated_data):
        course = Course.objects.create(
            owner=validated_data.get('owner'),
            difficulty=validated_data.get('difficulty'),
            price=validated_data.get('price'),
            rating=validated_data.get('rating'),
        )

        course.tags.set(validated_data.get('tags'))
        return course

    def update(self, instance, validated_data):
        print(validated_data)
        instance.difficulty = validated_data.get('difficulty', instance.difficulty)
        instance.rating = validated_data.get('rating', instance.rating)
        instance.price = validated_data.get('price', instance.price)
        instance.tags.set(validated_data.get('tags', instance.tags))
        instance.is_deleted = validated_data.get('is_deleted', instance.is_deleted)
        instance.table_of_content = validated_data.get('table_of_content',
instance.table_of_content)
        # instance.own = validated_data.get('is_deleted', instance.is_deleted)

        instance.save()
        return instance


class CourseTitleDeserializer(serializers.ModelSerializer):
    class Meta:
        model = CourseTitle
        fields = '__all__'

    def create(self, validated_data):
        return CourseTitle.objects.create(
            course=validated_data['course'],
            title=validated_data['title'],
            language=validated_data['language']
        )
```

```
    def update(self, instance, validated_data):
        instance.title = validated_data.get('title', instance.title)
        instance.language = validated_data.get('language', instance.language)

        instance.save()

        return instance


class CourseDescriptionDeserializer(serializers.ModelSerializer):
    class Meta:
        model = CourseDescription
        fields = '__all__'

    def create(self, validated_data):
        return CourseDescription.objects.create(
            course=validated_data['course'],
            language=validated_data['language'],
            description=validated_data['description']
        )

    def update(self, instance, validated_data):
        instance.description = validated_data.get('description', instance.description)
        instance.language = validated_data.get('language', instance.language)

        instance.save()

        return instance


class CourseAuthorsDeserializer(serializers.ModelSerializer):
    class Meta:
        model = CourseAuthors
        fields = '__all__'

    def create(self, validated_data):
        return CourseAuthors.objects.create(
            course=validated_data['course'],
            language=validated_data['language'],
            authors=validated_data['authors']
        )

    def update(self, instance, validated_data):
        instance.authors = validated_data.get('authors', instance.authors)
        instance.language = validated_data.get('language', instance.language)

        instance.save()

        return instance


class CourseCoverDeserializer(serializers.ModelSerializer):
    class Meta:
        model = CourseCover
        fields = '__all__'

    def create(self, validated_data):
        return CourseCover.objects.create(
            course=validated_data['course'],
            language=validated_data['language'],
            cover=validated_data['cover']
        )


class SubscriptionTitleSerializer(serializers.ModelSerializer):
    class Meta:
        model = SubscriptionTitle
        fields = '__all__'


class SubscriptionDescriptionSerializer(serializers.ModelSerializer):
    class Meta:
        model = SubscriptionDescription
```

```
        fields = '__all__'


class TaskCourseSerializer(serializers.ModelSerializer):
    tags = TagsSerializer(many=True)
    titles = CourseTitleSerializer(many=True)

    class Meta:
        model = Course
        fields = ['tags', 'titles']


# Start serializers for SubscriptionAPI ----------------------------------
class CourseSubscriptionSerializer(serializers.ModelSerializer):
    titles = CourseTitleSerializer(many=True)

    class Meta:
        model = Course
        fields = ['titles']


class SubscriptionInfoCourseSerializer(serializers.ModelSerializer):
    course = CourseSubscriptionSerializer()

    class Meta:
        model = CourseSubscription
        exclude = ['subscription']


class SubscriptionInfoSerializer(serializers.ModelSerializer):
    course = CourseSubscriptionSerializer()

    class Meta:
        model = CourseSubscription
        fields = '__all__'


class SubscriptionSerializer(serializers.ModelSerializer):  # сериалайзер возвращает список
всех подписок
    titles = SubscriptionTitleSerializer(many=True)
    descriptions = SubscriptionDescriptionSerializer(many=True)
    subscription_info = SubscriptionInfoSerializer(
        many=True)  # сериалайзер возвращает информацию о подписанных

    class Meta:
        model = Subscription
        fields = ['duration', 'price', 'titles', 'descriptions']
```

Додаток Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Керівника економічної частини Доцента Касьяненко Л.В.
(прізвище, ім'я, по батькові, вчене звання)

на кваліфікаційну роботу

Студента ІІ курсу групи 122м-20-1 Чуба Євгена Олексійовича
(прізвище, ім'я, по батькові)

На тему: Розробка архітектури сховищ великих даних для системи дистанційної освіти.

.

«___»_____2022 р. _____
(підпис)

Додаток В

ПЕРЕЛІКЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Чуб.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Чуб.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація_Чуб.ppt	Презентація кваліфікаційної роботи.