

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "магістр"

на тему:

Розробка системи управління контентом web-порталу

закладу післядипломної освіти

Виконав: студент групи 6141 мз


(підпис) **Комарова А.П.**

Керівник роботи:

доцент кафедри ІУСТ, к.т.н.,
(посада, науковий ступінь вчене звання)


(підпис) **Кроптя А.В.**

м. Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра інформаційних управляючих систем та технологій
Спеціальність 122 "Комп'ютерні науки"
Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

В. Ажищев

к.т.н, доц. Ажищев В.Ф.

"___" _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "магістр"

Студенту Комаровій Анні Петрівні
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка системи управління контентом web-порталу закладу післядипломної освіти

Керівник роботи доцент кафедри ІУСТ, к.т.н. Кроптя А.В.

Затверджені наказом ректора № 1355-уч. від "14" листопада 2025 року.

2. Термін подання роботи: 15 грудня 2025 р.

3. Вихідні дані до проєкту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів) _____

Аналіз стану інтернет-технологій для створення систем управління контентом, існуючих рішень та засобів розробки, Розробка проєктних рішень по системі, Розробка робочої документації системи управління контентом web-порталу МОІППО, Розрахунок економічного ефекту від впровадження ІС, Охорона праці, Охорона навколишнього середовища

5. Перелік презентаційних матеріалів: Content Management System, Структура МОІППО, Існуючі рішення - CMS, Типова архітектура CMS, Порівняння існуючих CMS, Рішення з засобів розробки, Схема функціональної структури, Діаграми варіантів використання, Логічна модель даних, Діаграма діяльності, Діаграма класів, Приклади екранів web-порталу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Кроптя А.В.	05.09.2024	22.12.2024
2	Кроптя А.В.	23.12.2024	15.06.2025
3	Кроптя А.В.	15.06.2025	25.09.2025
4	Кроптя А.В.	25.09.2025	14.11.2025
5	Кроптя А.В.	25.09.2025	14.11.2025
6	Кроптя А.В.	25.09.2025	14.11.2025

7. Дата видачі завдання 05 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Дослідження та аналіз предметної галузі	22.12.2024	
2	Розробка концепції	15.03.2025	
3	Розділ з розробки проектних рішень по системі (та її частинах)	15.06.2025	
4	Реалізація проекту та розробка робочої документації	25.09.2025	
5	Обчислення економічної ефективності	14.11.2025	
6	Вирішення питань охорони праці та навколишнього середовища	14.11.2025	
7	Оформлення пояснювальної записки	14.11.2025	
8	Подання магістерської роботи на перевірку на плагіат	16.11.2025	
9	Подання магістерської роботи рецензенту	20.11.2025	
10	Подання магістерської роботи на державний захист	15.12.2025	

Студент


(підпис)

Комарова А.П.
(ПІБ)

Керівник роботи


(підпис)

Кроптя А.П.
(ПІБ)

Анотація

Представлена робота присвячена дослідженню інтернет-технологій та створенню системи управління контентом web-порталу Миколаївського обласного інституту післядипломної педагогічної освіти у складі системи керування МОІППО та призначена для автоматизації процесів викладення інформації на web-сторінку закладу освіти, уніфікації доступу до інформації, викладеної на web-порталі закладу освіти.

В роботі аналізуються існуючі рішення, досліджуються засоби створення таких систем, формуються вимоги до системи, розробляються концепції системи і проводиться вибір найкращої, розробляється технічне завдання. Далі проводиться розробка основних проектних рішень за технічним, інформаційним та програмним забезпеченням, формується робоча документація за програмним забезпеченням. Також робота містить розділи з економіки, охорони праці, охорони навколишнього середовища.

Дипломна робота виконана на 127 аркушах машинописного тексту, містить 17 рисунків, 7 таблиць, 4 додатки.

Ключові слова: web-портал, CMS, Java, Spring Boot, Hibernate

Abstract

The work is devoted to the development of Internet technologies for creating a content management system for the web-portal of the Mykolaiv Regional Institute of Postgraduate Pedagogical Education as part of the MRIPPE Management System and is intended for automation of the processes of information sharing on the web-site of the educational institution, unification of access to information provided on the institute's web-portal.

The paper analyzes the existing solutions, examines the instruments for creating such systems, creates requirements for the system, develops the concept of the system and chooses the best, developed technical task. Further, the development of basic design solutions for technical, information and software, and the working documentation for software is formed. Also, the work contains sections on economics, labor protection, and environmental protection.

The master's paper is stated on 127 sheets of the typewritten text, contains 17 figures, 7 tables, 4 applications.

Keywords: web portal, CMS, Java, Spring Boot, Hibernate

ЗМІСТ

ВСТУП.....	
РОЗДІЛ 1. АНАЛІЗ СТАНУ ІНТЕРНЕТ-ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМ УПРАВЛІННЯ КОНТЕНТОМ, ІСНУЮЧИХ РІШЕНЬ ТА ЗАСОБІВ РОЗРОБКИ.....	
1.1 Основні функції та застосування систем управління контентом.....	
1.2 Різновиди систем управління контентом.....	
1.3 Аналіз предметної області.....	
1.4 Аналіз існуючих рішень для систем управління контентом.....	
1.5 Вибір технологій та середовища розробки.....	
1.6 Засоби розробки інтерфейсу.....	
1.7 Засоби тестування продукту.....	
1.8 Вибір ORM.....	
1.9 Вибір бази даних.....	
1.10 Динамічне відображення даних.....	
1.11 Вибір засобів розробки серверної частини.....	
РОЗДІЛ 2. РОЗРОБКА ПРОЕКТНИХ РІШЕНЬ ПО СИСТЕМІ.....	
2.1 Загальносистемні проектні рішення.....	
2.2 Рішення по організаційному забезпеченню.....	
2.3 Рішення по інформаційному забезпеченню.....	
2.4 Рішення по технічному забезпеченню.....	
2.5 Опис алгоритмів функціонування системи.....	
2.6 Рішення по програмному забезпеченню.....	
РОЗДІЛ 3. РОЗРОБКА РОБОЧОЇ ДОКУМЕНТАЦІЇ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ WEB-ПОРТАЛУ МОППО.....	
РОЗДІЛ 4. РОЗРАХУНОК ЕКОНОМІЧНОГО ЕФЕКТУ ВІД ВПРОВАДЖЕННЯ ІС.....	
4.1 Вступ до проблеми.....	
4.2 Розрахунок вартості створення інформаційної системи управління контентом web-порталу.....	
4.3 Розрахунок економічної ефективності.....	

4.4 Підсумки.....	
РОЗДІЛ 5. ОХОРОНА ПРАЦІ.....	
5.1 Аналіз небезпечних і шкідливих факторів в університеті.....	
5.2 Розрахунок системи штучного освітлення приміщення.....	
5.3 Розробка заходів щодо зменшення впливу шкідливих факторів.....	
5.4 Розробка заходів щодо зменшення впливу шкідливих факторів.....	
РОЗДІЛ 6. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА.....	
6.1 Вступ до проблеми.....	
6.2 Забруднення навколишнього середовища підприємством з розробки програмного забезпечення.....	
6.3 Розробка заходів щодо зменшення забруднення.....	
ВИСНОВКИ.....	
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	
ДОДАТКИ	
Додаток А Технічне завдання на розробку.....	
Додаток Б Програма та методика випробувань.....	
Додаток В Тексти програмних модулів системи.....	

ВСТУП

Система управління контентом (англ. Content Management System, CMS) це – програмне забезпечення, призначене для створення, організації та підтримки веб-сайтів та інших інформаційних ресурсів в Інтернеті або локальних комп'ютерних мережах. На сьогодні існує широкий спектр систем управління контентом, серед яких представлені як комерційні, так і безкоштовні рішення, що реалізовані на основі різних технологічних платформ.

Функціональна сутність CMS полягає у відокремленні змістової частини веб-ресурсу від його дизайну. Користувачеві надається можливість обрати шаблон – попередньо підготовлену структуру сторінки з визначеним оформленням, яку необхідно лише наповнити відповідною інформацією. Більшість сучасних систем управління контентом використовують візуальні редактори типу WYSIWYG (What You See Is What You Get), що забезпечують інтуїтивно зрозумілий інтерфейс для додавання та редагування матеріалів без необхідності володіння спеціальними технічними навичками. Важливо зазначити, що веб-сайт у межах CMS не є статичним набором сторінок: його вміст формується динамічно. Додані матеріали зберігаються у базі даних (наприклад, MySQL) і використовуються для генерації сторінок у відповідь на запити користувачів.

Післядипломна освіта являє собою систему підвищення кваліфікації осіб, які вже здобули вищу освіту. Післядипломна педагогічна освіта забезпечує можливість професійного розвитку педагогів та інших працівників освітньої сфери. Для оперативного обміну інформацією, пов'язаною з навчальним процесом (новинами, лекційними матеріалами, статтями, виступами, вебінарами та їх записами), доцільним є використання веб-порталу як найбільш ефективного та зручного інструмента комунікації.

Актуальність дипломної роботи зумовлена відсутністю на даний час зручної, адаптованої до потреб інститутів післядипломної педагогічної освіти системи управління контентом. Наявні технологічні рішення або не враховують специфіку освітнього контенту (його змістові, структурні та формальні

особливості), або є надмірно складними для користувачів, які не мають спеціальної технічної підготовки.

Метою дипломної роботи є розроблення інформаційної системи управління контентом веб-порталу Миколаївського обласного інституту післядипломної педагогічної освіти.

Об'єктом дослідження виступають інтернет-технології, що застосовуються для створення веб-порталів. Предметом дослідження є інформаційна система управління контентом веб-порталу.

Для досягнення поставленої мети визначено такі завдання:

- здійснити огляд сучасних систем управління контентом;
- провести аналіз існуючих рішень у сфері контент-менеджменту;
- дослідити інтернет-технології, що використовуються для створення систем управління контентом;
- проаналізувати основні інструменти, застосовувані під час розроблення веб-порталів;
- розробити тестовий веб-портал із реалізованою можливістю використання системи управління контентом.

РОЗДІЛ 1. Аналіз стану інтернет-технологій для створення систем управління контентом, існуючих рішень та засобів розробки

1.1 Основні функції та застосування систем управління контентом

Система управління вмістом (англ. *Content Management System*, CMS) – це інформаційна система або програмний комплекс, призначений для організації спільного процесу створення, редагування, упорядкування та публікації вмісту (контенту). Такі системи забезпечують централізоване управління інформаційними ресурсами та підтримують повний цикл роботи з контентом у межах веб-сайтів або інших інформаційних платформ.

Основні функції CMS охоплюють надання інструментів для створення та редагування вмісту, організацію колективної роботи над матеріалами, управління контентом (зберігання, контроль версій, регулювання доступу, керування потоками документів), а також публікацію інформації у структурованому вигляді, зручному для навігації та пошуку.

У системі управління вмістом можуть зберігатися різноманітні типи даних: текстові документи, мультимедійні матеріали (відео, зображення), контактна інформація, наукові дані тощо. CMS широко застосовуються для зберігання, перегляду та публікації документації, а контроль версій є однією з ключових переваг таких систем, особливо у випадках, коли над контентом працює група користувачів.

Типовими сферами застосування CMS є:

- блоги та форуми (WordPress, phpBB, vBulletin);
- інтернет-магазини (Magento, OpenCart, osCommerce);
- соціальні мережі (InstantCMS, Social Engine);
- персональні сайти (WordPress, Monstra);
- корпоративні веб-ресурси (Joomla, Drupal);
- - інформаційні портали (DLE, Drupal).

Водночас більшість систем управління вмістом характеризуються високою гнучкістю налаштування та можуть бути адаптовані для створення веб-ресурсів різного призначення. Одним із найбільш універсальних і поширених рішень є WordPress, який дозволяє реалізувати широкий спектр проєктів – від персональних сайтів до масштабних порталів та інтернет-магазинів.

1.2 Різновиди систем управління контентом

У загальному випадку системи управління контентом поділяються на два основні типи:

1. системи управління контентом масштабу підприємства (Enterprise Content Management System, ECM);
2. системи управління веб-контентом (Web Content Management System, WCMS).

Управління корпоративним контентом (Enterprise Content Management, ECM) охоплює процеси зберігання, оброблення, організації та доставки цифрових документів і інших типів інформаційних ресурсів у межах організації. Контент, що підлягає управлінню, зазвичай має слабку структурованість і може включати файли різних форматів, електронні документи з неоднорідними наборами полів та інші інформаційні об'єкти.

ECM-система – це програмне забезпечення, призначене для комплексного управління корпоративним контентом. У багатьох випадках ECM-системи розглядають як окремий різновид систем управління вмістом. На пострадянському просторі поняття ECM часто ототожнюють із системами електронного документообігу (СЕД), що зумовлено подібністю їх функціонального призначення.

Система управління веб-вмістом (Web Content Management System, WCMS) – це програмний комплекс, який забезпечує створення, редагування, контроль та організацію веб-сторінок. WCMS широко застосовуються для розроблення блогів, персональних сайтів, інтернет-магазинів та інших

веб-ресурсів, орієнтованих на користувачів, які не мають спеціальної підготовки у сфері програмування.

З огляду на те, що ЕСМ-системи мають розгалужену внутрішню класифікацію за предметними галузями (HRM, DMS, CRM, ERP), термін CMS у практиці веб-розроблення поступово став використовуватися як синонім WCMS, тобто системи управління веб-сайтами. Такі CMS забезпечують управління текстовим і графічним наповненням веб-ресурсу, надаючи користувачеві інтерфейс для роботи з контентом, засоби зберігання та публікації інформації, а також автоматизуючи процеси розміщення даних у базах даних і їх подання у форматі HTML.

Існує значна кількість готових систем управління контентом, зокрема безкоштовних. За принципом роботи їх можна класифікувати на три основні типи:

1. *Генерація сторінок за запитом.*

У таких системах використовується зв'язка «модуль редагування → база даних → модуль подання». Модуль подання формує сторінку під час кожного запиту, використовуючи дані з бази. Зміни вмісту здійснюються через модуль редагування. Динамічна генерація сторінок створює додаткове навантаження на сервер, яке може бути суттєво зменшене завдяки механізмам кешування, що підтримуються сучасними веб-серверами.

2. *Генерація сторінок під час редагування.*

Системи цього типу формують статичні сторінки одразу після внесення змін до контенту. Такий підхід знижує інтерактивність взаємодії користувача з веб-ресурсом, але зменшує навантаження на сервер.

3. *Змішаний тип.*

Поєднує переваги двох попередніх підходів. Найпоширенішим варіантом є використання кешування: сторінка генерується один раз, а подальші звернення обслуговуються з кешу, що значно прискорює завантаження. Кеш може оновлюватися автоматично (за часом або при зміні певних розділів сайту) або

вручну. Інший варіант – збереження окремих інформаційних блоків під час редагування та формування сторінки з цих блоків під час запиту.

1.3 Аналіз предметної області

Післядипломна освіта – це система професійного розвитку та підвищення кваліфікації осіб, які вже здобули вищу освіту. Попри те, що за назвою вона належить до сфери освіти, за своїм змістом післядипломна освіта значною мірою охоплює науково-дослідну діяльність, результати якої можуть слугувати підставою для присудження відповідного наукового ступеня.

У законодавстві України післядипломна освіта визначається як спеціалізоване вдосконалення освіти та професійної підготовки особи шляхом поглиблення, розширення й оновлення її професійних знань, умінь і навичок або здобуття іншої спеціальності на основі раніше отриманого освітньо-кваліфікаційного рівня та набутого практичного досвіду.

Відповідно до державного визначення, післядипломна освіта забезпечує безперервність і наступність освітнього процесу та включає такі форми:

- перепідготовка – здобуття іншої спеціальності на основі наявного освітньо-кваліфікаційного рівня та практичного досвіду, у тому числі здобуття ступеня MBA;
- спеціалізація – набуття здатності виконувати окремі професійні завдання та обов’язки, що мають специфічний характер у межах певної спеціальності;
- розширення профілю (підвищення кваліфікації) – формування здатності виконувати додаткові професійні функції в межах спеціальності;
- стажування – набуття практичного досвіду виконання професійних завдань і обов’язків за певною спеціальністю.

Особа, яка пройшла перепідготовку та успішно склала державну атестацію, отримує документ про вищу освіту відповідного рівня. У разі успішного проходження стажування, спеціалізації або підвищення кваліфікації видається документ про післядипломну освіту.

Післядипломна освіта здійснюється закладами вищої освіти, що мають відповідну акредитацію, або їх структурними підрозділами, у тому числі на підставі укладених договорів про співпрацю.

У Миколаєвській області функції з надання післядипломної педагогічної освіти виконує заклад вищої освіти «Миколаївський обласний інститут післядипломної педагогічної освіти». Інститут організовує та проводить курси підвищення кваліфікації для працівників освітньої сфери, зокрема для вчителів. Створення, методичний супровід та оновлення змісту таких курсів забезпечують методисти інституту. Крім того, важливим завданням є оперативне інформування слухачів курсів та науково-педагогічних працівників про нові нормативні документи Міністерства освіти і науки України, результати проведених заходів, а також іншу актуальну інформацію, пов'язану з освітньою та педагогічною діяльністю.

Структура МОІППО:

1. Директор.
2. Заступники директора.
3. Кафедри інституту:

3.1. Кафедра філософії освіти, теорії й методики суспільствознавчих предметів.

3.2. Кафедра теорії й методики дошкільної та початкової освіти.

3.3. Кафедра теорії й методики мовно-літературної та художньо-естетичної освіти.

3.4. Кафедра теорії й методики природничо-математичної освіти та інформаційних технологій.

3.5. Кафедра педагогіки, психології та менеджменту освіти.

4. Лабораторії:

4.1. Науково-методична лабораторія медіаосвіти, інструментальної медіадидактики та освітнього дизайну.

4.2. Науково-методична лабораторія розвитку математичної, інформатичної та STEM-освіти.

4.3. Навчально-методична лабораторія практичної психології, соціальної роботи та інклюзивного навчання.

4.4. Навчально-методична лабораторія моніторингу якості освіти та сертифікації.

4.5. Навчально-методична лабораторія управління розвитком регіональної освіти.

4.6. Навчально-методична лабораторія акмеологічного розвитку.

4.7. Лабораторія організації навчального процесу та інформаційного забезпечення.

4.8. Лабораторія редакційно-видавничої діяльності.

5. Навчальний кабінет «Освіта Миколаївщини».

6. Бухгалтерія.

7. Бібліотека.

8. Юрист.

1.4 Аналіз існуючих рішень для систем управління контентом

1.4.1 Система управління контентом Joomla!

Joomla! – це система управління контентом, розроблена мовами програмування PHP та JavaScript і призначена для роботи з реляційними системами керування базами даних, зокрема MySQL та іншими промисловими СУБД. Програмне забезпечення поширюється на умовах вільної ліцензії GNU GPL.

У базовій конфігурації CMS Joomla! містить мінімальний набір інструментів, який може бути розширений за потреби шляхом встановлення додаткових модулів, плагінів та шаблонів. Такий підхід дає змогу уникнути перевантаження адміністративної панелі зайвими елементами, зменшує навантаження на сервер та оптимізує використання дискового простору

хостингу. Система підтримує багатомовність інтерфейсу як фронтальної, так і адміністративної частини, а каталог розширень містить значну кількість мовних пакетів, що встановлюються стандартними засобами адміністрування.

Структурно Joomla! складається з ядра, до якого підключаються компоненти. Компоненти, залежно від функціонального призначення, можуть включати модулі, плагіни та шаблони. Модулі відповідають за відображення окремих блоків на сторінках користувачів; плагіни забезпечують додаткову функціональність і інтегруються в адміністративну панель; шаблони визначають уніфіковане оформлення веб-сайту. Розроблення власних розширень здійснюється шляхом створення плагінів мовою PHP із використанням API Joomla! для забезпечення взаємодії з ядром системи.

На рис. 1.1 наведено схему сайту побудованого за допомогою Joomla!.

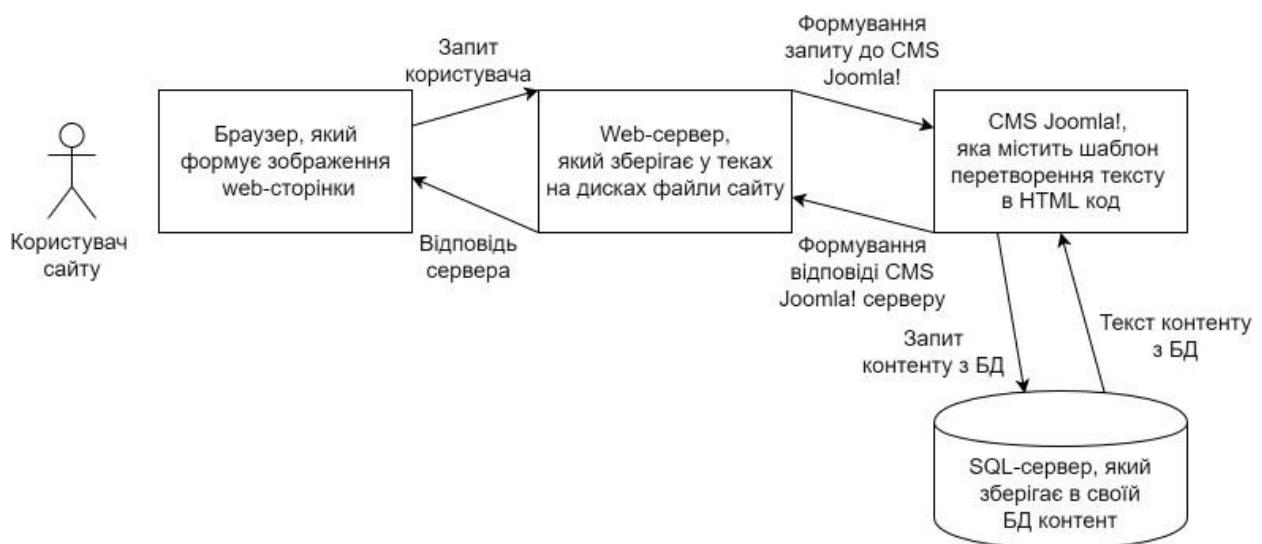


Рис. 1.1 – Схема сайту побудованого за допомогою Joomla!

1.4.2 Система управління контентом WordPress

WordPress – система управління вмістом з відкритим програмним кодом, яка завдяки простоті встановлення та використання набула широкого поширення у створенні веб-сайтів різного рівня складності – від персональних блогів до повнофункціональних веб-ресурсів. Архітектура системи, що поєднує механізми

тем і плагінів, забезпечує можливість гнучкого конструювання веб-проектів практично будь-якого типу. Програмна платформа реалізована мовою PHP та використовує базу даних MySQL.

WordPress є однією з найпоширеніших систем управління контентом у світі. На її основі створено понад 60 мільйонів веб-сайтів, що становить близько 40% усіх ресурсів, побудованих на CMS. Система є безкоштовною та має відкритий вихідний код, що сприяє активному розвитку екосистеми плагінів, віджетів і тем, які дозволяють повністю адаптувати сайт до потреб користувача.

WordPress особливо добре підходить для невеликих сайтів і блогів, оскільки не потребує спеціальних технічних навичок для встановлення та базового налаштування. Це робить систему популярною серед початківців. Користувач може оперативно змінювати параметри сайту, публікувати контент і керувати ним, зберігаючи при цьому належний рівень безпеки даних.

Разом із тим WordPress має певні обмеження. Зокрема, спрощеність використання супроводжується меншою гнучкістю у глибокому редагуванні зовнішнього вигляду та структури сайту. Для реалізації складних дизайнерських рішень або модифікації архітектури сторінок зазвичай потрібна участь кваліфікованих фахівців – веб-розробників або дизайнерів.

На рис. 1.2 наведено вигляд архітектури сайту побудованого за допомогою WordPress.

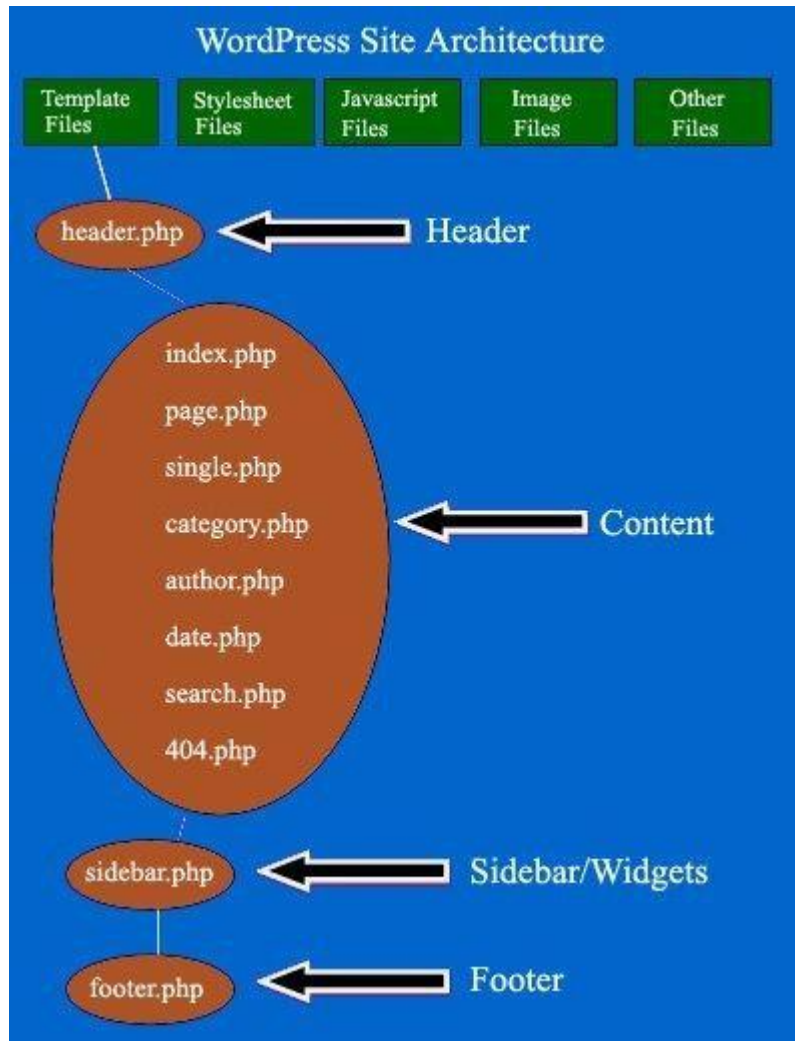


Рис 1.2 – Архітектура сайту побудованого за допомогою WordPress

1.4.3 Система управління контентом Drupal

Drupal – система управління вмістом (CMS), яка також може використовуватися як каркас для веб-додатків (CMF). Платформа розроблена мовою програмування PHP та використовує реляційні системи керування базами даних, зокрема MySQL, PostgreSQL та інші. Drupal є вільним програмним забезпеченням, що поширюється під ліцензією GPL і розвивається міжнародною спільнотою розробників-ентузіастів.

Однією з ключових особливостей Drupal є гнучка схема організації структури веб-сайту на основі таксономії. Таксономія – це механізм, який дає змогу створювати необмежену кількість тематичних категорій для контенту та

пов'язувати їх із модулями, що відповідають за введення та відображення інформації. Категорії можуть мати вигляд плоских або ієрархічних списків, а також складних структур, у яких окремий елемент може мати кілька «батьківських» та кілька дочірніх елементів. Такий підхід забезпечує можливість формування різних варіантів структуризації вмісту за допомогою одних і тих самих модулів. Наприклад, легко створюється наскрізний список ключових слів для всіх матеріалів сайту.

Інша важлива концепція з'явилася з упровадженням у Drupal розширення Content Construction Kit (ССК). ССК дає змогу доповнювати документи новими полями різних типів – від полів для введення URL-адрес та електронної пошти до полів для зберігання та відображення мультимедійних файлів. Крім того, за допомогою додаткових модулів, таких як Node Reference, можна організовувати зв'язки між документами без використання механізму таксономії. На рис. 1.3 наведено структуру системи управління контентом Drupal.

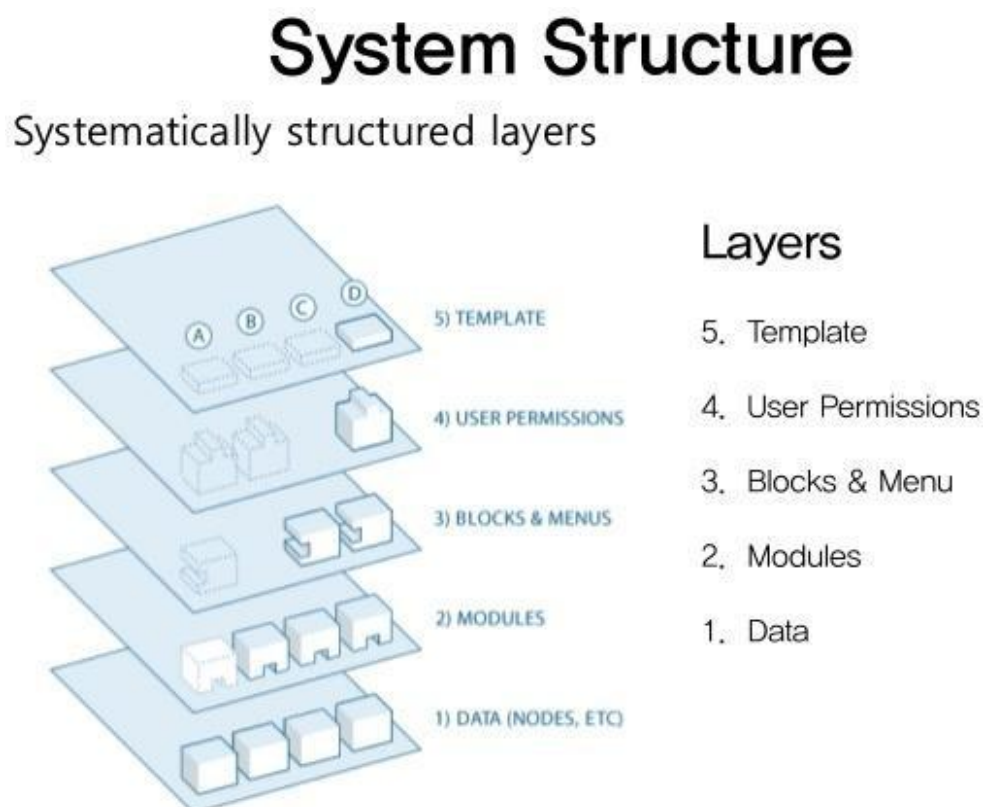


Рис. 1.3 – структура системи управління контентом Drupal

1.4.4 Система управління контентом UMI.CMS

UMI.CMS – комерційна мультисайтова система управління контентом, розроблена командою фахівців компанії «Юмісофт». Система була представлена на ринку у 2007 році. Програмна платформа реалізована мовою PHP та використовує як основне сховище даних реляційну базу MySQL.

За інформацією пресслужби компанії «Юмісофт», станом на травень 2015 року на платформі UMI.CMS функціонувало близько 800 000 веб-ресурсів різного типу, зокрема сайти-візитки, каталоги товарів, інтернет-магазини та портали.

UMI.CMS є кросплатформним програмним забезпеченням. Рекомендованою типовою конфігурацією для її роботи є веб-сервер Apache під керуванням операційної системи Linux, що доступна на більшості віртуальних хостингів. Виробник гарантує коректне функціонування системи саме на цій конфігурації та надає відповідну технічну підтримку.

Для роботи UMI.CMS на сервері Apache мають бути встановлені та активовані модулі `mod_rewrite` і `mod_auth`. У конфігурації сервера повинна бути дозволена директива `AllowOverride` зі значенням `All` або `Options FileInfo AuthConfig`, а директива `AccessFileName` має бути встановлена на `.htaccess`.

Інтерпретатор PHP повинен бути зібраний із підтримкою бібліотек `zLib`, `GD`, `libxml`, `mb_string`, `iconv`, `xsl`, `SimpleXML`, `xmlreader`. Режим `safe_mode` має бути вимкнений. Необхідно дозволити опцію `allow_url_fopen` або забезпечити наявність бібліотеки `cURL`. Обсяг пам'яті, виділений для виконання скриптів (директива `memory_limit` у файлі `php.ini`), повинен становити не менше 32 МБ (робота при 16 МБ можлива, але не гарантується). Починаючи з версії 2.8.4.4, система потребує підтримки JSON, що необхідно для роботи файлового менеджера `elFinder` та інтеграції зі сторонніми сервісами.

UMI.CMS використовує базу даних MySQL версії 5 і вище з підтримкою механізму InnoDB (робота на версіях 4.1+ можлива, але не рекомендована). Система також допускає використання альтернативних джерел даних. Крім того,

UMI.CMS підтримує технологію динамічного завантаження змін за допомогою AJAX.

1.4.5 Система управління контентом Hippo CMS

Hippo CMS – система управління вмістом з відкритим програмним кодом, розроблена мовою Java. Модульна архітектура системи та підтримка відкритих стандартів забезпечують її гнучкість, масштабованість і високу інтеграційну сумісність. Одним із ключових принципів Hippo CMS є чітке розмежування вмісту та логіки його представлення, що дає змогу контент-редакторам створювати матеріали один раз і публікувати їх у різних каналах: на веб-сайтах, мобільних версіях сайтів, у сервісах Facebook, через REST-інтерфейси, у цифрових журналах, мобільних застосунках тощо.

Система також надає веб-розробникам інструменти для створення нових каналів або налаштування існуючих шаблонів сторінок за допомогою менеджера каналів і шаблонів. Завдяки модулю релевантності Hippo CMS підтримує персоналізацію контенту: матеріали можуть бути орієнтовані на конкретних користувачів з урахуванням їхнього поточного контексту, історії взаємодії та профілів, сформованих маркетинговими інструментами.

Hippo CMS підтримує широкий спектр відкритих стандартів, зокрема JCR та REST, що спрощує розроблення та інтеграцію з наявними інформаційними системами. Рівень доставки Hippo (HST) забезпечує засоби для швидкого створення веб-сайтів та інших каналів, включно з REST-ендпойнтами та RSS-стрічками. Як інтерфейс користувача Hippo CMS, так і рівень доставки відзначаються високою гнучкістю, можливістю глибокого налаштування та розширення. На рис. 1.4 наведено структуру системи управління контентом Hippo CMS.

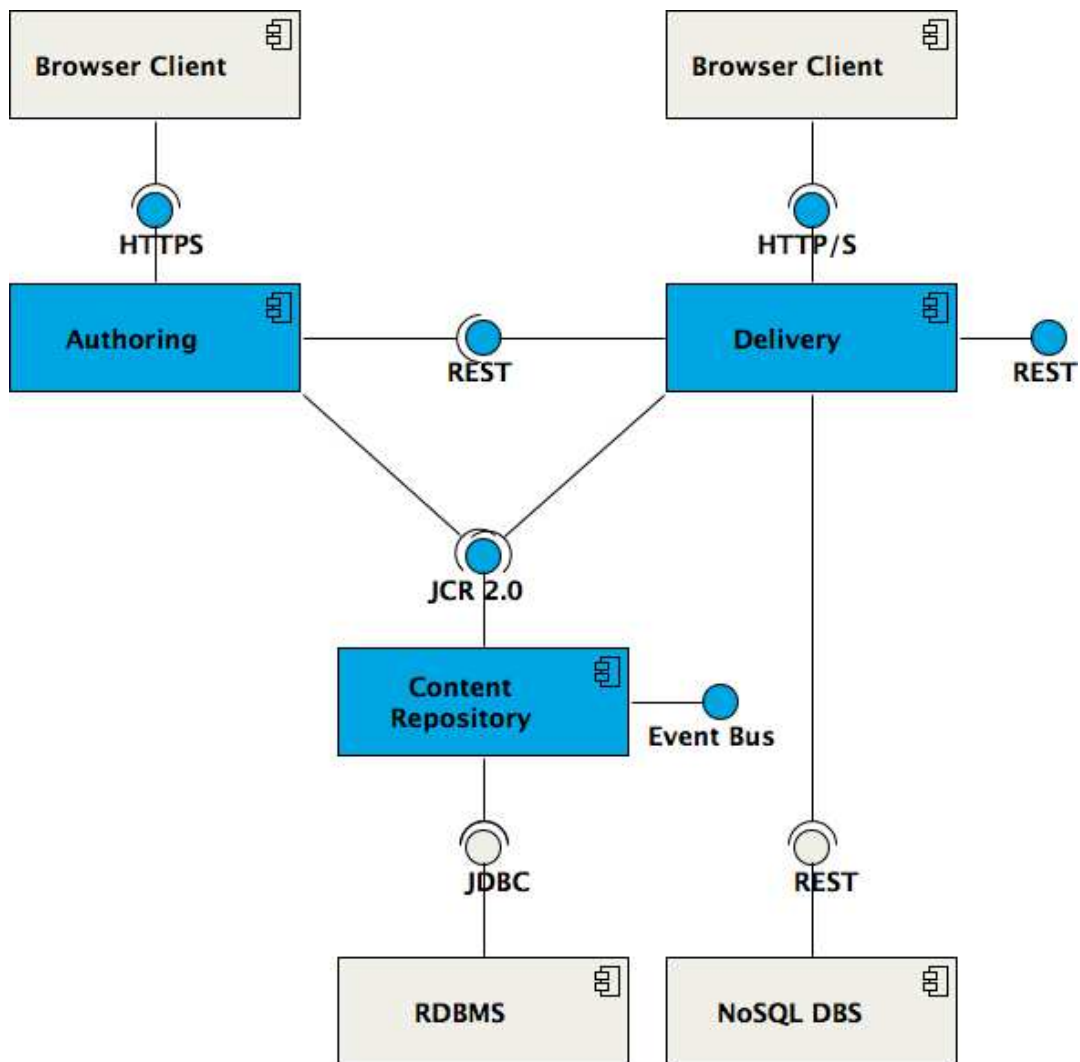


Рис. 1.4 – структура системи управління контентом Hippo CMS

1.4.6 Експертні оцінки для порівняння існуючих рішень

У таблиці 1.1 наведено порівняння існуючих систем управління контентом за шкалою від 1 до 10, де 1 – найгірша оцінка, 10 – найкраща оцінка. Усі дані базуються на експертних оцінках.

Таблиця 1.1 – Порівняння існуючих систем управління контентом.

Критерії	Joomla!	WordPress	Drupal	UML.CMS	Hippo CMS
Зручність для контент-менеджера	8	10	8	10	7
Зручність для розробника	8	8	7	10	7
Функціональність	10	8	6	10	6
Вбудовані модулі	10	9	8	4	7
Підтримка мультимедійного контенту	7	10	9	6	7
Менеджер розсилки	9	6	9	9	8
Надійність	7	7	6	9	6
Швидкість роботи	7	9	9	7	6
Якість підтримки користувачів	7	7	6	9	5
Безпека	6	6	4	10	4
Асортимент готових шаблонів дизайну	10	9	8	4	7
Початкова SEO-оптимізація	5	8	4	3	5

Слід зазначити, що безкоштовні системи управління контентом мають низку переваг, зокрема широкий вибір готових шаблонів та значну кількість вбудованих модулів. Водночас такі рішення часто характеризуються недостатнім рівнем захисту даних, нижчою продуктивністю та обмеженими можливостями технічної підтримки, незважаючи на наявність великих користувацьких спільнот.

Комерційні системи, навпаки, зазвичай забезпечують вищий рівень безпеки та стабільну цілодобову підтримку користувачів, проте їх вартість може

бути надмірно високою для окремих організацій. У зв'язку з цим доцільним є розгляд можливості розроблення індивідуальної системи управління контентом для веб-порталу МОІППО, що дозволить врахувати специфічні потреби установи та забезпечити оптимальний баланс між функціональністю, безпекою та витратами.

1.5 Вибір технологій та середовища розробки

Розроблювана веб-орієнтована система є безкоштовною, тому всі технології та інструменти, що використовуються під час її створення, мають бути вільними для застосування та не вимагати фінансових витрат. З огляду на це було обрано мову програмування Java, яка не потребує придбання ліцензій та забезпечує потужний інструментарій для створення веб-додатків.

Java - об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems у 1995 році як ключовий компонент платформи Java. З 2010 року її розвитком займається компанія Oracle, яка придбала Sun Microsystems. У стандартній реалізації програми на Java компілюються у байт-код, що під час виконання інтерпретується віртуальною машиною Java (JVM), адаптованою до конкретної платформи.

Синтаксис Java значною мірою успадкований від мов C та C++. За основу взято об'єктну модель C++, однак її було модифіковано з метою усунення типових помилок програмування та спрощення процесу створення об'єктно-орієнтованих програм. Частина операцій, які в C/C++ виконує програміст, у Java автоматизовано та передано на рівень віртуальної машини. Оскільки Java створювалася як платформонезалежна мова, вона має обмежені низькорівневі можливості роботи з апаратним забезпеченням, що певною мірою знижує продуктивність порівняно, наприклад, із C++. За потреби виконання низькорівневих операцій Java дозволяє викликати підпрограми, написані іншими мовами.

Першочерговою мотивацією створення Java була необхідність у платформонезалежній мові програмування, придатній для розроблення

програмного забезпечення, що вбудовується у побутові електронні пристрої, такі як мобільні телефони чи засоби дистанційного керування. Згодом більшість популярних веб-браузерів отримали можливість запускати Java-аплети всередині веб-сторінок. Компанія Sun Microsystems представила Java 2 (спочатку J2SE 1.2), у межах якої було реалізовано кілька конфігурацій для різних типів платформ: J2EE – для серверних застосунків, J2ME – для пристроїв з обмеженими ресурсами.

Період становлення Java збігся з активним розвитком глобальної мережі World Wide Web, що суттєво вплинуло на подальший напрям еволюції мови. Оскільки Web також потребувала платформонезалежних рішень, акценти в розвитку Java змістилися від побутової електроніки до інтернет-орієнтованого програмування.

До початкових цілей створення Java належали:

- забезпечення простого, об'єктно-орієнтованого та звичного синтаксису;
- надійність і безпечність виконання програм;
- збереження незалежності від архітектури та переносимості;
- висока продуктивність;
- підтримка інтерпретації, багатониттєвості та динамічного зв'язування модулів.

Під «незалежністю від архітектури» розуміється здатність програми, написаної мовою Java, виконуватися на будь-якій підтримуваній апаратній або програмній платформі без змін у вихідному коді та без повторної компіляції. Стандартні бібліотеки Java забезпечують уніфікований доступ до платформозалежних функцій, таких як графічна обробка, багатопотоковість та робота з мережами.

У деяких реалізаціях JVM байт-код може бути скомпільований у машинний код до або під час виконання програми для підвищення продуктивності. Основною перевагою байт-коду є його портативність, однак

інтерпретація призводить до певних втрат швидкодії порівняно зі статично скомпільованими мовами, що історично спричинило репутацію Java як «повільної» мови.

Цей недолік значною мірою усунуто завдяки впровадженню сучасних методів оптимізації у JVM. Одним із таких методів є just-in-time (JIT)-компіляція, яка перетворює байт-код у машинний код під час першого виконання програми та кешує результат. Це забезпечує значне прискорення виконання порівняно з чистою інтерпретацією, хоча й потребує додаткових ресурсів на компіляцію під час роботи. Більш складні віртуальні машини застосовують динамічну рекомпіляцію, що передбачає аналіз поведінки програми та вибірккову оптимізацію її найбільш ресурсомістких частин – так званих «гарячих точок» (hot spots). Поєднання JIT-компіляції та динамічної рекомпіляції дозволяє суттєво підвищити швидкість Java-програм без втрати їх портативності.

Ще одним підходом до оптимізації байт-коду є статична компіляція, або ahead-of-time (AOT)-компіляція. Цей метод передбачає безпосереднє перетворення байт-коду у машинний код, подібно до традиційних компіляторів. Такий підхід забезпечує вищу продуктивність порівняно з інтерпретацією, однак супроводжується втратою переносимості: програма, скомпільована AOT-методом, може виконуватися лише на цільовій платформі, для якої здійснено компіляцію.

Продуктивність офіційної віртуальної машини Java істотно зросла порівняно з ранніми версіями. Деякі дослідження демонструють, що швидкість JIT-компіляторів наближається до продуктивності традиційних компіляторів, які генерують машинний код. Водночас ефективність компілятора не завжди прямо корелює зі швидкістю виконання програми, тому реальна продуктивність може бути визначена лише шляхом ретельного тестування в конкретному середовищі.

Apache Commons – це проєкт Apache, спрямований на створення та підтримку багаторазово використовуваних компонентів для Java. Вихідний код

репозиторію Apache Commons доступний для модифікації всім учасникам Apache Software Foundation, які зобов'язані дотримуватися встановлених принципів розроблення. Бібліотеки Apache Commons широко застосовуються у таких проєктах, як Apache Tomcat, Struts, Hibernate та інших.

Згідно зі статутом, метою проєкту є «створення та підтримка пакетів, написаних мовою Java, призначених для використання у розробленні серверних застосунків і спроектованих таким чином, щоб їх можна було незалежно інтегрувати в будь-який більший продукт або фреймворк».

У структурі Apache Software Foundation Apache Commons є проєктом верхнього рівня. Усередині нього виділяють численні проєкти другого рівня, кожен з яких має власний веб-сайт, окремий каталог у спільному репозиторії вихідного коду, власну систему відстеження помилок і відповідає за розроблення одного конкретного компонента (бібліотеки) Java. Активні підпроєкти мають кількох постійних учасників, тоді як неактивні можуть не мати активних розробників взагалі.

JavaMail – це програмний інтерфейс (API) для мови Java, призначений для надсилання та отримання електронної пошти за протоколами SMTP, POP3 та IMAP. Пакет JavaMail інтегрований у платформу Java EE, проте також доступний як окремий модуль для використання у середовищі Java SE.

Spring Security – це комплексний фреймворк безпеки, орієнтований на корпоративні застосунки, створені на основі Java EE. Особливу увагу в ньому приділено підтримці проєктів, розроблених із використанням Spring Framework, який є одним із провідних рішень для створення корпоративного програмного забезпечення на платформі Java.

Spring Security застосовують у тих випадках, коли стандартні механізми безпеки, визначені специфікаціями сервлетів Java EE або EJB, виявляються недостатніми для типових сценаріїв корпоративних систем. Незважаючи на наявність цих стандартів, вони не є повністю переносимими на рівні архівів WAR або EAR. У разі зміни серверного середовища зазвичай виникає потреба у

значному обсязі додаткового налаштування механізмів безпеки. Використання Spring Security дає змогу уникнути цих труднощів і забезпечує доступ до широкого набору гнучких та розширюваних засобів захисту.

Безпека застосунків охоплює дві ключові сфери: аутентифікацію та авторизацію (контроль доступу). Саме на ці аспекти спрямована функціональність Spring Security. Аутентифікація – це процес встановлення особи (принципала), яка намагається виконати певну дію в системі, тобто користувача, пристрою або іншої сутності. Авторизація – це процес визначення того, чи має встановлений принципал право виконувати відповідну дію. Таким чином, авторизація можлива лише після успішного завершення аутентифікації.

Spring Framework – це фреймворк з відкритим програмним кодом, що реалізує принципи інверсії управління та надає контейнер для керування компонентами Java-застосунків. Базові можливості Spring Framework можуть бути використані в будь-якому Java-проєкті, а додаткові модулі забезпечують підтримку створення веб-додатків на платформі Java EE. Водночас Spring Framework не нав'язує розробнику жорсткої моделі програмування, що сприяло його популярності як альтернативи або доповнення до моделі Enterprise JavaBeans (EJB) у спільноті Java-розробників.

IntelliJ IDEA – комерційне інтегроване середовище розробки, створене компанією JetBrains і призначене для роботи з різними мовами програмування, зокрема Java, Python, Scala, PHP та іншими. Середовище постачається у двох редакціях: безкоштовній версії Community Edition із обмеженою функціональністю та повнофункціональній комерційній версії Ultimate Edition, для якої активні розробники відкритих проєктів можуть отримати безкоштовну ліцензію. Вихідні коди Community-версії поширюються відповідно до ліцензії Apache 2.0. Готові бінарні збірки доступні для операційних систем Linux, macOS та Windows.

Eclipse (англ. eclipse – «затемнення») – вільне модульне інтегроване середовище розробки програмного забезпечення, що розробляється та

підтримується Eclipse Foundation. До його складу входять численні проєкти, зокрема платформа Eclipse, інструментарій для програмування мовою Java, засоби контролю версій, конструктори графічних інтерфейсів та інші компоненти. Середовище написане переважно мовою Java та може використовуватися для створення Java-застосунків, а також застосунків іншими мовами за допомогою відповідних плагінів. Підтримуються такі мови, як Ada, C, C++, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з Ruby on Rails), Scala, Clojure та Scheme. Серед спеціалізованих середовищ розробки варто відзначити Eclipse ADT (Ada Development Toolkit), Eclipse CDT для C/C++, Eclipse JDT для Java та Eclipse PDT для PHP.

NetBeans IDE – вільне інтегроване середовище розробки, що підтримує мови Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python та Groovy. Середовище може встановлюватися як у мінімальній конфігурації для окремих мов, так і у повному наборі компонентів. За замовчуванням NetBeans IDE забезпечує підтримку розроблення застосунків для платформ J2SE та J2EE.

Вихідні коди NetBeans поширюються під ліцензіями GPLv2 та CDDL. Проєкт тривалий час підтримувався компанією Sun Microsystems, а після її придбання – компанією Oracle. Наразі розвиток NetBeans здійснюється незалежною спільнотою розробників (NetBeans Community) та організацією NetBeans.org.

NetBeans IDE доступне для операційних систем Microsoft Windows, GNU/Linux, FreeBSD та Solaris (як для архітектури SPARC, так і x86). Для інших платформ передбачена можливість самостійної збірки середовища з вихідних кодів.

1.6 Засоби розробки інтерфейсу

Для розроблення веб-додатка на Java було обрано технологію JavaServer Faces (JSF). JavaServer Faces – це компонентно-орієнтований фреймворк, що реалізує архітектурний шаблон Model–View–Controller (MVC) та призначений для створення веб-інтерфейсів у Java EE-застосунках. На відміну від більшості

MVC-фреймворків, які ґрунтуються на обробленні запитів, JSF використовує підхід, орієнтований на компоненти. Стан компонентів інтерфейсу зберігається між запитами та відновлюється під час повторного звернення, що спрощує керування взаємодією користувача з додатком.

Переваги JSF значною мірою зумовлені наявністю офіційної специфікації, яка визначає загальні принципи побудови фреймворків цього типу. Специфікація забезпечує сумісність різних реалізацій JSF, дозволяючи створювати фреймворки з різною внутрішньою архітектурою, але єдиними базовими вимогами. Еталонна реалізація JSF найбільш ефективна для невеликих веб-сайтів та застосунків із типовими інтерфейсними сценаріями.

Архітектурний шаблон Model–View–Controller (MVC) широко застосовується під час проєктування програмного забезпечення. Він передбачає поділ системи на три логічні складові:

- Модель (Model) – відповідає за зберігання даних та реалізацію бізнес-логіки;
- Представлення (View) – забезпечує відображення даних користувачеві;
- Контролер (Controller) – обробляє події, що виникають у результаті дій користувача, та координує взаємодію між Моделлю і Представленням.

Основною метою MVC є відокремлення даних від інтерфейсу користувача, що дозволяє мінімізувати вплив змін у представленні на логіку роботи з даними та навпаки. Такий підхід сприяє створенню гнучкої архітектури, яка полегшує подальший розвиток, модифікацію та розширення програмного забезпечення, а також забезпечує можливість повторного використання окремих компонентів.

У межах MVC Модель інкапсулює ядро даних і функціональність їх оброблення, не залежачи від способів введення чи виведення інформації. Представлення може містити кілька взаємопов'язаних елементів, таких як таблиці, форми чи інші візуальні компоненти. Контролер відстежує події, що виникають унаслідок дій користувача (натискання кнопок, введення тексту,

переміщення курсора тощо), та транслює їх у запити до Моделі або компонентів Представлення.

Відокремлення Моделі від Представлення забезпечує незалежність механізмів відображення інформації. У разі внесення змін до Моделі через Контролер відповідні візуальні компоненти автоматично оновлюються, відображаючи актуальний стан даних. Така організація сприяє впорядкованості структури програмного забезпечення та зменшує його загальну складність.

1.7 Засоби тестування продукту

Під час розроблення моделі було застосовано технологію розробки через тестування (Test-Driven Development, TDD). TDD – це методологія створення програмного забезпечення, що передбачає виконання коротких ітерацій, кожна з яких розпочинається з написання тестів, які визначають необхідні зміни або нову функціональність. Метою кожної ітерації є розроблення коду, що забезпечує успішне проходження попередньо підготовлених тестів, після чого здійснюється рефакторинг для узгодження та оптимізації отриманого рішення.

Однією з ключових особливостей TDD є те, що написання тестів до створення коду прискорює процес внесення змін і підвищує передбачуваність поведінки системи. Методологія вимагає від розробника створення автоматизованих модульних тестів, які формулюють вимоги до коду безпосередньо перед його реалізацією. Кожен тест містить набір умов, що можуть виконуватися або не виконуватися; у разі їх виконання тест вважається пройденим, що підтверджує відповідність поведінки програми очікуванням розробника.

На практиці модульні тести охоплюють критичні та нетривіальні фрагменти коду: ті, що часто змінюються, впливають на роботу значної частини системи або мають велику кількість залежностей. Ефективне застосування TDD потребує швидкої реакції середовища розробки на невеликі модифікації коду, а також архітектури, побудованої на використанні незалежних або слабо пов'язаних компонентів, що істотно спрощує тестування.

TDD не лише забезпечує перевірку коректності роботи програмного забезпечення, а й впливає на його архітектурний дизайн. Орієнтуючись на тести, розробники швидше формують уявлення про необхідну користувацьку функціональність, завдяки чому елементи інтерфейсу та логіки взаємодії визначаються ще до завершення реалізації системи. Вимоги до якості тестів є такими самими високими, як і до основного програмного коду, оскільки саме тести визначають надійність, передбачуваність і підтримуваність програмного продукту.

1.8 Вибір ORM

ORM (англ. Object-Relational Mapping, об'єктно-реляційне відображення) – це технологія програмування, що забезпечує узгодження реляційних баз даних із концепціями об'єктно-орієнтованих мов програмування, створюючи своєрідну «віртуальну об'єктну базу даних».

В об'єктно-орієнтованому програмуванні програмні об'єкти моделюють сутності реального світу. Наприклад, адресна книга може містити перелік осіб, кожна з яких має кілька телефонних номерів і кілька адрес. У термінах ООП такі сутності представляються об'єктами класу «Людина», що містять поля «ім'я», «список телефонів» та «список адрес».

Основна проблема полягає у перетворенні таких об'єктів у форму, придатну для зберігання у файлах або реляційних базах даних, а також у подальшому відтворенні цих об'єктів зі збереженням їх властивостей і взаємозв'язків. Такі об'єкти називають постійними (англ. persistent). Історично було запропоновано кілька підходів до розв'язання цього завдання.

Створено значну кількість ORM-фреймворків, які усувають необхідність ручного перетворення об'єктів для їх зберігання у реляційних базах даних. Деякі з них надають бібліотеки класів, здатні автоматично виконувати відповідні перетворення. Маючи опис таблиць у базі даних та відповідні об'єкти у програмі, такі фреймворки автоматично формують SQL-запити та перетворюють результати у відповідні об'єкти. Наприклад, під час запиту об'єкта «Людина»

ORM-система самостійно сформує SQL-запит, виконає його та перетворить отримані дані на об'єкти «Номер телефону».

З погляду розробника ORM-система має виглядати як постійне сховище об'єктів: програміст створює об'єкти та працює з ними звичним чином, а їх збереження у реляційній базі даних відбувається автоматично.

На практиці, однак, ORM-рішення не є повністю прозорими. Вони частково обмежують можливість ігнорувати специфіку бази даних, а транзакційний шар може бути повільним або неефективним, особливо з огляду на згенерований SQL. Це може призводити до зниження продуктивності та збільшення споживання пам'яті порівняно з програмами, у яких SQL-запити формуються вручну.

Попри це ORM значно зменшує обсяг рутинного та помилконебезпечного коду, прискорюючи розробку. Більшість сучасних ORM-фреймворків дозволяють за потреби явно визначати SQL-запити, що використовуються під час збереження, завантаження або пошуку даних.

Java Persistence API (JPA) – стандартизований інтерфейс для ORM-фреймворків у середовищі Java. Він є частиною EJB 3 та J2EE 5, хоча може використовуватися незалежно. JPA виник як відповідь на популярність вільного ORM-фреймворку Hibernate та потребу у незалежному від конкретної реалізації стандарті.

Hibernate – один із найпоширеніших ORM-фреймворків для платформи Java, що є вільним програмним забезпеченням. Він забезпечує зручний механізм відображення об'єктно-орієнтованої моделі даних на реляційну базу даних. У межах цього проєкту саме Hibernate було обрано як основний засіб взаємодії з базою даних.

Метою Hibernate є усунення необхідності ручного програмування типових операцій доступу до даних. Фреймворк може використовуватися як під час створення нової бази даних, так і для роботи з уже існуючими. Hibernate автоматично встановлює відповідність між класами та таблицями, а також між

типами даних Java та SQL. Він генерує SQL-запити, виконує операції читання та запису, обробляє результати та забезпечує сумісність із різними СУБД, що суттєво скорочує час розробки.

Hibernate підтримує прозору персистентність POJO-об'єктів (Plain Old Java Objects). Єдиною обов'язковою вимогою до класу є наявність конструктора за замовчуванням.

Фреймворк також надає SQL-подібну мову Hibernate Query Language (HQL), яка дозволяє виконувати запити до об'єктної моделі. Крім того, Hibernate підтримує Criteria API – об'єктно-орієнтовану альтернативу HQL, що забезпечує більш гнучке та типобезпечне формування запитів.

1.9 Вибір бази даних

База даних – це впорядкована сукупність даних, організованих відповідно до певної концептуальної моделі, яка визначає характеристики цих даних та взаємозв'язки між їх елементами. Така сукупність призначена для підтримки однієї або кількох сфер застосування. У загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані в ній організовуються відповідно до обраної моделі представлення, тому сучасна база даних, окрім власне даних, включає їх опис і може містити засоби для їх оброблення.

У сучасних інформаційних системах використовують два основні типи баз даних: реляційні та нереляційні (NoSQL).

Нереляційні бази даних (NoSQL)

NoSQL-бази даних застосовують альтернативні механізми зберігання та отримання даних порівняно з традиційною реляційною моделлю. Вони набули широкого поширення у задачах, пов'язаних із великими обсягами даних (big data) та обробленням інформації в режимі реального часу. Термін Not Only SQL підкреслює, що такі системи можуть підтримувати SQL-подібні структури та мови запитів, але не обмежуються ними.

MongoDB – документоорієнтована система керування базами даних з відкритим кодом, яка не потребує попереднього визначення схеми. Вона займає проміжне місце між високопродуктивними системами типу «ключ–значення» та реляційними СКБД, забезпечуючи гнучкість формування запитів. MongoDB підтримує зберігання документів у JSON-подібному форматі, створення індексів, ефективне зберігання великих бінарних об'єктів, журналювання операцій, роботу за парадигмою Map/Reduce, реплікацію та побудову відмовостійких конфігурацій.

Apache Cassandra – розподілена NoSQL-система, орієнтована на створення масштабованих і надійних сховищ великих масивів даних, представлених у вигляді хеш-структур. Спочатку розроблена у Facebook, у 2009 році вона була передана до Apache Software Foundation. Cassandra використовується у промислових рішеннях компаній Cisco, IBM, Cloudkick, Reddit, Digg, Rackspace, Twitter та інших.

Реляційні бази даних

Реляційна база даних ґрунтується на реляційній моделі, у якій дані подаються у вигляді нормалізованих відношень. Для роботи з такими базами застосовують реляційні системи керування базами даних (СКБД).

MySQL – реляційна СКБД з відкритим кодом, створена як альтернатива комерційним рішенням. Спочатку вона була близькою за структурою до mSQL, однак з часом значно розширила функціональність. MySQL є однією з найпоширеніших СКБД, особливо у сфері динамічних веб-застосунків, завдяки широкій підтримці з боку різних мов програмування.

PostgreSQL – об'єктно-реляційна СКБД, яка є альтернативою як комерційним продуктам (Oracle Database, Microsoft SQL Server, IBM DB2), так і системам з відкритим кодом (MySQL, Firebird, SQLite). На відміну від багатьох інших проєктів, PostgreSQL не контролюється однією компанією; її розвиток забезпечується спільнотою розробників та організацій, зацікавлених у впровадженні сучасних технологій. Сервер PostgreSQL написаний мовою C і

зазвичай поширюється у вигляді вихідних текстів, які користувач компілює самостійно.

Oracle Database – об’єктно-реляційна СКБД, розроблена компанією Oracle Corporation, що належить до класу високопродуктивних корпоративних рішень.

Обґрунтування вибору СКБД

Для реалізації проєкту було обрано MySQL, оскільки вона є безкоштовною, легко встановлюється та налаштовується, а також забезпечує необхідний рівень продуктивності. MySQL добре підходить як для вбудовування у Java-застосунки, так і для оброблення транзакцій у режимі реального часу.

1.10 Динамічне відображення даних

AJAX (Asynchronous JavaScript and XML) – це технологія, що забезпечує взаємодію веб-клієнта з сервером без необхідності перезавантаження веб-сторінки. Завдяки цьому зменшується час відгуку, а веб-додаток за рівнем інтерактивності наближається до настільних програм. Наприклад, під час натискання кнопки «Голосувати» браузер надсилає запит на сервер, а сервер повертає відповідь про успішне прийняття голосу без оновлення сторінки.

Ключовою характеристикою AJAX, що відображена у першій літері аббревіатури, є асинхронність. Це означає, що після надсилання запиту браузер може продовжувати виконувати інші дії – відображати повідомлення про очікування, дозволяти прокручування сторінки тощо.

Технологія AJAX ґрунтується на поєднанні таких компонентів:

- (X)HTML та CSS – для структурування та стилізації інформації;
- DOM-моделі, з якою працює JavaScript на стороні клієнта для забезпечення динамічного відображення та взаємодії з даними;
- XMLHttpRequest – для асинхронного обміну даними з веб-сервером (у деяких фреймворках можуть використовуватися альтернативні механізми, такі як IFrame або SCRIPT-теги);
- JSON – як один із найпоширеніших форматів обміну даними, хоча можуть застосовуватися й інші формати: HTML, текст, XML, EBML тощо.

Типовий AJAX-додаток складається щонайменше з двох частин: клієнтської, що виконується у браузері та реалізована переважно мовою JavaScript, і серверної, що може бути написана, наприклад, мовами Ruby, Java або PHP.

У традиційній синхронній моделі браузер надсилає запит і очікує завершення всіх операцій на сервері. Сервер виконує необхідні дії – звернення до бази даних, формування відповіді – після чого повертає результат. Усі процеси виконуються послідовно, а мережеві затримки збільшують час очікування. Користувач у цей час не може взаємодіяти зі сторінкою.

На відміну від цього, у моделі AJAX браузер підтримує постійний канал зв'язку з сервером або періодично надсилає коригувальні запити. Необхідні зміни передаються цим каналом без перезавантаження сторінки. Проблема застарілого контексту при цьому безпосередньо пов'язана із забезпеченням цілісності даних, за яку, як і у випадку валідації форм, остаточну відповідальність несе сервер.

1.11 Вибір засобів розробки серверної частини

У процесі розроблення системи було використано Spring Framework, насамперед як контейнер Inversion of Control (IoC). Spring Framework забезпечує розв'язання широкого спектра завдань, з якими стикаються Java-розробники та організації, що створюють інформаційні системи на основі платформи Java. Завдяки великій кількості функціональних можливостей складно виокремити один домінуючий структурний компонент фреймворку. Важливою причиною його популярності є те, що Spring не є жорстко прив'язаним до Java Enterprise Edition, хоча й має глибоку інтеграцію з нею.

Spring Framework найбільш відомий як джерело розширень, необхідних для ефективної розробки складних бізнес-застосунків без використання громіздких моделей програмування, які історично домінували в індустрії. Крім того, Spring запровадив низку функціональних можливостей, що згодом стали

стандартними у сучасних підходах до розроблення, у тому числі поза межами платформи Java.

Фреймворк пропонує узгоджену модель розроблення, придатну для більшості типів застосунків, створених на основі Java. Вважається, що Spring реалізує підхід, заснований на найкращих індустріальних практиках, роблячи їх доступними для широкого кола Java-проектів.

Центральним елементом Spring Framework є контейнер Inversion of Control, який забезпечує конфігурацію та керування Java-об'єктами за допомогою механізмів відображення. Контейнер відповідає за повний життєвий цикл об'єктів: їх створення, виклик методів ініціалізації та конфігурацію шляхом встановлення залежностей між ними. Об'єкти, створені контейнером, називають керованими об'єктами (beans). Зазвичай конфігурація контейнера здійснюється шляхом завантаження XML-файлів, що містять визначення beans та необхідні параметри для їх створення.

Отримання об'єктів у Spring можливе двома способами:

- Пошук залежності (Dependency Lookup) – об'єкт самостійно звертається до контейнера з метою отримання екземпляра за ім'ям або типом.
- Впровадження залежності (Dependency Injection, DI) – контейнер сам передає необхідні залежності іншим об'єктам через конструктор, властивості або фабричні методи.

У межах даної системи було застосовано саме підхід Dependency Injection.

Впровадження залежності – це процес надання зовнішніх залежностей програмному компоненту. DI є специфічною формою інверсії управління, коли створення та конфігурація залежностей передаються зовнішньому механізму, що відповідає принципу єдиної відповідальності. У межах IoC-контейнера частина об'єктів створюється звичайним способом, тоді як інша частина – контейнером відповідно до конфігурації.

У традиційному підході, коли об'єкту потрібен доступ до певного сервісу, він самостійно отримує посилання на нього або звертається до service locator. У

випадку DI об'єкт лише декларує властивість для зберігання посилання на сервіс, а контейнер автоматично надає відповідну реалізацію під час створення об'єкта.

Перевагою DI є гнучкість: легко створювати альтернативні реалізації сервісів і визначати, яка саме реалізація має бути використана, наприклад, у конфігураційному файлі, без змін у коді залежних компонентів. Це особливо корисно під час модульного тестування, оскільки дозволяє просто підміняти реальні сервіси тестовими «заглушками».

Водночас надмірне використання DI може ускладнювати супровід програмного забезпечення, оскільки для розуміння поведінки системи необхідно аналізувати не лише вихідний код, а й конфігураційні файли. Останні часто не відображаються у засобах розробки так само зручно, як код, якщо IDE не має спеціальної підтримки відповідного фреймворку.

Висновки

Вибір платформи та фреймворку для розроблення програмного забезпечення визначає низку критично важливих факторів, що безпосередньо впливають на якість, продуктивність і масштабованість кінцевого продукту. Саме тому в даному розділі було здійснено аналіз доступних інструментів та обґрунтовано вибір технологій для створення системи управління контентом веб-порталу.

З огляду на специфіку сучасних інтернет-технологій, добір засобів розроблення має ґрунтуватися на таких критеріях, як вимоги до продуктивності програмного забезпечення, оптимальний рівень використання ресурсів, складність проєкту та кваліфікація розробника.

Доцільним є застосування фреймворку Spring MVC у поєднанні з мовою програмування Java, оскільки наявність великої кількості готових рішень, бібліотек і модулів істотно прискорює процес розроблення. Крім того, більшість технологій, пов'язаних із Java, є безкоштовними, що робить їх економічно вигідними для реалізації проєкту.

Розроблення програмного продукту на основі Spring MVC здійснюється у середовищі IntelliJ IDEA, яке забезпечує зручні засоби інтеграції сторонніх бібліотек і модулів, а також підтримує широкий спектр інструментів для ефективної роботи розробника.

Розділ 2. Розробка проектних рішень по системі

2.1 Загальносистемні проектні рішення

2.1.1 Організаційна структура

Призначення системи управління контентом полягає у централізації інформаційної структури контенту та забезпеченні користувачів зручним і швидким інтерфейсом, який дозволяє отримувати, опрацьовувати та використовувати інформацію для розв'язання управлінських та інших прикладних завдань.

Для реалізації зазначених функцій система повинна включати сховище даних та засоби введення й виведення інформації. Сховище має підтримувати автоматичну взаємодію з наявними інформаційними системами, а також можливість розподілу на кілька фізичних серверів. Важливою вимогою є забезпечення одночасної роботи значної кількості користувачів із гарантуванням оперативного збереження цілісності даних. Оскільки точну кількість одночасних підключень передбачити неможливо, система повинна бути розрахована на високі навантаження. Користувачі можуть працювати як у межах МОІППО, так і поза його інфраструктурою.

Архітектурними підходами, що відповідають зазначеним вимогам, є клієнт-серверна та сервіс-орієнтована архітектури.

У межах тришарової клієнт-серверної архітектури сховище даних фізично розміщується на окремому сервері, тоді як робочі місця користувачів оснащуються засобами доступу до нього. Між клієнтами та сховищем функціонує проміжний шар, який відповідає за оброблення запитів і забезпечує логічну взаємодію між рівнями системи.

Сервіс-орієнтована архітектура ґрунтується на модульному підході, що передбачає використання сервісів зі стандартизованими інтерфейсами. В її основі лежать принципи повторного використання функціональних компонентів, усунення дублювання функцій, уніфікації типових операційних процесів. Компоненти такої системи можуть бути розподілені між різними вузлами мережі та функціонувати як незалежні, слабо пов'язані та взаємозамінні

сервіси. Інтерфейси сервісів забезпечують інкапсуляцію деталей реалізації, включно з операційною системою, платформою, мовою програмування та особливостями розробника.

З практичної точки зору клієнт-серверна архітектура є більш зручною для використання, оскільки доступ до веб-порталу (серверної частини) не потребує спеціалізованого програмного забезпечення: достатньо стандартного веб-браузера, наявного в будь-якій сучасній операційній системі. Саме тому цей підхід було обрано як базовий для розроблення програмного забезпечення системи управління контентом веб-порталу.

2.1.2 Функціональна структура

У процесі функціонування система взаємодіє з підсистемою управління користувачами, що забезпечує виконання ними відповідних функцій. Структурна схема цієї взаємодії наведена на рисунку 2.1.

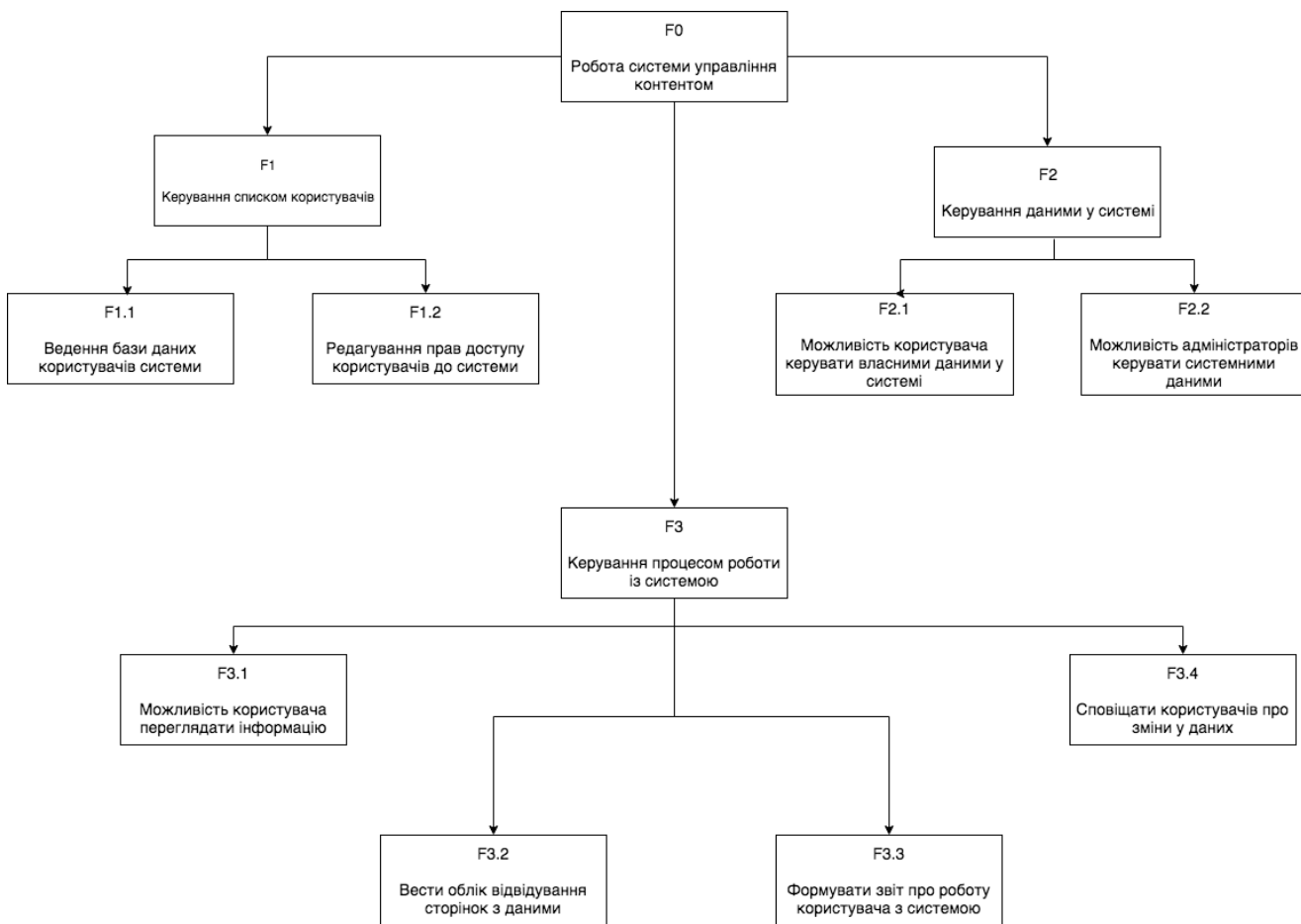


Рис. 2.1. Схема функціональної структури

Користувачі системи, відповідно до своїх повноважень, мають доступ до таких функціональних можливостей:

- Користувач — перегляд інформаційних сторінок та внесення змін до власних профільних даних.
- Методист — перегляд і редагування даних на сторінках, управління дистанційними курсами, а також виконання обмежених функцій з адміністрування користувачів.
- Адміністратор — повний доступ до управління користувачами (створення, редагування, видалення), а також до управління даними системи.

2.1.2 Опис функцій, що автоматизуються.

Завданням системи, яка розробляється, є автоматизація таких основних функцій користувача:

F1.1. Забезпечити користувачам можливість реєстрації в системі, управління власним обліковим записом, включно з можливістю його видалення за потреби.

Процеси: ведення обліку даних користувачів системи.

F1.2. Забезпечити можливість редагування прав доступу користувачів до сервісів системи.

Процеси: призначення користувачам відповідних ролей у системі.

F2.1. Надавати користувачам можливість розміщувати власні дані в системі та керувати ними відповідно до індивідуальних потреб.

Процеси: розроблення інтерфейсу користувача для управління даними.

F2.2. Забезпечити адміністраторам можливість повноцінного управління даними, що зберігаються в системі.

Процеси: створення адміністративного інтерфейсу для керування даними; процеси можуть уточнюватися або змінюватися в ході розроблення чи експлуатації системи.

F3.1. Надавати користувачам можливість переглядати інформацію, доступну відповідно до їхніх прав.

Процеси: створення інтерфейсу користувача для перегляду інформаційних матеріалів.

F3.2. Забезпечити ведення обліку відвідувань сторінок із даними.

Процеси: формування статистики переглядів сторінок порталу користувачами.

F3.3. Формувати звіти про роботу користувача в системі.

Процеси: розроблення механізмів формування відповідних звітів.

F3.4. Сповіщати користувачів про зміни в даних.

Процеси: створення інтерфейсу та механізмів сповіщення користувачів у разі необхідності.

Для впровадження системи необхідно створити та налаштувати єдину базу даних для збереження інформації (статті, новини, накази, навчальні матеріали тощо), та інформацію про користувачів. Також треба розробити програмне забезпечення яке буде дозволяти зручно адмініструвати систему.

2.1.3 Опис постановки задачі

На підставі аналізу вимог, обмежень та попередніх концепцій реалізації системи сформульовано постановку задачі. Необхідно розробити систему управління та оброблення даних, яка відповідатиме зазначеним функціональним вимогам. Система має забезпечувати надійне збереження інформації та безпечний доступ до неї. Крім того, вона повинна підтримувати одночасну безперебійну роботу значної кількості користувачів, причому зміни, внесені адміністраторами та модераторами, мають негайно відображатися для всіх інших користувачів.

Програмне забезпечення повинно бути реалізоване на основі технологічної платформи Spring MVC / MySQL / Payara / Java / IntelliJ IDEA та забезпечувати інтуїтивно зрозумілий графічний веб-інтерфейс користувача.

2.1.4 Загальний опис системи

Система призначена для створення та оброблення даних у межах діяльності навчального закладу (інституту). До її основних функціональних характеристик належать:

- підтримання єдиної інформаційної бази з розподіленим доступом;
- забезпечення авторизованого доступу до даних відповідно до встановлених прав;
- оперативне відображення змін, внесених із будь-якого робочого місця;
- можливість формування завдань для генерації додатків до дипломних робіт;
- збереження згенерованих системою даних з метою подальшого редагування або відновлення інформації у разі виникнення помилок у роботі системи.

Система має інтуїтивно зрозумілий графічний інтерфейс користувача, що мінімізує витрати на навчання персоналу, та розроблена з максимально можливим використанням наявного в установі обладнання й програмного забезпечення.

2.1.5 Захист від несанкціонованого доступу

Методи захисту даних на персональних комп'ютерах є надзвичайно різноманітними як за своїм призначенням, так і за технічним способом реалізації. Їх умовно поділяють на механічні, апаратні та програмні. У даній системі захист від несанкціонованого доступу реалізовано на системному рівні програмними засобами. Для роботи з системою користувач повинен пройти процедуру авторизації; доступ до даних і функцій надається лише після введення коректного імені користувача та відповідного пароля.

Авторизація здійснюється за допомогою спеціально розробленого механізму, який передбачає створення алгоритму безпечної перевірки та збереження інформації про користувачів, а також інтерфейсу для взаємодії з цими даними (наприклад, форми введення облікових даних). Крім того, адміністратор системи повинен мати можливість змінювати права доступу користувачів та інші параметри конфігурації.

Для доступу до окремих категорій даних (наприклад, інформації про напрям підготовки) можуть застосовуватися додаткові технічні або програмні

засоби захисту, такі як використання карток доступу та відповідних пристроїв їх зчитування.

2.2 Рішення по організаційному забезпеченню

2.2.1 Організаційна структура інституту

Система призначена для використання широким колом користувачів, що представляють різні структурні підрозділи закладу освіти (відділи, факультети, кафедри тощо). За наявної організаційної структури кожен підрозділ має власних фахівців, відповідальних за підтримку локальних інформаційних систем.

Розроблення системи пропонується здійснювати на базі обчислювального центру закладу, який має найбільший досвід та технічні можливості у сфері інформаційних технологій (зокрема, адміністрування серверів, підтримка мережевої інфраструктури тощо). Оскільки основною метою системи є інтеграція наявних інформаційних ресурсів у єдиний комплекс, потреба в окремому супроводі локальних систем у структурних підрозділах відпадає. Фахівців, які вивільняються внаслідок цього, доцільно залучити до роботи обчислювального центру для забезпечення підтримки нової системи на апаратному та програмному рівнях.

Крім того, функціонування системи потребує розроблення спеціалізованих сервісів для забезпечення роботи користувачів. Відповідальність за створення та підтримку таких сервісів пропонується покласти на кафедри інформаційних технологій. У межах подальшої автоматизації передбачається поступовий перехід до електронного документообігу.

2.2.2 Заходи щодо організаційного забезпечення

Ефективне функціонування системи потребує впровадження організаційних заходів, спрямованих на підтримання актуальності даних про поточні процеси та забезпечення своєчасної й коректної реакції користувачів на інформацію, що надається системою. До таких заходів належать:

- навчання користувачів роботі в системі відповідно до їхніх функціональних обов'язків;

- проведення тестової експлуатації системи в умовах, максимально наближених до реальних;
- мінімізація часу між оновленням версій даних і фактичним відображенням змін, тобто оперативна реакція адміністраторів і модераторів на зміни у виробничих процесах;
- обов'язкова участь усіх структурних підрозділів у формуванні даних, регулярний оперативний контроль взаємопов'язаних процесів керівниками підрозділів та контроль виробничих циклів;
- моніторинг і оцінювання коректності роботи користувачів;
- внесення необхідних коригувань до системи у випадках, коли дії користувачів призводять до порушення коректності роботи її алгоритмів.

2.2.3 Організаційно-кадрові заходи

Для забезпечення належного функціонування апаратних і програмних засобів, а також для виконання поточних завдань із налаштування, встановлення та відновлення загальносистемного програмного й апаратного забезпечення, у структурі університету доцільно передбачити введення штатної посади системного адміністратора. Альтернативним варіантом є залучення зовнішніх фахівців шляхом аутсорсингу, тобто передачі ІТ-функцій спеціалізованим стороннім організаціям.

2.3 Рішення по інформаційному забезпеченню

2.3.1 Опис інформаційного забезпечення системи

Одним із ключових елементів інформаційного забезпечення системи є визначення її внутрішнього стану, що відображається у моделі класів. На цьому етапі розглядаються постійні сутності бізнес-процесів. Бізнес-процес розуміється як діяльність, що має вхідний продукт, створює додану вартість та формує вихідний продукт для внутрішнього або зовнішнього споживача. Класи, які описують такі процеси, надалі називатимемо класами-сутностями або класами предметної області.

Для побудови моделі класів формується словник даних, що ґрунтується на сутностях, визначених під час бізнес-аналізу:

- Користувач — особа, яка взаємодіє з системою.
- Модератор — методист, що має можливість змінювати окремі дані (наприклад, інформацію про курси).
- Адміністратор — модератор із розширеними повноваженнями щодо управління системою.
- Модерація — процес управління користувачами та даними.
- Перегляд — отримання результатів роботи системи (зокрема, перегляд статей і матеріалів).

На цьому етапі проектування сутності, визначені у словнику даних, розглядаються як кандидати для формування класів майбутньої системи. Модель бізнес-сутностей наведена на рисунку 2.2.

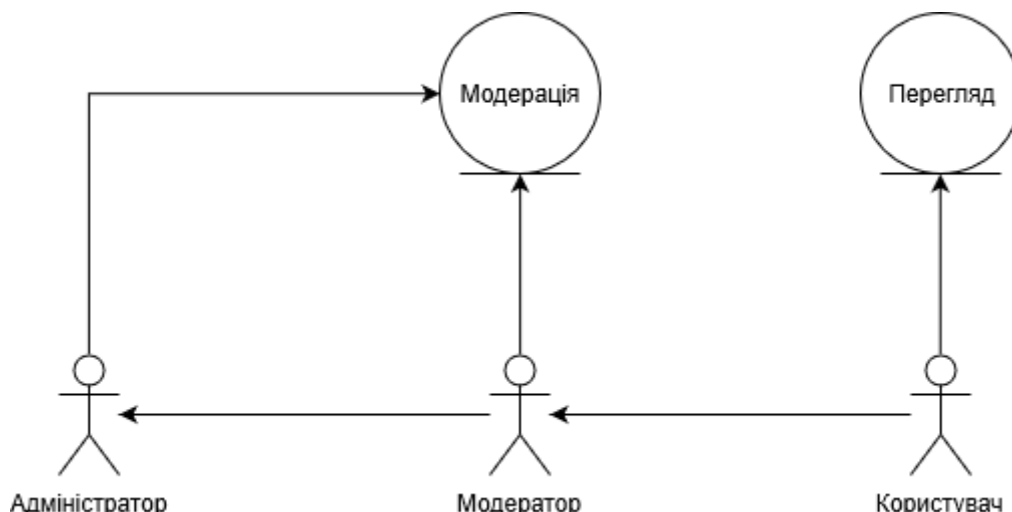


Рис. 2.2 - Діаграма бізнес-сутностей системи.

Іншим підходом до відображення внутрішнього стану системи є побудова логічної моделі бази даних. У низці випадків логічну модель також називають концептуальною моделлю даних. Концептуальна модель являє собою узагальнену модель предметної області, що включає перелік взаємопов'язаних понять, які використовуються для її опису, а також їх властивості,

характеристики, класифікаційні ознаки, типи, ситуації та закономірності перебігу процесів у межах цієї області.

Логічна модель даних подана на рисунку 2.3.

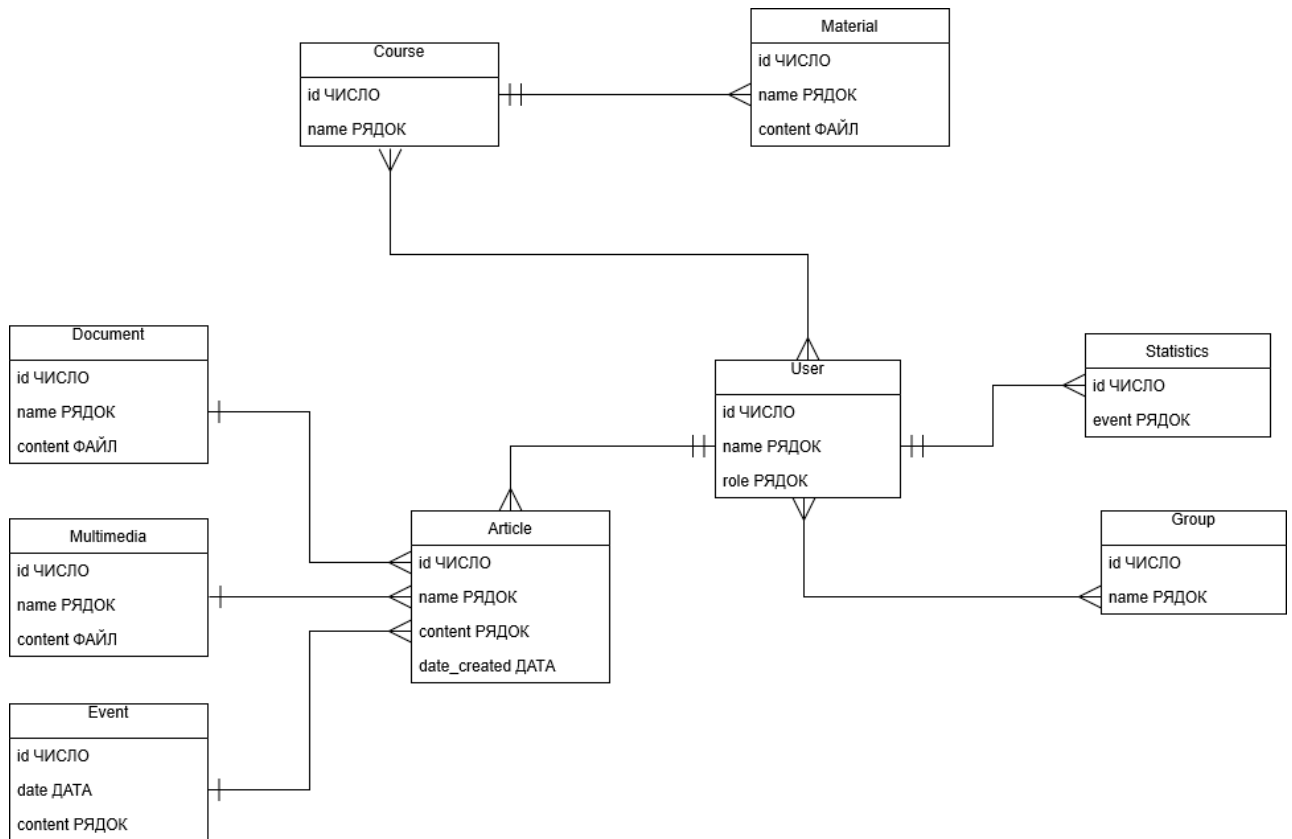


Рис. 2.3 – Логічна модель даних системи.

У наведеній діаграмі представлено такі сутності:

- User — відомості про користувача системи.
- Group — інформація про групу користувачів.
- Article — дані, подані у форматі статті.
- Document — інформація про документ, прикріплений до статті.
- Multimedia — відомості про мультимедійні матеріали, що супроводжують статтю.
- Event — інформація про заходи.
- Statistics — дані про дії користувачів у системі.
- Course — інформація про дистанційні курси.

- Material — відомості про матеріали, що входять до складу дистанційних курсів.

На цьому етапі проєктування визначені сутності словника даних розглядаються як кандидати для формування класів майбутньої системи.

2.3.2 Опис організації даних

Визначивши функціональні можливості системи, можна встановити потоки даних, що супроводжують процеси реалізації кожної з функцій. З цією метою будується діаграма потоків даних (DFD), наведена на рисунку 2.4.

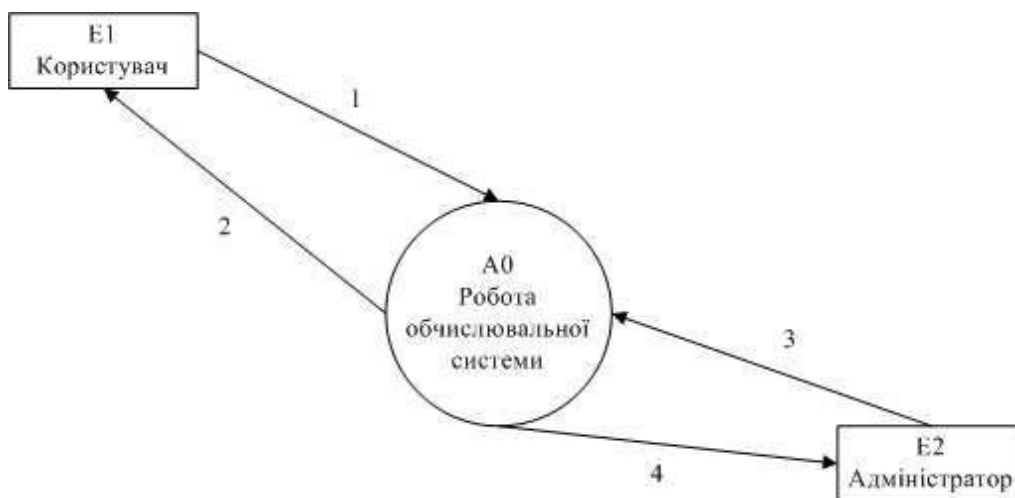


Рис. 2.4 – Діаграма потоків даних системи

Опис потоків даних:

1. Процес аутентифікації користувача (введення логіна та пароля), керування даними в системі, перегляд статей і матеріалів.
2. Формування статистики використання обчислювальних ресурсів за визначений період, отримання переліку даних, що містяться в системі, а також інформації про сторінки, відвідані користувачем.
3. Управління доступом користувачів до сервісів системи, налаштування системних параметрів, виділення додаткових ресурсів за потреби.
4. Формування статистики використання ресурсів системи всіма користувачами за певний період, отримання даних про поточне навантаження, підготовка звітів щодо роботи користувачів у системі.

Для опису процесів взаємодії користувачів із системою застосовуються діаграми UML. До канонічних діаграм UML належать:

- діаграма варіантів використання (use case diagram);
- діаграма класів (class diagram);
- діаграма діяльності (activity diagram);
- діаграма компонентів (component diagram);
- діаграма розгортання (deployment diagram).

Кожна з цих діаграм деталізує та конкретизує певний аспект моделі складної системи в термінах мови UML. Діаграма варіантів використання є найбільш загальною концептуальною моделлю, що слугує основою для побудови інших діаграм. Діаграма класів відображає статичну структуру системи та є логічною моделлю її побудови. Діаграми кооперації та послідовностей представляють динамічні аспекти функціонування системи. Діаграми станів і діяльності призначені для моделювання поведінки системи. Діаграми компонентів і розгортання описують фізичну структуру системи та належать до її фізичної моделі.

Діаграма варіантів використання, або прецедентів, зображена на рис. 2.5.



Рис. 2.5 – Діаграма прецедентів системи.

Декомпозиція варіанту використання «Керувати матеріалами» для сутності «Адміністратор» показана на рис. 2.6.



Рис. 2.6 – Діаграма декомпозиції прецеденту
«Керувати матеріалами».

Діаграма розгортання компонентів системи зображена на рис. 2.7.

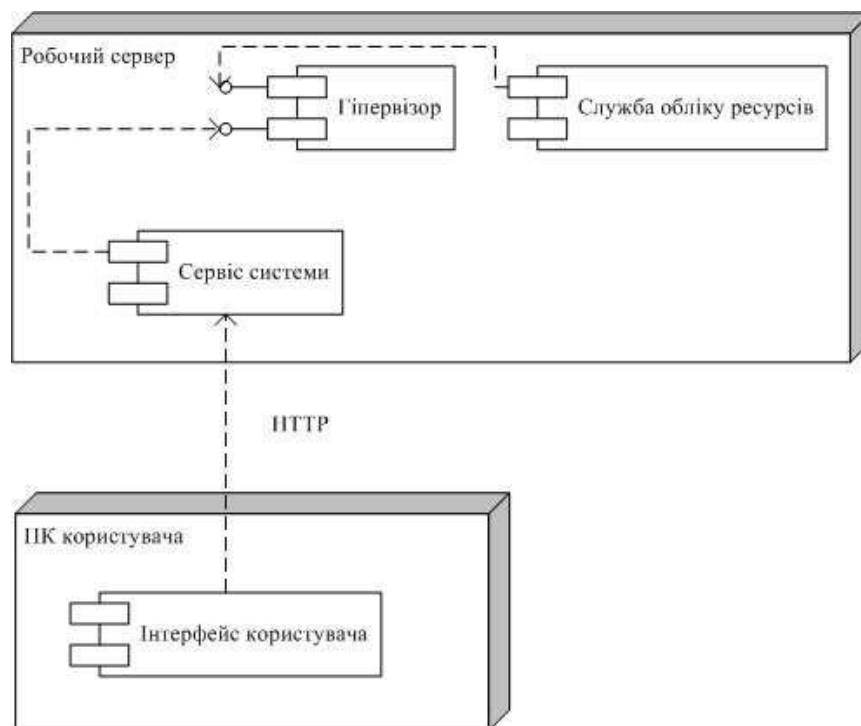


Рис. 2.7 – Діаграма компонентів системи

На рис. 2.8 показана діаграма діяльності користувача, який працює з системою.

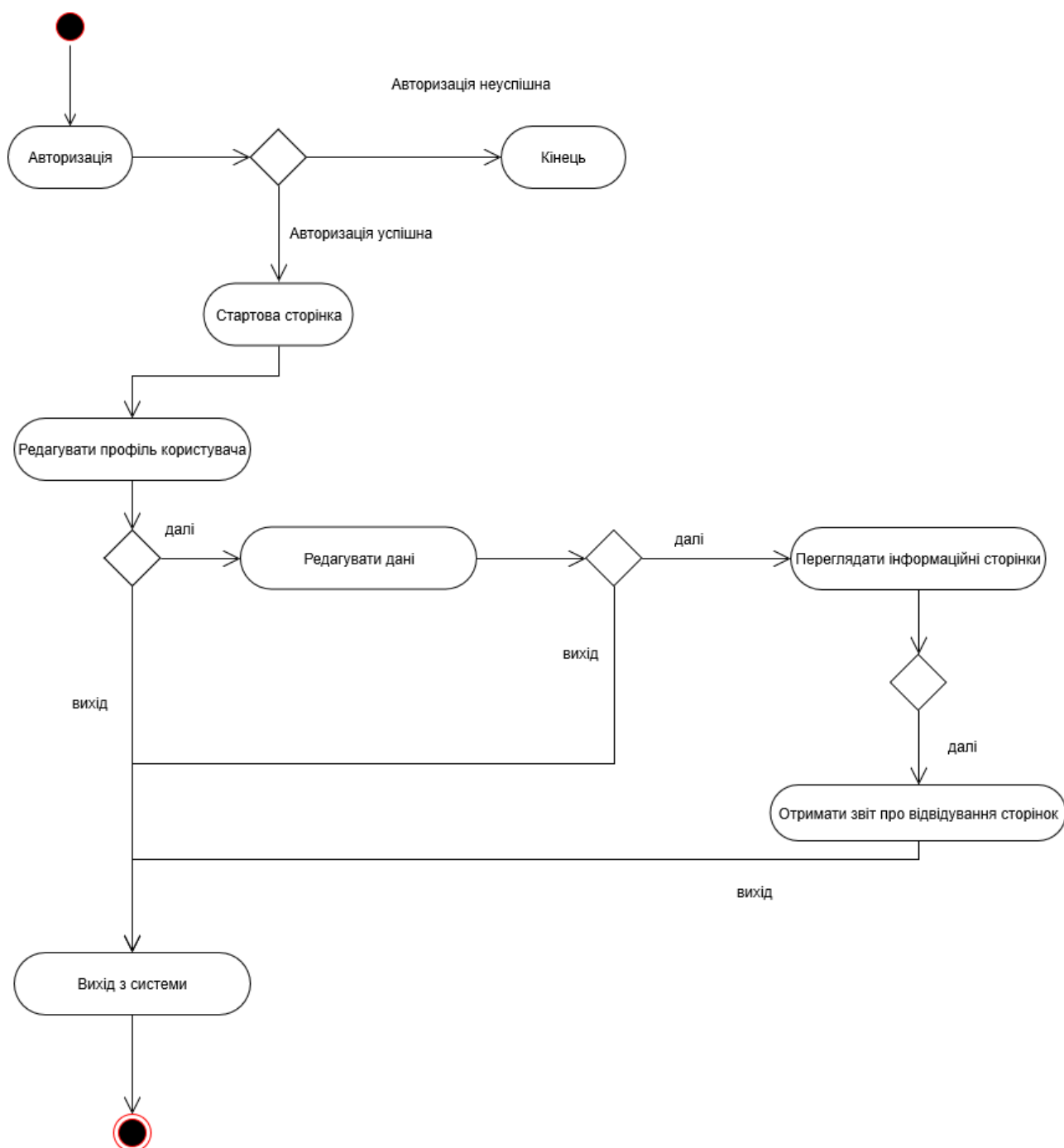


Рис. 2.8 – Діаграма діяльності

2.4 Рішення по технічному забезпеченню

2.4.1 Схема автоматизації

Рациональний вибір технічних засобів, необхідних для впровадження автоматизованої системи, значною мірою визначає успішність або неуспішність

усього проєкту. На підставі прийнятих загальносистемних рішень було визначено склад та організацію комплексу технічних засобів.

Вимоги до продуктивності

Обчислювальний комплекс має забезпечувати достатній рівень продуктивності для розв'язання передбачуваних завдань за умови мінімізації фінансових витрат на його придбання. Важливо також гарантувати мінімальний час реакції системи під час роботи в інтерактивних режимах, які становлять значну частину експлуатаційних сценаріїв. Це є необхідною умовою для забезпечення комфортної роботи користувачів.

Проектowana система не висуває підвищених вимог до обчислювальної потужності. Її можна віднести до класу облікових систем, у яких загальна продуктивність визначається швидкістю операцій введення та виведення даних за неповного завантаження центрального процесора. Таким чином, будь-яке сучасне апаратне рішення, представлене на ринку, задовольнятиме вимоги до продуктивності. Розподіл програмних компонентів системи дозволяє уникнути додаткового навантаження на наявні клієнтські робочі місця та не потребує їх модернізації.

Вимоги до надійності

Поставлене завдання передбачає стандартні вимоги до надійності апаратних засобів без необхідності забезпечення повної відмовостійкості. Основною вимогою є гарантування збереження даних. Для цього передбачено такі заходи:

- підключення сервера до електромережі через пристрій безперебійного живлення (UPS);
- використання на сервері бази даних апаратного RAID-контролера з можливістю віддзеркалення дисків, що є економічно доцільним рішенням.

Вимоги до експлуатаційних характеристик

На підставі наведених вимог можна зробити висновок про доцільність використання для сервера бази даних та клієнтських робочих місць стандартного

апаратного забезпечення на платформі x64. До сервера бази даних висувається традиційний набір вимог, характерний для подібних рішень.

2.4.2 Структурна схема комплексу обчислювальних засобів

Проектована програмна система реалізується за клієнт-серверною архітектурою, у зв'язку з чим апаратне забезпечення включатиме обчислювальну підсистему та мережу передавання даних. Принципова структура системи, подана у вигляді схеми розміщення, наведена на рисунку 2.9.

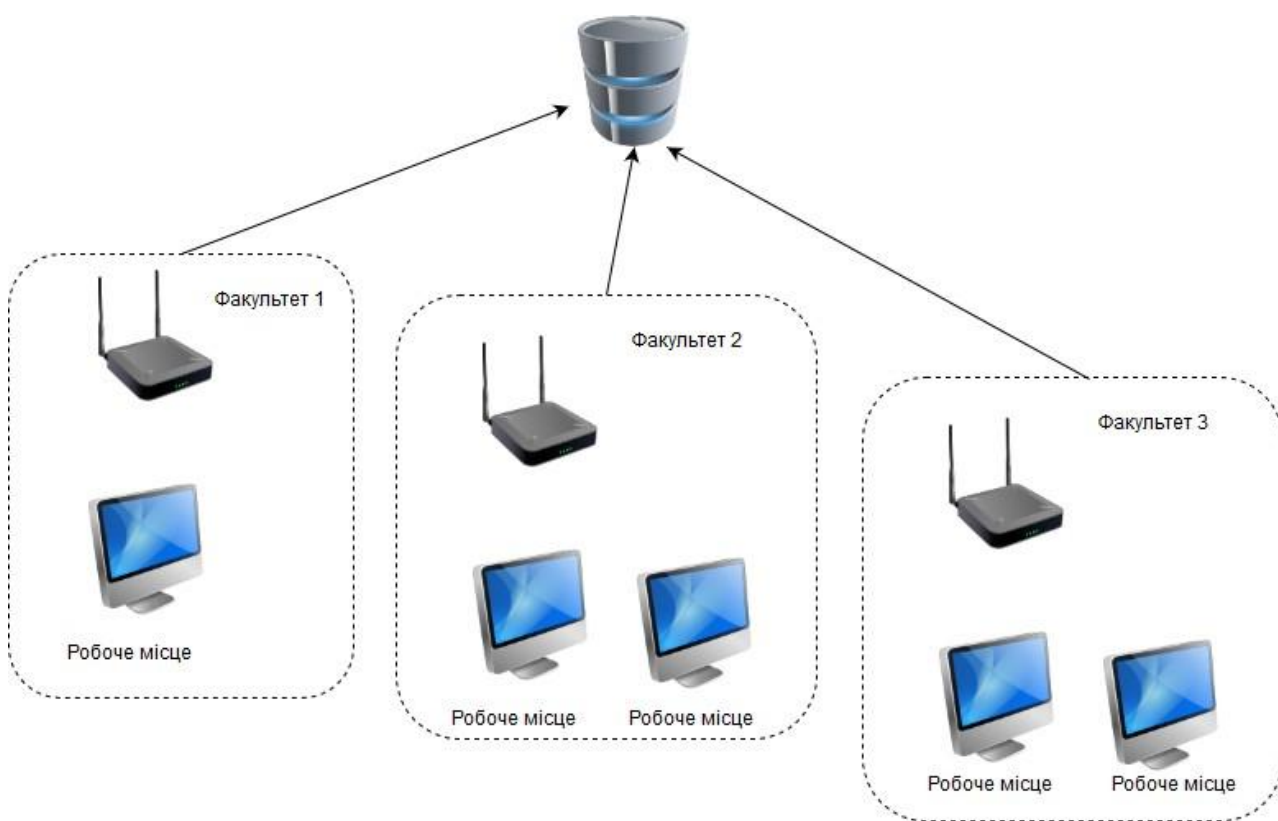


Рис. 2.9 - Схема розміщення комплексу технічних засобів.

2.4.3 Опис комплексу технічних засобів

За результатами вибору концепції системи пропонуються такі конфігурації технічних засобів, описаних у попередніх підрозділах (таблиці 2.1 – 2.5).

Таблиця 2.1 - Конфігурація сервера балансування навантаження.

Компонент	Специфікація
Центральний процесор	Intel Pentium G6950 2x2,8 GHz
Оперативна пам'ять, Гб	4, з можливістю розширення
Жорсткий диск, Гб	20

Таблиця 2.2 - Конфігурація робочих серверів.

Компонент	Специфікація
Центральний процесор	дво- або чотириядерний Intel Xeon
Оперативна пам'ять, Гб	8, з можливістю розширення
Жорсткий диск, Гб	4x200, hot swap, RAID 0,1,5,10

Таблиця 2.3 - Конфігурація серверів сховища даних.

Компонент	Специфікація
Центральний процесор	Intel Pentium G6950 2x2,8 GHz
Оперативна пам'ять, Гб	8, з можливістю розширення
Жорсткий диск, Гб	2x500, hot swap, RAID 0,1,10

Таблиця 2.4 - Конфігурація інструментів серверу управління системою.

Компонент	Специфікація
Центральний процесор	Intel Pentium G6950 2x2,8 GHz
Оперативна пам'ять, Гб	4, з можливістю розширення
Жорсткий диск, Гб	120

Таблиця 2.5 - Конфігурація мережевого середовища системи

Компонент	Специфікація
Технологія створення мережі	FastEthernet, GigabitEthernet 100/1000/10000 Мбіт/с
Мережеве середовище	оптоволокну, UTP5

2.5 Опис алгоритмів функціонування системи

Для уточнення алгоритмів функціонування розроблюваної системи застосовуються засоби об'єктно-орієнтованого аналізу та моделювання поведінки об'єктів.

У межах цього підходу розглядається як загальна поведінка системи (з метою визначення типів, характеру та послідовності взаємодій між класами), так і реалізація функціональних можливостей окремих класів.

Одним із ключових алгоритмів роботи системи є процес створення та публікації статей на веб-порталі. Діаграма класів підсистеми створення та публікації статей, сформована на основі аналізу відповідних сутностей та операцій над ними, наведена на рисунку 2.10.

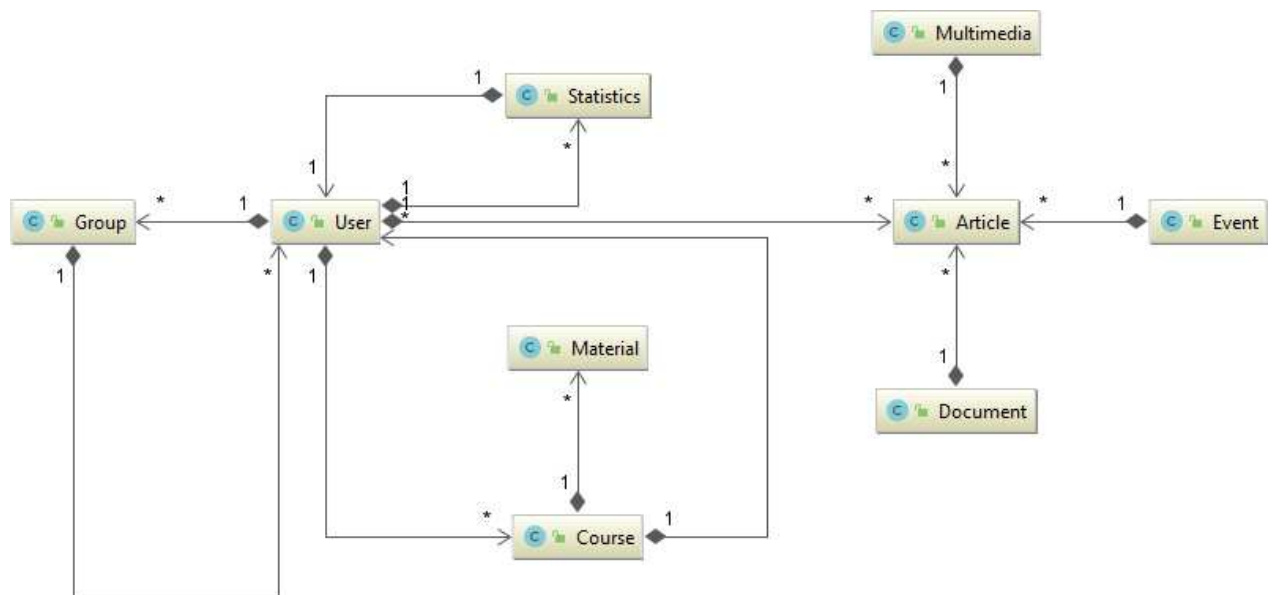


Рис. 2.10 - Діаграма класів підсистеми створення та публікації статей

Класи, що відповідають сутностям фізичної моделі даних, створено для подальшого використання в ORM-середовищі.

Іншим важливим алгоритмом функціонування системи є взаємодія з базою даних, зокрема процеси авторизації користувачів та збереження результатів їхньої роботи. Діаграма класів підсистеми авторизації та збереження даних, сформована на основі аналізу відповідних сутностей і операцій над ними, наведена на рисунку 2.11.

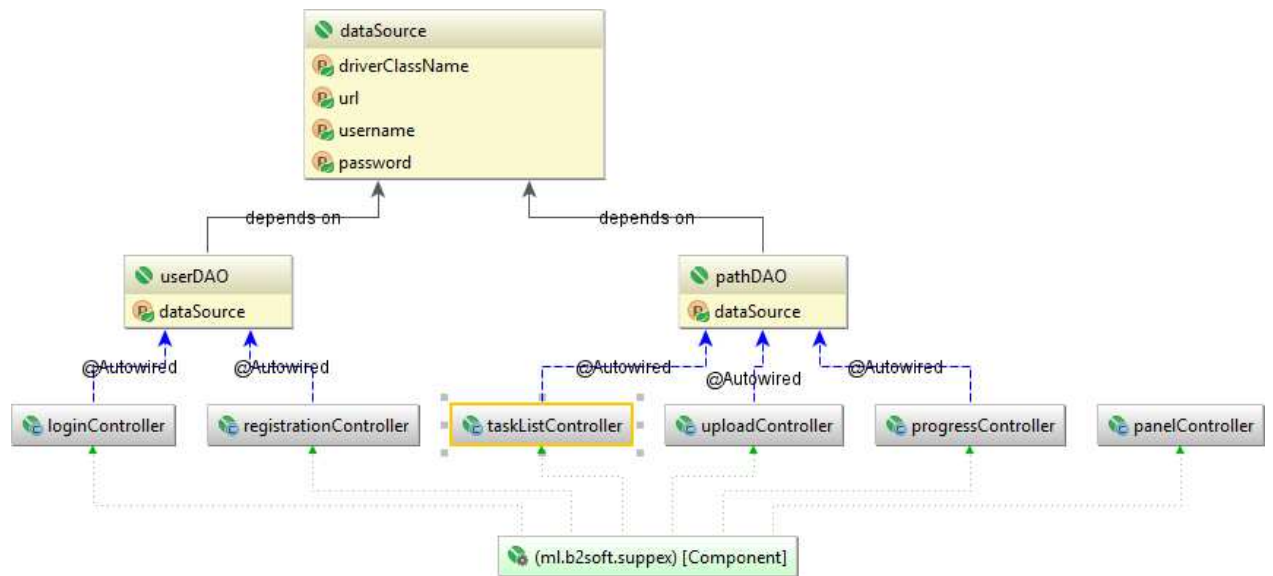


Рис. 2.11 - Діаграма класів підсистеми авторизації та збереження даних

dataSource – клас моделі даних для збереження;

userDAO – клас, який є моделлю даних про користувача для збереження у БД;

pathDAO – клас, який є моделлю даних про результати роботи користувачів у БД;

loginController, registrationController – класи для управління процесами авторизації та реєстрації;

taskListController – клас для управління завданнями системи;

uploadController – клас для управління отриманням даних;

progressController, panelController – класи для відображення інформації користувачам.

Розглянуті вище діаграми класів відображають логічну модель статичної структури системи. На цих діаграмах подано виключно структурні взаємозв'язки, які не залежать від часових характеристик реакції системи на зовнішні події. На відміну від такого статичного подання, загальний алгоритм поведінки системи може бути описаний за допомогою діаграми станів (statechart diagram).

Опис поведінки системи є важливим для визначення характеру взаємодії між її компонентами та уточнення причинно-наслідкових зв'язків, що зумовлюють виконання певних дій. Алгоритм поведінки системи, поданий у нотації діаграми станів, наведено на рисунку 2.12.

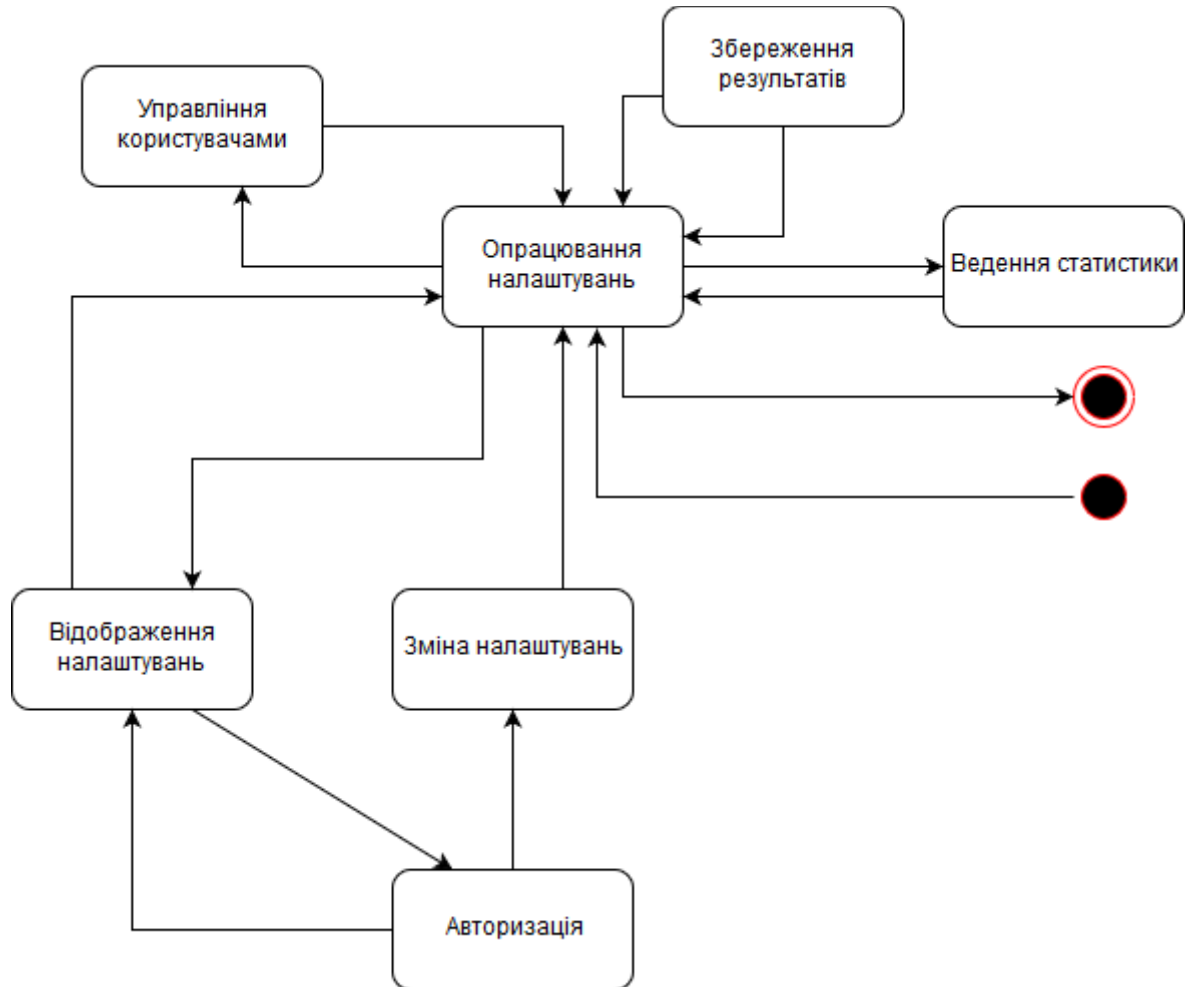


Рис. 2.12 - Загальний алгоритм поведінки системи.

2.6 Рішення по програмному забезпеченню

2.6.1 Структура і функції частин програмного забезпечення

Функціональність і структура проєктованої автоматизованої системи, поряд із моделлю її поведінки, визначаються також моделлю станів. Формування моделі станів здійснюється шляхом уточнення моделі бізнес-сутностей, побудованої на основі вимог користувачів до функцій системи та оброблюваних даних. Вхідними даними для цього етапу проєктування є моделі сутностей, а

результатом — модель специфікації проєктних конструкцій програмного забезпечення.

Відповідно до прийнятого в цій роботі підходу об'єктно-орієнтованого аналізу та проєктування, подальша специфікація проєктних рішень виконується у вигляді діаграм класів. Класи визначають стан програмної системи (через значення атрибутів) та значною мірою її поведінку (через операції або методи).

Сформульовані користувацькі вимоги на цьому етапі розглядаються як обмеження. Поряд із моделлю сутностей для проєктування класів використовуються раніше побудовані діаграми прецедентів (рисунок 2.1), відповідно до рекомендацій методології Rational Unified Process. Модель класів узгоджується зі словником даних, сформованим на основі аналізу прецедентів і таким чином є текстовим аналогом моделі сутностей.

Програмне забезпечення розроблюваної системи складатиметься з програмного комплексу, розміщеного на серверах. Для роботи користувача не передбачається встановлення додаткового програмного забезпечення — взаємодія здійснюється через веб-браузер.

Модуль авторизації забезпечує вхід користувачів до системи.

Модуль управління списком користувачів призначений для ведення облікових даних, реєстрації нових користувачів, зміни персональної інформації та видалення профілів. Адміністратор за допомогою цього модуля може керувати правами доступу.

Модуль управління контентом надає модераторам можливість завантажувати дані до системи. Адміністратор або модератор може додавати, редагувати та видаляти інформаційні сторінки й матеріали.

Модуль обліку відвідувань забезпечує збирання статистики перегляду інформаційних сторінок користувачами.

Модуль журналів фіксує всі дії користувачів у системі. Ці дані є важливими для діагностики збоїв, оскільки дають змогу встановити причини виникнення аварійних ситуацій. Модуль звітів дозволяє користувачам і адміністраторам переглядати статистику роботи системи за вибраний період. Користувач може

переглядати статистику використання власних ресурсів, тоді як адміністратор отримує доступ як до звітів про використання ресурсів, так і до даних журналів подій.

Модуль технічної підтримки забезпечує комунікацію між користувачем і адміністратором у разі виникнення проблем або незадоволеності роботою системи.

2.6.2 Фізична модель бази даних

Для забезпечення можливості зберігання даних у реляційній базі даних на основі моделі класів було сформовано фізичну модель даних. На цьому етапі серед усіх класів було виокремлено постійні класи, тобто такі, об'єкти яких повинні зберігати свої значення не лише під час роботи програми. Після цього здійснено відображення постійних класів на таблиці фізичної моделі даних: атрибути класів трансформовано у поля реляційних таблиць, а асоціації між класами — у зв'язки між таблицями з використанням механізму зовнішніх ключів. Для забезпечення цілісності даних у межах кожної сутності застосовано первинні ключі.

Обрана система керування базами даних не підтримує зберігання ієрархій спадкування класів, тому під час побудови фізичної моделі даних ієрархії було відображено за принципом «одна таблиця — один клас». Метод згортання типів у супертип, за якого ієрархія представлена однією таблицею, було відхилено через значну надмірність даних, що виникала б у результаті.

Розроблену фізичну модель даних наведено на рисунку 2.13.

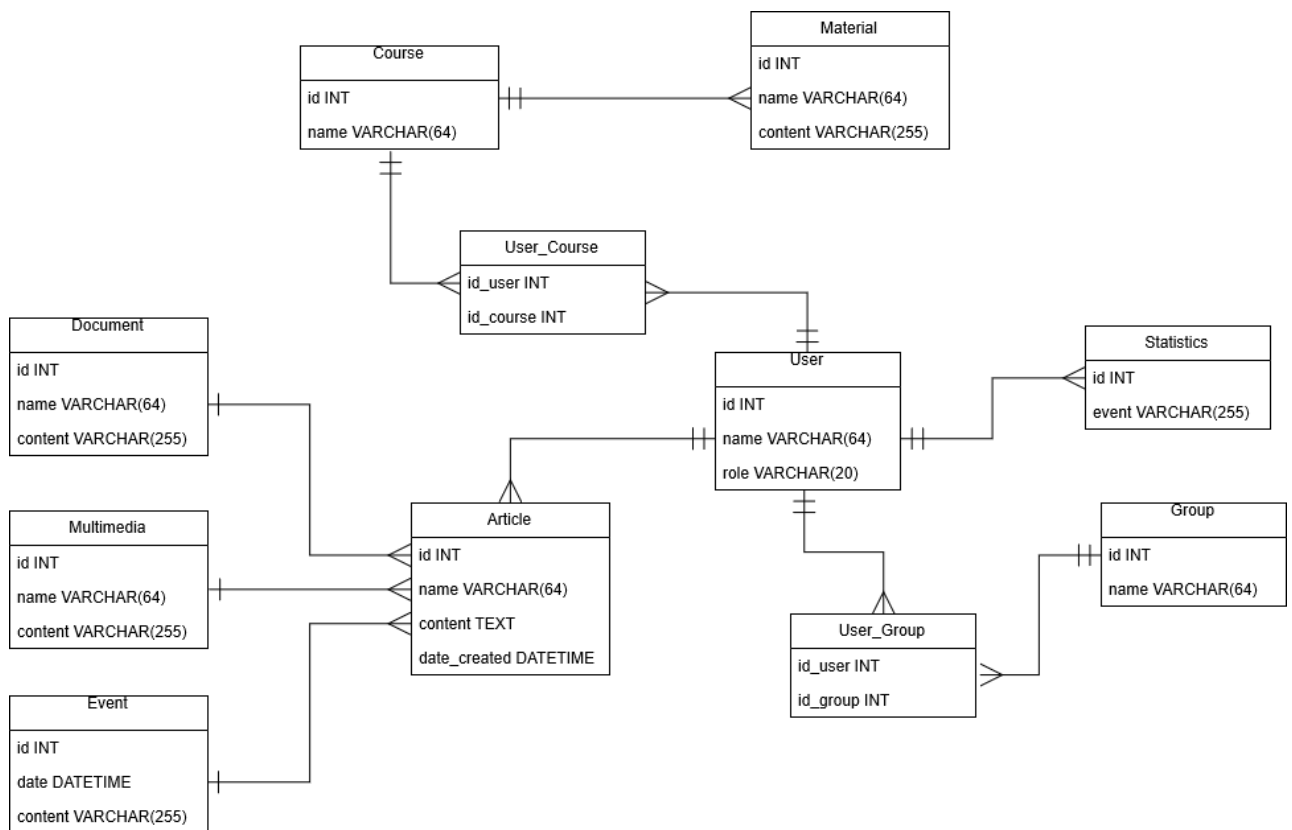


Рис. 2.13 Фізична модель бази даних

2.6.3 Методи і засоби розробки програмного забезпечення

Під час аналізу та проєктування Системи застосовувався об'єктно-орієнтований підхід (ООП), що зумовлює необхідність використання відповідних принципів і підходів під час розроблення програмного забезпечення. Основними принципами ООП є спадкування, інкапсуляція та поліморфізм.

Застосування об'єктно-орієнтованого підходу у процесі розроблення забезпечує такі переваги:

- використання логічних моделей, отриманих на етапах аналізу та проєктування, для первинної генерації програмного коду;
- скорочення часу розробки завдяки повторному використанню програмних компонентів;
- можливість налаштування та тестування системи на основі поведінки об'єктів без необхідності знання їх внутрішньої реалізації;
- повноцінне використання можливостей обраних засобів розроблення програмного забезпечення.

Для проєктування та аналізу Системи застосовувалося UML-середовище з відкритим кодом ArgoUML, розроблене спільнотою Tigris Software (Tigris.org). UML є де-факто стандартом галузі у сфері моделювання програмних систем. ArgoUML використовується на всіх етапах проєктування — від аналізу до автоматичної генерації коду — та підтримує роботу з усіма основними інструментами UML.

Основним засобом розроблення програмного забезпечення обрано мову програмування Java. У проєкті застосовуються такі технології: Servlets, EJB3, JSPx та JSF. Як середовище розроблення використано IntelliJ IDEA 2025.

IntelliJ IDEA — комерційне інтегроване середовище розроблення від компанії JetBrains, що підтримує широкий спектр мов програмування (Java, Python, Scala, PHP тощо). Продукт доступний у двох версіях: безкоштовній Community Edition та повнофункціональній комерційній Ultimate Edition. Community-версія поширюється за ліцензією Apache 2.0 та має збірки для Linux, macOS і Windows.

До складу IntelliJ IDEA включено напрацювання, створені у співпраці з компанією Google, яка використала це середовище як основу для Android Studio. Завдяки цьому значно розширено можливості розроблення застосунків для платформи Android.

Community Edition підтримує плагіни для тестування (TestNG, JUnit), систем контролю версій (CVS, Subversion, Mercurial, Git), засобів складання (Maven, Ant, Gradle), а також мов програмування Java, Scala, Clojure, Groovy та Dart. Середовище містить інструменти для розроблення Android-застосунків, візуальний конструктор інтерфейсів Swing UI Designer, XML-редактор, редактор регулярних виразів, засоби перевірки коректності коду, модуль керування завданнями та інструменти імпорту/експорту проєктів з Eclipse. Передбачено інтеграцію з системами відстеження помилок (JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse).

Комерційна версія Ultimate Edition додатково підтримує інші мови програмування (PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL),

технології Java EE, UML-діаграми, засоби аналізу покриття коду, а також роботу з фреймворками (Rails, Grails, GWT, Spring, Play Framework, Hibernate) та інтеграцію з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

У проєкті також використано бібліотеку XMLBeans, розроблену Apache. XMLBeans забезпечує доступ до XML-даних у вигляді Java-об'єктів, використовуючи XML-схеми для генерації відповідних інтерфейсів і класів, що дозволяє природно працювати з XML-структурами у Java-середовищі.

Для підвищення швидкості розроблення та надійності системи застосовано архітектурний шаблон MVC (Model–View–Controller). Цей шаблон передбачає поділ системи на три взаємопов'язані компоненти:

- Model — відповідає за зберігання даних, логіку та правила роботи застосунку;
- View — забезпечує представлення даних користувачеві;
- Controller — обробляє вхідні дії користувача та передає відповідні команди моделі або вигляду.

Використання MVC забезпечує гнучкість архітектури, спрощує внесення змін, сприяє повторному використанню компонентів та підвищує структурованість великих систем. Однією з реалізацій цього шаблону в середовищі Java є Spring MVC — фреймворк з відкритим кодом, що підтримує інверсію керування та широко застосовується для створення веб-застосунків на платформі Java EE. Spring Framework не нав'язує конкретної моделі програмування та часто використовується як альтернатива або доповнення до моделі Enterprise JavaBeans (EJB).

Для функціонування системи необхідна наявність Java-орієнтованого веб-сервера, встановленого на серверному обладнанні. Можуть бути використані такі серверні платформи:

- JBoss
- Payara
- GlassFish
- Tomcat

На сервері баз даних має бути встановлена серверна система керування базами даних (СКБД). Необхідною є сучасна реляційна СКБД, що забезпечує високу продуктивність, здатна працювати з великими обсягами даних та підтримує стандарт SQL-92. Крім того, СКБД повинна функціонувати під керуванням операційної системи Windows. До серверних СКБД, що відповідають зазначеним вимогам, належать:

- MySQL
- MS SQL
- PostgreSQL

На клієнтських робочих станціях може використовуватися будь-яка операційна система, що підтримує встановлення віддаленого мережевого з'єднання та має графічний інтерфейс. Рекомендованими є MS Windows, Linux та інші сучасні ОС.

Програмне забезпечення, з яким працює користувач, являє собою веб-оглядач із підтримкою стандартів HTML 5.0, JavaScript та CSS 3 (наприклад, MS Edge, Firefox, Opera, Chrome, Safari тощо).

Висновок

У межах розділу було здійснено комплексне обґрунтування архітектури, функціональності та технічних рішень для розроблюваної автоматизованої системи. На основі аналізу бізнес-процесів сформовано модель сутностей, логічну та фізичну моделі даних, а також побудовано UML-діаграми, що відображають структурні та поведінкові аспекти системи. Використання об'єктно-орієнтованого підходу забезпечило узгодженість між моделями та програмною реалізацією, а також створило передумови для масштабованості та подальшого розвитку системи.

Було визначено вимоги до продуктивності, надійності та експлуатаційних характеристик, що дозволило сформувати оптимальну конфігурацію апаратних і програмних засобів. Обґрунтовано вибір клієнт-серверної архітектури, сучасних серверних СКБД, Java-орієнтованих технологій та інструментів розроблення.

Окрему увагу приділено модульній структурі програмного забезпечення, що включає підсистеми авторизації, управління користувачами, контентом, журналами подій, статистикою та технічною підтримкою.

Таким чином, у розділі сформовано цілісну концепцію побудови системи, яка охоплює логічне, фізичне та програмно-технічне проєктування. Отримані результати створюють надійну основу для подальшої реалізації, тестування та впровадження автоматизованої системи в експлуатацію.

РОЗДІЛ 3. РОЗРОБКА РОБОЧОЇ ДОКУМЕНТАЦІЇ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ WEB-ПОРТАЛУ МОІППО

Система управління контентом веб-порталу МОІППО розроблена з метою підвищення ефективності роботи методистів навчального закладу. Автоматизовані функції системи спрямовані на обробку та управління інформаційними ресурсами.

До основних функцій, що підлягають автоматизації, належать:

- надання користувачам можливості переглядати інформаційні матеріали (новини, статті, ресурси);
- забезпечення адміністраторам системи інструментів для управління даними, що зберігаються у системі;
- інші вимоги не є визначальними та можуть уточнюватися або змінюватися в процесі розроблення чи експлуатації.

Під час впровадження програмного забезпечення визначено такі ключові цілі:

- забезпечення простого та інтуїтивно зрозумілого інтерфейсу користувача;
- можливість доступу до системи через мережу Internet з будь-якого комп'ютера;
- підтримка багатокористувацького режиму роботи;
- висока масштабованість системи;
- забезпечення високої продуктивності;
- надійність функціонування;
- гарантування конфіденційності інформації.

Для ефективного використання програмного забезпечення необхідно впровадити відповідні організаційні та технічні заходи, визначені у Розділі 2. З огляду на зростання складності програмного забезпечення, важливим є підвищення кваліфікації співробітників, які працюватимуть із системою. Експлуатація програмного забезпечення має здійснюватися відповідно до

документів «Технічне завдання» (Додаток А) та «Програма та методика випробувань» (Додаток Б).

Висновок

У цьому розділі було визначено призначення, функціональні можливості та ключові вимоги до системи управління контентом веб-порталу МОППО. Система орієнтована на підвищення ефективності роботи методистів шляхом автоматизації процесів обробки, зберігання та представлення інформаційних матеріалів. Сформульовано основні функції, що забезпечують доступ користувачів до контенту, можливість адміністрування даних та підтримку гнучкого розширення функціональності.

Окреслено цілі впровадження програмного забезпечення, серед яких — забезпечення зручності користування, доступності через мережу Internet, багатокористувацького режиму, масштабованості, продуктивності, надійності та конфіденційності даних. Підкреслено важливість організаційних і технічних заходів, необхідних для ефективної експлуатації системи, а також потребу у відповідному підвищенні кваліфікації персоналу.

Експлуатація системи має здійснюватися відповідно до нормативної документації, зокрема «Технічного завдання» та «Програми та методики випробувань», що забезпечує контроль якості та відповідність програмного забезпечення встановленим вимогам.

РОЗДІЛ 4. РОЗРАХУНОК ЕКОНОМІЧНОГО ЕФЕКТУ ВІД ВПРОВАДЖЕННЯ ІС

4.1 Вступ до проблеми

Інформаційні системи сьогодні широко застосовуються в організаціях України незалежно від форми власності. Їх використання постійно зростає, а самі системи еволюціонують відповідно до вимог користувачів та технологічного розвитку.

Системи управління контентом не є принципово новою технологією, однак як концепція створення інформаційних систем і надання ІТ-послуг вони перебувають на етапі активного становлення. Очікується, що зі зростанням їх функціональних можливостей дедалі більше підприємств використовуватимуть такі системи для формування власної інформаційної інфраструктури та розв'язання прикладних обчислювальних завдань.

Розроблення інформаційної системи потребує одноразових витрат, пов'язаних зі створенням програмного забезпечення та придбанням необхідних технічних засобів, а також поточних витрат, що забезпечують функціонування її підсистем. Економічний ефект від експлуатації системи визначається з урахуванням витрат на її підтримку, а співвідношення отриманого ефекту до витрат на створення характеризує економічну ефективність капітальних вкладень. Розрахунок економічних показників здійснюється на основі чинних на момент оцінювання цін, тарифів і ставок заробітної плати.

4.2 Розрахунок вартості створення інформаційної системи управління контентом web-порталу

Загальні витрати (на проектування, реалізацію та впровадження системи)

$$З = C_{п-р} + C_{тз} + C_{пз} + C_{впр},$$

де $C_{п-р}$ – витрати на проектування і реалізацію;

$C_{тз}$ – витрати на придбання або модернізацію обчислювальної техніки, периферійних пристроїв, засобів зв'язку, допоміжного устаткування та

оргтехніки (з урахуванням витрат на транспортування, монтаж, налагодження і запуск), виробничо-господарського інвентарю;

$C_{пз}$ – витрати на придбання та інсталяцію програмного забезпечення (ПЗ), необхідного для розробки;

$C_{впр}$ – витрати на впровадження системи (навчання персоналу тощо).

Витрати на проектування та реалізацію розраховуються за формулою

$$C_{п-р} = C_з + C_{від} + C_м + C_н,$$

де $C_з$ – заробітна плата розробників;

$C_{від}$ – відрахування на соціальні заходи (відповідно до чинного законодавства (становить 47,5% від фонду оплати праці);

$C_м$ - експлуатація та амортизація, $C_м = Z_{експл} + Z_{амор}$;

$C_н$ – накладні витрати у розмірі 50–150% від витрат на оплату праці (включають оплату праці управлінського персоналу з нарахуваннями, службові відрядження, консультаційно-інформаційні послуги, ремонт і технічне обслуговування основних фондів, за винятком персональних комп'ютерів).

1) Тривалість розроблення підсистеми становить 3 місяці (експертна оцінка, заснована на досвіді створення аналогічних систем). Місячна заробітна плата розробника становить 12 000 грн. Кількість розробників – одна особа.

$$C_з = 1 * 12000 * 3 = 36000 \text{ грн.}$$

$$C_{від} = 36000 * 0,475 = 17100 \text{ грн.}$$

2) Витрати на амортизацію ЕОМ, на яких виконується розробка:

$$Z_{амор} = (Ц_{еом} / T_{сл}) * (T_{разр} / 12),$$

де $Ц_{еом}$ = 20000 грн. – балансова вартість ЕОМ;

$T_{сл}$ = 5 років – термін служби ЕОМ;

$T_{разр}$ = 0,5 року – час, необхідний для розробки системи, виражений в місяцях, і тому ділиться на 12, щоб одержати число років.

$$Z_{амор} = (20000 / 5) * 0,5 = 2000 \text{ (грн.)};$$

3) витрати на експлуатацію ЕОМ, на якій виконується розробка, полягають в оплаті споживаної нею електричної енергії:

$$Z_{\text{експл}} = P_{\text{ЕОМ}} * T_{\text{разр}} * N_{\text{рд}} * N_{\text{рч}} * E_{\text{ст}},$$

де $P_{\text{ЕОМ}} = 0,4$ кВт/год – потужність ЕОМ;

$T_{\text{разр}} = 3$ місяці – тривалість розробки;

$N_{\text{рд}} = 21$ день – число робочих днів у місяці;

$N_{\text{рч}} = 8$ годин – число годин у робочому дні;

$E_{\text{ст}} = 0.9$ грн – вартість 1 кВт електроенергії

$$Z_{\text{експл}} = 0,4 * 3 * 21 * 8 * 0,9 = 181,44 \text{ (грн.)}$$

$$C_{\text{м}} = 2000 + 181,44 = 2181,44 \text{ (грн.)}$$

$$C_{\text{н}} = 36000 * 0,5 = 18000 \text{ (грн.)}$$

$$C_{\text{п-р}} = 36000 + 17100 + 2181,44 + 18000 = 73281,44 \text{ (грн.)}$$

Витрати на програмне забезпечення не враховуються, оскільки передбачається використання безкоштовних рішень з відкритим вихідним кодом. Витрати на апаратне забезпечення становлять 18 594 грн (придбання сервера, комутаційного обладнання та матеріалів). До витрат на впровадження належать витрати на навчання персоналу, монтаж і налаштування мережі, придбання обладнання та програмного забезпечення.

Витрати на впровадження системи наведено в таблиці 4.1.

Таблиця 4.1 – Витрати на впровадження системи.

Найменування інвестицій	Сума, грн
Вартість устаткування	18594,00
Вартість програмного забезпечення	-
Вартість підготовки кадрів	500,00
Монтаж і налаштування мереж	300,00
Разом на впровадження системи	19394,00

Узагальнюючи проведені розрахунки, можна зазначити, що сумарні витрати на проєктування, реалізацію та впровадження системи становлять:

$$З = C_{п-р} + C_{тз} + C_{пз} + C_{впр} = 73281,44 + 19394 = 92675,44 \text{ (грн.)}$$

Витрати на створення та експлуатацію системи в перший рік експлуатації

$$З_{се} = З + З_{ер},$$

де $З_{ер}$ – експлуатаційні витрати на технічне забезпечення за рік.

$$З_{ер} = A_{об} + З_{м} + З_{е},$$

де $A_{об}$ – амортизаційні відрахування на устаткування;

$З_{м}$ – витрати на основні і допоміжні матеріали;

$З_{е}$ – витрати на електроенергію.

Амортизаційні відрахування на устаткування становлять 60% його балансової вартості на рік відповідно до чинного законодавства

$$A_{об} = 19394 * 0,6 = 11636,4 \text{ (грн.)}$$

У масштабах підприємства річні витрати на основні та допоміжні матеріали (зокрема носії даних, папір тощо) визначаються на рівні 3% від вартості основного устаткування

$$З_{м} = 19394 * 0,03 = 581,82 \text{ (грн.)}$$

Річний обсяг робіт ПК у нормо-годинах

$$\Phi_{м} = 253,3 * T_{с},$$

де $T_{с}$ – середнє місячне завантаження устаткування (близько 5 годин);

253,3 – середня кількість робочих днів у році.

$$\Phi_{м} = 253,3 * 5 = 1266,5 \text{ (годин.)}$$

Витрати на електроенергію під час 1266,5 годин роботи устаткування в рік при потужності ЕОМ 0,4 кВт/год та ціні 0,9 грн за 1 кВт електроенергії складуть

$$З_{е} = 1266,5 * 0,9 * 0,4 = 455,94 \text{ (грн.)}$$

Експлуатаційні витрати на технічне забезпечення за рік

$$З_{ер} = 11636,4 + 581,82 + 455,94 = 12674,16 \text{ (грн.)}$$

Отже, у перший рік витрати на створення та експлуатацію системи складуть

$$З_{се} = 92675,44 + 12674,16 = 105349,6 \text{ (грн.)}$$

4.3 Розрахунок економічної ефективності

Економічна ефективність використання інформаційної системи полягає у наступному:

- зменшення часу на обслуговування системи за рахунок її відмовостійкості та автоматичного масштабування;
- можливість надання користувачам інформації в оренду.

Це зменшення відбудеться за рахунок зменшення складності користування системою, збільшення надійності, стабільності та рівня безпеки. Визначимо, що це зменшення складе 50% від сучасного.

Однак, не піддається прямій грошовій оцінці зменшення кількості помилкових та необережних рішень, підвищення оперативності керування, поліпшення організації роботи та своєчасне отримання необхідної інформації про пріоритетність варіантів рішень тощо.

Використання даної системи дозволяє вивільнити 0,1 робочого часу. Так як з системою працює біля 50 користувачів, то система умовно вивільнить 5 робітників.

Визначимо пряму економічну ефективність, ґрунтуючись на тому, що впровадження системи вивільнить 5 робітників. Річна платня за рахунок вивільнення робочого часу складе

$$\Delta C_n = 5000 * 5 * 12 - 12674,16 = 287325,84 \text{ (грн.)}$$

Річний економічний ефект розраховується за наступною формулою:

$$E_{\text{год}} = \Delta C_n - E_n * k,$$

де E_n – нормативний коефіцієнт економічної ефективності капітальних вкладень у галузь і для обчислювальної техніки приймається 0,5;

k – додаткові капітальні вкладення з урахуванням витрат на проектування, створення і функціонування системи.

$$E_{\text{год}} = 287325,84 - 0,5 * 105349,6 = 234651,04 \text{ (грн.)}$$

Строк окупності підсистеми

$$T = k / \Delta C_n$$

Тобто

$$T = 105349,6 / 287325,84 = 0,37 \text{ (року)}$$

Узагальнюючи результати розрахунків, можна стверджувати, що витрати на створення розподіленої інформаційної системи повністю окупляться протягом 0,37 року.

Висновок

У результаті проведених розрахунків встановлено, що витрати на створення системи обліку робочого часу, які включають проєктування, реалізацію та впровадження, становлять 92 675,44 грн. Щорічні експлуатаційні витрати оцінюються у 12 674,16 грн. Таким чином, сумарні витрати на модернізацію та експлуатацію системи в перший рік її функціонування становитимуть 105 349,6 грн.

Умовне вивільнення трудових ресурсів еквівалентне п'яти фахівцям. Річний фонд оплати праці, що вивільняється завдяки впровадженню системи, становить 287 325,84 грн. За таких умов витрати на створення системи повністю окупляться протягом 0,37 року.

Отримані значення річного економічного ефекту та строку окупності свідчать про економічну доцільність і обґрунтованість розроблення та впровадження інформаційної системи. Варто також зазначити, що економічну ефективність можна підвищити шляхом пропонування системи стороннім користувачам як програмного продукту. Це дозволить скоротити строк окупності та забезпечити додатковий прибуток від реалізації програмного забезпечення.

5. ОХОРОНА ПРАЦІ

Вступ

Охорона праці є невід'ємною складовою організації трудової діяльності в будь-якій сфері. Особливої актуальності питання забезпечення безпечних і здорових умов праці набувають у галузі інформаційних технологій, де фахівці значну частину робочого часу здійснюють професійну діяльність за персональними комп'ютерами. Розроблення інформаційних систем, як один із ключових напрямів діяльності ІТ-спеціалістів, передбачає тривалу роботу з комп'ютерним обладнанням, моніторами та іншими технічними засобами.

Порушення вимог охорони праці під час роботи з комп'ютерною технікою може спричинити розвиток низки професійно зумовлених захворювань, зокрема синдрому зап'ясткового каналу, комп'ютерного зорового синдрому, захворювань опорно-рухового апарату тощо. Крім того, неналежна організація робочого місця, недостатній рівень освітлення, невідповідні параметри мікроклімату у виробничих приміщеннях здатні негативно позначатися на самопочутті та працездатності розробників ІС.

У зв'язку з цим дотримання вимог охорони праці та створення належних умов для збереження здоров'я і безпеки фахівців, задіяних у процесі розроблення інформаційних систем, є одним із пріоритетних завдань керівників підприємств та організацій. Забезпечення безпечних умов праці не лише сприяє поліпшенню фізичного та психологічного стану працівників і зниженню ризиків виникнення професійних захворювань, але й позитивно впливає на продуктивність праці та якість кінцевого програмного продукту.

У цьому розділі дипломної роботи, присвяченої розробленню інформаційної системи підтримки волонтерської діяльності, буде розглянуто

основні аспекти охорони праці в галузі інформаційних технологій, зокрема у контексті проєктування та розроблення ІС. Особливу увагу буде приділено аналізу нормативно-правових актів, що регламентують питання охорони праці, вимогам до організації робочого місця розробника, забезпеченню безпечних умов праці та мінімізації впливу шкідливих виробничих факторів, регламентації режиму праці та відпочинку, а також специфіці застосування заходів електробезпеки.

5.1. Нормативно-правові акти, що регулюють питання охорони праці при роботі з персональними комп'ютерами та іншими пристроями для розробки вебзастосунків

Питання охорони праці під час роботи з персональними комп'ютерами (ПК) та іншими пристроями, що застосовуються для розробки інформаційних систем, регулюються низкою нормативно-правових актів України. Основним документом, що визначає вимоги безпеки й захисту здоров'я працівників при роботі з екранними пристроями, є Наказ Міністерства соціальної політики України «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» від 14.02.2018 № 207. Дія цього наказу поширюється на всіх суб'єктів господарювання незалежно від форми власності та організаційно-правової структури, які здійснюють діяльність, пов'язану із застосуванням екранних пристроїв.

Відповідно до зазначеного документа, роботодавець зобов'язаний забезпечити відповідність робочих місць працівників ергономічним, організаційно-технічним і санітарно-гігієнічним вимогам. Зокрема, роботодавець має інформувати працівників про умови праці та наявність небезпечних або шкідливих виробничих чинників, які залишаються

невирішеними, а також забезпечити проведення інструктажів з питань охорони праці та пожежної безпеки під час роботи з екранними пристроями.

Окрім згаданого нормативного акта, питання охорони праці при роботі з ПК регламентуються й іншими нормативними документами, наприклад, ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», який містить санітарно-гігієнічні вимоги до умов праці, організації робочих місць з ВДТ ЕОМ, параметрів виробничого середовища, освітлення, рівнів шуму тощо.

Електробезпека при експлуатації ПК і периферійних пристроїв забезпечується виконанням положень Правил улаштування електроустановок (ПУЕ), Правил технічної експлуатації електроустановок споживачів (ПТЕЕС), Правил безпечної експлуатації електроустановок споживачів (ПБЕЕС). Згідно з пунктом 1.7.110 ПУЕ, конструкція та розміщення комп'ютерного обладнання повинні відповідати встановленим вимогам електробезпеки і забезпечувати зручність його технічного обслуговування.

Строге дотримання вимог НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів», а також інших державних стандартів і санітарних норм, є обов'язковим для всіх роботодавців та працівників, що використовують ПК і суміжне обладнання для розробки вебзастосунків. Лише системний підхід і суворе виконання положень нормативно-правових актів щодо охорони праці дають змогу створити безпечні та комфортні умови для ІТ-фахівців, істотно знизити ризики виникнення професійних захворювань і травматизму.

5.2. Вимоги до організації робочого місця розробника інформаційної системи

Організація робочого місця розробника вебзастосунків відіграє ключову роль у забезпеченні безпечних та комфортних умов праці, а також у підвищенні продуктивності та ефективності роботи спеціаліста. При облаштуванні робочого простору необхідно враховувати комплекс ергономічних, санітарно-гігієнічних та організаційно-технічних вимог, що регламентуються відповідними нормативно-правовими документами.

Одним із основних нормативних документів, що визначає вимоги до робочого місця з персональним комп'ютером, є ДСТУ ISO 9241-5:2004 «Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози». Згідно з цим стандартом, робоче місце розробника вебзастосунків має бути достатньо просторим для зміни робочого положення та забезпечувати зручний доступ до всіх необхідних елементів обладнання.

Робочий стіл повинен мати площу, достатню для розміщення комп'ютера, периферійних пристроїв, документації та інших необхідних матеріалів. Рекомендовані розміри робочої поверхні становлять: ширина – не менше 1200 мм, глибина – не менше 800 мм. Висота робочої поверхні має забезпечувати правильне положення рук користувача на клавіатурі та відповідати антропометричним даним працівника. Оптимальна висота столу варіюється в межах 680–800 мм.

Робоче крісло розробника повинно бути підйомно-поворотним, з регулюванням висоти та кута нахилу спинки, що забезпечує надійну підтримку тіла у зручному положенні. Сидіння та спинка крісла мають бути

м'якими, з заокругленими краями та виготовленими з повітропроникного матеріалу. Для зменшення статичного навантаження на м'язи верхніх кінцівок рекомендується використовувати стаціонарні або регульовані підлокітники.

Монітор слід розташовувати на відстані 600–700 мм від очей користувача, причому верхня точка екрану має знаходитися на рівні очей або трохи нижче. Кут нахилу екрана повинен становити 10–15° при перпендикулярному розташуванні площини екрану до лінії зору користувача. Клавіатуру слід розміщувати на робочій поверхні або на спеціальній полиці на рівні ліктів користувача з кутом нахилу до горизонтальної площини 15–20°.

Для запобігання перенавантаженню зорового аналізатора та розвитку комп'ютерного зорового синдрому необхідно забезпечити належний рівень освітленості робочої зони. Рекомендована освітленість робочого столу становить 300–500 лк, а освітленість екрана монітора не повинна перевищувати 300 лк. Слід уникати прямого світла в очі та відблисків на екрані монітора.

Мікроклімат приміщення, у якому працює розробник вебзастосунків, має відповідати санітарним нормам. Оптимальна температура повітря складає 22–24°C у холодний період року та 23–25°C – у теплий, відносна вологість – 40–60%, швидкість руху повітря – не більше 0,1 м/с. Для підтримки комфортних параметрів мікроклімату рекомендується застосовувати системи кондиціонування або вентиляції.

Дотримання ергономічних та санітарно-гігієнічних вимог до організації робочого місця розробника вебзастосунків створює оптимальні умови для праці, знижує ризик розвитку професійних захворювань опорно-рухового апарату та органів зору, а також підвищує працездатність і ефективність роботи фахівця.

5.3. Забезпечення безпечних умов праці та мінімізація впливу шкідливих факторів при роботі з комп'ютерним обладнанням

Забезпечення безпечних умов праці та мінімізація впливу шкідливих факторів під час роботи з комп'ютерним обладнанням є одним із ключових завдань охорони праці в сфері інформаційних технологій. Незважаючи на те, що професійна діяльність розробника вебзастосунків не належить до категорії робіт з підвищеною небезпекою, вона супроводжується низкою потенційних ризиків для здоров'я працівника.

До основних шкідливих і небезпечних факторів, які можуть негативно впливати на стан здоров'я розробника під час роботи з комп'ютерною технікою, належать: підвищений рівень електромагнітного випромінювання, статичне електричне поле, неіонізуюче електромагнітне випромінювання оптичного діапазону, підвищений рівень шуму та вібрації, незадовільні параметри мікроклімату, недостатнє або нераціональне освітлення робочої зони, а також психофізіологічні фактори, такі як розумове перенавантаження, перевантаження зорового та опорно-рухового апарату.

Для мінімізації впливу зазначених факторів та створення безпечних умов праці необхідно дотримуватися комплексу вимог та рекомендацій. Зокрема, відповідно до ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», рівень електромагнітного випромінювання на відстані 50 см навколо відеотерміналу за електричною складовою не повинен перевищувати 2,5 В/м у діапазоні частот 5 Гц – 2 кГц та 0,25 В/м у діапазоні частот 2 – 400 кГц. Напруженість електростатичного поля на робочому місці з ВДТ не повинна перевищувати гранично допустиме значення – 20 кВ/м.

Для зниження впливу електромагнітного випромінювання рекомендується використовувати монітори, що відповідають сучасним

стандартам безпеки, зокрема MPR II або TCO. Також слід дотримуватися регламентованих перерв при роботі з комп'ютером, під час яких доцільно виконувати комплекс вправ для очей, шиї, плечей, спини та кистей рук.

Важливим аспектом забезпечення безпечних умов праці є підтримка оптимальних параметрів мікроклімату у виробничому приміщенні. Згідно з ДСанПіН 3.3.2-007-98, на робочих місцях з ВДТ необхідно забезпечувати температуру повітря 22–24°C у холодний період року та 23–25°C у теплий, відносну вологість 40–60% і швидкість руху повітря не більше 0,1 м/с. У приміщеннях з комп'ютерним обладнанням слід щоденно проводити вологе прибирання та регулярно провітрювати приміщення.

Для запобігання перенавантаженню зорового аналізатора під час роботи з комп'ютером необхідно забезпечити раціональне освітлення робочої зони. Освітленість робочого столу у зоні розташування документів має становити 300–500 лк. Яскравість світильників загального освітлення в зоні кутів випромінювання 50°–90° відносно вертикалі в подовжній та поперечній площинах не повинна перевищувати 200 кд/м², а яскравість відблисків на екрані ВДТ – 40 кд/м².

Для зниження ризику розвитку синдрому зап'ястного каналу та інших захворювань опорно-рухового апарату рекомендується використовувати ергономічні клавіатури та маніпулятори типу «миша». Крім того, необхідно дотримуватися правильної робочої пози: сидіти прямо, не сутулячись; лікті повинні бути зігнуті під кутом 70–90°, стегна – розташовані горизонтально, а стопи – на підлозі або на спеціальній підставці.

Не менш важливим фактором забезпечення безпеки праці є дотримання правил електробезпеки. Комп'ютерне обладнання та периферійні пристрої повинні підключатися до електромережі через справні штепсельні роз'єми. Забороняється використовувати саморобні подовжувачі, нестандартне

опалювальне електрообладнання, а також залишати увімкнені прилади без нагляду.

Для запобігання ураженню електричним струмом у разі випадкового контакту зі струмопровідними частинами обладнання необхідно забезпечити наявність заземлення або занулення. Крім того, рекомендовано періодично перевіряти стан ізоляції електропроводів та ефективність заземлюючих пристроїв.

Висновок

Дотримання наведених вище вимог і рекомендацій щодо забезпечення безпечних умов праці та мінімізації впливу шкідливих факторів під час роботи з комп'ютерним обладнанням дає змогу зберегти здоров'я і працездатність розробника вебзастосунків, а також підвищити ефективність і якість його професійної діяльності.

6. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

6.1 Вступ до проблеми

Охорона навколишнього середовища здійснюється через різні засоби, зокрема й правові механізми. У сфері права захист охоплює майже всі складові природного середовища. Основними сучасними нормативно-правовими актами, які визначають основи організації охорони природи в Україні, є Закони «Про охорону навколишнього природного середовища» (25 червня 1991 р.), «Про охорону атмосферного повітря» (16 жовтня 1992 р.), «Про природно-заповідний фонд України» (16 червня 1992 р.), «Про тваринний світ» (3 березня 1993 р.), «Про карантин рослин» (30 червня 1993 р.) тощо. Крім того, питання використання і захисту природи регулюють також кодекси – земельний, водний, лісовий, про надра, а також закони «Про плату за землю» (3 липня 1992 р.), «Про ветеринарну медицину» (25 червня 1992 р.). Важливим є й затверджений Верховною Радою Порядок щодо обмеження, тимчасової заборони чи припинення діяльності підприємств, установ, організацій та об'єктів у разі порушень законодавства про охорону довкілля.

Види права природокористування включають землекористування, водокористування, лісокористування, користування надрами, тваринним світом та природно-заповідним фондом. Природокористування — це раціональне застосування природних ресурсів людиною для задоволення своїх потреб. Ключові принципи цього права: цільове використання, плановість, тривалість, ліцензування, а також врахування його особливої ролі для суспільства. Виокремлюють загальне й спеціальне природокористування стосовно землі, вод, лісу, надр, тваринного світу тощо. Згідно із Законом «Про охорону навколишнього природного середовища», усі громадяни мають право на загальне користування природними ресурсами безкоштовно та без спеціальних дозволів. Це стосується відвідування парків, скверів, водойм,

лісу, збору дикорослих ягід, грибів, горіхів тощо. Стаття 13 Конституції України гарантує це право кожному. Спеціальне природокористування виникає, коли частину ресурсів держава передає у визначених законом випадках індивідам або організаціям на певний термін і за відповідну плату; передача закріплюється спеціальним дозволом.

Права природокористувачів супроводжуються екологічними правами та обов'язками, що інтенсивно розвиваються в юридичній науці. За Конституцією України, кожен має право на безпечне довкілля і відшкодування шкоди, завданої порушенням цього права. Також гарантується доступ до інформації про стан довкілля та її поширення, що закріплено і в Законі «Про охорону навколишнього природного середовища». Над цим правом лежить обов'язок держави впроваджувати санітарно-гігієнічні заходи для покращення стану довкілля. Екологічні права громадян захищаються у судовому порядку. Разом із цим Закон передбачає певні обов'язки, незалежно від статусу природокористувача: усі зобов'язані берегти природу, використовувати ресурси раціонально та не завдавати їм шкоди. Для суб'єктів спеціального використання природних ресурсів встановлюються додаткові обов'язки, включно зі сплатою за використання та забруднення довкілля згідно з нормативами і лімітами. Контроль у цій сфері здійснюють як державні органи, так і громадські організації шляхом перевірок, нагляду, обстежень та експертиз. Державний контроль здійснюють Ради народних депутатів, державні адміністрації та Міністерство охорони навколишнього природного середовища разом із його місцевими структурами.

Основні пріоритети охорони довкілля та раціонального використання природних ресурсів включають:

- забезпечення екологічної безпеки ядерних об'єктів і захист населення й довкілля від радіації, а також мінімізацію негативних наслідків аварії

на Чорнобильській АЕС;

- покращення екологічного стану річкових басейнів України та якості питної води;
- стабілізацію й поліпшення екологічної ситуації у містах і промислових центрах Донецько-Придніпровського регіону;
- будівництво та модернізацію міських очисних каналізаційних споруд;
- боротьбу із забрудненням Чорного й Азовського морів та покращення їхнього стану;
- формування збалансованої системи природокористування, структурну перебудову економіки, екологізацію процесів у промисловості, енергетиці, будівництві, сільському господарстві та транспорті;
- збереження біологічного й ландшафтного різноманіття, розвиток заповідної справи.

Для досягнення цих цілей передбачено такі завдання:

- зменшення рівня радіаційного забруднення до мінімуму;
- захист повітря від забруднення, особливо у великих містах і промцентрах;
- захист і збереження земель від виснаження, забруднення та нераціонального використання;
- збереження та розширення природних ландшафтів, посилення охорони заповідних і рекреаційних територій;
- підвищення стійкості лісів і їхніх екологічних функцій;
- знешкодження, утилізація та захоронення побутових і промислових відходів;
- запобігання забрудненню вод, скорочення й припинення скиду забруднених стоків у водойми, захист підземних вод;

- збереження й відродження малих річок, управління водними ресурсами за басейновим принципом;
- завершення формування державної системи моніторингу довкілля;
- створення систем прогнозування, запобігання й оперативного реагування на надзвичайні ситуації природного та техногенного характеру;
- екологічний супровід конверсії військово-промислового комплексу;
- контроль за екологічною діяльністю Збройних Сил України;
- розробка ефективних механізмів природокористування;
- впровадження економічних важелів впливу на систему природокористування;
- формування системи екологічної освіти, виховання й інформування.

6.2 Забруднення навколишнього середовища підприємством з розробки програмного забезпечення

Основним джерелом забруднення навколишнього середовища підприємством є використання автотранспорту. Підприємство впливає на стан атмосферного повітря, однак не спричиняє забруднення водного чи ґрунтового середовища.

У сучасних умовах автомобільний транспорт є ключовим фактором погіршення якості атмосферного повітря, особливо в межах великих міст. Атмосфера урбанізованих територій характеризується постійним надходженням забруднюючих речовин, що істотно відрізняє її параметри від природного стану. У міських поселеннях спостерігаються найвищі рівні антропогенних навантажень, унаслідок чого змінюються мікрокліматичні фактори та фізико-хімічні властивості повітряного басейну. Згідно з

метеорологічними дослідженнями, температура повітря у містах переважно вища, ніж у передмісті, а також частіше фіксуються атмосферні опади, тумани та смоги – густі утворення, що містять дим, кіптяву і вихлопні гази. Прозорість міської атмосфери є нижчою порівняно із сільською місцевістю, а запиленість та тумани суттєво зменшують проникнення сонячної радіації.

Якість атмосферного повітря у містах дедалі більше погіршується через викиди автотранспортних засобів. Збільшення кількості автомобілів призводить до зростання об'ємів забруднювачів, зокрема оксидів вуглецю, вуглеводнів, оксидів азоту, сажі, а також свинцевих і сірчаних сполук, концентрація яких значно підвищилась. Міська інфраструктура не розрахована на існуючу кількість автотранспорту, що спричиняє короткі ділянки безперервного руху (400–600 м) та низьку середню швидкість (12–20 км/год), збільшуючи витрати пального у 3–4 рази, відповідно посилюючи викиди шкідливих речовин.

Автотранспорт також є джерелом специфічних видів забруднення, зокрема пилу від стирання шин, який у значних обсягах потрапляє у повітря. Окрім продуктів згоряння пального, автомобільний транспорт сприяє накопиченню свинцю в довкіллі. В Україні продовжують використовувати бензин із вмістом свинцю 0,36 г/л, тоді як у Великій Британії, Німеччині та США цей показник становить 0,013–0,15 г/л.

6.3 Розробка заходів щодо зменшення забруднення

Природоохоронна діяльність охоплює всі заходи, спрямовані на підтримку якості довкілля на рівні, що гарантує стійкість біосфери. Вона включає масштабні державні програми зі збереження унікальних природних територій та різноманіття видів, організацію наукових досліджень, підготовку екологічних фахівців і просвітницьку роботу серед населення. Крім того,

підприємства здійснюють локальні дії: очищають стічні води та відхідні гази, знижують використання природних ресурсів тощо. Здебільшого такі заходи реалізуються інженерними методами.

Виділяють два основні напрями природоохоронної діяльності підприємств. Перший — очищення шкідливих викидів, який сам по собі є малоефективним, оскільки не завжди дозволяє повністю припинити надходження забруднюючих речовин у біосферу. Зменшення забруднення одного компонента часто призводить до посилення забруднення іншого. Наприклад, впровадження вологих фільтрів у системах газоочистки зменшує забруднення повітря, але збільшує забруднення води. Уловлені речовини можуть отруювати значні площі землі. Очищувальні споруди скорочують рівень забруднення, проте цілком проблему не вирішують, адже самі генерують нові, хоча й менш об'ємні, але концентровані відходи. Діяльність більшості очисних установок потребує значних енергетичних ресурсів, створюючи додатковий екологічний ризик.

Значна частина забруднювачів, які витрачають кошти на їхнє знешкодження, є речовинами, що вже мають виробничу цінність і можуть бути використані у господарстві. Для інтеграції еколого-економічних рішень важливо поєднати процес очищення шкідливих викидів із утилізацією уловлених речовин.

Другий напрям — усунення причин забруднення шляхом розробки маловідходних або безвідходних технологій, які дають змогу раціонально використовувати сировину і максимально утилізувати шкідливі речовини. Однак зараз не для всіх галузей знайдено економічно прийнятні рішення щодо зменшення відходів та їх утилізації, тому необхідно діяти комплексно у двох напрямках.

Слід пам'ятати, що жодні очисні споруди чи безвідходні технології не здатні відновити стійкість біосфери, якщо буде перевищено граничні показники перетворення природних систем, згідно із законом незамінності біосфери. Критичними є використання понад 1% енергії біосфери й трансформація понад 10% природних територій (правило одного і десяти відсотків). Тому технічний прогрес не замінює необхідності перегляду суспільних пріоритетів, стабілізації населення, створення заповідників та інших заходів.

Основні напрями захисту атмосфери від автотранспортних викидів:

- розвиток і виробництво автомобілів із високоефективними, малотоксичними двигунами, включаючи дизелізацію транспорту;
- впровадження дієвих систем нейтралізації вихлопних газів;
- зниження токсичності моторних палив;
- оптимізація організації транспортного руху в містах та вдосконалення дорожнього будівництва для забезпечення безперервного руху на автомагістралях.

Висновок

Отже, забезпечення ефективного захисту навколишнього середовища вимагає інтегрованого підходу, який поєднує впровадження технологічних інновацій із цілеспрямованою трансформацією суспільних цінностей. Необхідно не лише застосовувати сучасні методи очищення та нейтралізації забруднень, а й оптимізувати використання природних ресурсів і сприяти збереженню екосистем. Лише всебічне поєднання технологічних рішень і свідомої екологічної відповідальності дозволить підтримати стабільність біосфери для майбутніх поколінь.

ВИСНОВКИ

У межах виконаної роботи здійснено комплексне дослідження та розроблення системи управління контентом веб-порталу МОШПО. Проєкт охоплював повний цикл створення інформаційної системи – від аналітичного опрацювання предметної області до формування технічних, організаційних та програмних рішень, а також підготовки супровідної документації.

У процесі дослідження було реалізовано такі основні напрями:

- проведено систематизований огляд сучасних систем управління контентом, визначено їхні ключові характеристики, архітектурні підходи та сфери застосування;
- проаналізовано існуючі програмні рішення у сфері контент-менеджменту, оцінено їх функціональні можливості, масштабованість, гнучкість, вимоги до інфраструктури та потенціал адаптації до потреб освітніх установ;
- виконано аналіз інтернет-технологій, що можуть бути використані для створення систем управління контентом, зокрема веб-фреймворків, мов програмування, серверних платформ, протоколів взаємодії та засобів забезпечення інформаційної безпеки;
- досліджено інструментарій, який застосовується під час розроблення веб-порталів, включно з технологіями клієнтської та серверної частини, засобами інтеграції, бібліотеками інтерфейсних компонентів та системами керування базами даних;
- розроблено веб-портал із підтримкою функцій управління контентом. Для цього було проведено детальне вивчення предметної області, сформовано вимоги до системи, розроблено концептуальну модель, підготовлено технічне завдання, визначено загальносистемні рішення, а також рішення щодо організаційного, інформаційного, технічного та програмного забезпечення. Окремо створено повний комплект супровідної документації, необхідної для впровадження та експлуатації системи.

У межах загальносистемних рішень визначено архітектурний стиль системи, структуру її компонентів, побудовано дерево функцій та описано механізми взаємодії з іншими інформаційними системами, що функціонують у закладі освіти. Особливу увагу приділено забезпеченню сумісності, масштабованості та можливості подальшого розвитку системи.

Організаційні рішення передбачали аналіз та оптимізацію розподілу функціональних обов'язків між фахівцями підрозділів ЗВО, які залучені до роботи з порталом. Було визначено ролі користувачів, порядок взаємодії між ними, а також процедури адміністрування та супроводу системи.

У частині інформаційного забезпечення сформовано логічну та фізичну модель даних, визначено структуру інформаційних потоків, а також ухвалено рішення щодо використання реляційної бази даних як основи для зберігання та оброблення інформації. Обґрунтовано вибір СУБД з урахуванням вимог до надійності, продуктивності та безпеки.

Технічні рішення включали розроблення схеми автоматизації системи, визначення комплексу технічних засобів, необхідних для її функціонування, а також проєктування схеми інтеграції веб-порталу у наявну інфраструктуру ЗВО. Розглянуто питання серверного розміщення, мережевої взаємодії, резервного копіювання та захисту даних.

На етапі розроблення програмного забезпечення було обрано методологію створення системи, визначено структуру програмного комплексу, описано функціональне призначення його модулів та механізми їхньої взаємодії. Особливу увагу приділено забезпеченню модульності, розширюваності та підтримуваності програмного коду.

Крім того, проведено економічну оцінку розробленого проєкту, у межах якої визначено витрати на створення та впровадження системи, оцінено економічну доцільність її використання та потенційний ефект від автоматизації процесів управління контентом.

Розроблену систему планується впровадити у МОІППО, що сприятиме підвищенню ефективності інформаційної взаємодії та оптимізації процесів управління контентом. Окрім того, система може бути адаптована та впроваджена в інших закладах освіти, що підкреслює її універсальність та практичну цінність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання дипломного проектування зі спеціальності 7.080401 «Інформаційні управляючі системи та технології». – К.В. Кошкін, С.Б. Приходько, С.В. Суслов, В.І. Тимофєєв. – Миколаїв: УДМТУ, 2003. – 52 с.
2. Кріс Шефер, Кларенс Хо, Роб Харроп. Spring 4 для професіоналів, APress, 2016
3. Роберт Лафоре, Структури даних та алгоритми Java, SAMS, 2016
4. Бретт Маклафлін. Об'єктно-орієнтований аналіз та проектування, O'REILLY Media, 2013
5. Сейед Тахагхогхі, Х'ю Є. Вільямс. Посібник з MySQL, O'REILLY Media, 2007
6. Герберт Шилдт. Java8. Повне керівництво, 9-е изд., К.:Вільямс, 2015
7. Система керування вмістом. Вікіпедія — вільна енциклопедія. [Електроний ресурс]. – Режим доступу URL: <http://uk.wikipedia.org/wiki/CMS>.
8. Каталог CMS. CMS Magazine [Електроний ресурс]. – Режим доступу URL: <http://www.cmsmagazine.ru/catalogue>.
9. Інжекція SQL-кода. Вікіпедія — вільна енциклопедія. [Електроний ресурс]. – Режим доступу URL: http://uk.wikipedia.org/wiki/SQL_injection.
10. Межсайтовий скриптинг. Вікіпедія — вільна енциклопедія. [Електроний ресурс]. – Режим доступу URL: <http://uk.wikipedia.org/wiki/Xss>.
11. Лановий О.Ф., Кобзєв С.В., Калякін С.В., Системи управління контентом і безпека WEB-сайтів. Системи обробки інформації. Харків; ХНУВС, 2010
12. Безпека життєдіяльності. Під редакцією Білова Н.А. – К.: Наука, 2011.– 364с.
13. Вимоги до організації робочого місця користувача комп'ютера: веб-сайт. URL: <https://zp.dsp.gov.ua/articles/a-657.html> (дата звернення: 14.05.2025).

14. Охорона праці користувачів комп'ютерів: веб-сайт. URL: https://pidru4niki.com/1821100138196/bzhd/ohorona_pratsi_koristuvachiv_kompyuteriv (дата звернення: 14.05.2025).
15. ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». URL: <https://zakon.rada.gov.ua/rada/show/v0007282-98> (дата звернення: 14.05.2025).

ДОДАТКИ

Додаток А – Технічне завдання на розробку

1 Вступ

1.1 Назва – Система управління контентом web-порталу МОІППО

1.2 Призначення програми: надати можливість зручно та швидко управляти контентом необхідним для виконання діяльності інституту післядипломної педагогічної освіти.

1.3 Область застосування: МОІППО

2 Призначення розробки

Призначена для спрощення, уніфікації, прискорення роботи методистів вищого навчального закладу післядипломної педагогічної освіти, підвищення ефективності та помилко-безпечності робіт працівників інституту.

3 Вимоги до системи

3.1 Вимоги до функціональних характеристик

Система буде містити наступні функціональні характеристики:

- Управління інформацією про новини, накази, події, які мають відношення до діяльності МОІППО;
- Своєчасне доведення інформації до слухачів курсів та інших користувачів web-порталу;
- Управління користувачами;
- Перегляд створеної інформації.

3.2 Вимоги до надійності

3.2.1 Вимоги до забезпечення надійності функціонування програми

Надійне (стійке) функціонування системи має бути забезпечено надійним (стійким) функціонуванням бездротової мережі і мережі електропостачання в університеті, або у іншому місці, де буде розміщений сервер.

3.2.2 Відмова виконання програмного продукту через некоректні дії користувача.

- Перевірка формату вводу даних для коректної роботи.
- Функція автоматичного backup кожен тиждень.

3.3 Умови експлуатації

Данною програмою будуть користуватися працівники, які є впевненими користувачами, та не потребують додаткового навчання по даному ПО, але можуть вимагати додаткову допомогу у роботі з вихідними даними.

3.4 Вимоги до інформаційної та програмної сумісності

3.4.1 Вимоги до інформаційних структур і методів розв'язання

Вимоги до інформаційних структур і методів розв'язання відсутні.

3.4.2 Вимоги до вихідних кодів і мов програмування

Додаткові вимоги не пред'являються.

3.4.3 Вимоги до програмних засобів , які використовуються програмою

Програмні засоби що використовуються мають бути вільно поширювані

3.5 Вимоги до маркування та упаковки

3.5.1 Вимоги до маркування :

- Читабельність тексту, чіткість ілюстрацій;
- Достовірність інформації про систему
- Нанесення на Install CD позначок про дане ПО.

3.5.2 Вимоги до упаковки:

Вимоги до упаковки відсутні.

3.6 Вимоги до транспортування та зберігання

Вимоги до транспортування програми відсутня.

3.7 Спеціальні вимоги

Спеціальні вимоги відсутні.

4 Вимоги до програмної документації

4.1 Попередній склад програмної документації

Попередній склад програмної документації повинен включати в себе:

- технічне завдання;
- устав проекту.

4.2 Склад програмної документації

Програмна документація складається з наступних програмних документів:

- Устав проекту;
- Технічне завдання ;
- Опис системи ;
- Код програми ;
- Керівництво користувача.

Сертифікація містить список всіх програмних документів .

Технічне завдання містить опис завдання роботи, склад функціональних характеристик програми, опис вимог для розробки.

Опис програми містить опис логічної структури програми, форматів вхідних та вихідних даних, використовуваних технічних засобів.

Код програми містить опис блоків програми і запис блоків в програмному коді.

Керівництво користувача містить інформацію , необхідну для ознайомлення користувача з програмним продуктом.

5 Техніко-економічні показники

5.1 Економічні переваги розробки

Орієнтовна економічна ефективність не розраховується.

6 Стадії та етапи розробки програмного забезпечення

6.1 Стадії розробки

Розробка повинна бути проведена в 5 стадій:

1. Технічне завдання
2. Ескізний проект
3. Технічний проект
4. Робочий проект
5. Впровадження

Додаток Б - Програма і методика випробувань

Об'єкт випробувань

Назва об'єкта випробувань: «МОІППО CMS» , надалі називається «Програма». Програма дозволяє користувачеві пройти автентифікацію та отримати доступ до свого інтерфейсу.

Ціль випробувань

Перевірка придатності програми до використання за призначенням, її відповідність заданим вимогам і програмним документам. Перевірка реально досягнутих характеристик програми.

Вимоги до програми

Програма повинна задовольняти наступним функціональним характеристикам:

- 1) перевірка правильності імені та паролю користувача;
- 2) надання користувачеві доступу до свого інтерфейсу.

Користувач повинен вказати ім'я та пароль.

Вимоги до програмної документації

У комплект програмної документації повинні входити наступні документи:

- Загальний опис системи.
- Посібник користувача – адміністратора.

- Посібник користувача – відповідального модератора.
- Посібник користувача – клієнта.
- Інструкція з формування і ведення бази даних.
- Посібник адміністратора з установки програмного забезпечення на серверах.
- Програма і методика випробувань.
- Текст програми.

Засоби і порядок випробувань

Вимоги до складу і параметрів технічних засобів:

Сервер на базі i586_64 або x86.

Вимоги до складу і параметрів програмних засобів :

- 1) операційна система Windows;
- 2) встановлене з'єднання з сервером баз даних MariaDB 10.1

Група людей, що будуть проводити випробування, повинна складатися з автора програми (розробника) і представника і/або самого замовника. При наявності відповідної можливості, ця група може бути розширена програмістом, спеціалістом-тестувальником і фахівцем-проектувальником, що не брали участь у розробці програми.

Попередньо (не пізніше 3 днів) до початку випробувань уточнюється кінцевий склад групи, що буде проводити випробування. Кожен член групи одержує необхідну документацію до програми (список вимог до програми, опис програми, керівництво користувача, програму і методику випробувань).

Порядок проведення випробувань:

- послідовно виконуються контрольні тести; результати заносяться у відповідний журнал;
- виробляється аналіз отриманих результатів і встановлюється відповідність програми заданим вимогам і програмним документам.

Методи випробувань

Робота програми залежить тільки від двох параметрів, що вводяться користувачем: від імені та паролю користувача.

Іспити будуть проводитися методом «чорної шухляди».

Тести будуть створюватися на основі класів еквівалентності.

Класи еквівалентності виділяються шляхом вибору кожної вхідної умови і розбивкою його на двох або більш груп. Розрізняють два типи класів еквівалентності: правильні класи еквівалентності, що представляють правильні вхідні дані програми, і неправильні класи еквівалентності, що представляють всі інші можливі стани умов (помилкові вхідні значення). При цьому існує ряд правил:

- один тест може покривати тільки один неправильний клас еквівалентності;
- один тест повинний покривати як можна більше число непокритих правильних класів еквівалентності.

Виділені класи еквівалентності для складання тестів мають унікальні номери і приводяться в таблиці Б.1.

Таблиця Б.1 – Класи еквівалентності

Операція, що тестується	Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
Перевірка імені та паролю користувача	Ім'я правильне	Так (1)	Ні (2)
	Пароль правильний	Так (3)	Ні (4)
Надання користувачеві доступу до свого інтерфейсу	Належність користувача до інтерфейсу встановлена	Так (5)	Ні (6)

Тести

Тести складені на основі класів еквівалентності таблиці Б.1. Тести використовують наступні початкові умови:

- 1) користувач – user1, його пароль – pass1 (власник товару);
- 2) користувач – admin, його пароль – adMpaSSw0rd (адміністратор системи).

Тест 1

Класи еквівалентності:

1, 3, 5

Вхідні дані:

Ім'я користувача: user1

Пароль користувача: pass1

Результат:

Ім'я та пароль користувача пізнані. Належність до власників товарів встановлена. Наданий доступ до інтерфейсу власника товарів.

Тест 2

Класи еквівалентності:

2, 3, 5

Вхідні дані:

Ім'я користувача: user2

Пароль користувача: pass1

Результати:

Ім'я та пароль користувача не пізнані. Належність до користувачів системи не встановлена. Видане повідомлення про помилку введення імені чи паролю. Доступу до системи не надано.

Тест 3

Класи еквівалентності :

1, 4, 5

Вхідні дані:

Ім'я користувача: user1

Пароль користувача: pass2

Результати:

Ім'я та пароль користувача не пізнані. Належність до користувачів системи не встановлена. Видане повідомлення про помилку введення імені чи паролю. Доступу до системи не надано.

Тест 4

Класи еквівалентності:

1, 3, 5

Вхідні дані:

Ім'я користувача: admin

Пароль користувача: adMpaSSw0rd

Результати:

Ім'я та пароль користувача впізнані. Наданий доступ до адміністрування системи.

Додаток В - Тексти програмних модулів системи

Оскільки система складається з багатьох модулів та підсистем, нижче наведено тексти деяких (не усіх) програмних модулів.

Файл 1 – ArticleController.java

```
package com.moippocms.cms.controllers;

import java.util.Calendar;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.TimeZone;
import java.util.UUID;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.moippocms.cms.MCMSArticlesCache;
import com.moippocms.cms.MCMSAuthenticationResult;
import com.moippocms.cms.MCMSCacheFactory;
import com.moippocms.cms.MCMSAdminDataStorage.AdminQueryOperator;
import com.moippocms.cms.MCMSAdminDataStorage.AdminSortOperator;
import com.moippocms.cms.cmsdata.MCMSArticle;
import com.moippocms.cms.cmsdata.MCMSResource;
import com.moippocms.cms.engine.DefaultMCMSCacheFactory;
import com.moippocms.cms.engine.MCMSAdminDataStorageListener;
import com.moippocms.cms.exception.MCMSException;
import com.moippocms.cms.exception.MCMSIOException;

public class ArticleController extends Controller implements
MCMSAdminDataStorageListener{
    private ArticleValidator validator;
    private MCMSArticlesCache wbArticleCache;
    public ArticleController()
    {
        validator = new ArticleValidator();
        MCMSCacheFactory wbCacheFactory =
DefaultMCMSCacheFactory.getInstance();
        wbArticleCache = wbCacheFactory.getArticlesCacheInstance();
        adminStorage.addStorageListener(this);
    }

    public<T> void notify (T t, AdminDataStorageOperation o, Class<? extends
Object> type)
    {
        try
        {
            if (type.equals(MCMSArticle.class))
            {
                wbArticleCache.Refresh();
            }
        } catch (MCMSIOException e)
        {
            // do nothing
        }
    }
}
```

```

    }

    public void create(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String jsonRequest = httpServletToolbox.getBodyText(request);
            MCMSArticle article =
(MCMSArticle)jsonObjectConverter.objectFromJSONString(jsonRequest,
MCMSArticle.class);
            Map<String, String> errors = validator.validateCreate(article);

            if (errors.size()>0)
            {
                httpServletToolbox.writeBodyResponseAsJson(response, "",
errors);
                return;
            }

            article.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).
getTime());

            article.setExternalKey(adminStorage.getUniqueId());
            article.setVersion(UUID.randomUUID().toString());
            MCMSArticle newArticle = adminStorage.addWithKey(article);

            MCMSResource resource = new
MCMSResource(newArticle.getExternalKey(), newArticle.getTitle(),
MCMSResource.ARTICLE_TYPE);
            try
            {
                adminStorage.addWithKey(resource);
            } catch (Exception e)
            {
                // do not propagate further
            }

            returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(newArticle));
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

        } catch (Exception e)
        {
            Map<String, String> errors = new HashMap<String, String>();

            errors.put("", MCMSErrors.WB_CANT_CREATE_RECORD);
            httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
        }
    }
}

```



```

    public void getAll(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            Map<String, Object> additionalInfo = new HashMap<String,
Object> ();

            String sortParamDir =
request.getParameter(SORT_PARAMETER_DIRECTION);
            String sortParamProp =
request.getParameter(SORT_PARAMETER_PROPERTY);

            List<MCMSArticle> articles = null;

            if (sortParamDir != null && sortParamProp != null)
            {
                if (sortParamDir.equals(SORT_PARAMETER_DIRECTION_ASC))
                {
                    additionalInfo.put(SORT_PARAMETER_DIRECTION,
SORT_PARAMETER_DIRECTION_ASC);
                    additionalInfo.put(SORT_PARAMETER_PROPERTY,
sortParamProp);

                    articles =
adminStorage.getAllRecords(MCMSArticle.class, sortParamProp,
AdminSortOperator.ASCENDING);
                } else if
(sortParamDir.equals(SORT_PARAMETER_DIRECTION_DSC))
                {
                    additionalInfo.put(SORT_PARAMETER_DIRECTION,
SORT_PARAMETER_DIRECTION_DSC);
                    additionalInfo.put(SORT_PARAMETER_PROPERTY,
sortParamProp);

                    articles =
adminStorage.getAllRecords(MCMSArticle.class, sortParamProp,
AdminSortOperator.DECENDING);
                } else
                {
                    articles =
adminStorage.getAllRecords(MCMSArticle.class);
                }
            } else
            {
                articles = adminStorage.getAllRecords(MCMSArticle.class);
            }

            List<MCMSArticle> result = filterPagination(request, articles,
additionalInfo);

            returnJson.put(DATA,
jsonObjectConverter.JSONArrayFromListObjects(result));

```

```

        returnJson.put (ADDITIONAL_DATA,
jsonObjectConverter.JSONObjectFromMap (additionalInfo));

        httpServletToolbox.writeBodyResponseAsJson (response,
returnJson, null, authenticationResult);

    } catch (Exception e)
    {
        Map<String, String> errors = new HashMap<String, String>();

        errors.put("", MCMSErrors.WB_CANT_GET_RECORDS);
        httpServletToolbox.writeBodyResponseAsJson (response,
jsonObjectConverter.JSONObjectFromMap (null), errors);
    }
}

public void get (HttpServletRequest request, HttpServletResponse response,
String requestUri) throws MCMSException
{
    org.json.JSONObject returnJson = new org.json.JSONObject();
    MCMSAuthenticationResult authenticationResult =
this.handleAuthentication (request);
    if (! isRequestAuthenticated (authenticationResult))
    {
        httpServletToolbox.writeBodyResponseAsJson (response,
returnJson, null, authenticationResult);
        return ;
    }

    try
    {
        String key = (String) request.getAttribute ("key");
        MCMSArticle article = adminStorage.get (key, MCMSArticle.class);
        returnJson.put (DATA,
jsonObjectConverter.JSONFromObject (article));
        httpServletToolbox.writeBodyResponseAsJson (response,
returnJson, null, authenticationResult);
    } catch (Exception e)
    {
        Map<String, String> errors = new HashMap<String, String>();

        errors.put("", MCMSErrors.WB_CANT_GET_RECORDS);
        httpServletToolbox.writeBodyResponseAsJson (response,
jsonObjectConverter.JSONObjectFromMap (null), errors);
    }
}

public void getExt (HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
{
    org.json.JSONObject returnJson = new org.json.JSONObject();
    MCMSAuthenticationResult authenticationResult =
this.handleAuthentication (request);
    if (! isRequestAuthenticated (authenticationResult))
    {
        httpServletToolbox.writeBodyResponseAsJson (response,
returnJson, null, authenticationResult);
        return ;
    }

    try
    {
        String extKey = (String) request.getAttribute ("key");

```

```

        List<MCMSArticle> articles =
adminStorage.query(MCMSArticle.class, "externalKey", AdminQueryOperator.EQUAL,
extKey);
        MCMSArticle article = (articles.size()>0) ? articles.get(0) :
null;
        returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(article));
        httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

    } catch (Exception e)
    {
        Map<String, String> errors = new HashMap<String, String>();

        errors.put("", MCMSErrors.WB_CANT_GET_RECORDS);
        httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
    }
}

    public void delete(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String key = (String)request.getAttribute("key");
            MCMSArticle article = adminStorage.get(key, MCMSArticle.class);
            adminStorage.delete(key, MCMSArticle.class);

            try
            {
                if (article != null)
                {
                    adminStorage.delete(article.getExternalKey(),
MCMSResource.class);
                }
            } catch (Exception e)
            {
                // do not propagate further
            }

            returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(article));
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

        } catch (Exception e)
        {
            Map<String, String> errors = new HashMap<String, String>();

            errors.put("", MCMSErrors.WB_CANT_DELETE_RECORD);
            httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
        }
    }
}

```

```

    }
}

public void update(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
{
    org.json.JSONObject returnJson = new org.json.JSONObject();
    MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
    if (! isRequestAuthenticated(authenticationResult))
    {
        httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
        return ;
    }

    try
    {
        String key = (String)request.getAttribute("key");
        String jsonRequest = httpServletToolbox.getBodyText(request);
        MCMSArticle article =
(MCMSArticle)jsonObjectConverter.objectFromJSONString(jsonRequest,
MCMSArticle.class);
        article.setExternalKey(key);
        Map<String, String> errors = validator.validateUpdate(article);

        if (errors.size()>0)
        {
            httpServletToolbox.writeBodyResponseAsJson(response, "",
errors);
            return;
        }

        article.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).
getTime());
        article.setVersion(UUID.randomUUID().toString());

        MCMSArticle newArticle = adminStorage.update(article);

        MCMSResource resource =
new MCMSResource(newArticle.getExternalKey(), newArticle.getTitle(),
MCMSResource.ARTICLE_TYPE);
        try
        {
            adminStorage.update(resource);
        } catch (Exception e)
        {
            // do not propagate further
        }
        returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(newArticle));
        httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

    } catch (Exception e)
    {
        Map<String, String> errors = new HashMap<String, String>();

        errors.put("", MCMSErrors.WB_CANT_UPDATE_RECORD);
        httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
    }
}

```

Файл 2 – FileController.java

```
package com.moippocms.cms.controllers;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TimeZone;
import java.util.UUID;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItemIterator;
import org.apache.commons.fileupload.FileItemStream;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.fileupload.util.Streams;
import org.apache.commons.io.IOUtils;

import com.moippocms.cms.MCMSAuthenticationResult;
import com.moippocms.cms.MCMSFilePath;
import com.moippocms.cms.MCMSAdminDataStorage;
import com.moippocms.cms.MCMSCacheFactory;
import com.moippocms.cms.MCMSFileInfo;
import com.moippocms.cms.MCMSFileStorage;
import com.moippocms.cms.MCMSFilesCache;
import com.moippocms.cms.MCMSAdminDataStorage.AdminQueryOperator;
import com.moippocms.cms.MCMSAdminDataStorage.AdminSortOperator;
import com.moippocms.cms.cmsdata.MCMSFile;
import com.moippocms.cms.cmsdata.MCMSResource;
import com.moippocms.cms.cmsdata.MCMSUri;
import com.moippocms.cms.engine.DefaultMCMSCacheFactory;
import com.moippocms.cms.engine.MCMSAdminDataStorageListener;
import com.moippocms.cms.exception.MCMSException;
import com.moippocms.cms.exception.MCMSIOException;
import com.moippocms.cms.utility.ContentTypeDetector;

public class FileController extends Controller implements
MCMSAdminDataStorageListener{
    public static final String PUBLIC_BUCKET = "public";

    private FileValidator validator;
    private MCMSFilesCache filesCache;

    private static final int MAX_DIR_DEPTH = 25;

    public FileController()
    {
        validator = new FileValidator();
        MCMSCacheFactory wbCacheFactory =
DefaultMCMSCacheFactory.getInstance();
        filesCache = wbCacheFactory.getFilesCacheInstance();
        adminStorage.addStorageListener(this);
    }
}
```

```

    public<T> void notify (T t, AdminDataStorageOperation o, Class<? extends
Object> type)
    {
        try
        {
            if (type.equals(MCMSFile.class))
            {
                filesCache.Refresh();
            }
        } catch (MCMSIOException e)
        {
            // TBD
        }
    }

    public static String getDirectoryFullPath(String externalKey,
MCMSAdminDataStorage adminStorage) throws MCMSException
    {
        if (externalKey == null || externalKey.length() == 0)
        {
            return "";
        }
        String result = "";
        String parentExtKey = externalKey;
        boolean completed = false;
        for(int i = 0; i< MAX_DIR_DEPTH; i++)
        {
            if (parentExtKey != null && parentExtKey.length()>0)
            {
                MCMSFile parent = getDirectory(parentExtKey, adminStorage);
                if (parent != null)
                {
                    result = parent.getFileName() + "/" + result;
                    parentExtKey = parent.getOwnerExtKey();
                } else
                {
                    throw new MCMSException("MCMSFile should not be null");
                }
            } else
            {
                completed = true;
                break;
            }
        }
        if (completed)
        {
            if (result.startsWith("/")) result = result.substring(1);
            if (result.endsWith("/")) result = result.substring(0,
result.length()-1);

            return result;
        } else
        {
            throw new MCMSException("Cannot calculate full dir path");
        }
    }

    public static String getDirectoryFullPath(String externalKey, Map<String,
MCMSFile> files) throws MCMSException
    {
        if (externalKey == null || externalKey.length() == 0)
        {

```

```

        return "";
    }
    String result = "";
    String parentExtKey = externalKey;
    boolean completed = false;
    for(int i = 0; i< MAX_DIR_DEPTH; i++)
    {
        if (parentExtKey != null && parentExtKey.length()>0)
        {
            MCMSFile parent = files.get(parentExtKey);
            if (parent != null && parent.getDirectoryFlag() != null &&
parent.getDirectoryFlag() == 1)
            {
                result = parent.getFileName() + "/" + result;
                parentExtKey = parent.getOwnerExtKey();
            } else
            {
                throw new MCMSEException("MCMSFile should not be null");
            }
        } else
        {
            completed = true;
            break;
        }
    }
    if (completed)
    {
        if (result.startsWith("/")) result = result.substring(1);
        if (result.endsWith("/")) result = result.substring(0,
result.length()-1);

        return result;
    } else
    {
        throw new MCMSEException("Cannot calculate full dir path");
    }
}

    public static MCMSFile getDirectory(String externalKey,
MCMSAdminDataStorage adminStorage) throws MCMSEException
    {
        List<MCMSFile> result = adminStorage.query(MCMSFile.class,
"externalKey", AdminQueryOperator.EQUAL, externalKey);
        if (result.size() == 1)
        {
            MCMSFile file = result.get(0);
            if (file.getDirectoryFlag() != null && file.getDirectoryFlag() ==
1)
            {
                return file;
            }
        }
        return null;
    }

    private String getDirectoryFromLongName(String longName)
    {
        int index = longName.lastIndexOf('/');
        if (index>0)
        {
            return longName.substring(0, index);
        }
        return null;
    }

```

```

    }
    private String getFileNameFromLongName(String longName)
    {
        int index = longName.lastIndexOf('/');
        if (index>0)
        {
            return longName.substring(index+1);
        }
        return longName;
    }

    private List<MCMSFile> getFilesFromDirectory(MCMSFile directory) throws
MCMSEException
    {
        if (null == directory || directory.getDirectoryFlag() != 1)
        {
            return new ArrayList<MCMSFile>();
        }

        String ownerExtKey = directory.getExternalKey();
        return adminStorage.query(MCMSFile.class, "ownerExtKey",
AdminQueryOperator.EQUAL, ownerExtKey);
    }

    private MCMSFile getFileFromDirectory(MCMSFile directory, String fileName)
throws MCMSEException
    {
        String ownerExtKey = "";
        if (null != directory)
        {
            ownerExtKey = directory.getExternalKey();
        }
        Set<String> propertyNames = new HashSet<String>();
        propertyNames.add("ownerExtKey");
        propertyNames.add("fileName");
        HashMap<String, AdminQueryOperator> operators = new HashMap<String,
AdminQueryOperator>();
        operators.put("ownerExtKey", AdminQueryOperator.EQUAL);
        operators.put("fileName", AdminQueryOperator.EQUAL);

        HashMap<String, Object> values = new HashMap<String, Object>();
        values.put("ownerExtKey", ownerExtKey);
        values.put("fileName", fileName);

        List<MCMSFile> result = adminStorage.queryEx(MCMSFile.class,
propertyNames, operators, values);
        if (result.size() == 1)
        {
            return result.get(0);
        }
        return null;
    }

    private void deleteFile(MCMSFile file, int level) throws MCMSEException,
IOException
    {
        //we need to protect from infinite loops
        if (level > MAX_DIR_DEPTH) return;

        if (file.getDirectoryFlag() != null && file.getDirectoryFlag() == 1)
        {
            List<MCMSFile> files = getFilesFromDirectory(file);

```



```

        for(MCMSFile afile: files)
        {
            deleteFile(afile, level+1);
        }
    }
    if (file.getBlobKey() != null && file.getBlobKey().length()>0)
    {
        MCMSFilePath contentFile = new MCMSFilePath(PUBLIC_BUCKET,
file.getBlobKey());

        cloudFileStorage.deleteFile(contentFile);
    }
    adminStorage.delete(file.getExternalKey(), MCMSFile.class);
}

private MCMSFile createDirectory(MCMSFile parentdirectory, String dirName)
throws MCMSException
{
    MCMSFile file = new MCMSFile();
    file.setDirectoryFlag(1);
    file.setFileName(dirName);
    file.setSize(0L);
    file.setExternalKey(adminStorage.getUniqueId());

file.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).getTime()
);

    if (parentdirectory == null)
    {
        file.setOwnerExtKey("");
    } else
    {
        file.setOwnerExtKey(parentdirectory.getExternalKey());
    }
    file.setVersion(UUID.randomUUID().toString());

    file = adminStorage.addWithKey(file);

    MCMSResource resource = new MCMSResource(file.getExternalKey(),
file.getFileName(), MCMSResource.FILE_TYPE);
    try
    {
        adminStorage.addWithKey(resource);
    } catch (MCMSException e)
    {
        // do not propagate further
    }

    return file;
}

private void addFileToDirectory(MCMSFile parentDirectory, MCMSFile file,
InputStream is) throws MCMSException, IOException
{
    String dirPath = "";
    String filePath = null;
    if (parentDirectory != null)
    {
        dirPath = getDirectoryFullPath(parentDirectory.getExternalKey(),
adminStorage);
    }
    if (dirPath.length()>0)
    {
        filePath = dirPath + "/" + file.getFileName();
    }
}

```

```

        } else
        {
            filePath = file.getFileName();
        }

        MCMSFilePath cloudFile = new MCMSFilePath(PUBLIC_BUCKET, filePath);
        cloudFileStorage.storeFile(is, cloudFile);
        cloudFileStorage.updateContentType(cloudFile,
        ContentTypeDetector.fileNameToContentType(file.getFileName()));

        MCMSFileInfo fileInfo = cloudFileStorage.getFileInfo(cloudFile);
        file.setBlobKey(cloudFile.getPath());
        file.setHash(fileInfo.getCrc32());
        file.setSize(fileInfo.getSize());
        file.setContentType(fileInfo.getContentType());
        file.setAdjustedContentType(file.getContentType());
        file.setDirectoryFlag(0);
        if (parentDirectory != null)
        {
            file.setOwnerExtKey(parentDirectory.getExternalKey());
        } else
        {
            file.setOwnerExtKey("");
        }
        file.setVersion(UUID.randomUUID().toString());

        MCMSResource resource = new MCMSResource(file.getExternalKey(),
        file.getFileName(), MCMSResource.FILE_TYPE);

        adminStorage.delete(file.getExternalKey(), MCMSFile.class);
        adminStorage.addWithKey(file);

        try
        {
            adminStorage.delete(resource.getRkey(), MCMSResource.class);
            adminStorage.addWithKey(resource);
        } catch (MCMSEException e)
        {
            // do not propagate further
        }
    }

    /**
     * Checks and creates subdirectories from a path relative to the owner
     * @param subDirectory path that is relative to the owner directory
     * @param owner The owner directory, or null if the owner is the root
     * @return
     */

    private Map<String, MCMSFile> checkAndCreateSubDirectory(String
    fullDirectory, MCMSFile owner) throws MCMSEException
    {
        Map<String, MCMSFile> subfolderFiles = new HashMap<String, MCMSFile>();

        MCMSFile currentDirectory = owner;
        String advancePath = "";
        while (fullDirectory.length() > 0)
        {
            int index = fullDirectory.indexOf('/');
            if (index == 0)
            {

```

```

        fullDirectory = fullDirectory.substring(1);
        index = fullDirectory.indexOf('/');
    }
    String subDir = fullDirectory;
    if (index > 0)
    {
        subDir = fullDirectory.substring(0, index);
        fullDirectory = fullDirectory.substring(index+1);
    } else
    {
        fullDirectory = "";
    }
    if (subDir.length() > 0)
    {
        MCMSFile subDirFile = getFileFromDirectory(currentDirectory,
subDir);

        if (null == subDirFile)
        {
            subDirFile = createDirectory(currentDirectory, subDir);
        }
        if (advancePath.length() == 0)
        {
            advancePath = subDir;
        } else
        {
            advancePath = advancePath + "/" + subDir;
        }
        subfolderFiles.put(advancePath, subDirFile);
        currentDirectory = subDirFile;
    }
}

return subfolderFiles;
}

public void uploadFolder(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
{
    org.json.JSONObject returnJson = new org.json.JSONObject();
    MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
    if (! isRequestAuthenticated(authenticationResult))
    {
        httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
        return ;
    }

    try
    {
        ServletFileUpload upload = new ServletFileUpload();
        upload.setHeaderEncoding("UTF-8");
        FileItemIterator iterator = upload.getItemIterator(request);
        MCMSFile ownerFile = null;
        Map<String, MCMSFile> subfolderFiles = new HashMap<String,
MCMSFile>();

        while (iterator.hasNext()) {
            FileItemStream item = iterator.next();

            if (item.isFormField())
            {
                if (item.getFieldName().equals("ownerExtKey"))
                {
                    &&

```

```

        String ownerExtKey =
Streams.asString(item.openStream());
        ownerFile = getDirectory(ownerExtKey, adminStorage);

    } else
    if (!item.isFormField() &&
item.getFieldName().equals("file")) {

        String fullName = item.getName();
        String directoryPath =
getDirectoryFromLongName(fullName);
        String fileName = getFileNameFromLongName(fullName);

        Map<String, MCMSFile> tempSubFolders =
checkAndCreateSubDirectory(directoryPath, ownerFile);
        subfolderFiles.putAll(tempSubFolders);

        // delete the existing file
        MCMSFile existingFile =
getFileFromDirectory(subfolderFiles.get(directoryPath) , fileName);
        if (existingFile != null)
        {
            deleteFile(existingFile, 0);
        }

        // create the file
        MCMSFile file = new MCMSFile();
        file.setExternalKey(adminStorage.getUniqueId());
        file.setFileName(fileName);

file.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).getTime()
);

        file.setDirectoryFlag(0);

        addFileToDirectory(subfolderFiles.get(directoryPath),
file, item.openStream());

    }
}

returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(null));
    httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
    } catch (Exception e)
    {
        Map<String, String> errors = new HashMap<String, String>();
        errors.put("", MCMSErrors.WB_CANT_UPDATE_RECORD);
        httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
    }
}

public void upload(HttpServletRequest request, HttpServletResponse response,
String requestUri) throws MCMSException
{
    org.json.JSONObject returnJson = new org.json.JSONObject();
    MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
    if (!isRequestAuthenticated(authenticationResult))
    {
        httpServletToolbox.writeBodyResponseAsJson(response, returnJson,
null, authenticationResult);
    }
}

```

```

        return ;
    }

    try
    {
        ServletFileUpload upload = new ServletFileUpload();
        upload.setHeaderEncoding("UTF-8");
        FileItemIterator iterator = upload.getItemIterator(request);
        MCMSFile ownerFile = null;

        while (iterator.hasNext()) {
            FileItemStream item = iterator.next();
            if (item.isFormField() &&
item.getFieldName().equals("ownerExtKey"))
            {
                String ownerExtKey = Streams.asString(item.openStream());
                ownerFile = getDirectory(ownerExtKey, adminStorage);
            } else
            if (!item.isFormField() && item.getFieldName().equals("file")) {
                InputStream stream = item.openStream();
                MCMSFile wbFile = null;

                String fileName = getFileNameFromLongName(item.getName());

                if (request.getAttribute("key") != null)
                {
                    // this is an upload as update for an existing file
                    String key = (String)request.getAttribute("key");
                    wbFile = adminStorage.get(key, MCMSFile.class);

                    ownerFile = getDirectory(wbFile.getOwnerExtKey(),
adminStorage);

                    //old file need to be deleted from cloud
                    String oldFilePath = wbFile.getBlobKey();
                    if (oldFilePath != null && oldFilePath.length()>0)
                    {
                        // delete only if the blob key is set
                        MCMSFilePath oldCloudFile = new
MCMSFilePath(PUBLIC_BUCKET, oldFilePath);
                        cloudFileStorage.deleteFile(oldCloudFile);
                    }
                } else
                {
                    // this is a new upload
                    wbFile = new MCMSFile();
                    wbFile.setExternalKey(adminStorage.getUniqueId());
                }

                wbFile.setFileName(fileName);

                wbFile.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).getTime
());

                wbFile.setDirectoryFlag(0);
                addFileToDirectory(ownerFile, wbFile, stream);
            }
        }

        returnJson.put(DATA, jsonObjectConverter.JSONFromObject(null));
        httpServletToolbox.writeBodyResponseAsJson(response, returnJson, null,
authenticationResult);
    } catch (Exception e)
    {

```

```

        Map<String, String> errors = new HashMap<String, String>();
        errors.put("", MCMSErrors.WB_CANT_UPDATE_RECORD);
        httpServletToolbox.writeBodyResponseAsJson(response,
        jsonObjectConverter.JSONObjectFromMap(null), errors);
    }
}

    public void createDir(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSEException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String jsonRequest = httpServletToolbox.getBodyText(request);
            MCMSFile wbFile =
jsonObjectConverter.objectFromJSONString(jsonRequest, MCMSFile.class);
            Map<String, String> errors = validator.validateCreate(wbFile);

            if (errors.size()>0)
            {
                httpServletToolbox.writeBodyResponseAsJson(response, "{}",
errors);
                return;
            }
            wbFile.setAdjustedContentType(null);
            wbFile.setBlobKey(null);
            wbFile.setContentType(null);
            wbFile.setDirectoryFlag(1);
            wbFile.setHash(0L);
            wbFile.setPublicUrl(null);
            wbFile.setSize(0L);

            wbFile.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).getTime
());

            wbFile.setExternalKey(adminStorage.getUniqueId());
            wbFile.setVersion(UUID.randomUUID().toString());
            MCMSFile newFile = adminStorage.addWithKey(wbFile);

            MCMSResource resource =
new MCMSResource(newFile.getExternalKey(),
newFile.getFileName(),
MCMSResource.FILE_TYPE);
            try
            {
                adminStorage.addWithKey(resource);
            } catch (Exception e)
            {
                // do not propagate further
            }
            returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(newFile));
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

        } catch (Exception e)

```

```

        {
            Map<String, String> errors = new HashMap<String, String>();
            errors.put("", MCMSErrors.WB_CANT_CREATE_RECORD);
            httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
        }
    }

    public void update(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String key = (String)request.getAttribute("key");
            String jsonRequest = httpServletToolbox.getBodyText(request);
            MCMSFile wbfile =
(MCMSFile)jsonObjectConverter.objectFromJSONString(jsonRequest, MCMSFile.class);
            wbfile.setExternalKey(key);
            Map<String, String> errors = validator.validateUpdate(wbfile);

            if (errors.size()>0)
            {
                httpServletToolbox.writeBodyResponseAsJson(response, "",
errors);
                return;
            }
            MCMSFile existingFile = adminStorage.get(key, MCMSFile.class);

            existingFile.setLastModified(Calendar.getInstance(TimeZone.getTimeZone("GMT")).getTime());

            existingFile.setAdjustedContentType(wbfile.getAdjustedContentType());
            existingFile.setVersion(UUID.randomUUID().toString());
            MCMSFile newFile = adminStorage.update(existingFile);

            returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(newFile));
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
        } catch (Exception e)
        {
            Map<String, String> errors = new HashMap<String, String>();

            errors.put("", MCMSErrors.WB_CANT_UPDATE_RECORD);
            httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
        }
    }

    public void delete(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {

```

```

        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String key = (String)request.getAttribute("key");
            MCMSFile tempFile = adminStorage.get(key, MCMSFile.class);

            deleteFile(tempFile, 0);

            MCMSFile param = new MCMSFile();
            param.setExternalKey(key);

            returnJson.put(DATA,
jsonObjectConverter.JSONFromObject(param));
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

        } catch (Exception e)
        {
            Map<String, String> errors = new HashMap<String, String>();

            errors.put("", MCMSErrors.WB_CANT_DELETE_RECORD);
            httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
        }
    }

    public void getAll(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            Map<String, Object> additionalInfo = new HashMap<String,
Object> ();

            String sortParamDir =
request.getParameter(SORT_PARAMETER_DIRECTION);
            String sortParamProp =
request.getParameter(SORT_PARAMETER_PROPERTY);

            String ownerParam = "ownerExtKey";
            String ownerValue = request.getParameter("parent");
            if (ownerValue == null)
                ownerValue="";
            else
                ownerValue = ownerValue.trim();

```



```

List<MCMSFile> files = null;

if (sortParamDir != null && sortParamProp != null)
{
    if (sortParamDir.equals(SORT_PARAMETER_DIRECTION_ASC))
    {
        additionalInfo.put(SORT_PARAMETER_DIRECTION,
SORT_PARAMETER_DIRECTION_ASC);
        additionalInfo.put(SORT_PARAMETER_PROPERTY,
sortParamProp);

        files = adminStorage.queryWithSort(MCMSFile.class,
ownerParam , AdminQueryOperator.EQUAL, ownerValue, sortParamProp,
AdminSortOperator.ASCENDING);

    } else if
(sortParamDir.equals(SORT_PARAMETER_DIRECTION_DSC))
    {
        additionalInfo.put(SORT_PARAMETER_DIRECTION,
SORT_PARAMETER_DIRECTION_DSC);
        additionalInfo.put(SORT_PARAMETER_PROPERTY,
sortParamProp);

        files = adminStorage.queryWithSort(MCMSFile.class,
ownerParam , AdminQueryOperator.EQUAL, ownerValue, sortParamProp,
AdminSortOperator.DECENDING);

    } else
    {
        files = adminStorage.query(MCMSFile.class,
ownerParam, AdminQueryOperator.EQUAL, ownerValue);
    } else
    {
        files = adminStorage.query(MCMSFile.class, ownerParam ,
AdminQueryOperator.EQUAL, ownerValue);
    }

List<MCMSFile> result = filterPagination(request, files,
additionalInfo);
for(MCMSFile wbFile: result)
{
    setPublicFilePath(wbFile, cloudFileStorage);
}
MCMSFile ownerFile = null;
if (ownerValue.length()>0)
{
    // get the owner parent info
    List<MCMSFile> queryRes =
adminStorage.query(MCMSFile.class, "externalKey", AdminQueryOperator.EQUAL,
ownerValue);

    if (queryRes.size() == 1)
    {
        ownerFile = queryRes.get(0);
    }
}
returnJson.put(DATA,
jsonObjectConverter.JSONArrayFromListObjects(result));
org.json.JSONObject additionalDataJson =
jsonObjectConverter.JSONObjectFromMap(additionalInfo);
additionalDataJson.put("owner",
jsonObjectConverter.JSONObjectFromObject(ownerFile));

```

```

        String ownerOfOwner = "";
        if (ownerFile != null)
        {
            ownerOfOwner = ownerFile.getOwnerExtKey();
        }
        String dirPath = getDirectoryFullPath(ownerOfOwner, adminStorage);
        additionalDataJson.put("ownerFullDirectoryPath", dirPath);

        returnJson.put(ADDITIONAL_DATA, additionalDataJson);
        httpServletToolbox.writeBodyResponseAsJson(response, returnJson,
        null, authenticationResult);

    } catch (Exception e)
    {
        Map<String, String> errors = new HashMap<String, String>();

        errors.put("", MCMSErrors.WB_CANT_GET_RECORDS);
        httpServletToolbox.writeBodyResponseAsJson(response,
        jsonObjectConverter.JSONObjectFromMap(null), errors);
    }
}

private static void setPublicFilePath(MCMSFile wbFile, MCMSFileStorage
cloudFileStorage)
{
    if (wbFile.getDirectoryFlag() == null || wbFile.getDirectoryFlag() ==
0)
    {
        if (wbFile.getBlobKey() != null)
        {
            wbFile.setPublicUrl(cloudFileStorage.getPublicFileUrl(new
MCMSFilePath(PUBLIC_BUCKET, wbFile.getBlobKey())));
        }
    }
}

private org.json.JSONObject get(HttpServletRequest request, MCMSFile
wbFile) throws MCMSException
{
    try
    {
        setPublicFilePath(wbFile, cloudFileStorage);
        org.json.JSONObject returnJson = new org.json.JSONObject();
        returnJson.put(DATA,
        jsonObjectConverter.JSONFromObject(wbFile));

        org.json.JSONObject additionalData = new org.json.JSONObject();

        String dirPath = getDirectoryFullPath(wbFile.getOwnerExtKey(),
adminStorage);
        additionalData.put("ownerFullDirectoryPath", dirPath);

        String includeLinks = request.getParameter("include_links");
        if (includeLinks != null && includeLinks.equals("1"))
        {
            List<MCMSUri> uris = adminStorage.query(MCMSUri.class,
"resourceExternalKey", AdminQueryOperator.EQUAL, wbFile.getExternalKey());
            org.json.JSONArray arrayUris =
            jsonObjectConverter.JSONArrayFromListObjects(uris);
            additionalData.put("uri_links", arrayUris);
        }
        returnJson.put(ADDITIONAL_DATA, additionalData);
        return returnJson;
    }
}

```

```

        } catch (Exception e)
        {
            throw new MCMSEException("cannot get file details", e);
        }
    }

    public void get(HttpServletRequest request, HttpServletResponse response,
String requestUri) throws MCMSEException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String key = (String)request.getAttribute("key");
            MCMSFile wbFile = adminStorage.get(key, MCMSFile.class);

            returnJson = get(request, wbFile);
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);

        } catch (Exception e)
        {
            Map<String, String> errors = new HashMap<String, String>();

            errors.put("", MCMSErrors.WB_CANT_GET_RECORDS);
            httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
        }
    }

    public void getExt(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSEException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
            return ;
        }

        try
        {
            String extKey = (String)request.getAttribute("key");
            List<MCMSFile> wbFiles = adminStorage.query(MCMSFile.class,
"externalKey", AdminQueryOperator.EQUAL, extKey);
            MCMSFile wbFile = (wbFiles.size()>0) ? wbFiles.get(0) : null;
            returnJson = get(request, wbFile);
            httpServletToolbox.writeBodyResponseAsJson(response,
returnJson, null, authenticationResult);
        } catch (Exception e)
        {
            Map<String, String> errors = new HashMap<String, String>();

```

```

        errors.put("", MCMSErrors.WB_CANT_GET_RECORDS);
        httpServletToolbox.writeBodyResponseAsJson(response,
jsonObjectConverter.JSONObjectFromMap(null), errors);
    }

    public void downloadResource(HttpServletRequest request,
HttpServletResponse response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        }

        try
        {
            String key = (String)request.getAttribute("key");
            MCMSFile wbfile = adminStorage.get(key, MCMSFile.class);
            MCMSFilePath cloudFile = new MCMSFilePath(PUBLIC_BUCKET,
wbfile.getBlobKey());
            InputStream is = cloudFileStorage.getFileContent(cloudFile);
            response.setContentType(wbfile.getContentType());

            response.setHeader("Content-Disposition", "attachment;
filename=\"\" + wbfile.getFileName() + \"\");
            response.setContentLength(wbfile.getSize().intValue());
            OutputStream os = response.getOutputStream();
            IOUtils.copy(is, os);
            os.flush();
            IOUtils.closeQuietly(is);
            IOUtils.closeQuietly(os);
        } catch (Exception e)
        {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
        }
    }

    public void serveResource(HttpServletRequest request, HttpServletResponse
response, String requestUri) throws MCMSException
    {
        org.json.JSONObject returnJson = new org.json.JSONObject();
        MCMSAuthenticationResult authenticationResult =
this.handleAuthentication(request);
        if (! isRequestAuthenticated(authenticationResult))
        {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        }

        InputStream is = null;
        OutputStream os = null;
        try
        {
            String key = (String)request.getAttribute("key");
            MCMSFile wbfile = adminStorage.get(key, MCMSFile.class);
            MCMSFilePath cloudFile = new MCMSFilePath(PUBLIC_BUCKET,
wbfile.getBlobKey());
            is = cloudFileStorage.getFileContent(cloudFile);
            response.setContentType(wbfile.getAdjustedContentType());
            response.setContentLength(wbfile.getSize().intValue());

```

```
        os = response.getOutputStream();
        IOUtils.copy(is, os);
        os.flush();
    } catch (Exception e)
    {
        response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
    }
    finally
    {
        IOUtils.closeQuietly(is);
        IOUtils.closeQuietly(os);
    }
}
```