УДК 004.934

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

Рівень вищої освіти *другий (магістерський)*

*Дослідження ефективності використання систем зберігання даних для CMS платформ*
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-19-2*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва напряму)
Тип програми *освітньо-наукова*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*
(повна назва освітньої програми)

*Слива Є.К.*
(прізвище, ініціали)

Керівник проф. Рябченко І.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____ *Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

(повна назва)

Кафедра *Системотехніки*

(повна назва)

Рівень вищої освіти *другий (магістерський)*

Спеціальність *122 – Комп'ютерні науки*

(код і повна назва)

Тип програми *освітньо-наукова*

(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«____»_____ 20___ р.

# ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Сливі Євгену Костянтиновичу*

(прізвище, ім'я, по батькові)

1. Тема роботи *«Дослідження ефективності використання систем зберігання даних для CMS платформ»*

затверджена наказом по університету від « *23* » 03 2021 р. № *389 Ст*

2. Термін подання студентом роботи (проекту) *18 травня 2021 р.*

3. Вихідні дані до роботи *Функція: Розробка програмних додатків для оцінки єфективності різних СУБД. Організація даних: файлова з прямим доступом . Форма діалогу: консольний додаток. Перелік використовуваних програмних засобів: ОС Microsoft Windows 7 та вище, інтегроване середовище програмування. Технічне забезпечення: виділений сервер з якнайменш 16ГБ ОЗУ та чотирьох'ядерним процесором*

4. Перелік питань, що потрібно опрацювати в роботі *Вступ. Аналіз предметної області та постановка задачі. Порівняння моделей даних та їх використання для різних СУБД. Розробка утиліт які аналізують швидкість виконання операцій. Аналіз отриманих даних. Висновки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів, комп'ютерних ілюстрацій) *Плакати на аркушах формату А4: архітектура CMS системи, алгоритми обробри щапитів на отримання сторінки, приклади моделей даних, програмна реалізація тестування сценаріїв, діаграми прецедентів, механізм кешування, типи СУБД, реляційна модель даних, ієрархічна модель даних, концептуальна модель даних, класифікація систем зберігання даних.*

6. Консультанти розділів роботи (проекту)

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| *1* | *Отримання, аналіз завдання, уточнення плану роботи* | *23.03.2021* | |
| *2* | *Аналіз завдання, пошук літератури.* | *26.03.2021* | |
| *3* | *Постановка задачі та вибір методу її вирішення* | *03.04.2021* | |
| *4* | *Проведення експериментальних досліджень* | *18.04.2021* | |
| *5* | *Оформлення пояснювальної записки* | *27.04.2021* | |
| *6* | *Підготовка презентації* | *11.05.2021* | |
| *7* | *Подання закінченої роботи науковому керівникові* | *12.05.2021* | |
| *8* | *Подання роботи на рецензування* | *12.05.2021* | |
| *9* | *Попередній захист* | *14.05.2021* | |
| *10* | *Подання роботи до екзаменаційної комісії* | *18.05.2021* | |

Дата видачі завдання *23 квітня 2021 р.*

Студент _____ *Слива Є.К.*
(підпис)

Керівник роботи _____ *проф. Рябченко І.М.*
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 66 сторінок, 28 рисунків, 18 таблиць, 20 джерел інформації, 3 додатки.

СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, СИСТЕМА КЕРУВАННЯ КОНТЕНТОМ, РОЗРОБКА КОМПОНЕНТІВ, ВЕБ-СЕРВЕР, БАЛАНСУВАЛЬНИК НАВАНТАЖЕННЯ, КЕШУВАННЯ.

Об’єкт дослідження розробки – набір програмних утиліт які дозволяють оцинінити швидкість СУБД для CMS платформ.

Предмет дослідження – тестові сценарії використання CMS, програмні засоби, які тестують сценарії, та надають результат тестування.

Мета атестаційної роботи – розробка та аналіз програмних утиліт, що надають можливість оцінювання СУБД та їх характеристик які використовуються у системах управління контентом.

Методи розробки – методи проектування додатків для систем розпізнавання образів, методи об’єктно-орієнтованої розробки, методи розробки, базовані на функціональному підході, методи розробки розподіленої інформаційної системи.

Результати кваліфікаційної роботи – програмні засоби – утиліти . Програмні засоби призначені для аналізу характеристик ефективності інформаційної системи які використовують CMS платформи. Аналітична оцінка ефективності різних СУБД які інтегровані в системи керування контентом.

Область застосування – аналіз систем управління базами даних, надає можливість обрати найбільш ефективнішу систему зберігання даних для системи управління контентом.

# ABSTRACT

Master’s explanatory note: 66 pages, 28 figures, 18 tables, 20 sources, 3 applications.

DATABASE MANAGEMENT SYSTEM, CONTENT MANAGEMENT SYSTEM, COMPONENT DEVELOPMENT, WEB SERVER, LOAD BALANCER, CACHE.

The object of research is a set of software utilities that allow you to estimate the speed of the database for CMS platforms.

The subject of the research is test scenarios of using CMS, software tools that test scripts and provide test results.

The purpose of certification work is the development and analysis of software utilities that provide an opportunity to assess the database and their characteristics used in content management systems.

Development methods - methods of designing applications for image recognition systems, methods of object-oriented development, development methods based on a functional approach, methods of developing a distributed information system.

Results of qualification work - software - utilities. Software tools are designed to analyze the performance characteristics of information systems that use CMS platforms. Analytical evaluation of the effectiveness of various databases that are integrated into content management systems.

Scope - analysis of database management systems, provides an opportunity to choose the most efficient storage system for content management system.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – система управління базами даних.

СКБД – система керування базами даних.

CMS – content management system.

SQL – structured query language.

TTFB – time to first byte.

JCR – java content repository.

# ВСТУП

Зберігання даних для систем які побудовані на основі CMS платформ є дуже важливою частиною усієї інформаційної системи. Від вибору системи зберігання даних для CMS платформи залежить якість, та успішність того чи іншого бренду, компанії, організації інформаційної системи.

CMS платформи використовуються для сайтів із великою кількістю контенту. Найбільш розповсюдженими типами інформаційних систем які базуються на CMS прикладі – є інтернет магазини. Такі інформаційні системи мають категорії товарів для продажу, підкатегорії, сторінки з продуктом категорії, інформаційні сторінки. Крім того кожна сторінка може бути із різним контентом у відповідності до того як вона була сконфігурована контент автором.

За наш сучасний час існує багато систем які зберігають дані використовуючи за основу реляційну алгебру або так звану алгебру відношень. Ці системи називаються «Реляційна система керування базами даних». У кожної такої системи є свої сильні та слабкі сторони, котрі стають основними критеріями для вибору тієї чи іншої системи. Пройшло досить багато часу із моменту створення таких систем, але на сьогодні не можна сказати відразу, яка система зберігання даних буде ефективною. Це залежить від певних потреб тої чи іншої предметної області. Для якоїсь предметної області головним може бути якість транзакцій, для іншої швидкість запису даних, швидкість зчитування даних, консистентність даних та багато інших критеріїв, котрі один на одного можуть впливати.

До переваг CMS слід віднести наступні:

- Швидве та ефективне управління інформацією. Можливість підключення до наповнення сайту контентом не тільки веб розробника або редактора, але і всіх співробітників, які володіють тією чи іншою інформацією;
- Зменшення строків та кошти розробки тих чи інших функцій та надання додаткових сервісів;
- Підвищення якості розробки та зміни сайту. Так як усі розробки

пройшли багато ітерацію і неодноразовано протестовані, також кожна компонента вже використовується на багатьох сторінках;

− Зниження коштів подальших змін, за рахунок розподілення контенту та іх зберігання;

− Зниження вартості підтримки чи взагалі її повна відсутність. Так як власник сайту може сам змінювати будб-що, без участі розробника або веб мастера;

До основних недоліків CMS систем слід віднести наступні:

− Шаблонний дизайн та розміщення елементів сайту. Більшість популярних CMS систем не дозволяють швидко створювати сайти із власним дизайном, для цбого поторібна участь розробників;

− Неможливість додавання власних динамічних блоків. Ця особливість відноситься до CMS з багатьма користувачами, в які можна додавати динамічніблоки тільки за допомогою адміністрації сайту. В системи з одним користувачем можна додавати динамічні блоки самостійно, однак немає ніякого інструментарію для їх створення. Крім того в системах з одним користувачем немає глобального збереження динамічних блоків, в якому зберігалися б усі блоки, створені для даної системи управління контентом;

− Неуніверсальність CMS систем. Через обмежену низкудинамічних блоків та неможливості простого створення особистих динамічних блоків – CMS системи не мають можливості створювати сайти будь-якої складності;

− Неможливість роботи створеного сайту правцювати самостійно без CMS системи. Особливо ця проблема стає гострою для багатокористувацьких систем, бо якщо сайт створено на одній CMS системіЮ то дуже проблематично перенести його на іншу;

− Один інтерефейс для налаштування сайту і наповнення його інформацією Зазвичай в системах управління контентом інтерфейс налаштування динамічних блоків та інтерфейс оновлення сайту зміщені. Це ускладнює механізм розділення прав користувачів, а ткож навантажує користувача непротрібною інформацією.

У даній роботі детально розглядаються існуючі системи зберігання даних та їх характеристики які можуть використовуватись для CMS платформ і пропонується модель, система, чи специфікація системи зберігання даних, яка буде найефективнішою серед альтернатив для вирішення задач, котрих потребує CMS платформа.

Дивлячись на популярність CMS, велику кількість різних систем зберігання даних постає проблема того яку систему зберігання даних обрати для CMS. Це вибір на стадії проектування, який може не приносити швидкодійності чи нешвидкодійності на ранніх етапах роботи системи із невеликою кількістю контенту. Тому треба розуміти як система буде працювати в майбутьому з системою зберігання даних, мати можливість масштабувати таку систему.

Мета роботи полягає в аналізі існуючих систем зберігання даних які можуть використовуватися для CMS систем, для найефективнішого використання прикладним програмним забезпеченням за допомогою методів і алгоритмів теорії прийняття рішень.

Об'єктом дослідження є характеристики зберігання, додавання, індексування, видалення, отримання даних із різних типів та підтипів систем зберігання даних, які необхідні для редагування, оновлення, відображення контенту кінцевому користувачу тої чи іншої інформаційної системи побудованої за допомогою CMS платформи.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

## 1.1 Визначення CMS

Щоб залучити й утримати якомога більше відвідувачів за допомогою веб-сайту, вам необхідний привабливий контент. Тексти, зображення, відео та графіка не тільки забезпечують додаткову цінність ресурсу для користувачів, але і позитивно відзначаються пошуковими системами.

Після створення, будь-який контент повинен бути опублікований, а потім їм треба управляти, оновлювати і поширювати в Інтернеті. Незалежно від розміру веб-сайту, це трудомістке завдання, що реалізується за допомогою системи управління контентом (CMS).

В даний час на ринку є близько 300 різних CMS. При управлінні контентом з їх допомогою зовсім не потрібні навички програмування, а адміністрування зводитися до простих повторюваним функцій – наприклад, створення нових розділів і копіювання тексту з Word у вбудований редактор. При цьому використовується графічний користувальницький інтерфейс, який інтуїтивно зрозумілий більшості людей епохи смартфонів.

Саме програмне забезпечення являє собою інтерактивний ресурс, так зване веб-додаток. На практиці це працює таким чином, що співробітники заходять на свою сторінку входу, щоб потрапити на ту частину сайту, яка невидима для відвідувачів. Ця частина називається бекенда. Загальнодоступною частиною веб-сайту є інтерфейс.

Бекенда використовується для настройки ресурсу і управління контентом. Без CMS оператори веб-сайтів повинні були б редагувати сторінку для кожного зміни за допомогою редактора HTML, а потім завантажувати її на сервер за допомогою програми FTP. З CMS це більше не потрібно.

Простота управління робить ці системи ідеальними для підтримки як великих, так і односторінкових сайтів. Деякі з CMS безкоштовні, деякі з

відкритим вихідним кодом, що означає, що кожен власник може вносити правки під вимоги власних унікальних проектів [1].

Приклад наповнення сторінки різноманітним контентом який можна редагувати наведений на рис 1.1. При чому CMS не задає жорстких обмежень щодо контенту сторінки, і він може бути зміненим за бажанням контент автора. Будь який елемент сторінки може бути видалений або продубльований у інших секціях сторінки.

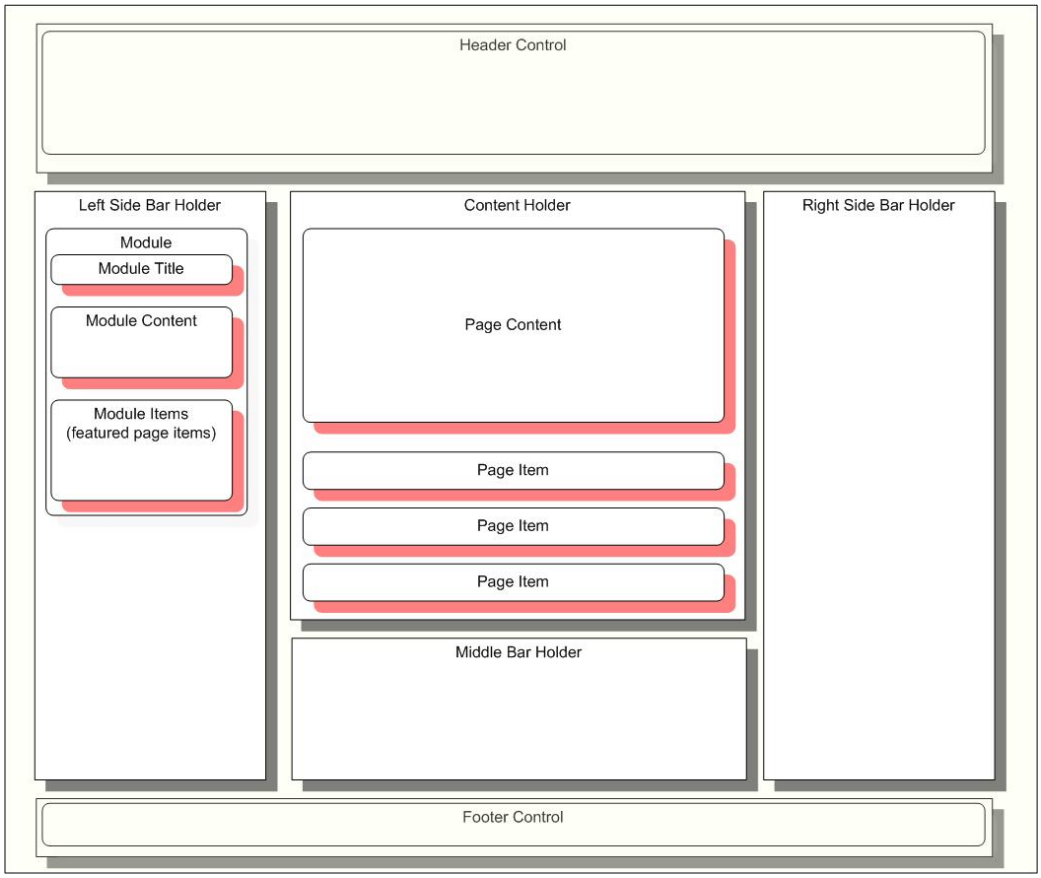


Рисунок 1.1 – Сторінка в CMS

## 1.2 Класифікація CMS

До сьогодні не розроблено досить чіткої класифікації систем управління контентом. Це відбувається тому що ринок CMS досить молодий і розробники цієї категорії програмних продуктів не мають чітких визначень та обмежень.

Досить складно їх розділити їх на якісь групи ще тому що вони досить сильно відрізняються одна від одної. Тому ту чи іншу класифікацію модна назвати у якійсь мірі умовною.

Під час аналізу предметної області було виділено 4 критерії класифікації CMS: 1) за областю призначення; 2) за видом росповсюдження; 3) за рівнем складності; 4) за способом роботи.

На рис. 1.2 приведена розроблена класифікація CMS [2].

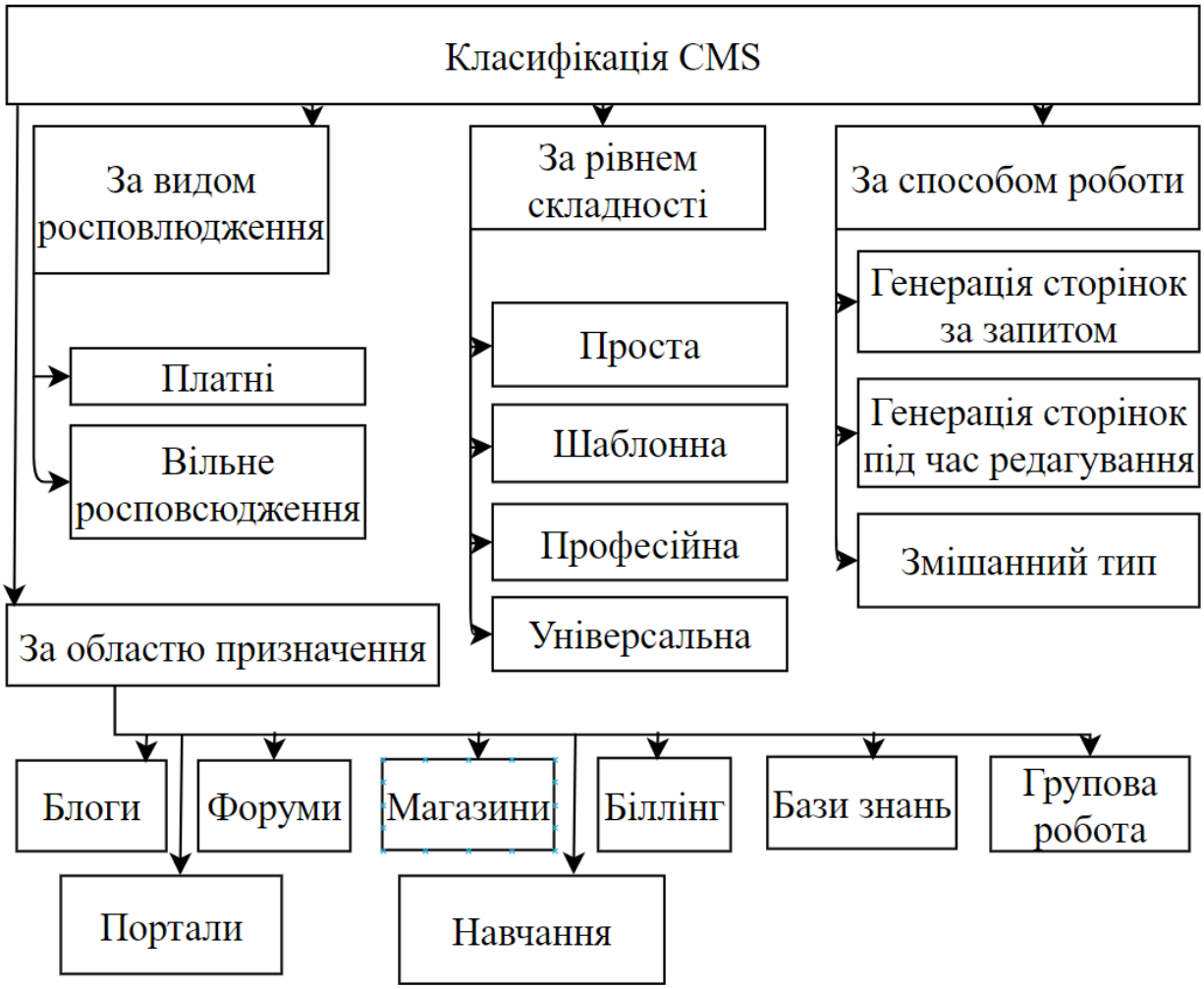


Рисунок 1.2 – Класифікація CMS

За способом росповсюдження системи управління контентом бувають платними і вільно роспосюджуванними. Із платними системами клієнти досить часто отримують супроводження і підтримку розробників.

Платні системи можна розділити на три цінові категорії:

– найбільш дешеві, які створені одним-двома WEB-розробниками чи невеликою групою розробників. Такі системи здебільше розроблюються як універсальні, для того щоб продати якомога більше копій потенційному покупцю (як правило правило представник малого бізнесу невеликих компаній);

– системи середнього цінового діапазону, які створюються для конкретного замовника;

– системи, створені гігантами розробки – Microsoft, Adobe, SAP та ін.

Для класифікації за рівнем складності можна виділити наступні характеристики:

− наявність тих чи інших функцій та модулів, зрозумілість та доступність для користувача;

− можливість функціонування системи на різних платформах, сумісність із базами даних, можливість підключення додаткових модулів;

− технологічність – використання технологій, які дозволяють підвищити безпеку та швидку роботу системи;

− потенціал системи.

Загалом, за рівнями складності CMS системи можна розділити на наступні групи:

− проста CMS система. Система збирається з різних програмних модулів, для кожної системи вони свої. Модулі тільки одни раз налаштовуються розробником, що жорстко закріпляє структуру проекту. Для подальшого змінення структури сайту та його параметрів необхідна участь технічного персоналу. Від користувача системи необхідне знання основ HTML. Сумісність: система сумісна з деякими платформами та типами СУБД. Спроба впровадження додаткових модулів, в залежності від використовуваних технологій , може привести до повної переробки проекту. Технологічність: динамічне формування сторінок, обмежена пропускна спосібність у діапазоні 3000-5000 відвідувачів за добу. Система встановлюється розробником. Розвиток сайту досягається за допомогою міграції сайту на іншу, більш технологічну CMS;

– шаблонна CMS система. Єдиний або набір модулів, з жорстко закріпленою структурою сайту. Система зберігає низку сервісних функцій, які дозволяють виконати стандартні операції з сайтом: сформувати стрічку новин, створити чи додати нову категорію, вибрати шаблон для роботи з інформацією, завантажити файл та розмістити картинку з текстом, додати атрибути тексту і т. д. Система сумісна з деякими платформами та типами СУБД. Спроба впровадження додаткових модулів також обмежена. Формування сторінок відбувається динамічно або із використанням кешування даних. В залежності від методів формування сторінок даний сайт може обробляти 5000-50000 користувачів за добу. Встановлюється розробником. Вдосконалюється за допомогою міграції на більш потужні CMS;

– професійна CMS система. Інтуітивно зрозумілий інтерфейс, візуалізація та розширені можливості редагування. Можливість зміни структури проекту – створення найрізноманітніших сайтів. Сумісність з різними апаратно-програмними платформами. Можливість підключення додаткових модулів як від розробника, так і прикладного ПО. Кешування сторінок які формуються динамічно. Пропускна спосібність не обмежена (залежить від апаратного забезпечення). Можливість вільного підключення додаткових модулів без порушення структури та ідеології керування WEB-ресурсом;

– універсальна CMS система. Має передові можливості управління контентом, налаштування функціональності системи, можливість зміни атрибутів об’єктів сайту. Система надає можливості для розробки нових сервісів та можливостей. Технічні ознаки. Розширені можливості API, наявність готових прикладних рішень, у тому числі з участю програмних продуктів сторонніх бібліотек. Наявність сертифікованої системи забезпечення безпеки – розподілення прав доступу до системи на корпоративному рівні. Кешування сторінок які формуються динамічно. Пропускна спосібність – не обмежена.

За способом роботи CMS системи можна розділити на три типи:

– Генерація сторінок за запитом. Системи такого типу працюють на основі моделі «Модуль редагування – База даних – Модуль відображення».

Модуль відображення генерує сторінку зі змістом при запиті на нього, на основі даних із системи зберігання даних. Інформація яка зберігається в системі збереженя даних змінюється за допомогою модуля редагування. На кожний запит створюється нова сторінка, що в свою чергу створює додаткове навантаження на системні ресурси. Навантаження може бути зменшеним за допомогою кешування, кешування загалом є в більшості сучасних WEB-серверів;

– Генерація сторінок під час редагування. Системи такого типу суть програми для редагування сторінок, які при внесенні змін у змісті сайту створюють набір статичних сторінок. У такий спосіб жертвою є інтерактивність користувача сайту зі змістом сайту.

– Змішаний тип. Цей тип комбінує у собі переваги перших двух. Може бути реалізований за допомогою кешування – модуль відображення генерує сторінку один раз, далі ця сторінка в декілька раз швидше буде відображатись, томущо буде діставатись із кешу. Кеш може оновлюватися як автоматично, наприклад за допомогою якогось таймеру, або при внесенні змін у контент сторінки, так і по запиту адміністратора. Інший підхід – збереження деяких інформаційних блоків на етапі редагування сайту, та створення сторінки із цих блоків при запиті відповідної сторінки користувачем.

Для класифікації за рівнем складності можна виділити наступні характеристики:

− наявність тих чи інших функцій та модулів, зрозумілість та доступність для користувача;

− можливість функціонування системи на різних платформах, сумісність із базами даних, можливість підключення додаткових модулів;

− технологічність – використання технологій, які дозволяють підвищити безпеку та швидку роботу системи.

За областю використання CMS системи можна розділити на наступні групи:

– Портали. Використовуються для інформаційних ресурсів, основною метою є максимальне спрощення публікації тої чи іншої статті або новини. Може включати в себе усю низку нижче згаданих систем управління контентом як окремі CMS модулі. Найбільш відомі представники даного класу: Bes-cms, CPG-Nuke, CoolPHP, AngelineCMS, вебZE, XOOPS, xNuke, Xaraya и др.

– Системи без SQL. Ця гілка в розробці CMS розвита досить слабо, так як використання у ролі зберігання інформації файлів замість таблиць бази даних з'єднано з великою кількістю невирішуємих проблем (таких, як одночасний запис в один файл декількома копіями скрипту). Перевага таких систем – в доступності модифікації контенту та можливості розміщення на бескоштовних хостингах. Є декілька реалізацій даної моделі: Cute News, Adobe Experience Manager, DeeLight CMS, SAPID, Progressive;

– Блог (від англійскої weblog, український термін «мережевий щоденник») – це сайт на якому знаходяться особисті нотатки автора. В основнуму нотатками є посилання на сайти, которі здаються володарю ресурсу цікавими, а також коментарії то них. Блог може містити у собі не тільки посилання, а також і просто елоктронний щоденник користувача. До цієї категорії можна віднести наступні CMS: bBlog, b2evolution, MyPHPblog, BLOG:CMS, pLog, Nucleus, pMachine Free, Textpattern, WordPress, Serendipity, XHP;

– Форуми – це інструмент для спілкування на сайті. Повідомлення в форумі у чомусь схожі на поштові – кожне з них має автора, тему та зміст. Але для того, щоб відправити повідомлення – потрібно заповнити відповідну форму на сайті. Головна функція форуму є в тому, що повідомлення в ньому об'єднані темою. Коли ви відповідаєте у форумі на чиєсь повідомлення, ваша відповідь буде «прив'язана» до початкового повідомлення. Форуми, які слід виділити: openBB, FUDforum, phpBB, Phorum, W-Agora, PunBB, Zorum, XMB, ExBB, vBulletin, IPB;

– Магазини. До магазинів віднесемо будь-який сайт, із якого можна замовити будь-який товар. У даному випадку у визначення «товару» може

входити абсолютно все, включаючи час доступу у Інтернет, хвилини мобільного з'язку. CMS, які дозволяють створювати віртуальний магазин: osc2nuke, MyMarket, Zen Cart, osCommerce;

– Групова работа (Groupware) – комплекс програмного забезпечення, які дозволяють організувати роботу підприємства, відношення з клієнтами в інтернеті. Зазвичай є повністю закритою частиною сайту або частинно закриту із можливістю відстеження сроків виконання поставлених завдань, росподілення ролей та часових нормативів. Іноді можна виносити питання на обговорювання та рішення керівництва. Зазвичай, використовують наступні CMS: eGroupWare, dotProject, phpCollab, MoreGroupware, PHProjekt;

– Навчання – дистанційна форма навчання із використанням Інтернету. Онлайн форма навчання уже багато років є потужною формою навчання у всьому світі, для майже усіх галузей навчання. Головним стратегічним направленням є швидке оновлення даних, та ефективне використання інформації. Таких дуже багато на наш час: Udemy, CouseInfo, LinkedIn learning.

– Бази знань – дозволяють накопичувати досвід множин розробників, працюючих або працювавших в той чи інший області знань. Кожна така база знань має свою специфічну структуру, тому ніяких загальних рішень на даний момент немає. Найпопулярніша база знань – RFC.

– Білінг – програмне забезпечення, дозволяє провайдерам та реселлерам працювати із рахунками клієнтів. Такі системи є невід'ємною частиною масштабної системи обліку використання послуг користувачами. Задача ситеми цієї категорії – у відображенні інформації про послуги, підключення нових послуг, змінення параметрів, прийому платежів і т.д. Часто такі системи створюються для прикладних задач.

– Панель адміністратору хостингу – до цбого класу відносяться такі продукти як Direct Admin та Control Panel. Немало хостинг провайдерів намагаються створити панель керування для користувача зостингу самостійно, але до вищезгаданих систем таким системам складно дотягнути свій рівень.

## 1.3 Класифікація баз даних

База даних (БД) - упорядкований набір логічно взаємопов'язаних даних, що використовується спільно, та призначений для задоволення інформаційних потреб користувачів. У технічному розумінні включно й система управління БД [3].

Система управління базами даних (СУБД) - це комплекс програмних і мовних засобів, необхідних для створення баз даних, підтримання їх в актуальному стані та організації пошуку в них необхідної інформації.

Централізований характер управління даними в базі даних передбачає необхідність існування деякої особи (групи осіб), на яку покладаються функції адміністрування даними, що зберігаються в базі.

Головним завданням БД є гарантоване збереження значних обсягів інформації та надання доступу до неї користувачеві або ж прикладній програмі. Таким чином БД складається з двох частин: збереженої інформації та системи управління нею. З метою забезпечення ефективності доступу записи даних організовують як множину фактів (елемент даних) [3].

Окреме місце в теорії та практиці займають просторові, тимчасові, або темпоральні і просторово-часові БД.

Ієрархічні бази даних можуть бути представлені як дерево, що складається з об'єктів різних рівнів. Верхній рівень займає один об'єкт, другий - об'єкти другого рівня і т.д.

Існує величезна кількість різновидів баз даних, що відрізняються за критеріями (наприклад, в Енциклопедії технологій баз даних визначаються понад 50 видів БД). Відзначимо тільки основні класифікації (наведені на Рис. 1.3).

Класифікація БД за моделлю даних:

- ієрархічні;
- мережеві;

- реляційні;
- об'єктні;
- об'єктно-орієнтовані;
- об'єктно-реляційні.

Класифікація БД за технологією фізичного зберігання:

- БД у вторинній пам'яті (традиційні);
- БД в оперативній пам'яті (in-memory databases);
- БД у третинній пам'яті (tertiary databases).

Класифікація БД за вмістом:

- географічні;
- історичні;
- наукові.
- мультимедійні.

Класифікація БД за ступенем розподіленості:

- централізовані (зосереджені);
- розподілені.


Рисунок 1.3 – Класифікація баз даних

Між об'єктами існують зв'язки, кожен об'єкт може включати в себе декілька об'єктів більш низького рівня. Такі об'єкти перебувають у відношенні предка (об'єкт більш близький до кореня) до нащадка (об'єкт більш низького

рівня), при цьому можлива ситуація, коли об'єкт-предок не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок. Об'єкти, що мають загального предка, називаються близнюками. Приклад схеми даних в ієрархічній базі даних наведено на Рис. 1.4.

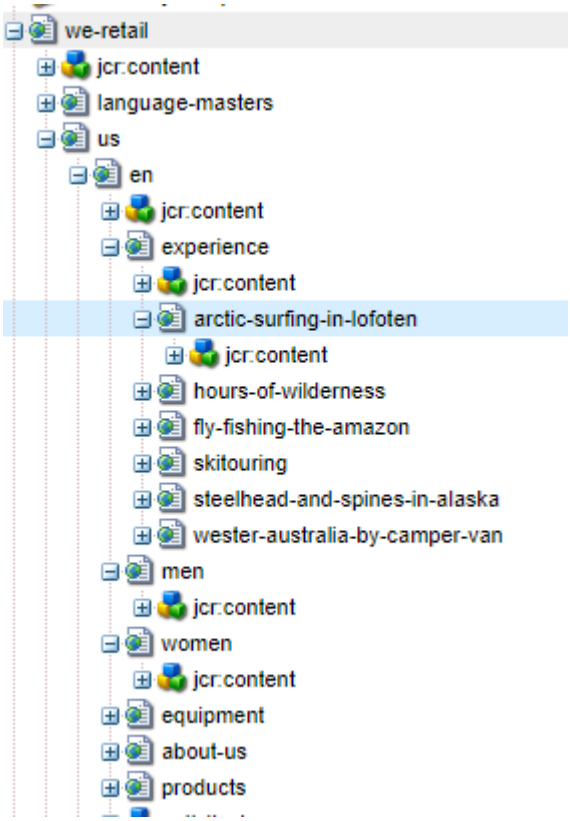


Рисунок 1.4 – Приклад схеми даних в ієрархічній базі даних

Мережеві бази даних подібні до ієрархічних, за винятком того, що в них є покажчики в обох напрямках, які з'єднують споріднену інформацію.

До основних понять мережевої моделі бази даних відносяться: рівень, елемент (вузол), зв'язок.

Вузол - це сукупність атрибутів даних, що описують деякий об'єкт. На схемі ієрархічного дерева вузли представляються вершинами графа. У мережній структурі кожен елемент може бути пов'язаний з будь-яким іншим елементом.

Незважаючи на те, що ця модель вирішує деякі проблеми, пов'язані з ієрархічною моделлю, виконання простих запитів залишається досить складним процесом.

Також, оскільки логіка процедури вибірки даних залежить від фізичної організації цих даних, то ця модель не є повністю незалежною від програми. Іншими словами, якщо необхідно змінити структуру даних, то потрібно змінити і додаток. Приклад схеми для мережевих баз даних наведено на Рис. 2.5.

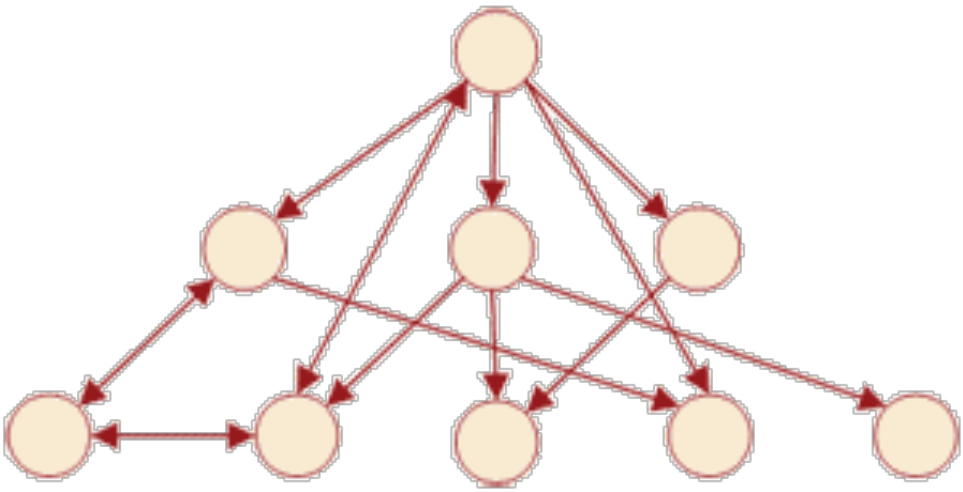

Рисунок 1.5 – Приклад схеми даних в мережевій базі даних

Реляційна модель орієнтована на організацію даних у вигляді двовимірних таблиць. Кожна реляційна таблиця являє собою двовимірний масив і має наступні властивості:

- кожен елемент таблиці - один елемент даних;
- всі осередки в стовпчику таблиці однорідні, тобто всі елементи в стовпчику мають однаковий тип (числовий, символьний тощо);
- кожен стовпчик має унікальне ім'я;
- однакові рядки в таблиці відсутні;
- порядок проходження рядків і стовпчиків може бути довільним.

Об'єктна СУБД ідеально підходить для інтерпретації складних даних, на відміну від реляційних СУБД, де додавання нового типу даних досягається

ціною втрати продуктивності або за рахунок різкого збільшення термінів і вартості розробки додатків. Об'єктна база, на відміну від реляційної, не вимагає модифікації ядра при додаванні нового типу даних. Новий клас і його екземпляри просто надходять у зовнішні структури бази даних. Система управління ними залишається без змін.

Об'єктно-орієнтована база даних (ООБД) - база даних, в якій дані оформлені у вигляді моделей об'єктів, що включають прикладні програми, які управляються зовнішніми подіями. Результатом поєднання можливостей (особливостей) баз даних і можливостей об'єктно-орієнтованих мов програмування є об'єктно-орієнтовані системи управління базами даних (ООСУБД). ООСУБД дозволяють працювати з об'єктами баз даних також, як з об'єктами у програмуванні в об'єктно-орієнтованих мовах програмування. ООСУБД розширює мови програмування, прозоро вводячи довготривалі дані, управління паралелізмом, відновлення даних, асоційовані запити й інші можливості.

Об'єктно-орієнтовані бази даних звичайно рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, які мають складну структуру.

Система, яка забезпечує об'єктну інфраструктуру і набір реляційних розширювачів, називається "об'єктно-реляційною".

Об'єктно-реляційні системи поєднують переваги сучасних об'єктно-орієнтованих мов програмування з такими властивостями реляційних систем як множинні представлення даних і високорівневі непроцедурні мови запитів.

За технологією обробки даних бази даних поділяються на централізовані й розподілені. Приклад схеми для реляційних баз даних наведено на Рис. 1.6.

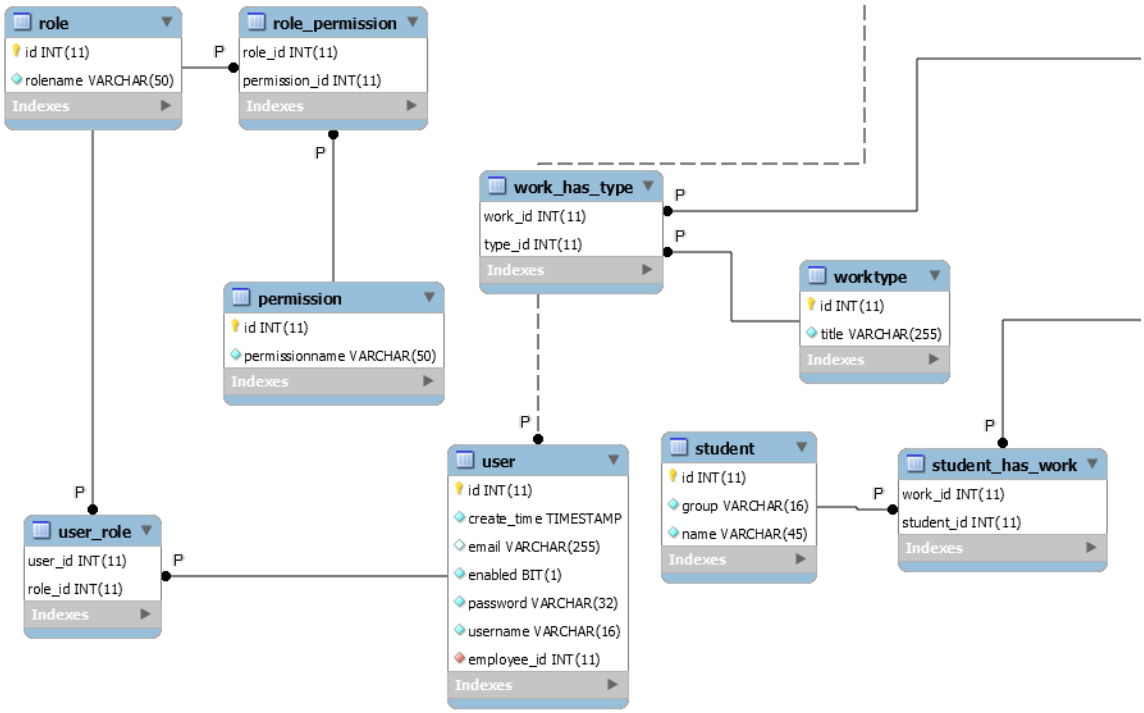


Рисунок 1.6 – Приклад схеми даних в реляційній базі даних

Централізована база даних зберігається у пам'яті однієї обчислювальної системи. Якщо ця обчислювальна система є компонентом мережі ЕОМ, можливий розподілений доступ до такої бази. Такий спосіб використання баз даних часто застосовують у локальних мережах ПК.

Розподілена база даних складається з декількох, можливо пересічних або навіть дублюючих одна одну частин, які зберігаються в різних ЕОМ обчислювальної мережі. Робота з такою базою здійснюється за допомогою системи управління розподіленою базою даних (СУРБД).

За способом доступу до даних бази даних поділяються на бази даних з локальним доступом і бази даних з віддаленим (мережевим) доступом.

Системи централізованих баз даних з мережевим доступом припускають різні архітектури подібних систем:

- файл-сервер;
- клієнт-сервер.

Файл-сервер. Архітектура систем БД з мережевим доступом передбачає виділення однієї з машин мережі в якості центральної (сервер). На такій машині зберігається спільно використовувана централізована БД. Усі інші машини

мережі виконують функції робочих станцій, за допомогою яких підтримується доступ користувальницької системи до централізованої бази даних. Файли бази даних відповідно до призначених для користувача запитів передаються на робочі станції, де в основному і проводиться обробка. При великій інтенсивності доступу до одних і тих же даних продуктивність інформаційної системи падає. Користувачі можуть створювати також на робочих станціях локальні БД, які використовуються ними монопольно.

Клієнт-сервер. У цій концепції мається на увазі, що крім зберігання централізованої бази даних центральна машина (сервер бази даних) повинна забезпечувати виконання основного обсягу обробки даних. Запит на дані, який видається клієнтом (робочою станцією), породжує пошук і вилучення даних на сервері. Витягнуті дані (але не файли) транспортуються по мережі від сервера до клієнта. Специфікою архітектури клієнт-сервер є використання мови запитів SQL.

## 1.4 Постановка задачі дослідження

Розробити модель даних інформації для CMS направлених систем, моделі інформації необхідно створити для різних систем управління базами даних.

Створити архітектуру CMS системи, яка покриває сценарії роботи із зміною та публікуванням контенту на роподілені сервери. До аналізу ефективності швидкодйності СУБД винести швидкість роботи усієї системи та окремих її модулів.

Спроєцювати розроблену концептуальну модель даних на фізичний рівень. Розробити алгоритми обробки запиту сторінок на існуючій моделі даних. Розробити сторінкому модель даних, яка не буде залежити від предметної області.

Розробити сценарії тестування системи, створити алгоритми та реалізацію цих алгоритмів. Виконати тестування системи за допомогою алгоритмів.

Проаналізувати дані отримані під час тестування. Визначити переваги таа недоліки кожної СУБД для систеи управління контентом.

Спроектувати оптимальну арзітектуру CMS системи, надати рекомендації що до використання тієї чи іншої системи зберігяння даних для систем управління контентом.

# 2 ПОРІВНЯННЯ МОДЕЛЕЙ ДАНИХ ТА ІХ ВИКОРИСТАННЯ ДЛЯ РІЗНИХ СУБД

## 2.1 Опис інфраструктури високонавантаженої CMS системи

CMS це не сайт де на одному й тому ж хостингу, в одному й тому ж місці віддається контент користувачам та контент авторам. Для створення контенту використовується так званий author instance[3]. Це програма – де контент автори конфігурують сторінки, редагують та публікують їх. Публікація контенту – це реплікація даних із author instance на delivery instances.

Delivety instance – це програма, яка надаж контент кінцевим користувачам системи, як правило, кожен сервер такої програми знаходиться на окремому фізичному носії. Це зроблелено для того щоб:

- розподілювати навантаження між серверами;
- у разі недоступності якогось із серверів delivery балансувальник буде використовувати тільки доступні сервери;

Крім того що є delivery instances для кожного з них є WEB-сервер, який відповідає за:

- кешування контенту, що пришвидшує доставку сторінок, зображень, медіа до кінцевого користувача. Кешування контенту дуже важливе у CMS системах, адже кожна сторінка не статична, вона складається із різних частин, які конфігуруються контент автором. Сторінка може оновлюватися це автоматично виклакає оновлення кешу;
- проксуванні певних запитів на іншосторонні ресурси або сервіси через певні цілі наприклад як передача хедерів, кук до іншостороннього ресурсу;
- додаткові інструменти які дозволяють конфігуруви безпеку сайту. Під безпекою мається на увазі перенаправлення на іншу сторінку із кодом 302, переписування внутрішніх посилань, додавання заголовків безпеки у запити;
- Віддача контенту під час недоступності веб серверу. Напринлад усі

delivery сервери недоступні з тої чи іншої причини, а WEB-сервери мають у себе сторінки які будуть відображати найпопулярніші помилки такі як 404, 400, 500, 502.

Інфраструктуру повної системи можна побачити на Рис 2.1, крім того можна додати Akamai для того щоб кінцевий користувач потрапляв на веб сервер, який дозволяє потрапляти на найближчий WEB-сервер, якщо це можливо. Далі система управління контентом буде розглядатися на такій архітектрурі, бо така архітектура використовується майже в усіх контент системах.

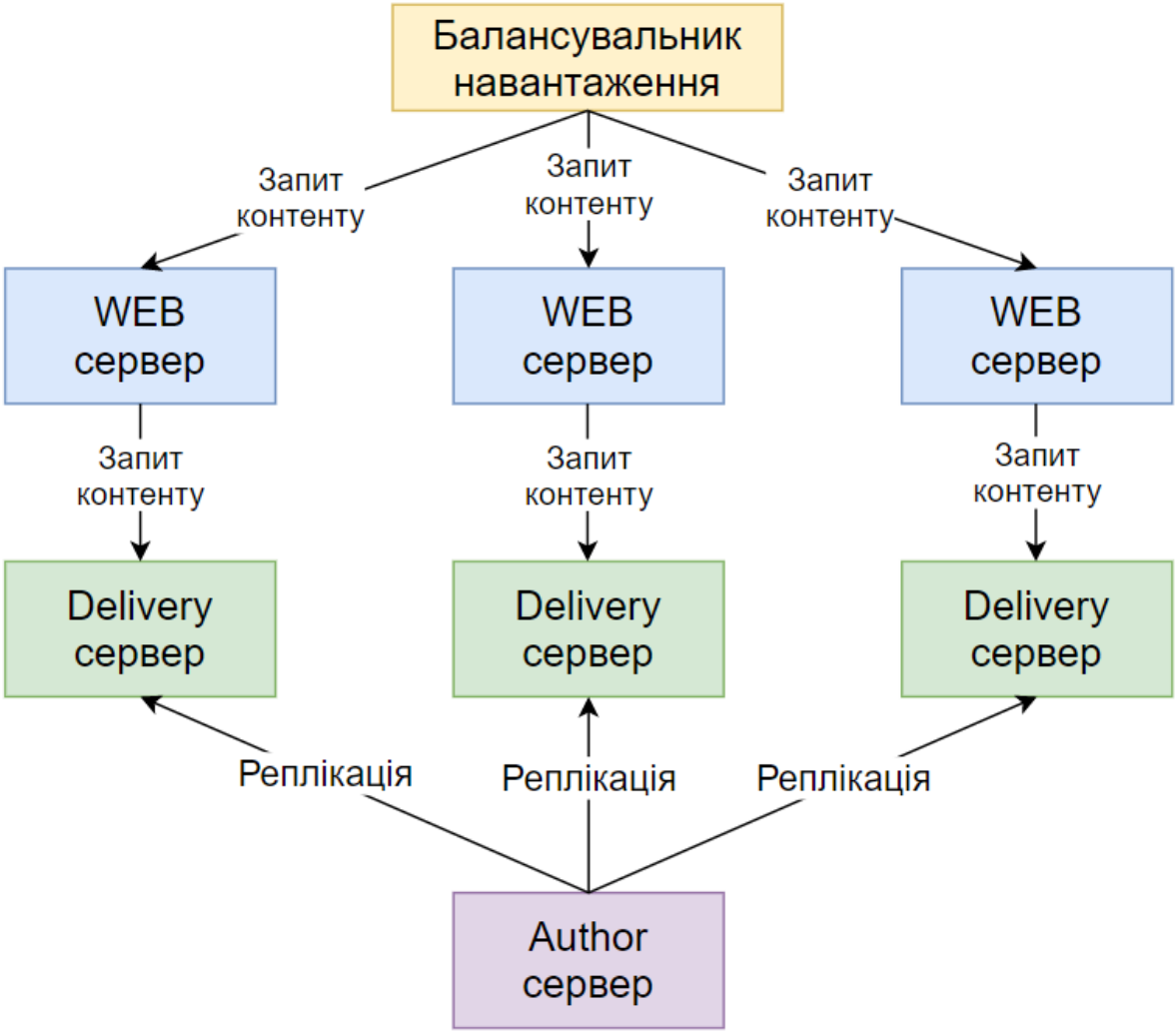


Рисунок 2.1 – Архітектура системи управління контентом

## 2.2 Опис моделі даних

Для опису моделі даних потрібна предметна область. Від тої чи іншої предметної області не буде залежити підхід до створення моделі даних для CMS системи, адже CMS маніпулює поняттями сторінки, та контенту сторінки.

Так як вибір предметної області майже не пливає на оцінку СУБД, то за предметну облать можна взяти систему дистанційного навчання, під час пандемії яка зараз триває, потребність у таких системах значно виросла.

Для того щоб описати модель даних – потрібно визначити основні функції системи. На Рис 2.2 та 2.3 та наведено основні функції системи, такі як створення курсу, створення тесту, тощо.

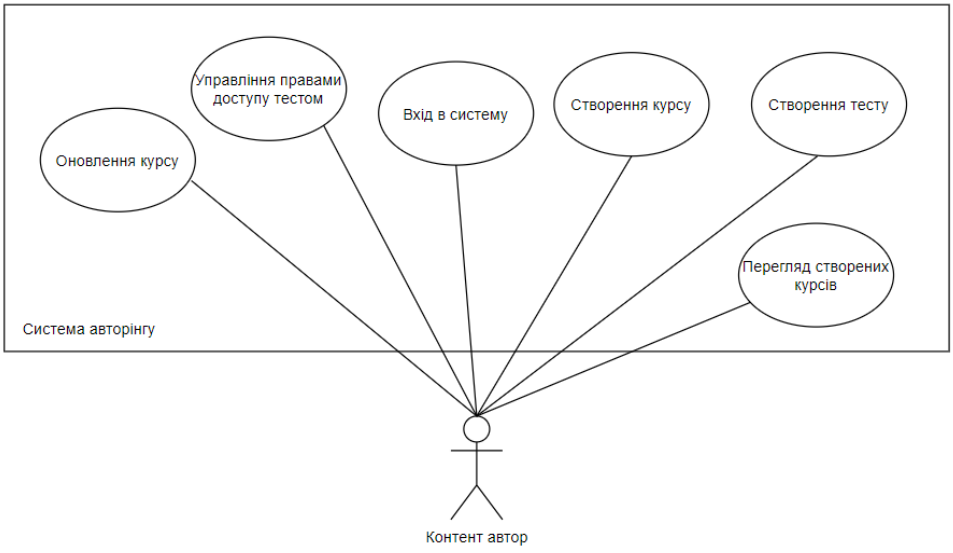


Рисунок 2.2 – Основні функції системи для контент автора

На базі описаних функцій показаних на UML Use Case діаграмі можна будувати модель даних. Діагарама надає можливість визначити сутності які будуть використовуватись в системі.

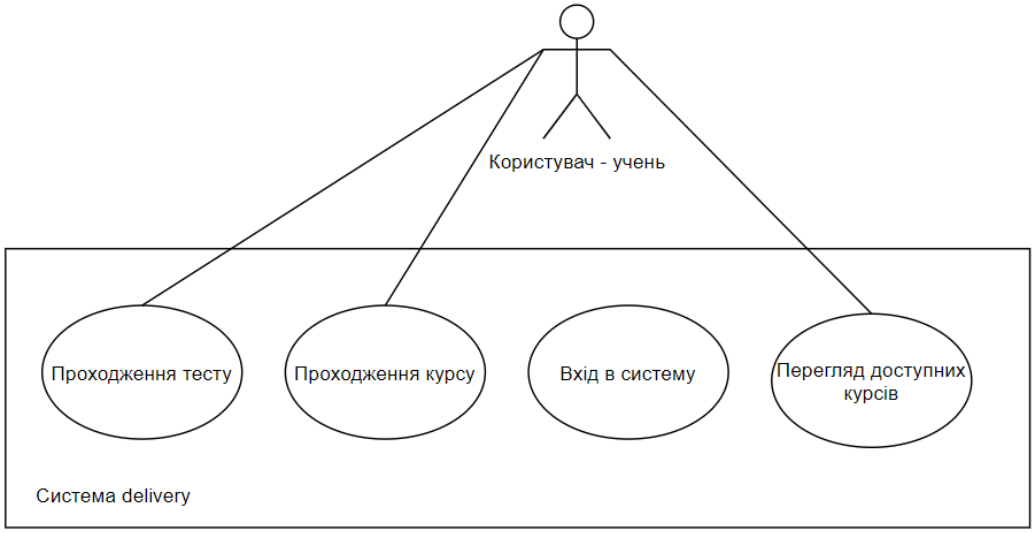


Рисунок 2.3 – Основні функції системи для учня

В табл 2.1 наведено опис сутності Контент Автора.

Таблиця 2.1 – опис сутності Контент Автора

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор користувача |
| firstName | string | Ім’я користувача |
| lastName | string | Прізвище користувача |
| email | string | Поштова адреса користувача |
| password | string | Пароль користувача |

В табл 2.2 наведено опис сутності Учня. Ця сутність є ідентичною до сутності Контент Автора, їх не можна об’єднувати, бо на author instance користувачі кчні будуь взагалі відсутні [4].

Таблиця 2.2 – опис сутності учня

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор користувача |
| firstName | string | Ім’я користувача |

*Продовження таблиці 2.2*

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| lastName | string | Прізвище користувача |
| email | string | Поштова адреса користувача |
| password | string | Пароль користувача |

В табл 2.3 наведено опис сутності Курсу. Такі параметри як mainImagePath та

previewImagePaths відповідають за те щоб відобразити те чи інше зображення для учня, коли той вибирає курс, mainImagePath буде відповідати за відображення зображення певного курсу при перегляді усих доступних курсів, previewImagePaths буде відповідати за детальніші зображення курсу, наприклад користувач захоче дізнатися більше про курс не тільки з його опису, а ще й з точки зору як курс оформлений.

Таблиця 2.3 – опис сутності Курсу

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор курсу |
| name | string | Назва курсу |
| description | string | Опис курсу |
| approximateTimeForComplete | number | Приблизна кількість хвилин для проходження курсу |
| mainImagePath | string | Шлях до голвного зображення курсу |
| previewImagePaths | string[] | Шлях до інших зображень курсу, який користувач може переглянути якщо обирає курс |
| createdBy | number | Id автора который створив курс |

В табл 2.4 наведено опис сутності Розділу Курсу.

Таблиця 2.4 – опис сутності Розділу

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор розділу |
| courseId | number | Номер курсу до якого відноситься розділ |
| name | string | Назва розділу |
| description | string | Опис розділу |
| approximateTimeForComplete | number | Приблизна кількість хвилин для проходження розділу курсу |

В табл 2.5 наведено опис сутності Заняття.

Таблиця 2.5 – опис сутності Заняття

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор заняття |
| chapterId | number | Номер розділу до якого відноситься заняття |
| name | string | Назва заняття |
| description | string | Опис заняття |
| approximateTimeForComplete | number | Приблизна кількість хвилин для проходження заняття |
| hmtlOfCourceContent | string | Контент заняття в html форматі |

В табл 2.6 наведено опис сутності Тесту. Тут ми маємо такі атрибути як lessonId,
chapterId, courseId. Якщо один атрибут має значення, то інші атрибути повинні бути пустими. Це томущо тест може відноситись або до заняття, або до розділу,

або до курсу, або взагалі це просто тест який ні до чого не відноситься, і який можна просто пройти.

Таблиця 2.6 – опис сутності Тесту

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор тесту |
| lessonId | number | Унікальний ідентифікатор заняття |
| chapterId | number | Унікальний ідентифікатор розділу |
| courseId | number | Унікальний ідентифікатор курсу |
| name | string | Назва тесту |
| description | string | Опис тесту |

В табл 2.7 наведено опис сутності Тестового Питання. Атрибут type відповідає за те що тестове питання може бути із типом, коли учень вибирає лиш одну правильну відповідь, або він може вибрати декілька відповідей.

Таблиця 2.7 – опис сутності Тестового питання

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор тесового питання |
| testId | number | id тесту до якого належить питання |
| name | string | Назва тесту |
| description | string | Опис тесту |
| type | string | multiple \| single тип для тестового питання |
| hmtlOfQuestionContent | string | Контент питання в html форматі |

Слід звернути увагу що є сутность які мають в собі html контент, в концеатуальній моделі даних неможливо, відразу дати чітку відповідь як буде зберігатися контент сторінки в якомусь репозиторії, томущо різні типи СУБД надають різні можливості, наприклад в реляційних субд потрібно будет

створювати додоткову табличю для кожного типу контенту, наприклад RTE, звичайне зображення, посилання, параграф, заголовок, тощо.

В табл 2.8 наведено опис сутності Тестової відповіді.

Таблиця 2.8 – опис сутності Тестової відповіді

| Назва атрибуту | Тип атрибуту | Опис атибуту |
|---|---|---|
| id | number | Унікальний ідентифікатор тесової відповіді |
| testQuestionId | number | id питання до якого належить відповідь |
| hmtlOfAnswerContent | string | Контент відповіді в html форматі |
| isCorrect | boolean | Чи правильна відповідь |

Для цих сутностей також була побудована концептуальна модель даних, яка не відноситься ні до якої реалізації певної СУБД. Ця модель лише візуалізація взаємоз'яків створених сутностостей. Для кожної СУБД буде створена своя фізична модель даних, яка може виглядати інакше. Концептуальну модель даних можна побачити на Рис. 2.4.

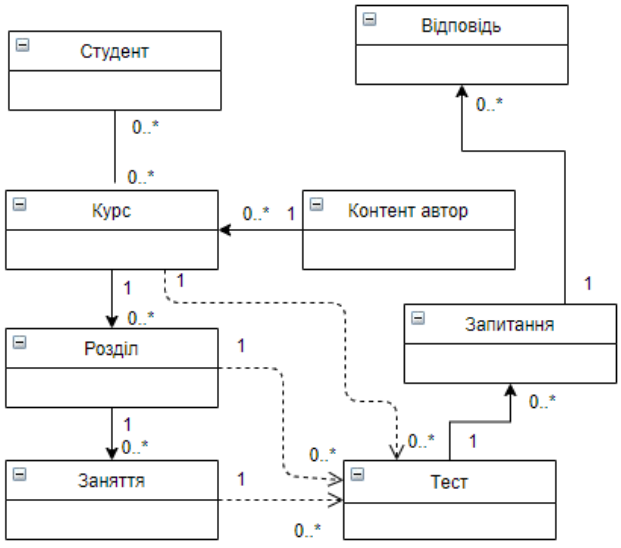


Рисунок 2.4 – Концептуальна схема даних для визначених сутностей предметної області

## 2.3 Створення логічної моделі даних для реляційних СУБД

Для реляційної моделі даних, не має різниці яку субд обрати, для того щоб побудувати модель даних, схема буде однакова для різних типів реляційних СУБД. Реляційну моделі даних можна створити на базі описаних сутностей. Тобто з описаної моделі нам потрібно буде сформувати зовнішні ключі, та поки-що відмовитись від контнтенту сторінок. Це буде лиш представленя даних, а не сторінок. Для сторінок потрібно буде перепобити створену схему. У нас з'являтіся такі сутності як, сторінка, компонент сторінки, тощо. Відображення сторінок та їх зберігання буде побудовано не на розробленій моделі даних, для реляційних баз даних, також потрібно буде розробити модель збереження сторінок, та логіку, як сторінки будуть оброблятися, при запиті на них. Реляційну модель даних, яка відображає розроблену модель даних можна побачити на Рис 2.5.

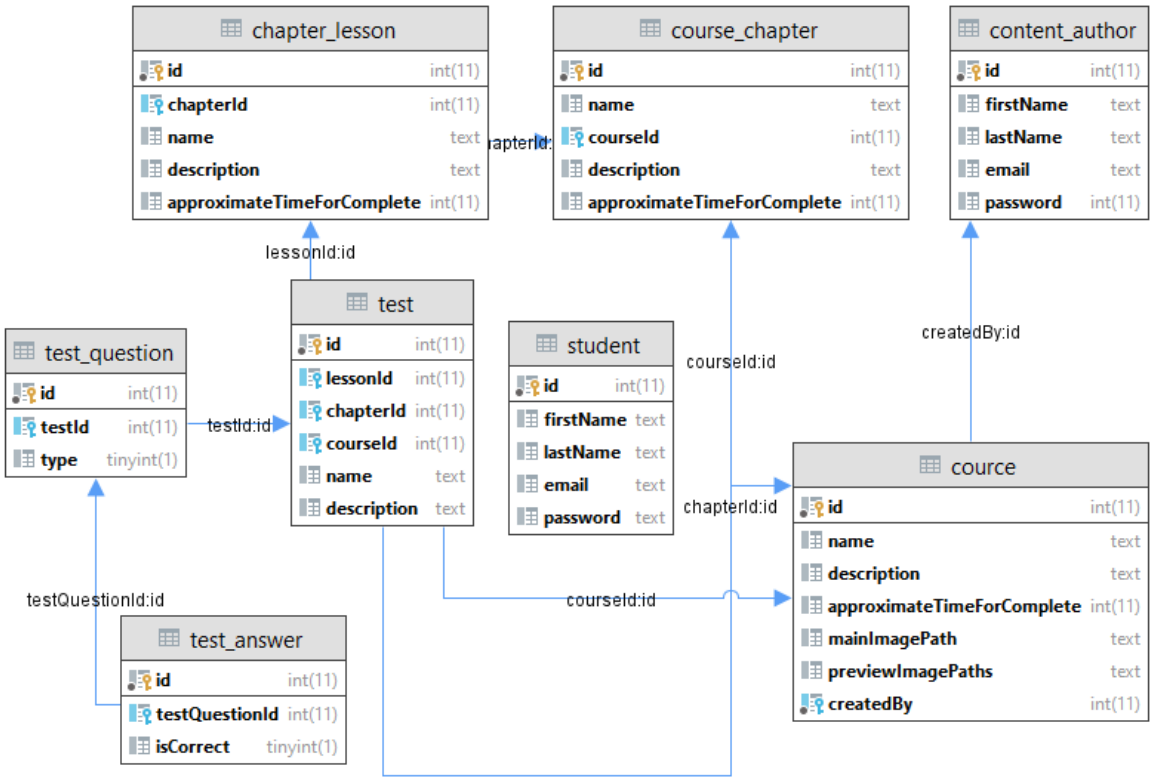


Рисунок 2.5 – Реляційна схема даних

Якщо розглядати механізм отримання сторінки, то нам потрібна сутність сторінки. Крім того ми не можемо робити friendly url із використанням реляційнох бази досить швидко, по перше, це зменшить швидкість розробки, по друге – зменшить швидкість отримання контенту. Тому для формування посилання, наприклад нам потрібно подивитись певний курс, ми повинні передавати id цього курсу як параметр запиту до сервера. На відміну від інших типів СУБД, ми не можемо досить швидко на сервері опрацьовувати запити із friendly-url виглядом.

Для контенту сторінки також необхідна окрема сутність у базі даних, яка буже зберігати компоненти, в яких будуть зовнішні ключі або на інші компоненти, або на сторінки. Також потрібна сутність яка буде визначати необіхідні дані, для певного типу сторінки, та певного типу компоненту. На Рис 2.6 можна побачити алгоритм обробки запиту на сторінку.

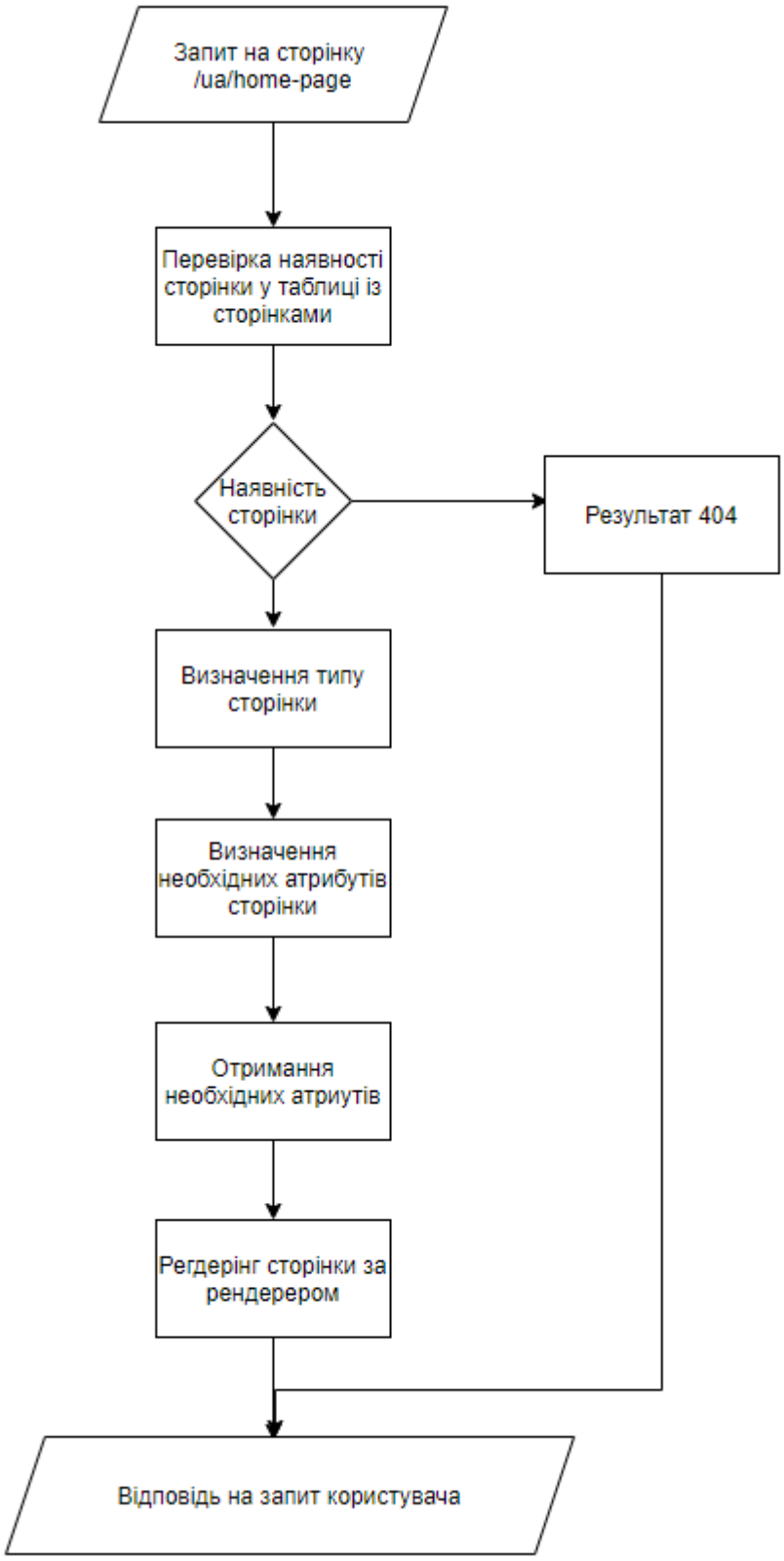


Рисунок 2.6 – Алгоритм обробки запитів із реляційною моделю даних

На Рис 2.7 наведена оновлена реляційна схема даних.

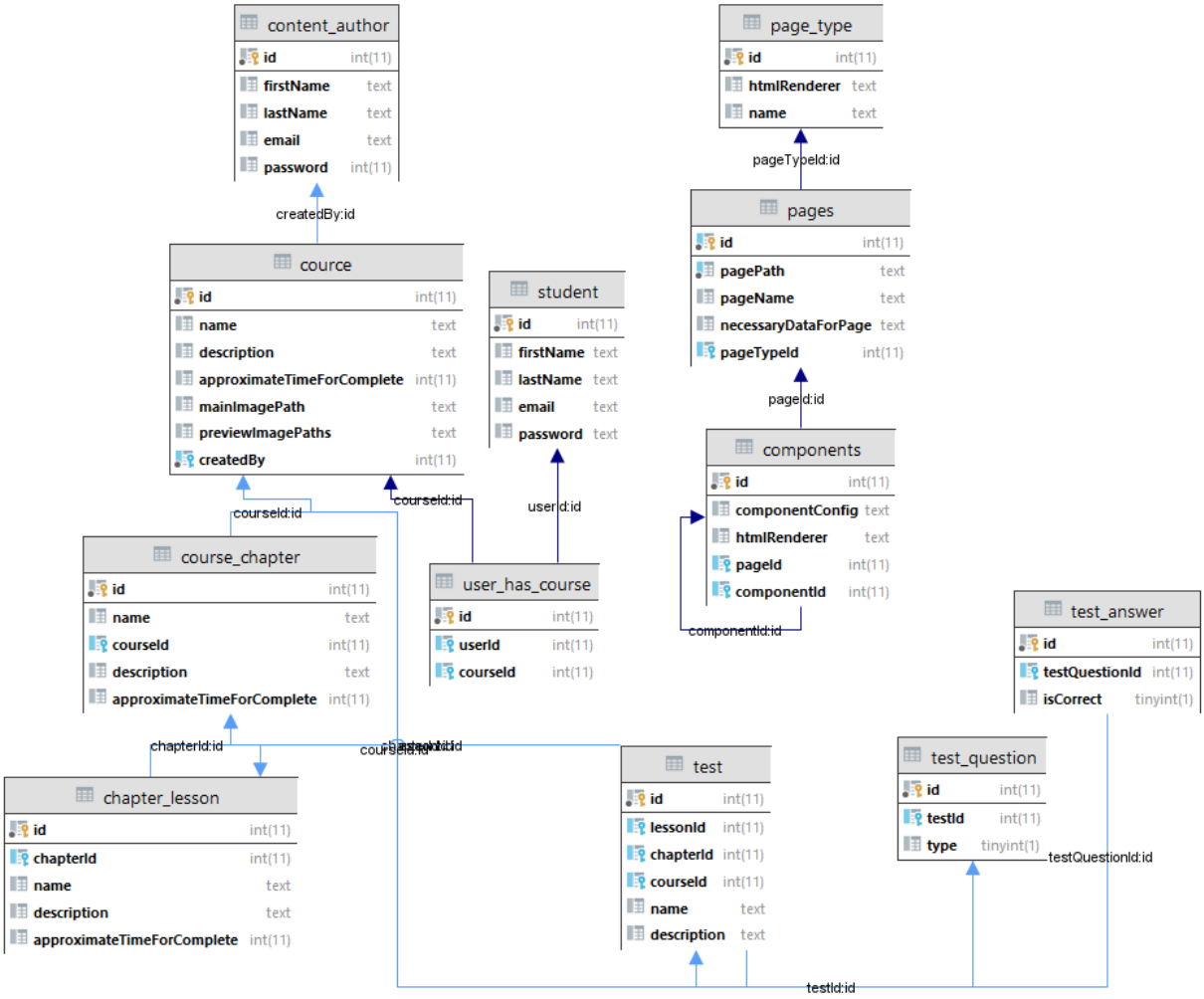


Рисунок 2.7 – Схема даних для ієрархічної структури

На приведеній схемі даних, ми таблицю components. Ця таблиця відповідає за збереження компонентів будь якого типу. Але різні компоненти можуть мати різну конфігурацію. Наприклад ми маємо сторінку з трьома компонентами – link, layout, text. Для ркожного з цих компонентів своя конфігурація. На Рис 2.8 можна побачити несконфігуровані компоненти на сторінці, на Рис. 2.9 можна побачити сконфігуровані компоненти.

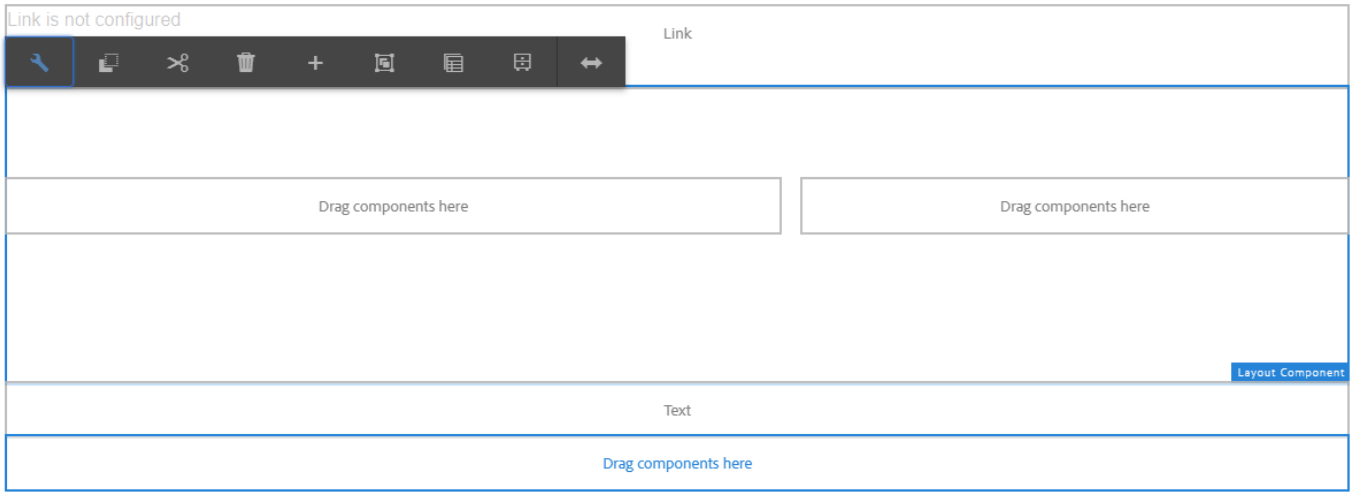


Рисунок 2.8– Приклад компонентів на сторінці

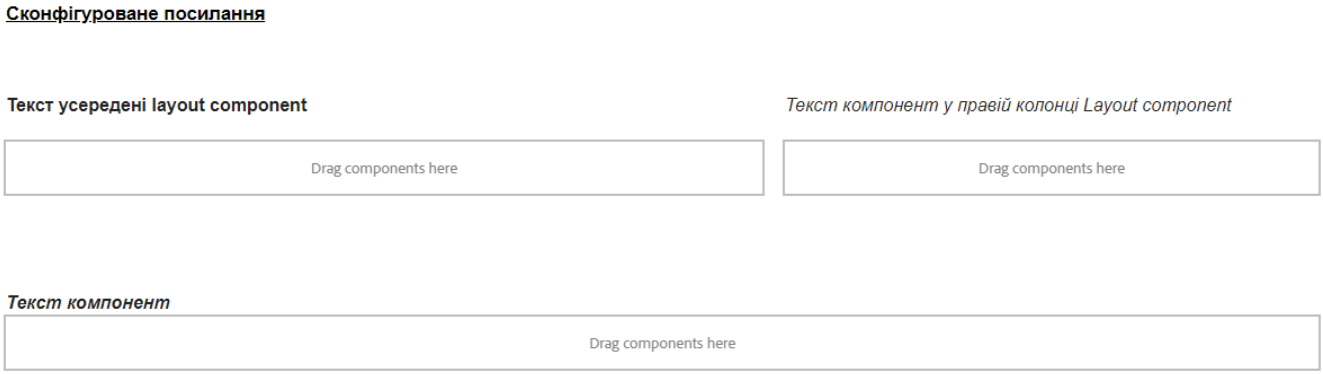


Рисунок 2.9 – Приклад компонентів на сторінці які сконфігуровані

Кожен компонент на сторінці має своє вікно конфігурації із різними параметрами. Відповідно, та конфігурація яка є у компонента, вона і буде зберігатися у базі даних, але в даній розробленій моделі, будь який компонент буде зберігати свої дані в одну таблицю, в певному форматі, чого зотілося б уникнути. Вікна конфігурації можна побачити на Рис 2.10, 2.11, 2.12 [5].

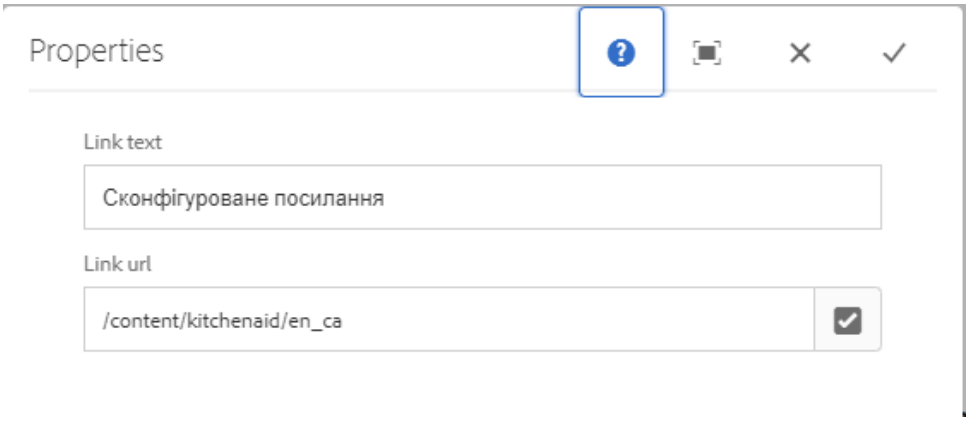


Рисунок 2.10 – Вікно конфігурації для link component

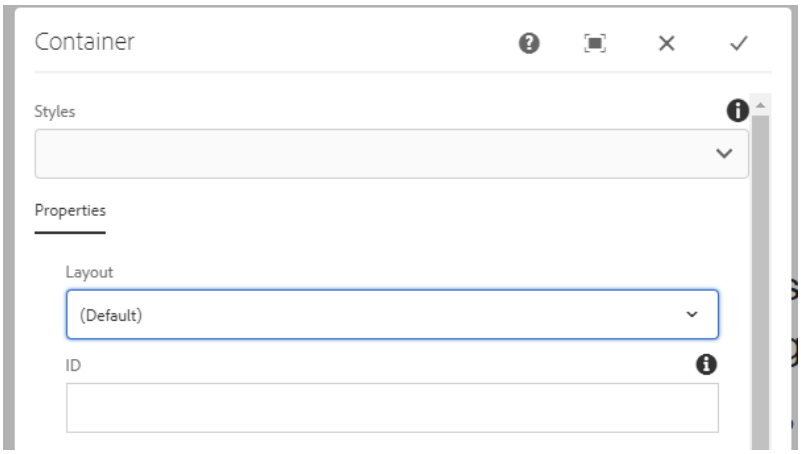


Рисунок 2.11 – Вікно конфігурації для layout component

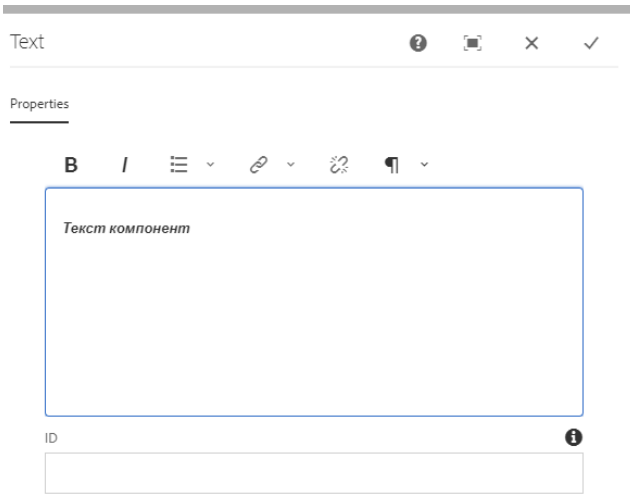


Рисунок 2.12 – Вікно конфігурації для text component

В таблиці 2.9 наведено структуру даних link component

Таблиця 2.9 – атрибути link компоненту

| Назва Атрибуту | Опис |
|---|---|
| linkText | Текст посилання |
| linkUrl | Посилання |

В таблиці 2.10 наведено структуру даних layout component

Таблиця 2.10 – атрибути layout компоненту

| Назва Атрибуту | Опис |
|---|---|
| styles | Специфічні стилі для цього компоненту |
| layout | Тип layout |
| htmlId | Id атрибут який повинен бути присвоєний в html |

В таблиці 2.11 наведено структуру даних text component

Таблиця 2.11 – атрибути layout компоненту

| Назва Атрибуту | Опис |
|---|---|
| text | Сконфігурований текст |
| htmlId | Id атрибут який повинен бути присвоєний в html |

Зрозуміло що це тільки приклад трьох компонентів, але CMS система може мати більш ніж 200 різних компонентів, де кожен буде важливий і буде використовуватися на сторінках певного типу, але додавати 200, зовнішніх ключів, до сторінки досить дорого по разувальним ресурсам, тому також необхідно створити деяку сутність, котра буде просто компонентом, і буде зв’язувати між собою певний компонент зі сторінкой, наприклад як дейкий uuid. На 2.13 можна побачити оновлену схему реляційної бази даних, із описании вище сутностями.

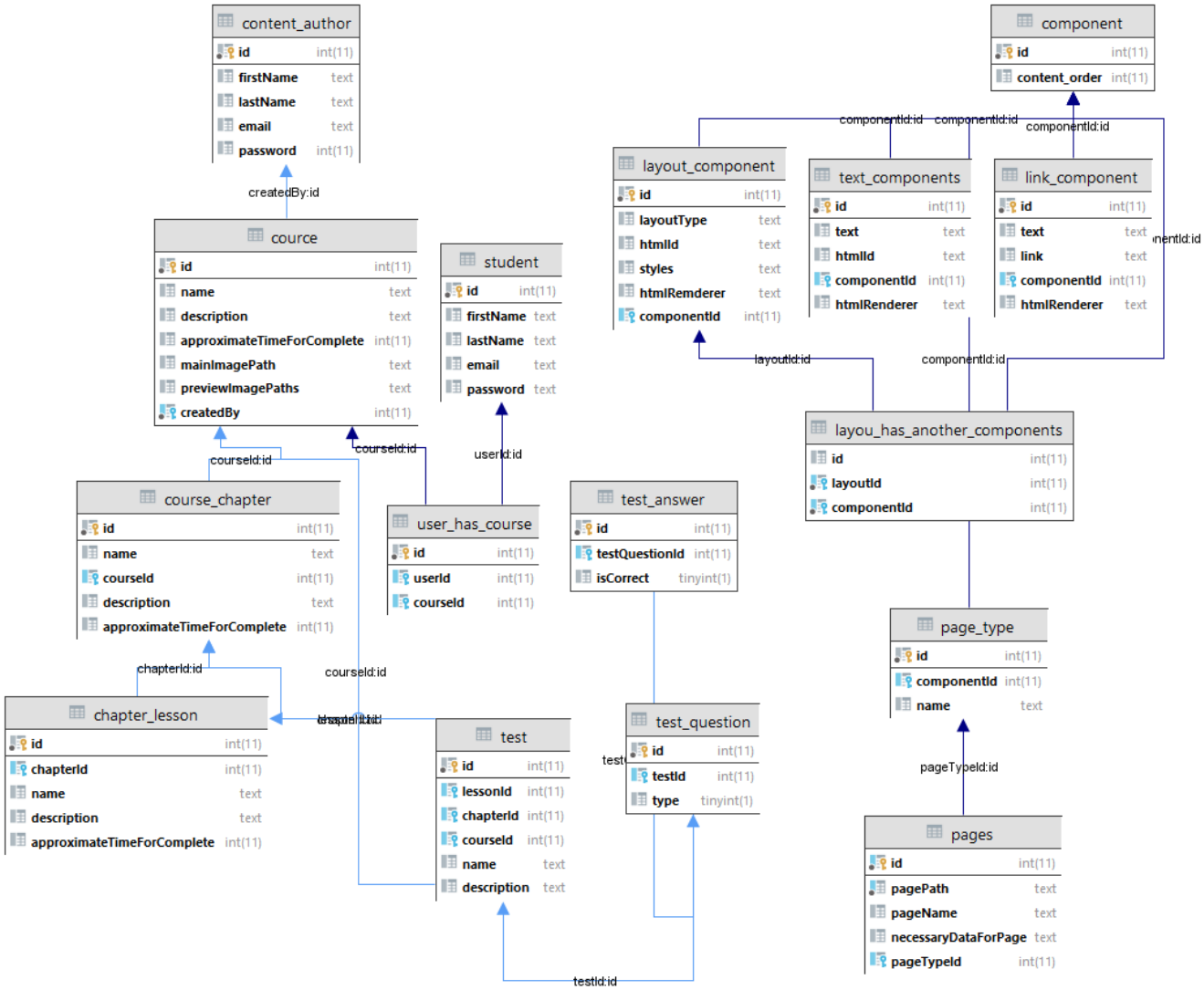


Рисунок 2.13 – Оновлена реляційна схема з таблицями під кожнний тип компонету

## 2.4 Створення моделі для JCR

### 2.4.1 Опис JCR

JCR маловідома специфікація, котра являє собою ієрархічний репозиторій. Абревіатура JCR розшиврофується як – Java Content Repository [6]. Дана специфікація буде розглядатися як ієрархічна система зберігання даних. JCR

відрізняється від RDBMS, томущо він є по-першре ієрархічним, по-друге – репозиторієм. Можна виділити наступні факти про JCR як систему зберігання даних:

- тільки специфікація;
- ієрархічна система хберігання даних, яка дозволяє зберігати контент в структурі, яка співпадає з потребами предметної області, де інформація яка відноситься одна до іншої, наприклад як зв'язок один до багатьох в реляційних базах даних, часто зберігажться в одному місці та може бути легко пронавігована;
- гнучкість, дозволяє контенту бути легко адаптованим за рахунок того що контент зберігаєтсья у вузлах(node) ієрархічної структури;
- повністю «schemaless», немає ніяких обмежень схеми даних [7];
- може бути використана тільки мовою Java;
- може зберігати контент будь де, будь то файл, архів, тощо, але специфікація це ієрархічний репозиторій, який надає контент у вигляді дерева;
- підтримка SQL запитів на мові SQL2 [8];
- підтримка full-text пошуку;

JCR зберігає дані без будь якої моделі даних. Є внутрішні тільки певні обмеження з точки зору створення вузлів. Кожен вузол повинен мати атрибут jcr:primaryType – який визначає тип вузла, значення цього атрибута можуть бути різними, але зазвичай це nt:unstructured, без будь яких обмежень, тому будемо зберігати контент саме у таких вузлах.

### 2.4.2 Розробка структури даних

У випадку із JCR зберігати сторінки дуже легко тому-що це ієрархічна ситстема, у створеній моделі даних чітко простежується те що курс має розділи, розділ має тести, тест має питання, а питання має відповіді, це все можна збегірати як дерево, ієрархію, у такому випадку нам не потрібно зберігати, відношення розділу до курсу за допомогою ідентифікатора, тому що ми будемо

мати ієрархію. Ієрархія буде відображати побудовану модель даних, з усіма необхідними атрибутами.

В створеній ієрархії моделі ми можемо також зберігати вузли з контент компоненнтами, які слід відображати на сторінці. Кожен такий компонент може мати свої атрибути, наприклад текст компонент – має текст контент, компонент з картинкою має налаштування картинки такі як ширина, висота, alt, шлях до зображення. На Рис 2.14 можна побачити структуру описаної моделі за допомогою ієрархічного представлення даних.

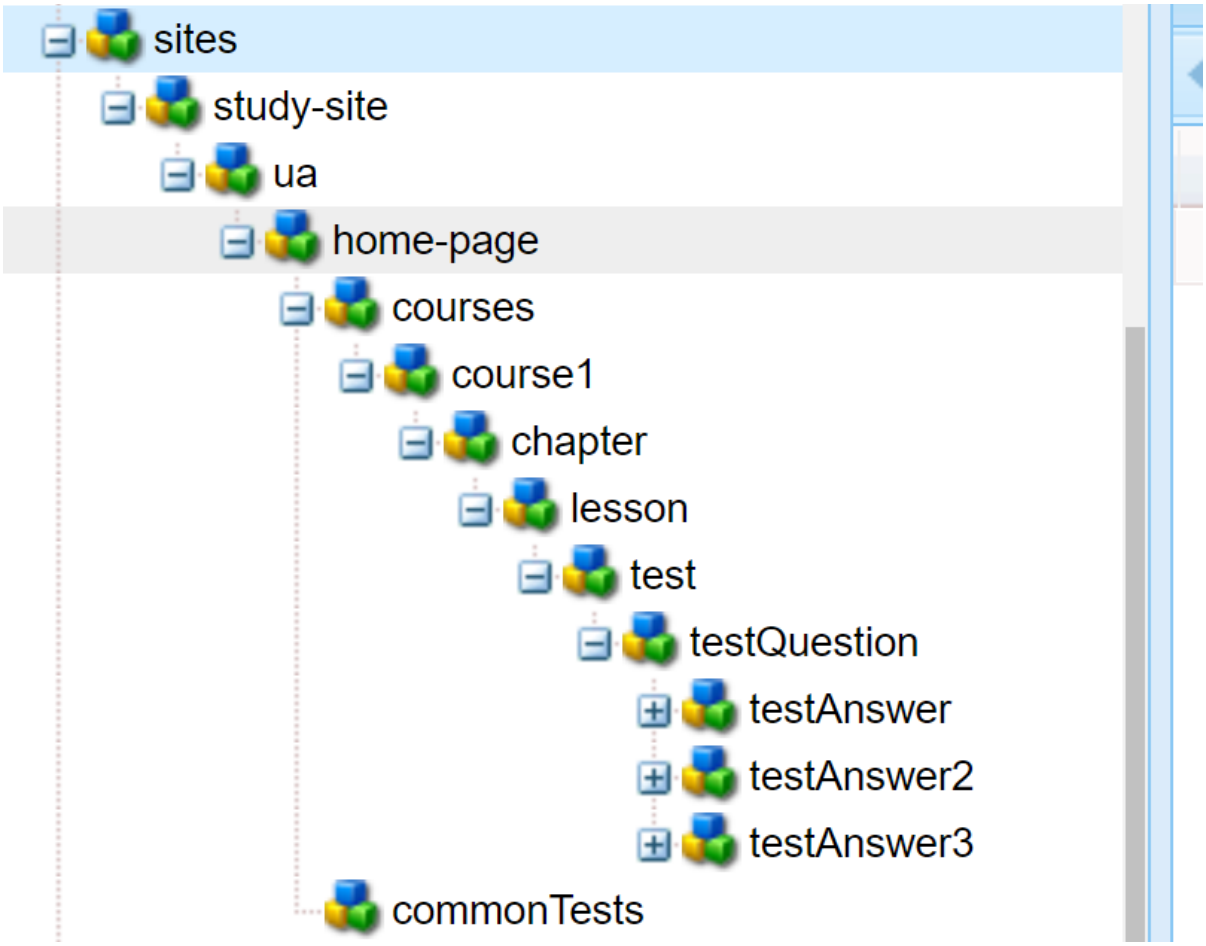


Рисунок 2.14 – Схема даних для ієрархічної структури

Дана структура відображає не тільки те, як контент зберігається, а те як можна буде отримати доступ до цбого контенту. Тобто, якщо називати вузли, які будуть зрозумілі у контекісті URL то, створити механіхзм, який буде віпдовідати

за маршрутизацію програми буде значно легше з точку зору розробника. А це зменшить кількість часу у декілька разів, при розробці проекту. В ієрархічній структурі такі назви не потрібно виносити в атрибути, бо шлях до вузла є унікальним, та можна використовувати ім’я вузла, як якийсь ідентифікатор жоступу до сторінки, тощо. Наприклад сайт може бути такої структури:

- наталог з курсами – cources;
- назва курсу яка зрозуміла англійською мовою;
- назва розділу, яка зрозуміла англійською мовою;
- назва заняття, яка зрозуміла англійською мовою;
- назва тесту, тестового питання, тестової відповіді не має значення, бо такі поняття можна показати нумерацією.

Оновлену структуру можна побачити на Рис 2.15.

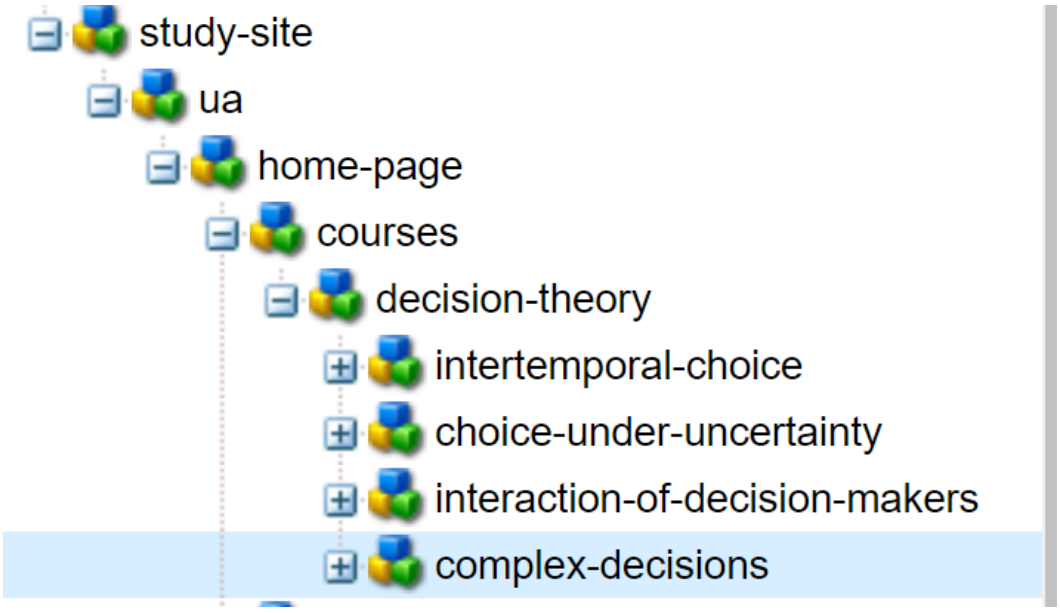


Рисунок 2.15 – Схема даних із зрозумілою назвою вузлів

Для того щоб відображати контент сторінок, нам потрібно знати як відображати контент сторінок. Для цього нам потрібен маркер, який визначає, чи можемо ми відобразати ресурс який запитуємо. Та якщо можемо, то як цей ресурс повинен відображатись [9].

Для рішення цієї проблеми у даній системі зберігання даних, мим можемо додати атрибут htmlRenderer, це атрибут, який буде зберігати шляї до html фрагменту, которий відповідає за відображення того чи іншого копомоненту, компонентом може бути сторінка, текст, хедер, футер, тощо.

Наприклад ми можемо відмальовувати головну сторінку сторінку home-page, тоді у вузлі home-page ми повинні мати атрибут htmlRenderer, із значенням наприклад, /render-scripts/pages/home-page, для курсу це /render-scripts/pages/course-page, для тесту – /render-scripts/pages/test-page, /render-scripts/pages/test-question. Для тестової відповіді немає сенсу малювати сторінку, бо відповіді будуть відображатися на сторінці із тестовим питанням [10].

Коли ми малюємо сторінку, то пи просто передаємо наш вузол на бекенд, бекенд в свою чергу перевірить наявність артибуту що відповідає за відображення, якщо цей атрибут присутній, то бекенд, спробує дістати із репозиторію рендерер веб сторінки, та педеасть до нього запитуємий вузол.

Слід зауважити, що звичайний html не може бути рендером, це повинна бути якась обгортка над html сторінкою, який шаблонний html додаток, до таких можна віднести Sightly, HTL, themleaf, JSP. Алоритм обробки запиту на сторінку можна побачити на Рис 2.16 [11].

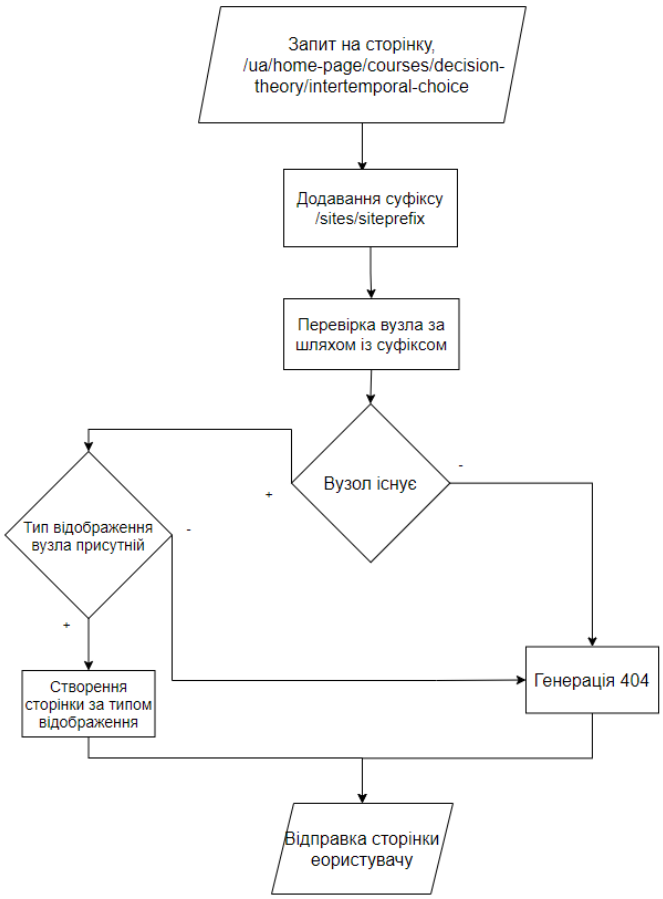


Рисунок 2.16 – Алгоритм обробки запиту на сторінку

## 2.5 Створення моделі для noSql

В якості noSql системи зберігання даних будемо розглядати mongoDB, по-перше – це класичний приклад noSql системи [12]. По-друге це найпопулярніша noSql система. MongoDB система управління базами даних яка маніпулює так званими документами. Структура даних – schemaless, це означає що не потрібно описувати схему даних. Дані будуть зберігатися в документах в JSON форматі. На Рис. 2.17 зображено приклад схеми зберігання даних.

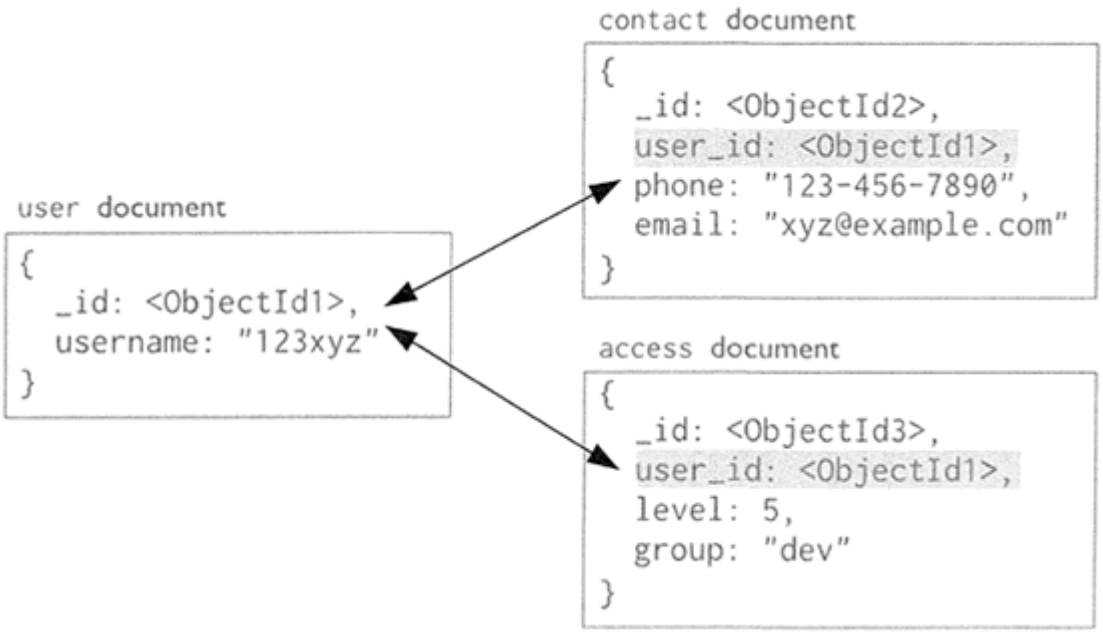


Рисунок 2.17 – Алгоритм обробки запиту на сторінку

Так як дана система зберігання даних дозволяє зберігати контент у будь якій формі, та краще за все підібрати ту структуру, яка по-перше буде працювати швидше, по друге, буде більш швидкою в плані її розробки та підтримки. Вже було розглянуто створення зберігання моделі даних в рамках ріляційної СУБД, та JCR, результати вийши максимально різними. У випадку із JCR ми зберігаємо контент ієрархічно. Якщо розглядати реляційну модель, то це таблиці які з'язані між собою за допомогою зовнішніх ключів. Слід розглянути обидва підходи у використанні mongoDB.

## 2.6 Механізм кешування

У якості WEB-серверу будемо використовувати Apache httpd сервер [13]. Apache дозволяє переписувати посилання на рівні серверу, додавати заголовки запиту та відповіді, робити перенапрвлення сторінок.

Для механізму кешування та балансування навантаження будемо використовувати Dispatcher модуль для Apache від Adobe [14]. Поперше функція балансувальника навантаження дозволяє передавати файли з декількох джерел

до кінцевого коричтувача [15]. Подруге модуль дозволяє кешувати контент. Приклад налаштування модуля для кешування контенту наведено на Рис 2.18

```
/cache
{
      /invalidate
      {
            /0000   { /glob "*" /type "deny" }
            /0001   { /glob "*.html" /type "allow" }
      }
}
```

Рисунок 2.18 – Схема даних із зрозумілою назвою вузлів

Кеш буде зберігатися на файловій системі операційної системи. на Рис 2.19 зображена дана структура даних. Доступ із Apache серверу до файлової системи операційної системи – є найшвидшим серед модливих, чи то з’єднання з базою даних, чи щось інше [16].

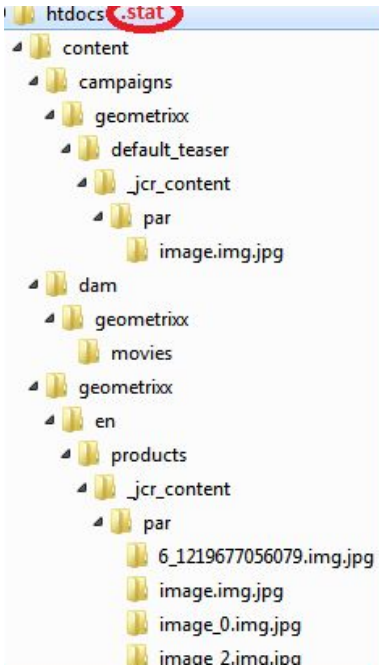


Рисунок 2.19 – Структура зберігання кешу

Це рішення для кешування є найкращим та найшвидшим, альтернативою може виступати nginX веб сервер. Дослідження в області, який веб сервер краще кешує неодноразово проводились різними коомпаніями та спеціалістами, і в більшості випадків серевер від Apache показував себе краще [17].

# 3 РОЗРОБКА УТИЛІТ ЯКІ ПРОВОДЯТЬ ТЕСТУВАННЯ СЦЕНАРІЇВ РОБОТИ СИСТЕМИ

Для оцінки роботи системи зберігання даних потрібно розглянути, систему в ріних місцях, але головним місцем є – швидкість доставки контенту від веб серверу до кінцевого користувача. Для оцінювання будемо розглядати декомпозовану систему, та систему взагалі. Різні функціональні місця системи розглядуються різними утилітами, також реалізація певних утиліт, для різних систем зберігання даних може бути виконана по різному, наприклад JCR – це JCR API, MySQL – SQL, mongoDB – collection api.

## 3.1 Завантаження сторінки з динамічним контентом

Даний тест має наступний сценарій – користувач іде на сторінку, сторінка відображає дані які потрібні користувачу, і які залежать від даного користувача, це значить, що для одно і того ж посилання контент сторінки двох різних користувачів можуть відрізнятися, наприклад через відсутність прав читати той чи інший контент, наприклад, певний курс має обмеження, по користувачам, або по групі користувачів, хто може його читати. Відповідно, якщо ми будемо запитувати, сторінку з усіма курсами, то користувачу А, відобразяться один перелік курсів, а користувачу Б – інший. Будемо розглядати сторінку з курсами так як вона динамічна, і залежить від поточного користувача. Також можна протестувати сторінку із сторінки з тестами, які не відносяться до жодного курсу, заняття, розділу, вони також можуть права доступу.

Тестування цього спенарію, не потребує тестування роботи із базою даних напряму, запит на сторінку на серверній частині буде працювати із базою даних.

Для того щоб протестувати цей сценарій ми можемо використовувати Chrome Light House [18]. Ця утиліта аналізує багато критеріїв, таких як SEO,

аналітику сторінки, якість коду сторінки, швидкість завантаження сторінки, та інше. Вигляд даного застосування можна побачити на Рис 3.1.

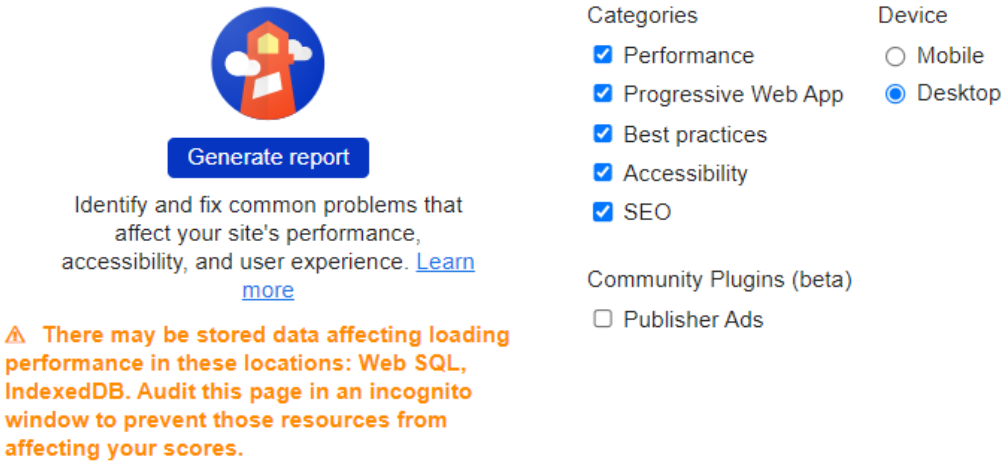


Рисунок 3.1 – Chrome lighthouse утиліта

## 3.2 Завантаження сторінки без динамічного контенту з кешем

Даний сценарій також може бути протестований за допомогою Chrome lighthouse. Але сторінка повинна бути без динамічного контенту, до таких сторінок відносяться сторінка з певним курсом, розділом, зняттям, тестом.

Слід зауважити, зо перед тестуванням цього сценарію необзідно прогріти кеш. Це значить що необхідно спочатку зробити запит на цю сторінку, щоб сторінка потрапила в кеш, а потім за допомогою утиліти її тестувати.

## 3.3 Завантаження сторінки без динамічного контенту без кешу

Цей сценарій такий як вищеописаний за однієї умови, тестування повинно відбуватись, ніби мито кожен раз ідемо за сторінкою, на коттру ні один користувач не потрапляв до цього. Тобто після кожного запиту нам потрібно оновлювати кеш статус, і знову робити запити. Утіліта chrome lighthouse не

підходить для даного сценарію, бо вона є браузерною, і не може оновити серверний кеш.

Для даного тестування необхідно написати власний скрипт файл, котрий буде робити необхідні операції. Алгоритм цього сценарію можна побачити на Рис 3.2.

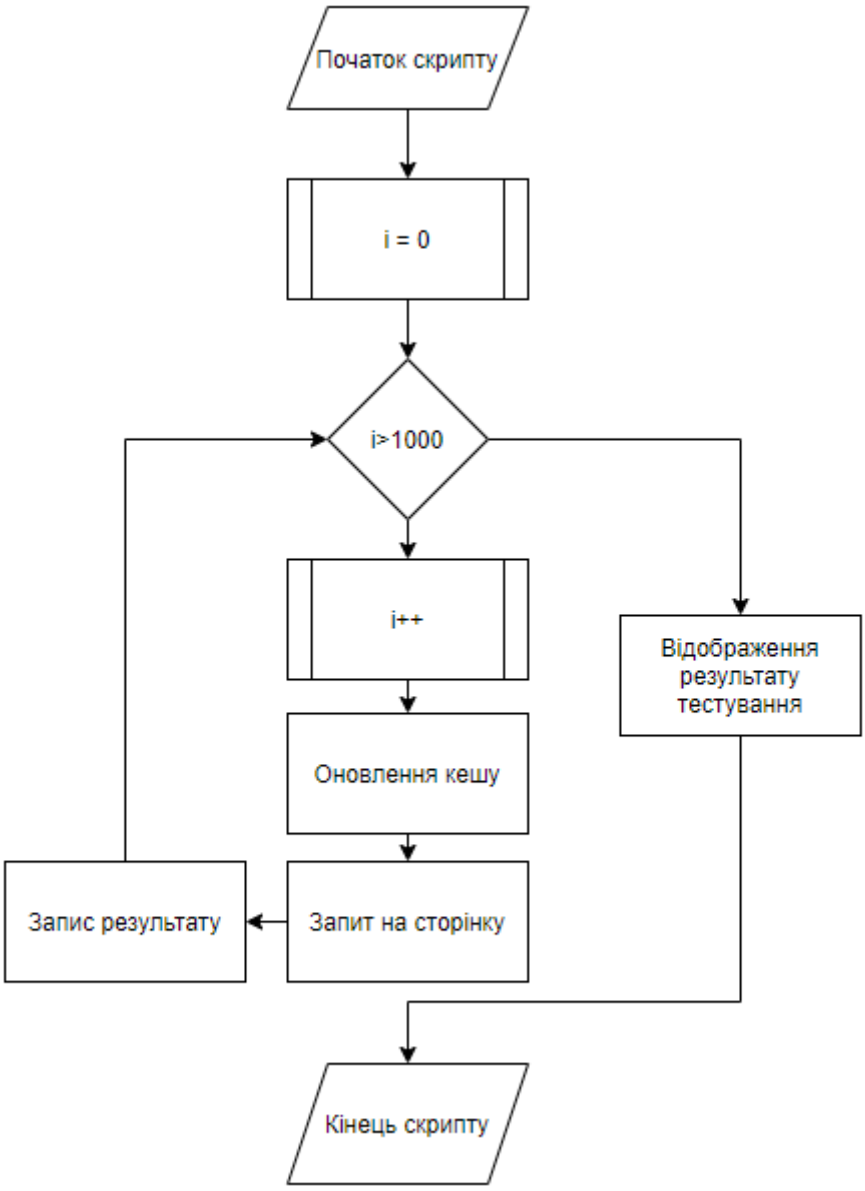


Рисунок 3.2 – Алгоритм тестування завантаження сторінки без динамічного контенту без кешу

Реалізація алгоритму зроблена на скриптовій мові програмування groovy [19][20]. Код алгоритму можна побачити на Рис 3.3.

```
def APACHE_CACHE_FLUSH_URLS = ['http://localhost:8080/flush',
'http://localhost:8081/flush',
'http://localhost:8082/flush',
'http://localhost:8083/flush']
def PAGE_URLS = ['http://localhost:4502/ua/home-page/courses/course1.html': [],
    'http://localhost:4503/ua/home-page/courses/course1.html': [],
    'http://localhost:4504/ua/home-page/courses.html?courseId=course1': [],
    'http://localhost:4505/ua/home-page/courses.html?courseId=course1': []   ]
def interationCount = 1000;
while(interationCount) {
    iterationCount--;
    APACHE_CACHE_FLUSH_URLS.each(url -> {fluchCache(url)})
    PAGE_URLS.each(map -> {makeRequestToPage(map.key(), map.value())})
}
def makeRequestToPage(pageUrl, results) {
    def response = http.get(pageUrl, {
        'login':'testStudent:QWERTY'
    });
    results << response.ttlTime()
}
def fluchCache(url) {
    http.post(url, {
        'login':'apache:apache'
    })
}
PAGE_URLS.each { key, value ->
    println key + ' ' + value/1000
}
```

Рисунок 3.3 – Реалізація алгоритму завантаження сторінки без динамічного контенту без кешу

## 3.4 Отримання контену бази даних із бекенду

Сценарій цього тесту наступний, нам потрібно отримати усі дані на серверній частині, але не починати рендерінг сторінки.

Для MySQL бази даних необхідно зробити SQL запити із серверної частини сайту. Для JCR – потрібно використати, JCR – API. JCR сервер, для MongoDB так само як і для SQL, бо в них однаковий алгоритм роботи.

## 3.5 Швидкість реплікації author – delivery

JCR підтримує реплікацію контенту за специфікацією в author-delivery архітектурі. Якщо говорити про mongoDB та mySQL, то тут поняття реплікації

відрізняється. Є так звані master та slaves, контент між ними однаковий, master відповідає за те щоб контент був однаковий на усих серверах, включаючи себе. Тому була створена додатвока логіка, яка займається реплікацією контенту з master, до master-delivery, який в свою чергу контролює контент на усіх slave серверах.

Такий сценарій можна описати наступним чином – author сервер публікує статтю, яка ще не існує на delivery серверах, і в цей же час, ми починаємо робити запити, на статтю, яка була опублікована. В той час коли нам прийшла стаття на відповіть ми зупиняємо лічильник, який очікув на публікацію статті.

## 3.6 Швидкість зворотньої реплікації

Так як у нас декілька делівері серверів і є балансувальник навантаження, то може статися така ситуація коли UGC збережеться на одному із серверів, потім, користувач закриж вікно, знові зробить запит на сайт, потрапить на інший дулівері сервер, а контент який він наприклад зберіг у своєму профілі, зберігся у попередньому делівері сервері. У такому випадку контент між делівері серверами повинен бути синхронізованим.

Для зворотньої реплікації буде використовуватись наступний сценарій – учень додає курс до свого профілю, і в цей час опитуються система збереження данних на наявність цього зв’язку – користувач і курс. Коли усі сервери будуть мати цей зв’язок користувач і курс, тоді можна зупиняти лічильник, який очікую успішну зворотню реплікацію.

Зворотня реплікація підтримується у архатектурі delivery-master – delivery-slaves автоматично для mongoDB, mySql. Для JCR це було імплементовано власноруч.

# 4 АНАЛІЗ ОТРИМАНИХ ДАНИХ

Розроблені сценарії та алгоритми їх виконання надали результати, для кожної СУБД. Для кожної СУБД окремий сценарій повторювався 1000 ітерацій, для того щоб скорегувати погрішності які могла викликати нестабільність, навантаженість операційної системи.

## 4.1 Аналіз результатів запитів сторінки з динамічним контентом без кешу

В таблиці 4.1 наведено результати тесту сценарію.

Таблиця 4.1 – результати запитів сторінки з динамічним контентом без кешу

| СУБД | Швидкість операції для 1000 ітерацій в мілісекундах |
|---|---|
| JCR | 2964 |
| MongoDB - ієрархічна структура | 3442 |
| MongoDB - реляційна структура | 5678 |
| Mysql | 4877 |

Ми бачимо що JCR, та MongoDB з ієрархічною структурою мають набагато кращі результати, ніж реалізації з ієрархічною структурою. Це пояснюється тим що ми уникаємо запитів до бази даних, які шукають записи у таблицях чи документах, за зовнішнім ключем.

Реляційна структура Mysql швидша за реляційну структуру MongoDB, це поясвнюється тим, що Mysql це реляційно-орієнтована субд на відміну від MongoDB.

Ієрархічна структура JCR працює швидше ха ірєрархічну структуру MongoDB. Коли MongoDB дістає з ієрархії даніпевного вузла, він також дістає

дані усіх дитячих вузлів на відміну від JCR.

## 4.2 Аналіз результатів запитів сторінки без динамічного контенту з кешем

В таблиці 4.2 наведено результати тесту сценарію.

Таблиця 4.2 – результати запитів сторінки без динамічним контентом з кешем

| СУБД | Швидкість операції для 1000 ітерацій в мілісекундах |
|---|---|
| JCR | 291 |
| MongoDB - ієрархічна структура | 289 |
| MongoDB - реляційна структура | 291 |
| Mysql | 285 |

Як можна спостерігати, усі СУБД мають однакову швидкість серверної відповіді на запит сторінки. Це відбувається, томущо сервер ніяк не задіяний до опрацювання, запиту контенту, сторінка є в кеші, і браухеру її віддає веб сервер.

## 4.3 Аналіз результатів запитів сторінки без динамічного контенту без кешу

В таблиці 4.3 наведено результати тесту сценарію.

Таблиця 4.3 – результати запитів сторінки без динамічним контентом без кешу

| СУБД | Швидкість операції для 1000 ітерацій в мілісекундах |
|---|---|
| JCR | 2878 |
| MongoDB - ієрархічна структура | 3359 |
| MongoDB - реляційна структура | 5583 |
| Mysql | 4820 |

Тут ми маємо майже той самий результат що і в першому тесті, за виключенням того, що загальний час зменшився приблизно на ~100мс. Це скоріш за все час на опрацювання динаміки сторінки.

## 4.4 Аналіз результатів отримання контену бази даних із бекенду

В таблиці 4.4 наведено результати тесту сценарію.

Таблиця 4.4 – результати отримання контенту бази даних без рендерингу

| СУБД | Швидкість операції для 1000 ітерацій в мілісекундах |
|---|---|
| JCR | 2269 |
| MongoDB - ієрархічна структура | 2828 |
| MongoDB - реляційна структура | 5101 |
| Mysql | 4298 |

Результати ідентичні до 1 та 3 сценарію, але від першого відрізняються приблизно на ~500мс, а від третього приблизно на ~400мс. Ця різниця, це ті операції, які не виконалися в сценаріях 1 та 3 сценарію, якщо ми будемо порівнювати даний тест з цими сценаріями. Операції які ми не виконали – рендер контенту та трансфер контенту. Можна зробити висновок що рендер контенту, та його тгрансфер до тонкого клієнту займає приблизно ~ 400 мс. Цей тест дав змогу побачити як виконується та чи інша операція без її тестсування

## 4.5 Аналіз результатів швидкості реплікації контенту

В таблиці 4.5 наведено результати тесту сценарію.

Таблиця 4.5 – швидкості реплікації контенту

| СУБД | Швидкість операції для 1000 ітерацій в мілісекундах |
|---|---|
| JCR | 720 |
| MongoDB - ієрархічна структура | 4980 |
| MongoDB - реляційна структура | 4894 |
| Mysql | 5601 |

За результатами тесту ми бачимо, що реплікація контенту проходить найшвидше для JCR. Це пояснюється тим, що ця СУБД підримує цю операцію на рівні ядра системи зберігання даних. Розробники цієї системи оптимізували її, протестували, тому вона працює набагато швидше ніж самостійно створена реплікація яка обробляється на сторонньому сервері.

## 4.6 Аналіз резільтатів швидкості зворотньої реплікації контенту

В таблиці 4.6 наведено результати тесту сценарію.

Таблиця 4.6 – результати швидкості зворотньої реплікації контенту

| СУБД | Швидкість операції для 1000 ітерацій в мілісекундах |
|---|---|
| JCR | 9756 |
| MongoDB - ієрархічна структура | 3652 |
| MongoDB - реляційна структура | 3709 |
| Mysql | 4973 |

Даний тест сценарію, відображає зеркальну ситуацію з попереднього тесту. Та зворотня реплікація яка працює на рівні ядра СУБД набагато швидша за самостійно створену реплікаю, але тут MongoDB виграє в MySQL, біль ніж секунду, що є досить важливим показником.

## 4.7 Загальний аналіз даних

Для початку необхідно згрупувати усі тести у відповідності до СУБД. Тобто зібрати таблицію з усіма тестами, та СУБД, але без значень, просто будемо додавати помарку, на найшвидвій СУБД у певному тесті. У Табл. 4.7 можна побачити ці результати.

Таблиця 4.7 – порівняння усих СУБД за тестами

| | JCR | MySql | MongoDB-JCR | MongoDB-MySql |
|---|---|---|---|---|
| Тест 1 | • | | | |
| Тест 2 | • | • | • | • |
| Тест 3 | • | | | |
| Тест 4 | • | | | |
| Тест 5 | • | | | |
| Тест 6 | | | • | • |

За данними результатми можна побачити, що JCR випереджає інші СУБД майже в усіх критеріях крім зворотнбої реплікації. Це означає що цю систему слід використовувати для CMS.

Проблема із зворотньої реплікацією досить важлива, томущо часто в CMS користувачі якимось чином зберігають свої дані на сервері і їх потрібно синхронізувати між усіма серверами делівері. Цю проблему можна вирішити за допомогою MongoDB. JCR дозволяє зберігати вузол в MongoDB. Потрібно створити відповідні колекції у MongoDB, наприклад профіль користувача, та налаштувати JCR так, щоб користувачі зберігалися у MongoDB колекціїї. При чому кожен делівері сервер буде мати зв’язок із виділенним слейвом для чього делівері сервара. А всі слейви будуть зв’язані за допомогою мастер серверу. Зворотня реплікація на автор сервер нам не потрібна. Так як на ньому працюють лише контент автори, і данич про студентів або учнів там не існує, тому і

реплікувати на цей сервер нічого не поторібно він і є реплікатором. Дану схему можна побачити на Рис. 4.1.

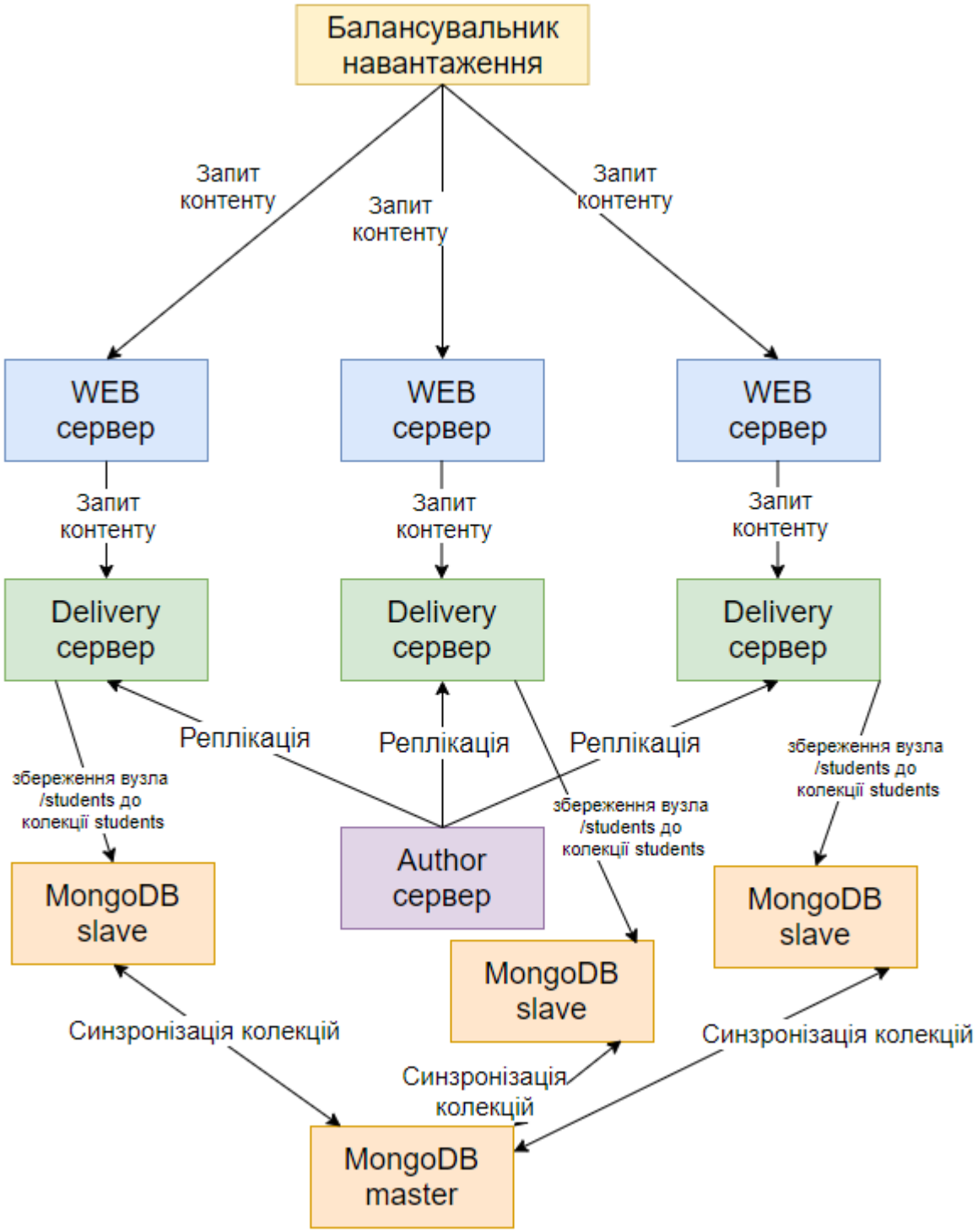


Рисунок 4.1 – Оптимальна архітектура CMS системи

Дана архітектура є оптимальною для CMS систем.

# ВИСНОВКИ

Під час виконання кваліфікаційної роботи виконано аналіз існуючих типів систем зберігання даних, класифіковано системи управління контентом. Визначено проблему у доцільності використяння тієї чи іншою системи збереження даних у системах управління контентом. Запропановано як оптимальну архітектуру усієї системи управління контентом, так і певну СУБД яку слід використовувати для CMS.

Проаналізовано, як система буде зберігати модель даних у різних СУБД. Розроблена модель даних певної предметної області. Розроблена та створена фізична модель даних для різних типів систем зберігання даних.

Розроблена бізнес логіка для кожної системи зберізання даних у контексті CMS, опрацювання запитів, маршрутизація запитів. Створена архітектура для високонавантаженої CMS із балансувальником навантаження, механізмом кешування, механізмом реплікації та зворотньої реплікації, хостингом декількох серверів для розподілення навантаження.

Створено тестові сценарії для різних типів навантаженя клієнтів системи. Розролені тести тестують систему в цілому, тестують окремі модулі системи. Даний підхід дозволив визначти сильні або слабкі сторони тієї чи іншої СУБД в різних ситуаціях. Використовуючи даний підхід, можна створити систему, із відповідною СУБД в залежності від найчастіших сценаріїв, які будуть відбуватися в системі. Комбінація різних створенії тестів дозволяє визначити як працює система в тому чи іншому сценарії, при чому тесту на цей сценарій немає.

Розроблено аналізуючі утиліти. Кожна утиліта відповідає до тестового сценарію. Утиліти це набір скрипт файлів, аоторі виконують певні операції, за пеаними алгоритмами та рахують час виконання операції. Час виконнаня операції визначає швидкість тієї чи іншої СУБД.

Кожна розроблена утиліта використана для аналізу даних різних СУБД, для отримання результатів ефективності системи у різних сценаріях роботи системи. Отримані результати швидкодійності системи у різних сценаріях із різними СУБД дозволила визначити найефективнішу СУБД для CMS. Також результати аналізу показали що, один сценарій, досить швидко виконується в одній системі зберігання даних, але цей сценарій єдиний який виконується повільно у найефективнішій СУБД. Було створено гібридну архітектуру, із гібридним зберіганням даних – це дозвозлять роботи гібриди які присутні в цій архітектурі.

Результати досліджень становлять практичний інтерес при розробці високонавантажуваної CMS системи. Результати дослідження можуть використовуватись під час вибору системи зберігання даних. Якщо жоден з розроблених сценаріїв не буде відповідати створювані системі, то опирвючісь на існуючі сценарії можна створити нові, які не менж ефективно проаналізують систему на стадії проектвання.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. «Що таке CMS? Огляд кращих CMS | Блог Хостинг Україна,» [Онлайновий]. Available: https://www.ukraine.com.ua/uk/blog/site-administration/chto-takoe-cms-obzor-luchshih-cms/. [Дата звернення: 14 4 2021].

2. С. Т. М.А. Курилов, «Классификация систем управления содержимым web-ресурсов и их использование,» 21 06 2010. [Онлайновий]. Available: Классификация систем управления. [Дата звернення: 14 4 2021].

3. «Бібліотека МДПУ ім.Б.Хмельницького,» [Онлайновий]. Available: http://lib.mdpu.org.ua/e-book/vstup/L5.htm. [Дата звернення: 14 4 2021].

4. «AEM Foundation Videos and Tutorials,» [Онлайновий]. Available: https://experienceleague.adobe.com/docs/experience-manager-learn/foundation/overview.html?lang=en. [Дата звернення: 15 4 2021].

5. Слива Є. К. Збереження описанної моделі даних у ієрархічнй та реляційній СУБД для CMS// “Modernization of today's science: expirience and trends”. Сінгапур (SGP), Singapore, 2021.

6. «Welcome to Apache Jackrabbit,» aPACHE, [Онлайновий]. Available: http://jackrabbit.apache.org/jcr/index.html. [Дата звернення: 15 14 2021].

7. J. H. Thomsen, 12 1 2016. [Онлайновий]. Available: https://eng.uber.com/schemaless-part-one/. [Дата звернення: 15 4 2021].

8. «Oak SQL-2 Query Grammar,» Apache, 30 03 2021. [Онлайновий]. Available: https://jackrabbit.apache.org/oak/docs/query/grammar-sql2.html. [Дата звернення: 15 4 2021].

9. «Getting Started with HTL,» Adobe, [Онлайновий]. Available: https://experienceleague.adobe.com/docs/experience-manager-htl/using/getting-started/getting-started.html?lang=en#getting-started. [Дата звернення: 15 4 2021].

10. «Thymeleaf,» [Онлайновий]. Available: https://www.thymeleaf.org/. [Дата звернення: 16 4 2021].

11. «Java Server Pages,» [Онлайновий]. Available: https://metanit.com/java/javaee/3.1.php. [Дата звернення: 16 4 2021].

12. «The database for modern applications,» [Онлайновий]. Available: https://www.mongodb.com/. [Дата звернення: 16 4 2021].

13. «What is the Apache HTTP Server Project?,» Apache, [Онлайновий]. Available: https://httpd.apache.org/ABOUT_APACHE.html. [Дата звернення: 16 4 2021].

14. И. А. Урняева, И. В. Гребенник, Д. В. Иванов та В. Г. Иванов, «Математическая модель задачи планирования передачи файла от нескольких источников потребителю,» *Системы обработки нформации,* pp. 82-85, 2015.

15. «What Is Load Balancing?,» NGINX, [Онлайновий]. Available: https://www.nginx.com/resources/glossary/load-balancing/. [Дата звернення: 16 4 2021].

16. «Adobe AEM Dispatcher Caching Basics,» [Онлайновий]. Available: http://www.adobe.com.by/pdf/meetup-4/Dispacther_Tips_and_Tricks.pdf. [Дата звернення: 17 4 2021].

17. «Nginx vs Apache: Web Server Showdown,» [Онлайновий]. Available: https://kinsta.com/blog/nginx-vs-apache/. [Дата звернення: 17 4 2021].

18. «Lighthouse,» Google, [Онлайновий]. Available: https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=ru. [Дата звернення: 17 4 2021].

19. «A multi-faceted language for the Java platform,» [Онлайновий]. Available: https://groovy-lang.org/. [Дата звернення: 18 4 2021].

20. Minukhin, S. V., Losev, M. U., & Sitnikov, D. E. (2019). «ANALYSIS OF WAYS FOR EXCHANGING DATA IN NETWORKS WITH PACKAGE COMMUTATION». Radio Electronics, Computer Science, Control, (4). https://doi.org/10.15588/1607-3274-2018-4-19