

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

на тему: Розробка системи управління контентом
web-порталу навчального закладу

Виконав: студент групи 4142

_____ Чугунов В.А.
(підпис)

Керівник роботи:

_____ Д.ф-м.н., проф.
(посада, науковий ступінь вчене звання)

_____ Ушкац М.В.
(підпис)

м. Миколаїв – 2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

к.т.н, доц. Гайда А.Ю.

" ____ " _____ 2022 р.

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

Студенту Чугунову В'ячеславу Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка системи управління контентом

web-порталу навчального закладу

Керівник роботи Д.ф-м.н., проф. Ушкац М.В.

Затверджені наказом ректора № 269-уч від "18" травня 2022 року.

2. Термін подання роботи: 20 червня 2022 р.

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

4. Перелік питань, що належать до розробки (найменування розділів) Аналіз стану інтернет-технологій для створення систем управління контентом, існуючих рішень та засобів розробки, Розробка проектних рішень по системі, Розробка робочої документації системи управління контентом web-порталу МПТУ, Охорона праці

5. **Перелік презентаційних матеріалів** Content Management System, Структура МПТУ, 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	Ушкац М.В.		
2	Ушкац М.В.		
3	Ушкац М.В.		
4			

7. Дата видачі завдання 14 лютого 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Студент

(підпис)

Чугунов В.А

(ПІБ)

Керівник роботи

(підпис)

Ушкац М.В.

(ПІБ)

Анотація

Представлена робота присвячена розробці інтернет-технологій для створення системи управління контентом web-порталу Миколаївського міського Палацу творчості учнів 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 _____ аркушах машинописного тексту, містить _____ рисунків, _____ таблиць, _____ додатків.

Abstract

The work is devoted to the development of Internet technologies for creating a content management system for the web-portal of the Mykolaiv City Palace of Pupil's Creativity 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 _____ sheets of the typewritten text, contains _____ figures, _____ tables, _____ applications.

Вступ.....	
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.....	
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.....	
3 Розробка робочої документації системи управління контентом web-порталу МПТУ.....	
4 Охорона праці.....	
4.1 Аналіз небезпечних і шкідливих факторів в університеті.....	
4.2 Розрахунок системи штучного освітлення приміщення.....	
4.3 Розробка заходів щодо зменшення впливу шкідливих факторів.....	
4.4 Розробка заходів щодо зменшення впливу шкідливих факторів.....	
Висновки.....	
Список використаних джерел.....	

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 для організації веб-сайтів чи інших інформаційних ресурсів в Інтернеті чи окремих комп'ютерних мережах. Існують різноманітні системи управління контентом, серед яких зустрічаються платні та безкоштовні, збудовані за різними технологіями.

Суть роботи CMS полягає в поділі вмісту сайту і його дизайну. Користувачеві надається можливість вибрати шаблон – заготовку, якій заздалегідь визначено оформлення сторінки, і залишається тільки заповнити її потрібною інформацією. Більшість систем управління контентом ґрунтується на використанні візуального редактора (WYSIWYG - від англ. What You See Is What You Get - «що бачиш, то і отримаєш») - програми, що дозволяє за допомогою інтуїтивно зрозумілого інтерфейсу додавати або змінювати інформацію на сайті. Варто відзначити, що сайт не є сукупністю сторінок як таких, а формується динамічно. Доданий контент зберігається в базі даних, наприклад, MySQL, і використовується при генерації сторінки після отримання відповідного запиту з боку клієнта

Післядипломна освіта – це система підвищення кваліфікації осіб, котрі вже мають вищу освіту. Післядипломна педагогічна освіта надає можливість підвищити кваліфікацію вчителів або іншим працівникам галузі освіти. Для обміну інформацією стосовно навчального процесу (новини, лекції, статті, виступи, вебінари та їх записи) може використовуватись web-портал, оскільки це найшвидший та зручний спосіб обмінюватись інформацією.

Актуальність дипломної роботи полягає в тому, що досі не існує зручної та ефективної системи управління контентом для інститутів післядипломної педагогічної освіти. Існуючі технологічні рішення не враховують особливості контенту (їх зміст, формат, структура), що використовується закладами вищої освіти, або є занадто складними для використання користувачами без спеціальної підготовки[1].

Мета дипломної роботи – розроблення інтернет технології для створення системи управління контентом web-порталу Миколаївського міського палацу творчості учнів. Для досягнення мети були поставлені наступні завдання:

- Провести огляд систем управління контентом;
- Провести огляд існуючих рішень для систем управління контентом;
- Проаналізувати існуючі інтернет-технології для створення систем управління контенту;
- Проаналізувати основні інструменти для розробки web-порталів;
- Розробити тестовий web-портал з можливістю використання системи управління контентом.

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 (контентом) (англ. Content management system, CMS) – інформаційна система або комп'ютерна програма, яка використовується для забезпечення і організації спільного процесу створення, редагування і управління вмістом, інакше – контентом (від англійського content).

Основними функціями CMS є надання інструментів для створення вмісту, організація спільної роботи над вмістом; управління вмістом: зберігання, контроль версій, дотримання режиму доступу, управління потоком документів; публікація вмісту; подання інформації у вигляді, зручному для навігації, пошуку.

В системі керування вмістом можуть перебувати найрізноманітніші дані: документи, фільми, фотографії, номери телефонів, наукові дані і так далі. Така система часто використовується для зберігання, управління, перегляду і публікації документації. Контроль версій є одним з основних її переваг, коли вміст змінюється групою осіб.

Як правило, CMS використовуються для таких сайтів:

- блог, форум (WordPress, phpBB, vBulletin);
- інтернет-магазин (Magento, OpenCart, osCommerce);
- соціальні мережі (InstantCMS, Social Engine);
- персональні сайти (WordPress, Monstra);
- корпоративні сайти (Joomla, Drupal);
- портали (DLE, Drupal).

Тим не менше, більшість CMS можуть бути гнучко налаштовані і можуть бути використані для розробки сайтів різної спрямованості. Наприклад, найбільш популярним і універсальним варіантом є WordPress, на

якому можливо створити практично будь-який проект: від особистого сайту до великого портала або інтернет-магазину.

1.2 Різновиди систем управління контентом

У загальному випадку системи управління контентом поділяються на:

- Систему управління контентом масштабу підприємства (англ. Enterprise Content Management System)
- Система управління веб-контентом (англ. Web Content Management System)

Управління корпоративним контентом (англ. Enterprise content management, ECM) – управління цифровими документами та іншими типами контенту, а також їх зберігання, обробка і доставка в рамках організації. Керована інформація (контент) передбачає слабку структурованість: це можуть бути файли різних форматів, електронні документи з різними наборами полів[7].

ECM-система - програмне забезпечення для управління корпоративним контентом. Часто ECM-системи вважаються особливим різновидом систем управління вмістом. На пострадянському просторі поняття ECM-системи часто трактується як подібне з поняттям «системи електронного документообігу» (СЕД).

Система управління веб-вмістом (Web Content Management System або WCMS) - програмний комплекс, що надає функції створення, редагування, контролю та організації веб-сторінок. WCMS часто використовуються для створення блогів, особистих сторінок і інтернет-магазинів і націлені на користувачів, мало знайомих з програмуванням.

В силу того, що ECMS мають глибоку внутрішню класифікацію за предметними галузями (HRM, DMS, CRM, ERP), термін CMS замінив собою WCMS, перетворившись на синонім системи управління сайтами. Подібні CMS дозволяють управляти текстовим і графічним наповненням веб-сайту, надаючи користувачу інтерфейс для роботи з вмістом сайту, зручні

інструменти зберігання і публікації інформації, автоматизуючи процеси розміщення інформації в базах даних і її видачі в HTML.

Існує безліч готових систем керування контентом сайту, в тому числі і безкоштовних. Їх можна розділити на три типи за способом роботи:

- Генерація сторінок за запитом. Системи такого типу працюють на основі зв'язки «Модуль редагування → База даних → Модуль представлення». Модуль представлення генерує сторінку з вмістом при запиті на нього, на основі інформації з бази даних. Інформація в базі даних змінюється за допомогою модуля редагування. Сторінки заново створюються сервером при кожному запиті, що в свою чергу створює додаткове навантаження на системні ресурси. Навантаження може бути багато разів знижена при використанні коштів кешування, які є в сучасних веб-серверах;
- Генерація сторінок при редагуванні. Системи цього типу служать для редагування сторінок, які при внесенні змін до змісту сайту створюють набір статичних сторінок. При такому способі в жертву приноситься інтерактивність між відвідувачем і вмістом сайту;
- Змішаний тип. Як зрозуміло з назви, поєднує в собі переваги перших двох. Може бути реалізований шляхом кешування - модуль представлення генерує сторінку один раз, надалі вона в кілька разів швидше завантажується з кешу. Кеш може оновлюватися як автоматично, після закінчення деякого терміну часу або при внесенні змін до певні розділи сайту, так і вручну по команді адміністратора. Інший підхід - збереження певних інформаційних блоків на етапі редагування сайту і збірка сторінки з цих блоків при запиті відповідної сторінки користувачем.

1.3 Аналіз предметної області

Позашкільна освіта є невід'ємним складником системи освіти, визначеної Конституцією України, законами України «Про освіту», «Про

позашкільну освіту», і спрямована на розвиток здібностей дітей та молоді у сфері освіти, науки, культури, фізичної культури і спорту, технічної та іншої творчості, здобуття ними первинних професійних знань, вмінь і навичок, необхідних для їх соціалізації, подальшої самореалізації та/або професійної діяльності.

Позашкільна освіта здобувається громадянами у закладах позашкільної освіти та інших закладах освіти як центрах позашкільної освіти незалежно від підпорядкування, типів і форм власності, в тому числі в школах соціальної реабілітації, міжшкільних навчально-виробничих комбінатах, закладах професійної (професійно-технічної) та фахової передвищої.

Іноземці та особи без громадянства, які перебувають в Україні на законних підставах, здобувають позашкільну освіту в порядку, встановленому для громадян України.

Система позашкільної освіти – освітня підсистема, що включає:

- державні, комунальні, приватні заклади позашкільної освіти;
- інші заклади освіти, як-от центри позашкільної освіти (заклади загальної середньої освіти, незалежно від підпорядкування, типів і форм власності, зокрема, школи соціальної реабілітації, міжшкільні навчально-виробничі комбінати, заклади професійної (професійно-технічної) та фахової передвищої освіти);
- гуртки, секції, клуби, культурно-освітні, спортивно-оздоровчі, науково-пошукові об'єднання на базі закладів загальної середньої освіти, навчально-виробничих комбінатів, закладів професійної та фахової передвищої освіти;
- клуби та об'єднання за місцем проживання незалежно від підпорядкування, типів і форм власності;
- культурно-освітні, фізкультурно-оздоровчі, спортивні та інші заклади освіти та установи;
- фонди, асоціації, діяльність яких пов'язана з функціонуванням позашкільної освіти;

- відповідні органи управління позашкільною освітою та науково-методичні установи;
- інші суб'єкти освітньої діяльності, що надають освітні послуги у системі позашкільної освіти.

Структура Миколаївського міського Палацу творчості учнів:

1. Директор.
2. Заступники директора.
3. Відділи Палацу:
 - 3.1. Відділ декоративно-ужиткового та образотворчого мистецтва.
 - 3.2. Відділ технічної творчості, туристсько-краєзнавчої та екологічної роботи.
 - 3.3. Відділ художньо-естетичного виховання.
 - 3.4. Відділ організаційно-масової роботи.
4. Лабораторії:
 5. Бухгалтерська служба.
 6. Бібліотека.
 7. Юрист.

1.4 Аналіз існуючих рішень для систем управління контентом

1.4.1 Система управління контентом Joomla!

Joomla! – система управління контентом, написана на мовах PHP і JavaScript, що використовує в якості сховища бази даних СУБД MySQL або інші стандартні промислові реляційні СУБД. Є вільним програмним забезпеченням, поширюваним під ліцензією GNU GPL.

CMS Joomla! включає в себе мінімальний набір інструментів при початковій установці, який доповнюється в міру необхідності. Це знижує захащення адміністративної панелі непотрібними елементами, а також знижує навантаження на сервер і економить місце на хостингу. Joomla! дозволяє відображати інтерфейс фронтальної та адміністративної частини

будь-якою мовою. Каталог розширень містить безліч мовних пакетів, які встановлюються штатними засобами адміністрування[17].

Структурно система являє собою ядро, до якого приєднуються компоненти. Компоненти можуть складатися по функціональному призначенню з модуля, плагіна і/або шаблону: модулі – елементи, які відображаються на сторінці користувачів у вигляді окремих блоків; плагіни – елементи, які відображаються на адміністративній панелі інструментів і надають додаткову функціональність; шаблони – елемент, який відповідає за уніфіковане оформлення всього сайту. Створення власного розширення здійснюється за допомогою написання плагіна на мові PHP з використанням для взаємодії API Joomla!.

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

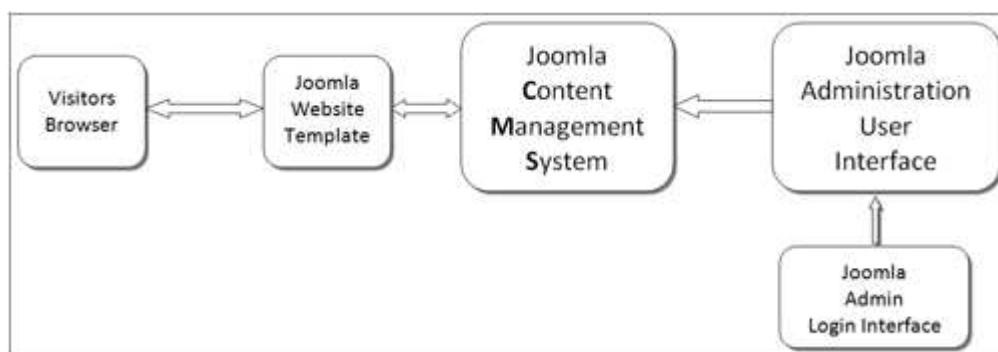


Рисунок 1.1 – Схема сайту побудованого за допомогою Joomla!

1.4.2 Система управління контентом WordPress

WordPress – система керування вмістом з відкритим кодом, яка через свою простоту в установленні та використанні широко застосовується для створення веб-сайтів. Сфера використання — від блогів до складних веб-сайтів. Вбудована система тем і плагінів у поєднанні з вдалою архітектурою дозволяє конструювати на основі WordPress практично будь-які веб-проекти. Написана мовою програмування PHP з використанням бази даних MySQL.

WordPress - це одна з найвідоміших систем управління контентом. На її базі створено вже понад 60 мільйонів сайтів по всьому світу, що складає 40% від усіх сайтів, заснованих на CMS. WordPress в першу чергу безкоштовна система з відкритим вихідним кодом. У неї входить безліч різних плагінів,

віджетів і тем, які дозволяють повністю налаштувати сайт під себе. WordPress ідеально підходить для невеликих блогів і сайтів, при цьому будь-які технічні навички не обов'язкові. Ось чому WordPress найчастіше радять новачкам. Можна оновити налаштування або швидко опублікувати контент і все це з дотриманням безпеки даних. Але незважаючи на всі переваги WP, у неї є свої недоліки: з огляду на простоту використання WordPress, система має обмеження при більш глибокому редагуванні зовнішнього вигляду сайту – потрібна допомога кваліфікованих спеціалістів (дизайнерів або веб-розробників), які допоможуть змінити структуру сторінок [17].

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

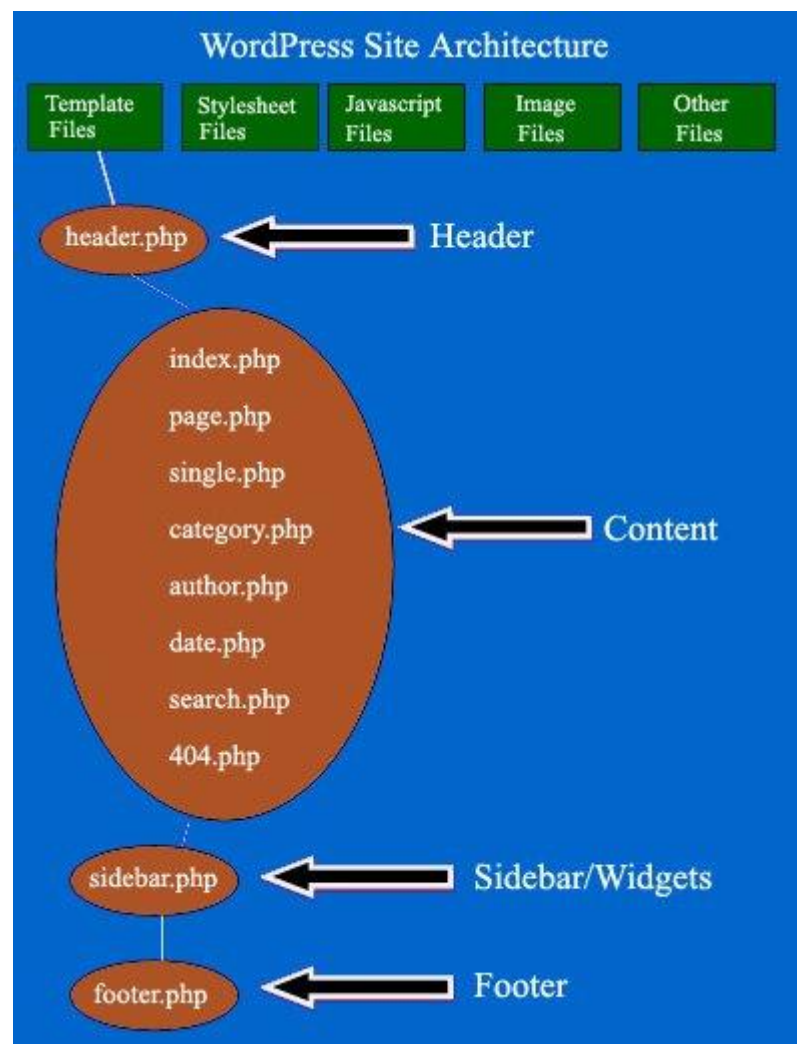


Рисунок 1.2 – Архітектура сайту побудованого за допомогою WordPress

1.4.3 Система управління контентом Drupal

Drupal – система управління вмістом (CMS), яка використовується також як каркас для веб-додатків (CMF), написана на мові PHP і використовує як сховище даних реляційну базу даних (підтримуються MySQL, PostgreSQL та інші). Drupal є вільним програмним забезпеченням, захищеним ліцензією GPL, і розвивається зусиллями ентузіастів зі всього світу.

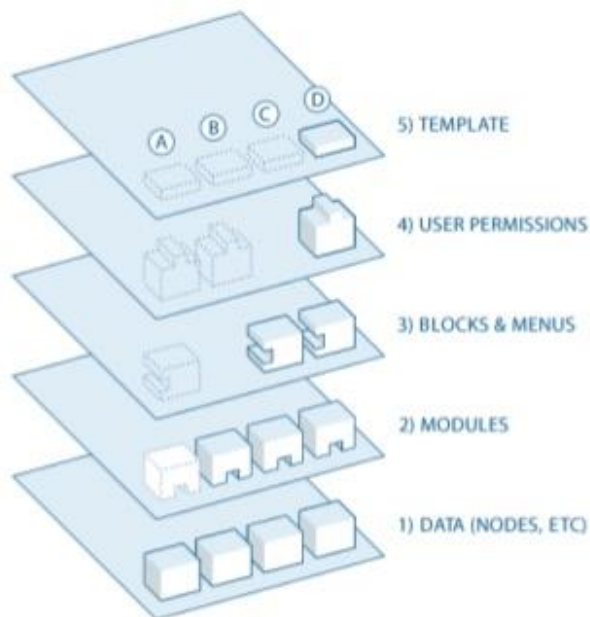
В Drupal пропонується гнучка схема організації структури сайту на основі таксономії. Таксономія - механізм, що дозволяє створювати будь-яку кількість тематичних категорій для вмісту сайту і асоціювати їх з модулями, що забезпечують введення і виведення інформації. Категорії можуть представляти плоскі або ієрархічні списки, або складні структури, де елемент може мати декілька «батьків» і кілька дочірніх елементів. За допомогою подібної схеми одними і тими ж модулями можлива організація різних варіантів структуризації вмісту. Наприклад, легко створюється наскрізний список «ключових слів» для всіх документів сайту і т. П.

Інша парадигма з'явилася зі створенням в Drupal розширення Content Construction Kit (ССК). ССК дозволяє доповнювати документи новими полями різних типів - від полів введення URL і email, до полів зберігання і відображення мультимедійних файлів. Також за допомогою додаткових модулів до ССК (наприклад, Node reference) можна організувати зв'язку між документами, не використовуючи механізм таксономії.

На рис. 1.3 наведено структуру системи управління контентом Drupal.

System Structure

Systematically structured layers



Layers

5. Template
4. User Permissions
3. Blocks & Menu
2. Modules
1. Data

Рисунок 1.3 – структура системи управління контентом Drupal

1.4.4 Система управління контентом UMI.CMS

UMI.CMS – комерційна мультисайтова система управління контентом, створена командою російських розробників «Юмісофт». У масовий продаж надійшла в 2007 році. Написана на мові програмування PHP і використовує базу даних MySQL.

За твердженням прес-служби компанії Юмісофт станом на травень 2015 року на платформі UMI.CMS працюють близько 800 000 сайтів різного масштабу: сайти-візитки, каталоги товарів, інтернет-магазини, портали і т.д.

UMI.CMS є кросплатформним програмним забезпеченням. Рекомендованої типовою конфігурацією для системи є веб-сервер Apache під операційною системою Linux, яка доступна практично на будь-якому віртуальному хостингу. Виробник UMI.CMS гарантує роботу системи на цій конфігурації і надає по ній підтримку.

Також до сервера Apache повинні бути встановлені і підключені модулі `mod_rewrite` і `mod_auth` (для інтеграції з 1С наявність модуля `mod_auth` обов'язково). В налаштуваннях повинна бути включена директива `AllowOverride` зі значенням "All" або "Options FileInfo AuthConfig", і при цьому директива `AccessFileName` повинна мати значення ".htaccess".

PHP повинен бути зібраний з підключеними бібліотеками `zLib`, `GD`, `libxml`, `mb_string`, `iconv`, `xsl`, `SimpleXML`, `xmlreader`. Режим `safe_mode` повинен бути вимкнений. Повинна бути дозволена опція `allow_url_fopen` або підключена бібліотека `cURL`. Об'єм пам'яті, виділеної на виконання скриптів (директива `memory_limit` в файлі `php.ini`), повинен становити не менше 32 мегабайт (установка і робота при 16 мегабайт можлива, але не гарантується). Починаючи з версії 2.8.4.4 і більш старших потрібна підтримка JSON. Це необхідно для роботи нового файлового менеджера `elFinder` і інтеграції з Яндекс.Метрика).

CMS використовує базу даних MySQL версії 5 і вище, з підтримкою InnoDB (робота на 4.1 і вище можлива, але не рекомендується). Дозволяє використовувати інші джерела даних. Підтримує технологію динамічного завантаження змін AJAX [17].

1.4.5 Система управління контентом Ніпро CMS

Ніпро CMS це система з відкритим кодом написана на мові програмування Java. Чиста модульна архітектура Ніпро і підтримка відкритих стандартів робить її ідеальним рішенням для гнучкого розвитку та відкритої інтеграції. Наголос Ніпро на відокремленні вмісту від логіки презентації дозволяє контент-редакторам створювати вміст один раз, а потім публікувати його на веб-сайтах, мобільних сайтах, службах Facebook, REST, цифрових журналах, мобільних додатках тощо. Ніпро дає змогу веб-майстру створювати нові канали або налаштовувати існуючі шаблони сторінок за допомогою менеджера каналів та шаблону. Використовуючи модуль релевантності, певний вміст може бути орієнтований на конкретних відвідувачів на основі поточного

контексту відвідувача, попередньої особистої історії та відповідних осіб, створених маркетинговою групою.

Nippo CMS підтримує безліч відкритих стандартів, таких як JCR та REST, щоб спростити розробку та інтеграцію з існуючими системами. Рівень доставки Nippo (HST) забезпечує інструменти для швидкого розвитку веб-сайтів та інших каналів, таких як кінцеві точки REST або RSS-канали. І користувацький інтерфейс Nippo CMS, і рівень доставки дуже добре налаштовуються і розширюються.

На рис. 1.4 наведено структуру системи управління контентом Nippo CMS.

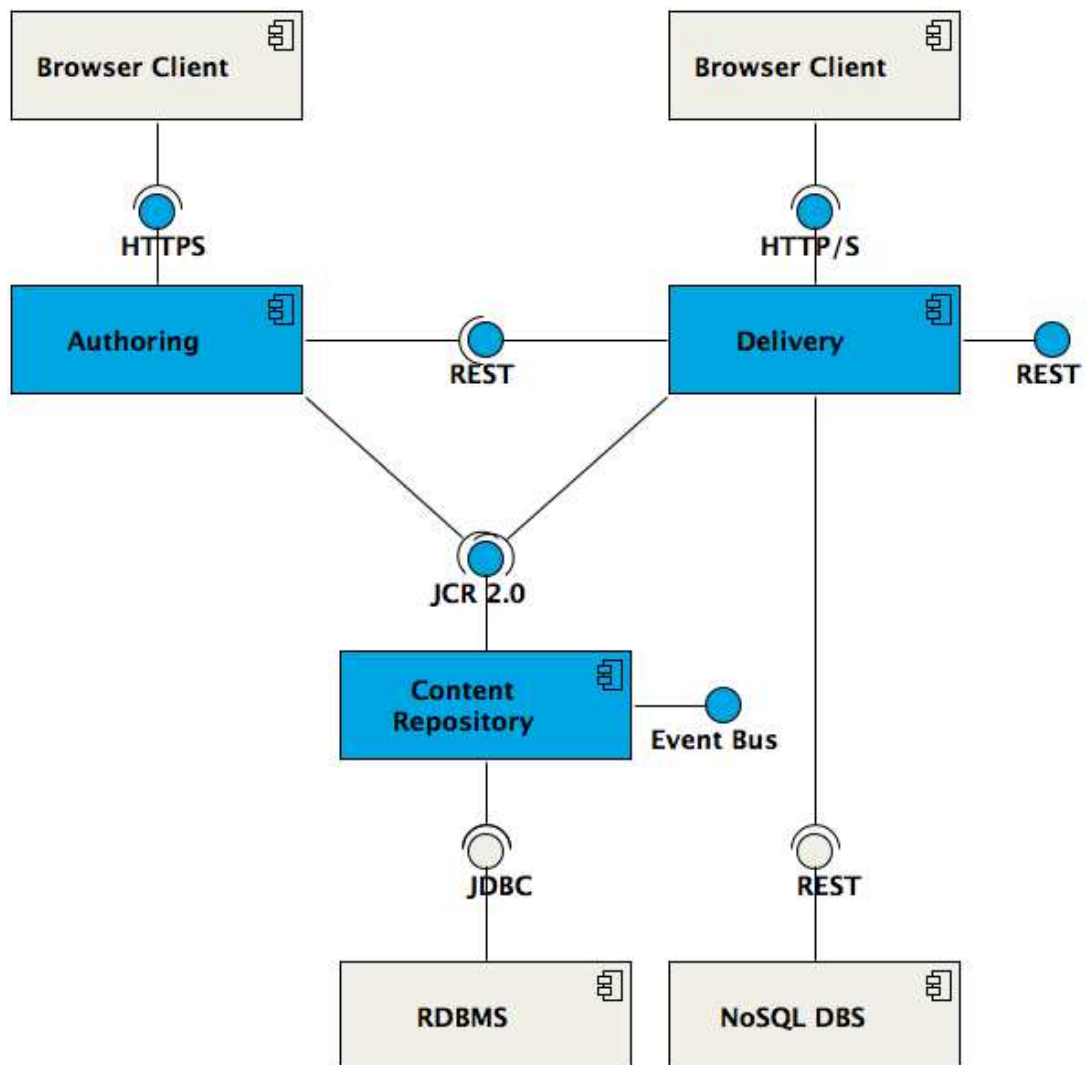


Рисунок 1.4 – структура системи управління контентом Nippo CMS

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	UMI.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 рішення мають переваги такі, як асортимент готових шаблонів та велика кількість вбудованих модулів, але такі системи зазвичай мають проблеми з безпекою даних, швидкістю роботи та підтримкою користувачів, не дивлячись на досить великі спільноти користувачів систем. Комерційні системи є більш захищеними та майже завжди мають цілодобову підтримку користувачів, але ціни можуть бути занадто великими. Тому є доцільним розглянути варіант створення індивідуального рішення системи управління контентом веб-порталу МОІППО.

1.5 Вибір технологій та середовища розробки

Дана веб-орієнтована система є безкоштовною, отже всі технології та середовище розробки повинні бути вільними для користування, тобто не потребувати фінансових витрат. Для цього було вирішено використовувати мову програмування Java, адже вона не потребує придбання ліцензій і надає потужний механізм для розробки веб-додатків.

Java - об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи [6].

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як платформи-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За

необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Досить скоро майже всі найпопулярніші тогочасні веб-оглядачі отримали можливість запускати «безпечні» для системи Java-аплети всередині веб-сторінок. “Sun Microsystems” випустила Java 2 (спершу під назвою J2SE 1.2), де було реалізовано декілька конфігурацій для різних типів платформ. Наприклад, J2EE призначалася для створення додатків застосунків, а значно урізана J2ME для приладів з обмеженими ресурсами, таких як мобільні телефони.

Період становлення Java збігся у часі з розквітом міжнародної інформаційної служби World Wide Web. Ця обставина відіграла вирішальну роль у майбутньому Java, оскільки Web теж вимагала платформо-незалежних програм. Як наслідок, були зміщені акценти в розробці Sun з побутової електроніки на програмування для Інтернет.

Початкові цілі:

- Синтаксис мови повинен бути "простим, об'єктно-орієнтовним та звичним";
- Реалізація має бути "безвідмовною та безпечною";
- Повинна зберегтися "незалежність від архітектури та переносимість";
- Висока продуктивність виконання;
- Мова має бути "інтерпретованою, багатонитевою, із динамічним зв'язуванням модулів";

Під “незалежністю від архітектури” мається на увазі можливість програми, на мові Java, працювати на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови [6].

Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just-in-time компіляція (JIT, що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його). Як результаті, така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про оточення виконання та про завантажені класи. До того ж, він може виявляти так звані гарячі точки (англ. hot spots) — частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносимості: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

Apache Commons – це проект Apache зосереджений на всі аспекти повторного використання компонентів Java.

Вихідний код репозиторія Apache Commons можуть перезаписати всі Apache Software Foundation учасники. Всі учасники повинні прочитати керівні принципи, що сприяють розробці. Утиліти проекту Apache Commons лежать в основі таких проектів як Apache Tomcat, Struts, Hibernate та ін.

Згідно зі статутом проекту, проект повинен «створювати і підтримувати пакети, написані на мові Java, призначені для використання в розробці серверних додатків, і спроектовані таким чином, щоб їх можна було використовувати незалежно в будь-якому більшому продукті або фреймворку.

В ієрархії проектів Apache Software Foundation Apache Commons є проектом "верхнього рівня". У свою чергу, Commons розбивається на безліч проектів «другого рівня». Кожен проект другого рівня має свій сайт, каталог в загальному репозиторії вихідного коду, каталог в загальній системі відслідковування помилок і має на меті розробку одного однойменного компонента (бібліотеки) Java. Активні проекти мають кілька постійних учасників, неактивні проекти можуть взагалі не мати постійних учасників.

JavaMail - це Java API, який використовується для відправки та отримання електронної пошти через SMTP, POP3 та IMAP. JavaMail вбудований в платформу Java EE, але також надає додатковий пакет для використання в Java SE.

Spring Security - надає комплексні служби безпеки для корпоративних додатків на базі Java EE. Особлива увага приділяється підтримці проєктів, створених з використанням Spring Framework, який є провідним рішенням Java EE для розробки корпоративного програмного забезпечення.

Люди використовують Spring Security з багатьох причин, але більшість з них залучаються до проєкту після виявлення функцій безпеки специфікації сервлетів Java EE або специфікації EJB не вистачає глибини, необхідної для типових сценаріїв корпоративних додатків. Незважаючи на присутність цих стандартів, важливо визнати, що вони не стерпні на рівні WAR або EAR. Тому, якщо ви переключаете серверні середовища, зазвичай потрібна велика робота по налаштуванню ТВ-тюнерів безпеки вашого застосування в нову цільову середу. Використання Spring Security дозволяє подолати ці проблеми, а також приносить вам десятки інших корисних налаштовуються функцій безпеки.

Існує дві основні галузі безпеки додатків: «аутентифікація» і «авторизація» (або «контроль доступу»). Це дві основні області, на які націлена Spring Security. «Аутентифікація» - це процес встановлення принципала, який, за їх твердженням, є «головним», це означає, що користувач, пристрій або будь-яка інша система може виконувати дію в вашому додатку). «Авторизація» відноситься до процесу прийняття рішення може принципалу дозволено виконувати дію в вашому додатку. Щоб досягти моменту, коли потрібен дозвіл на авторизацію, ідентифікатор принципала вже встановлено процесом аутентифікації.

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java. Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java

ЕЕ. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB) [2].

IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються в рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows.

Eclipse (вимовляється «ікліпс», від англійського «затемнення») — вільне модульне інтегроване середовище розробки програмного забезпечення. Розробляється і підтримується Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для програмістів на мові Java, системи контролю версій, конструктори GUI тощо. Написаний в основному на Java, може бути використаний для розробки застосунків на Java і, за допомогою різних плагінів, на інших мовах програмування, включаючи Ada, C, C++, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з каркасом Ruby on Rails), Scala, Clojure та Scheme. Середовища розробки зокрема включають Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C++, Eclipse JDT для Java, Eclipse PDT для PHP.

NetBeans IDE — вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній конфігурації. Середовище розробки NetBeans за замовчуванням підтримує розробку для платформ J2SE і J2EE.

Поширюється у сирцевих текстах під ліцензіями GPLv2 і CDDL. Проект NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun — Oracle, проте розробка NetBeans ведеться незалежно

співтовариством розробників (NetBeans Community) і компанією NetBeans.Org.

NetBeans IDE доступна для платформ Microsoft Windows, GNU/Linux, FreeBSD, і Solaris (як SPARC, так x86). Для інших платформ доступна можливість зібрати NetBeans самостійно із сирцевих текстів.

1.6 Засоби розробки інтерфейсу

Для розробки веб-додатку на Java було обрано використовувати технологію JavaServer Faces. JavaServer Faces (JSF) — це MVC каркас програмування, технологія для веб-додатків, що написані на Java. Він слугує для того, щоб полегшувати розробку користувацьких інтерфейсів для Java

EE додатків. На відміну від більшості MVC фреймворків, які керуються запитами, підхід JSF ґрунтується на використанні компонентів. Стан компонентів користувацького інтерфейсу зберігається, коли користувач запитує нову сторінку й потім відновлюється, якщо запит повторюється. Користь технології JSF обумовлена, в першу чергу, наявністю специфікації JSF. Специфікація дозволяє розробляти JSF фреймворки з різним призначенням та різною внутрішньою структурою. Вона лиш гарантує, що фреймворк буде підпорядкований певній структурі. Гарними сферами застосування еталонної реалізації JSF є невеликі веб-сайти. Модель-вид-контролер (англ. Model-View-Controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон поділяє систему на три частини: модель даних, представлення даних та керування. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (представлення) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен

полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах призводить до певної впорядкованості їх структури і робить їх зрозумілішими завдяки зменшенню складності. Архітектурний шаблон MVC поділяє програму на три частини. У тріаді до обов'язків компоненту Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Представлення (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель. Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

MVC поділяє цю частину системи на три самостійні частини: введення даних, компонент обробки даних і виведення інформації. Модель, як вже було відмічено, інкапсулює ядро даних і основний функціонал з їх обробки. Також компонент Модель не залежить від процесу введення або виведення даних. Компонент виводу представлення може мати декілька взаємопов'язаних областей, наприклад, різні таблиці і поля форм, в яких відображається інформація. У функції Контролера входить моніторинг за подіями, що виникають в результаті дій користувача (зміна положення курсора миші, натиснення кнопки або введення даних в текстове поле).

Зареєстровані події транслуються в різні запити, що спрямовуються компонентам Моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через Контролер внесе зміни до Моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

1.7 Засоби тестування продукту

При розробці моделі було використано технологію розробки через тестування. Розробка через тестування Test-driven development (TDD) — технологія розробки програмного забезпечення, яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести. Нарешті, програміст або група вдосконалюють код для погодження змін. Один із ключових моментів TDD полягає у тому, що підготовка тестів перед написанням самого коду пришвидшує процес внесення змін. Керована тестами розробка вимагає від розробника створення автоматизованих модульних тестів, які визначають вимоги до коду безпосередньо перед написанням самого коду. Тест містить перевірки умов, які можуть або виконуватися, або ні. Коли вони виконуються, кажуть, що тест пройдено. Проходження тесту підтверджує поведінка, передбачувана програмістом. На практиці модульні тести покривають критичні та нетривіальні ділянки коду. Це може бути код, схильний до частих змін, код, від роботи якого залежить працездатність великої кількості іншого коду, або код з великою кількістю залежностей. Середовище розробки повинно швидко реагувати на невеликі модифікації коду. Архітектура програми повинна базуватися на використанні багатьох сильно пов'язаних компонентів, які не залежать або майже не залежать один від одного, завдяки чому тестування коду спрощується. TDD не тільки пропонує перевірку коректності, а й впливає на дизайн програми. Спираючись на тести, розробники можуть швидше уявити, яка функціональність необхідна користувачеві. Таким чином, деталі інтерфейсу з'являються задовго до остаточної реалізації рішення. Зрозуміло, що до тестів застосовуються такі самі вимоги щодо якості коду, як і до основного коду.

1.8 Вибір ORM

ORM (англ. Object-relational mapping, Об'єктно-реляційна проекція) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи "віртуальну об'єктну базу даних".

При об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Як приклад можна розглянути адресну книгу, яка містить список людей разом з кількома телефонами і кількома адресами. В термінах об'єктно-орієнтованого програмування вони представлятимуться об'єктами класу “Людина”, які міститимуть наступний список полів: ім'я, список (або масив) телефонів і список адрес.

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відносин між ними. Ці об'єкти називають “постійними” (англ. persistent). Історично існує кілька підходів до рішення цієї задачі.

Розроблено безліч пакетів, що знімають необхідність в перетворенні об'єктів для зберігання в реляційних базах даних.

Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення автоматично. Маючи список таблиць в базі даних і об'єктів в програмі, вони автоматично перетворюють запити з одного вигляду в інший. В результаті запиту об'єкта “Людина” (з прикладу з адресною книгою) необхідний SQL-запит буде сформований і виконаний, а результати “магічним” чином перетворені в об'єкти “Номер телефону” всередині програми.

З погляду програміста система повинна виглядати як постійне сховище об'єктів. Він може просто створювати об'єкти і працювати з ними як завжди, а вони автоматично зберігатимуться в реляційній базі даних.

На практиці все не так просто і очевидно. Всі системи ORM звичайно проявляють себе в тому або іншому вигляді, зменшуючи в деякому роді

можливість ігнорування бази даних. Більш того, шар транзакцій може бути повільним і неефективним (особливо в термінах згенерованого SQL). Все це може привести до того, що програми працюватимуть повільніше і використовувати більше пам'яті, ніж програми, написані «вручну».

Але ORM позбавляє програміста від написання великої кількості коду, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість розробки. Крім того, більшість сучасних реалізацій ORM дозволяють програмістові при необхідності жорстко задати код SQL-запитів, який використовуватиметься при тих або інших діях (збереження в базу даних, завантаження, пошук тощо) з постійним об'єктом.

Java Persistence API — стандартизований інтерфейс для Java ORM фреймворків. Є частиною EJB 3 та J2EE 5, хоча може використовуватись незалежно від них. Виник через популярність вільного ORM фреймворку Hibernate, та бажання мати незалежний від конкретної реалізації стандарт.

Hibernate — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є вільним програмним забезпеченням. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

Для взаємодії з базою даних був вибраний ORM фреймворк Hibernate.

Метою Hibernate є звільнення розробника від значних типових завдань із програмування взаємодії з базою даних. Розробник може використовувати Hibernate як при розробці з нуля, так і для вже існуючої бази даних.

Hibernate піклується про зв'язок класів з таблицями бази даних (і типів даних мови програмування із типами даних SQL), і надає засоби автоматичної побудови SQL запитів й зчитування/запису даних, і може значно зменшити час розробки, який зазвичай витрачається на ручне написання типового SQL і JDBC коду. Hibernate генерує SQL виклики і звільняє розробника від ручної обробки результуючого набору даних, конвертації об'єктів і забезпечення сумісності із різними базами даних [18].

Hibernate забезпечує прозору підтримку збереження даних, тобто їхньої персистентності (англ. persistence) для «POJO»-об'єктів, тобто для звичайних Java-об'єктів; єдина сувора вимога до класу, що зберігається — конструктор за замовчуванням .

Hibernate забезпечує використання SQL-подібної мови Hibernate Query Language (HQL), яка дозволяє виконувати SQL-подібні запити, записані поряд з об'єктами даних Hibernate. Запити критеріїв надаються як об'єктно-орієнтована альтернатива до HQL.

1.9 Вибір бази даних

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. У загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки.

Існують реляційні бази даних і так звані NoSQL (Not Only SQL) бази даних.

NoSQL - база даних, що забезпечує інший механізм зберігання та видобування даних, ніж звичний підхід таблиць-відношень в реляційних базах даних.

NoSQL бази даних все більше і більше використовуються в задачах із застосуванням big data та real-time веб-застосунках. NoSQL системи також називають "Not only SQL" для підкреслення того, що вони можуть підтримувати SQL-подібну структуру та мову запитів.

Види нереляційних баз даних:

MongoDB - це документо-орієнтована система керування базами даних (СКБД) з відкритим кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують

даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій.

Apache Cassandra — розподілена система керування базами даних, що відноситься до класу noSQL-систем і розрахована на створення високомасштабованих і надійних сховищ величезних масивів даних, представлених у вигляді хеша.

Спочатку проект був розроблений в надрах Facebook і в 2009 році переданий під оруду фонду Apache Software Foundation. Промислові рішення на базі Cassandra розгорнуті для забезпечення сервісів таких компаній, як Cisco, IBM, Cloudkick, Reddit, Digg, Rackspace і Twitter.

Реляційна база даних — база даних, заснована на реляційній моделі даних. Слово "реляційний" походить від англ. relation. Для роботи з реляційними БД застосовують реляційні СКБД. Інакше кажучи, реляційна база даних — це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеня.

Види реляційних баз даних:

MySQL — була розроблений компанією «ТсХ» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування [5].

PostgreSQL – об'єктно-реляційна система керування базами даних. Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю систему керування базами даних та впроваджувати у неї найновіші досягнення.

Сервер PostgreSQL написаний мовою C. Звичайно розповсюджується у вигляді набору текстових файлів із початковий кодом. Для інсталяції необхідно відкомпілювати файли на своєму комп'ютері і скопіювати в деякий каталог. Весь процес детально описаний в документації.

Oracle – об'єктно-реляційна система керування базами даних від Oracle Corporation.

В якості бази даних була вибрана MySQL, у зв'язку з можливістю безкоштовного користування та легкістю встановлення та налаштування.

MySQL – реляційна система керування базами даних, призначена для вбудовування в Java-додатку або обробки транзакцій в реальному часі [5].

1.10 Динамічне відображення даних

AJAX (Asynchronous Javascript And Xml) – технологія для взаємодії з сервером без перезавантаження сторінок. За рахунок цього зменшується час відгуку і веб-додаток за інтерактивністю більше нагадує десктоп. Наприклад, при натисканні кнопки голосувати - з браузера на сервер буде відправлено повідомлення, а сервер відповість браузеру, що голос прийнятий.

Технологія AJAX, як вказує перша буква А в її назві - асинхронна, тобто браузер, відіславши запит, може робити що завгодно, наприклад, показати повідомлення про очікування відповіді, прокручувати сторінку, і т.п.

Технологія AJAX використовує комбінацію:

- (X) HTML, CSS для подачі і стилізації інформації

- DOM-модель, операції над якою виробляються JavaScript на стороні клієнта, щоб забезпечити динамічне відображення і взаємодія з інформацією

- XMLHttpRequest для асинхронного обміну даними з веб-сервером. У деяких AJAX-фреймворк і в деяких ситуаціях, замість XMLHttpRequest використовується IFrame, SCRIPT-тег або інший аналогічний транспорт.

- JSON часто використовується для обміну даними, однак будь-який формат підійде, включаючи форматований HTML, текст, XML і навіть який-небудь EBMML

Типовий AJAX-додаток складається як мінімум з двох частин. Перша виконується в браузері і написана, як правило, на JavaScript, а друга – знаходиться на сервері і написана, наприклад, на Ruby, Java або PHP.

У синхронної моделі браузер відправляє запит на сервер і чекає, поки той зробить всю необхідну роботу. Сервер виконує запити до бази даних, загортає відповідь у необхідний формат і виводить його. Браузер, отримавши відповідь, викликає функцію показу. Всі процеси виконуються послідовно, один за одним. Мережеві затримки включені від час очікування. Користувач не може займатися чимось іншим на цій самій сторінці, поки відбувається синхронний обмін даними.

Браузер тримає постійне з'єднання з сервером (або робить час від часу коригувальні запити), і потрібні зміни відсилаються цим каналом. Взагалі, проблема застарілого контексту безпосередньо відноситься до задачі цілісності даних. За кінцеву перевірку цілісності, як і при валідації форм, в будь-якому випадку несе відповідальність сервер.

1.11 Вибір засобів розробки серверної частини

Також у розробці був використаний Spring Framework, в першу чергу як Inversion of Control-контейнер.

Spring Framework забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники і організації, які хочуть створити інформаційну

систему, засновану на платформі Java. Через широкую функціональність важко визначити найбільш значущі структурні елементи, з яких він складається. Spring Framework не повністю пов'язаний з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

Spring Framework, ймовірно, найбільш відомий як джерело розширень (features), потрібних для ефективної розробки складних бізнес-додатків поза великогагових програмних моделей, які історично були домінуючими в промисловості. Ще одне його достоїнство в тому, що він ввів раніше невикористовувані функціональні можливості в сьогоденні панівні методи розробки, навіть поза платформи Java [2].

Цей фреймворк пропонує послідовну модель і робить її придатною до більшості типів додатків, які вже створені на основі платформи Java. Вважається, що Spring Framework реалізує модель розробки, засновану на кращих стандартах індустрії, і робить її доступною в багатьох областях Java.

Центральною частиною Spring Framework є контейнер Inversion of Control, який надає засоби конфігурації і управління об'єктами Java за допомогою відображення. Контейнер відповідає за управління життєвим циклом об'єкта: створення об'єктів, виклик методів ініціалізації і конфігурація об'єктів шляхом зв'язування їх між собою.

Об'єкти, які створюються контейнером, також називаються керованими об'єктами (beans). Звичайно конфігурація контейнера здійснюється шляхом завантаження XML-файлів, що містять визначення bean і надають інформацію, необхідну для створення bean.

Об'єкти можуть бути отримані одним з двох способів:

- Пошук залежності – шаблон проектування, в якому викликає об'єкт запитує у об'єкта-контейнера екземпляр об'єкта з певним ім'ям або певного типу.

- Впровадження залежності – шаблон проектування, в якому контейнер передає екземпляри об'єктів за їх іменем іншим об'єктам за допомогою конструктора, властивості або фабричного методу.

В данній системі було використане саме впровадження залежності - Dependency injection.

Впровадження залежності (DI) - процес надання зовнішньої залежності програмного компоненту. Є специфічною формою «інверсії управління» (англ. Inversion of control, IoC), коли вона застосовується для управління залежностями. У повній відповідності з принципом єдиної обов'язки об'єкт віддає піклуватися про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього спільного механізму.

Робота фреймворку, що забезпечує впровадження залежності, описується наступним чином. Додаток, незалежно від оформлення, виконується всередині контейнера IoC, що надається фреймворком. Частина об'єктів в програмі, як і раніше, створюється звичайним способом мови програмування, частина створюється контейнером на основі наданої йому конфігурації.

Умовно, якщо об'єкту потрібно отримати доступ до певного сервісу, об'єкт бере на себе відповідальність за доступ до цього сервісу: він або отримує пряме посилання на місцезнаходження сервісу, або звертається до відомого “сервіс-локатора” і запитує посилання на реалізацію певного типу сервісу. Використовуючи ж впровадження залежності, об'єкт просто надає властивість, яка в змозі зберігати посилання на потрібний тип сервісу. І коли об'єкт створюється, посилання на реалізацію потрібного типу сервісу автоматично вставляється в цю властивість (поле), використовуючи засоби середовища.

Впровадження залежності більш гнучке, тому оскільки легше створювати альтернативні реалізації даного типу сервісу, а потім вказувати, яка саме реалізація повинна бути використана в, наприклад, файлі конфігурації, без змін в об'єктах, які цей сервіс використовують. Це особливо

корисно в юніт-тестування, тому що вставити реалізацію “заглушки” сервісу в тестований об'єкт дуже просто.

З іншого боку, зайве використання впровадження залежностей може зробити програми більш складними і важкими у супроводі: оскільки для розуміння поведінки програми програмісту необхідно дивитися не тільки в вихідний код, а ще й в конфігурацію, а конфігурація, як правило, невидима для IDE, які підтримують аналіз посилань і рефакторинг, якщо явно не вказана підтримка фреймворків з впровадженнями залежностей.

Висновки

Від вибору платформи та фреймворку для розробки залежать багато факторів, які можуть впливати на кінцевий продукт. Саме тому у даному розділі були проаналізовані та обрані для розробки системи управління контентом веб-порталу наявні інструменти для розробки.

З огляду на особливості різних підходів при розробці інтернет-технологій вибор технологій та засобів розробки необхідно засновувати на таких факторах: вимоги до продуктивності ПЗ, оптимальний обсяг споживаних ресурсів, складність проекту і кваліфікованість програміста.

Доцільним є використання фреймворку Spring MVC, та мови програмування Java – кількість готових рішень та бібліотек може прискорити процес розробки. Також, технології розробки пов'язані з мовою програмування Java у своїй більшості є бешкоштовними.

Розробка програмного продукту за допомогою цього фреймворка ведеться у середовищі IntelliJ IDEA, що дозволяє додавати сторонні бібліотеки та модулі при необхідності використовуючи вбудовані засоби.

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, а також у наданні користувачам зручного та швидкого інтерфейсу, за допомогою якого користувачі матимуть можливість отримувати інформацію, керувати нею, а також для вирішення управлінських та інших задач.

Для вирішення цих задач необхідно, щоб система складалася зі сховища даних та засобів вводу/виводу даних з нього користувачами. При цьому сховище повинне мати можливість автоматичної взаємодії з існуючими системами, а також можливість розподілення на декілька фізичних серверів. Повинна бути можливість одночасної роботи великої кількості користувачів зі сховищем за умови оперативного збереження цілісності даних. Кількість користувачів, які одночасно працюють із системою, не може бути точно передбаченою, тому систему слід розраховувати на великі навантаження. Користувачі можуть бути розташованими як всередині МПТУ, так і за його межами.

Варіантами архітектури системи, що задовольняють поставленим вимогам, є клієнт-серверна та сервіс-орієнтована архітектури.

За трирівневою клієнт-серверною архітектурою сховище даних фізично розташоване на одній ЕОМ, а робочі місця кожного користувача обладнані засобами для обміну даними з цим сховищем. Між сховищем та користувачами додатково встановлюються компоненти проміжного шару для обробки запитів користувачів до сховища даних.

Сервіс-орієнтована архітектура є модульним підходом до розробки систем, заснованим на використанні сервісів (служб) зі стандартизованими інтерфейсами. В її основі лежать принципи багатократного використання функціональних елементів ІТ, ліквідації дублювання функціональності в ПЗ, уніфікації типових операційних процесів і т.д. Компоненти такої системи

можуть бути розподілені по різних вузлах мережі, і пропонуються як незалежні, слабо пов'язані, замінні сервіси-додатки. Інтерфейс компонентів такої системи надає інкапсуляцію деталей реалізації конкретного компонента (ОС, платформи, мови програмування, розробника, і т. ін.).

Клієнт-серверна архітектура є більш зручною з боку використання, адже для доступу до web-порталу (сервер) не потрібно специфічного програмного забезпечення, достатньо використання вбудованих можливостей операційних систем – браузерів (клієнт). Тому обираємо цей підхід як підхід до вирішення завдання розробки програмного забезпечення управління контентом web-порталу.

2.1.2 Функціональна структура

У процесі свого функціонування система взаємодіє з системою управління користувачами для виконання ними різних функцій. Схема цієї взаємодії, побудована показана на рисунку 2.1.

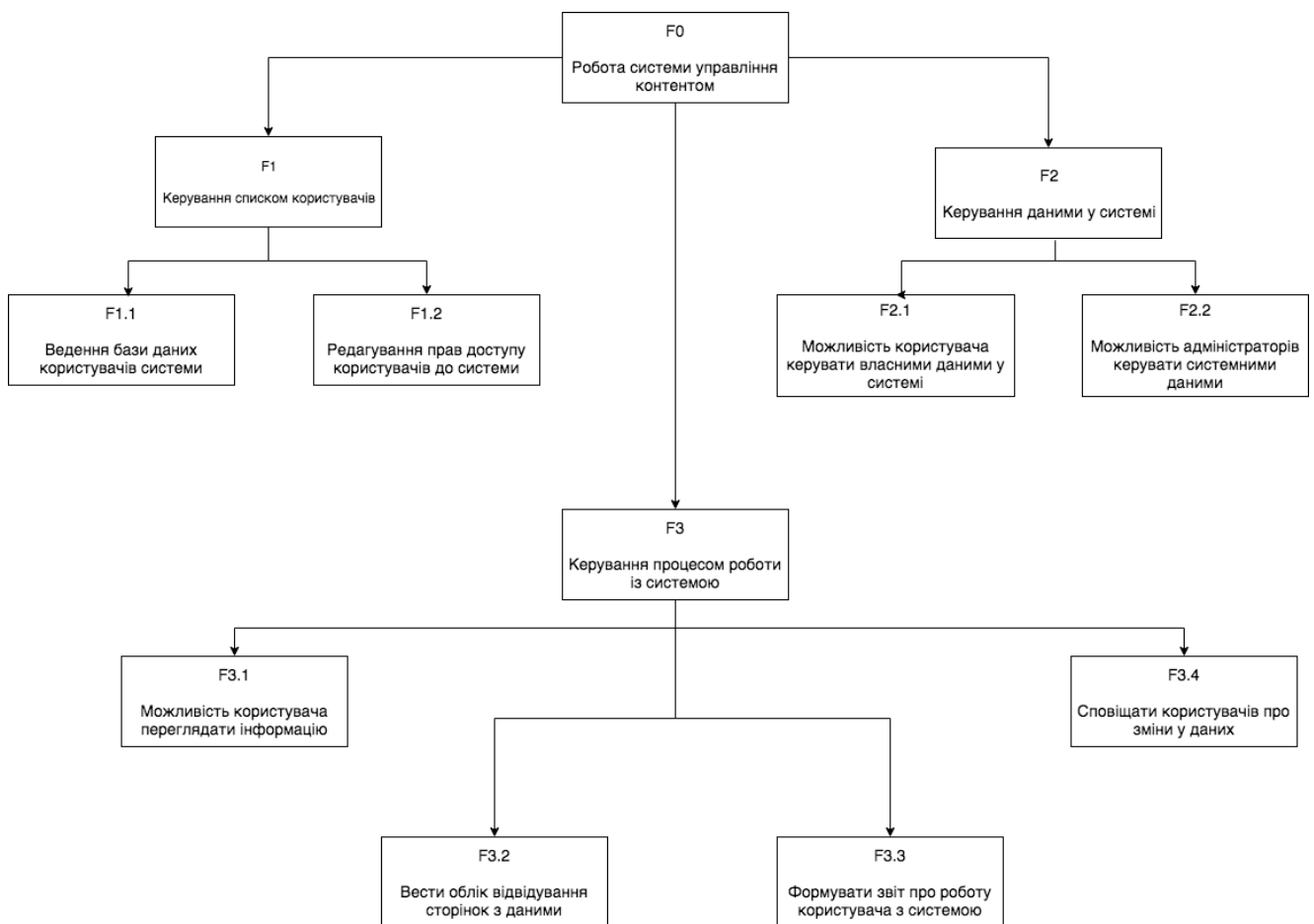


Рисунок 2.1. Схема функціональної структури

Користувачі системи під час виконання своєї роботи мають можливість використовувати наступні функції:

- Користувач – перегляд сторінок з даними, змінювати дані свого профілю;
- Методист – перегляд та редагування даних на сторінках, керування дистанційними курсами, обмежене керування користувачами;
- Адміністратор – перегляд та управління (зміна, видалення, створення нових) користувачами, управління даними.

2.1.2 Опис функцій, що автоматизуються.

Задачею системи, що створюється, є автоматизація наступних функцій користувача:

- F1.1) Надавати користувачам можливість реєструватися у системі, управляти своїм обліковим записом, у тому числі видаляти обліковий запис за необхідності;

Процеси, необхідні для реалізації цієї функції: облік даних користувачів системи.

- F1.2) Редагувати права доступу користувачів до сервісів системи

Процеси: виділення користувачам ролей у системі.

- F2.1) Надавати користувачам можливість розміщувати власні дані у системі та керувати ними відповідно до своїх потреб.

Процеси: створення інтерфейсу користувача для керування даними у системі.

- F2.2) Надавати адміністраторам системи можливість керувати даними, що зберігаються у системі.

Процеси: створення інтерфейсу адміністратора для керування даними у системі можуть змінюватися під час розробка або експлуатації.

- F3.1) Надавати користувачам можливість переглядати інформацію

Процеси: створення інтерфейсу користувача для перегляду доступної йому інформації.

- F3.2) Вести облік відвідування сторінок з даними

Процеси: ведення статистики використання перегляду сторінок порталу користувачами.

- F3.3) Формувати звіт про роботу користувача з системою

Процеси: формування звітів про роботу користувача з системою.

- F3.4) Сповіщати користувачів про зміни у даних

Процеси: створення інтерфейсу сповіщення користувачів за потреби

Для створення системи необхідно розробити єдину базу даних для збереження інформації (статті, новини, накази, навчальні матеріали тощо), інформацію про користувачів. Також повинно бути передбачене програмне забезпечення для адміністрування системи [4].

2.1.3 Опис постановки задачі

На підставі аналізу вимог і обмежень, а також попередніх концепцій реалізації системи, сформулюємо опис постановки задачі. Необхідно розробити систему управління та обробки даних, що буде задовольняти перерахованим вище функціональним вимогам. Необхідно також забезпечити надійність збереження і безпеку доступу до інформації Системи. Система повинна обслуговувати одночасну безперебійну роботу багатьом користувачам, причому зміни, внесені адміністраторами та модераторами, повинні бути доступні всім іншим користувачам негайно.

Програмне забезпечення повинне бути побудоване на платформі Spring MVC/MySQL/Tomcat/Java/IntelliJ IDEA і реалізовувати графічний інтуїтивно зрозумілий WEB інтерфейс користувача.

2.1.4 Загальний опис системи

Система являє собою інструмент створення та обробки даних у навчальному закладі – інституті. Основними функціональними характеристиками системи є:

- ведення загальної інформаційної бази з розподіленим доступом;
- авторизований доступ до даних на основі прав доступу;
- відображення змін, внесених з будь-якого робочого місця;
- можливість створення завдань генерації додатків до диплому;
- збереження даних, що генеруються системою, для подальшого редагування або відновлення інформації у разі помилок в роботі системи.

Система має звичний графічний інтерфейс користувача, що знижує витрати на навчання персоналу, і побудована з максимальним використанням існуючого на підприємстві в даний момент устаткування і програмного забезпечення.

2.1.5 Захист від несанкціонованого доступу

Методи захисту даних на персональних комп'ютерах надзвичайно різноманітні як по кінцевій меті, так і по технічному втіленню; їх можна розділити на механічні, апаратні і програмні. У даному випадку захист системи від несанкціонованого доступу реалізовано на системному рівні програмним методом. Для використання системи користувачі повинні пройти авторизацію. Доступ до даних і функцій дозволяється тільки при введенні правильного та імені та відповідного пароля.

Авторизація відбувається за рахунок розробленого механізму авторизації в системі. Для реалізації цього механізму, програміст повинен розробити алгоритм безпечної перевірки та збереження інформації про користувачів та інтерфейс роботи з цими даними (наприклад, форму для введення даних користувача. Крім того, адміністратор системи повинен мати можливість вносити зміни до прав користувачів та інших налаштувань.

Для отримання окремих даних (наприклад, інформації про напрям навчання) може використовуватись додаткові технічні або програмні засоби захисту інформації, такі як доступ до даних за картою доступу та пристроєм зчитування цієї картки доступу [11].

2.2 Рішення по організаційному забезпеченню

2.2.1 Організаційна структура інституту

Система розробляється для широкого кола користувачів, які представляють різні структурні підрозділи НЗ (відділи, гуртки та інші). За існуючих умов, кожен з підрозділів має власний штат спеціалістів, які займаються підтримкою відповідних інформаційних систем підрозділів.

Систему пропонується розробляти на базі обчислювального центру НЗ, який має найбільший доступ роботи з ІТ у НЗ (прокладення мереж, адміністрування серверів тощо). Оскільки призначенням системи є інтеграція існуючих інформаційних ресурсів у єдиний комплекс, відпадає необхідність підтримки власних інформаційних систем кожним з підрозділів. Спеціалістів, які при цьому вивільняються, пропонується приєднати до обчислювального центру НЗ з метою підтримки розроблюваної системи на апаратному та програмному рівнях. До того ж, створювана система потребує розробки відповідних сервісів для забезпечення роботи користувача. Відповідальність за це пропонується покласти на кафедри інформаційних технологій. Також за рахунок автоматизації передбачається поступовий перехід до електронного документообігу.

2.2.2 Заходи щодо організаційного забезпечення

Ефективна робота системи вимагає організаційних заходів, метою яких є підтримка даних про процеси, що відбуваються, в актуальному стані і грамотній своєчасній реакції користувачів на інформацію, яка надається системою:

- навчання користувачів роботі в системі згідно виконуваних функцій (посадових обов'язків);
- тестова експлуатація Системи у максимально наближених до реальних умовах;
- мінімізація часу, що проходить між зміною версій даних і зміни їхнього стану (іншими словами, негайна реакція адміністраторів та модераторів на зміни у виробничому процесі);

- обов'язкова участь у формуванні даних усіх суб'єктів підприємства; регулярний оперативний контроль зв'язаних процесів керівниками підрозділів і контроль виробничих циклів;
- моніторинг та оцінка коректності роботи користувачів;
- внесення необхідних змін до системи, якщо робота користувачів призводить до порушень у коректності роботи алгоритмів системи.

2.2.3 Організаційно-кадрові заходи

Для підтримки апаратних і програмних засобів у робочому стані, а також рішення поточних задач налаштування, установки і відновлення загальносистемного програмного й апаратного забезпечення в університеті повинна бути введена штатна одиниця системного адміністратора, або для цих цілей може застосовуватись аутсорсінг (outsourcing – надання ІТ-функцій сторонніми організаціями).

2.3 Рішення по інформаційному забезпеченню

2.3.1 Опис інформаційного забезпечення системи

Одну з основ інформаційного забезпечення системи складає визначення її внутрішнього стану, що буде відбито в моделі класів. На даному етапі нас цікавлять постійні сутності бізнесів-процесів. Бізнес-процес – це будь-яка діяльність, що має вхідний продукт, додає вартість до нього, та забезпечує вихідний продукт для внутрішнього або зовнішнього споживача. Вид класів, що забезпечують опис бізнес-процесів назвемо класами-сутностями (чи класами предметної області).

Для створення моделі класів складемо словник даних, ґрунтуючись на сутностях, виявлених на етапі бізнесу-аналізу.

Користувач – людина яка взаємодіє з системою.

Модератор – методист, який може змінювати деякі дані (наприклад, курси).

Адміністратор – модератор з виключними правами на керування системою.

Модерація – процес керування користувачами та даними.

Перегляд – отримання результатів роботи системи (перегляд статей та матеріалів).

На даному етапі проектування сутності словника даних стають кандидатами для класів проектованої системи. Модель бізнес-сутностей системи показана на рисунку 2.2.

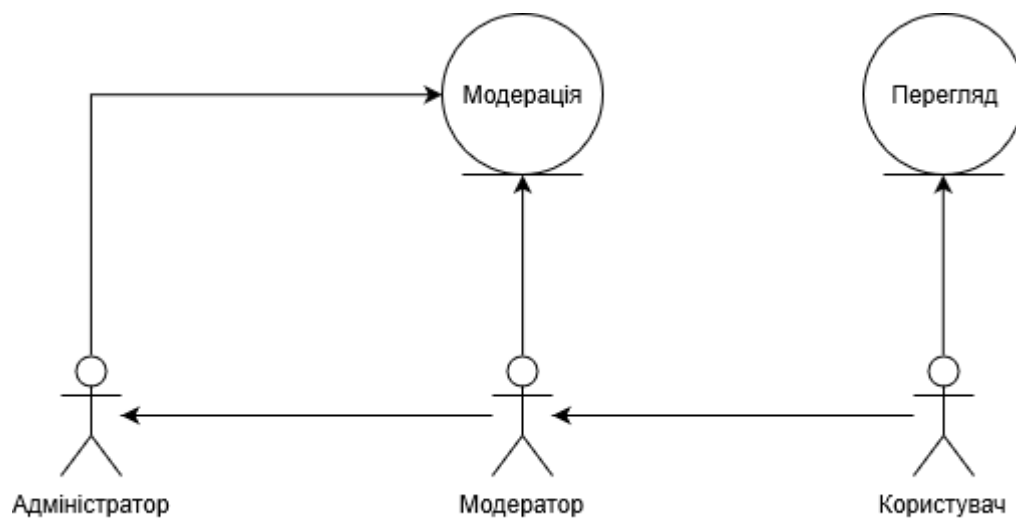


Рисунок 2.2 - Діаграма бізнес-сутностей системи.

Іншим способом змалювання внутрішнього стану системи є створення логічної моделі бази даних. Також логічну модель іноді називають концептуальною моделлю даних. Концептуальна модель — модель предметної області, що складається з переліку взаємопов'язаних понять, що використовуються для опису цієї області, разом з властивостями й характеристиками, класифікацією цих понять, за типами, ситуацій, ознаками в даній області і законів протікання процесів в ній. Логічна модель даних наведена на рисунку 2.3.

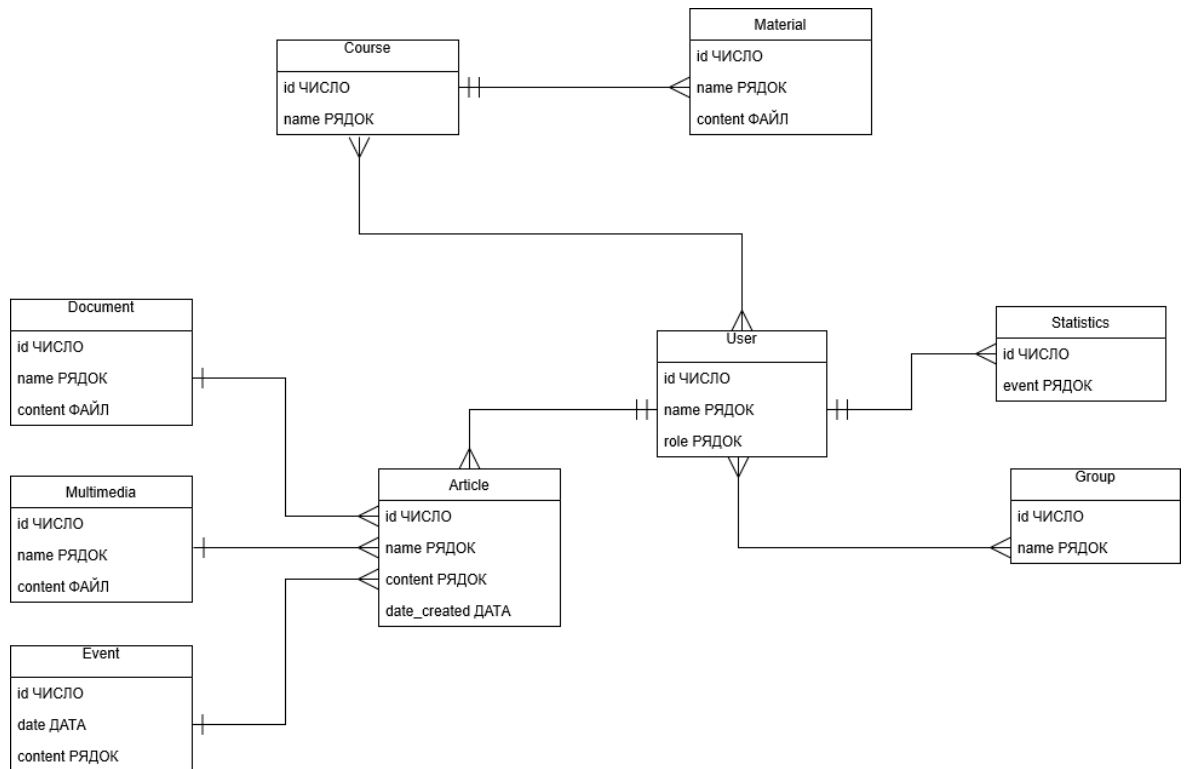


Рисунок 2.3 – Логічна модель даних системи.

На цій діаграмі зображені наступні сутності:

User – інформація про користувача системи.

Group – інформація про групу користувачів.

Article – інформація викладена у вигляді статті.

Document – інформація про документ прикріплений до статті.

Multimedia – інформація про мультимедійні дані, що супроводжує статтю.

Event – інформація про заходи.

Statistics – інформація про дії користувачів системи.

Course – інформація про дистанційні курси.

Material – інформація про матеріали, що містяться у дистанційних курсах.

На даному етапі проектування сутності словника даних стають кандидатами для класів проектованої системи.

2.3.2 Опис організації даних

Визначивши функції, які буде виконувати система, можливо з'ясувати потоки даних, які оточують процеси реалізації кожної функції. Для цього побудуємо діаграму потоків даних (DFD), яка представлена рисунком 2.4.

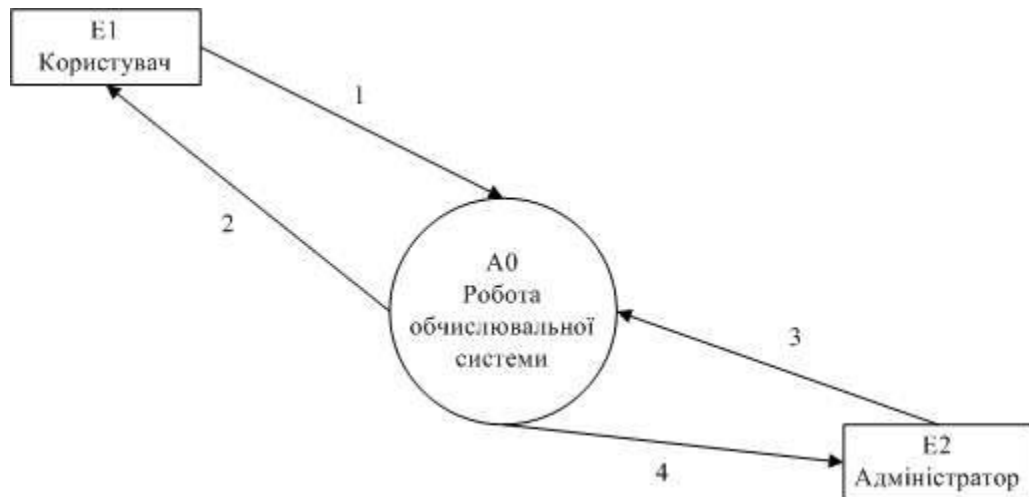


Рисунок 2.4 – Діаграма потоків даних системи

Опис потоків даних:

1. Проходження аутентифікації (введення логіну та паролю), керування даними у системі, перегляд статей та матеріалів;
2. Статистика використання обчислювальних ресурсів за певний період часу, список даних у системі, дані про відвідувані користувачем сторінки;
3. Управління доступом користувачів до сервісів системи, управління системними параметрами, виділення додаткових ресурсів користувачам за потреби.
4. Статистика використання ресурсів системи всіма користувачами за певний період часу, дані про поточне використання ресурсів, звіти про роботу користувачів із системою.

Для опису процесу використання системи використаємо діаграми UML.
Канонічні діаграми UML:

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

Кожна з цих діаграм деталізує і конкретизує різні представлення про модель складної системи в термінах мови UML. При цьому діаграма варіантів використання є найбільш загальною концептуальною моделлю складної системи, яка є базовою для побудови решти всіх діаграм. Діаграма класів, за своєю суттю, логічна модель, що відображає статичні аспекти структурної побудови складної системи. Діаграми кооперації і послідовностей є різновиди логічної моделі, які відображають динамічні аспекти функціонування складної системи. Діаграми станів і діяльності призначені для моделювання поведінки системи. І, нарешті, діаграми компонентів і розгортання служать для представлення фізичних компонентів складної системи і тому відносяться до її фізичної моделі.

Діаграма варіантів використання зображена на рис. 2.5.



Рисунок 2.5 – Діаграма варіантів використання системи.

На рис. 2.6 представлена декомпозиція варіанту використання «Керувати матеріалами» актора «Адміністратор».



Рисунок 2.6 – Діграма декомпозиції варіанту використання «Керувати матеріалами».

Діаграма компонентів системи зображена на рис. 2.7.

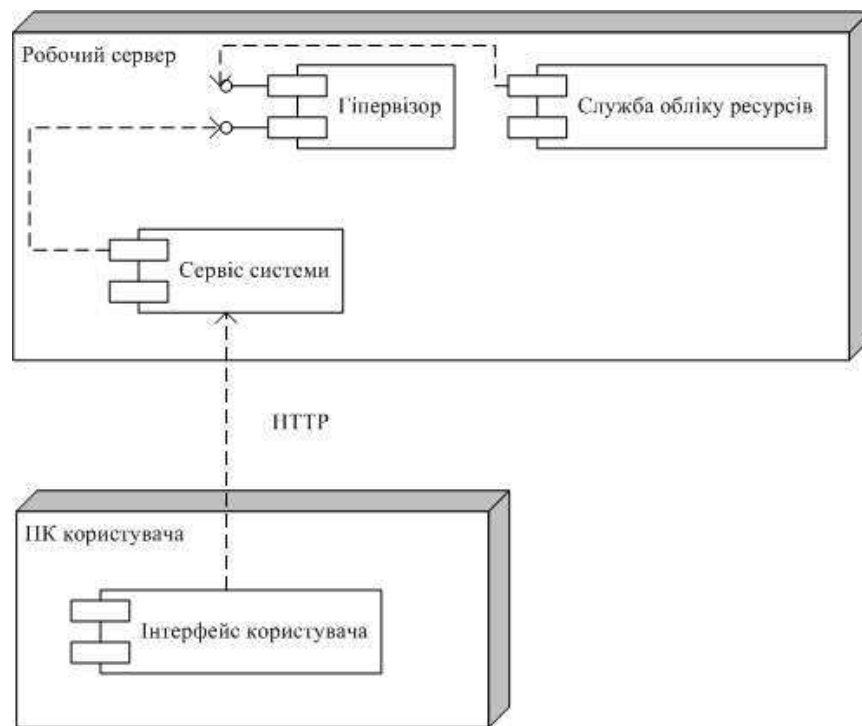


Рисунок 2.7 – Діграма компонентів системи

На рис. 2.8 зображено діаграму діяльності користувача при роботі з системою.

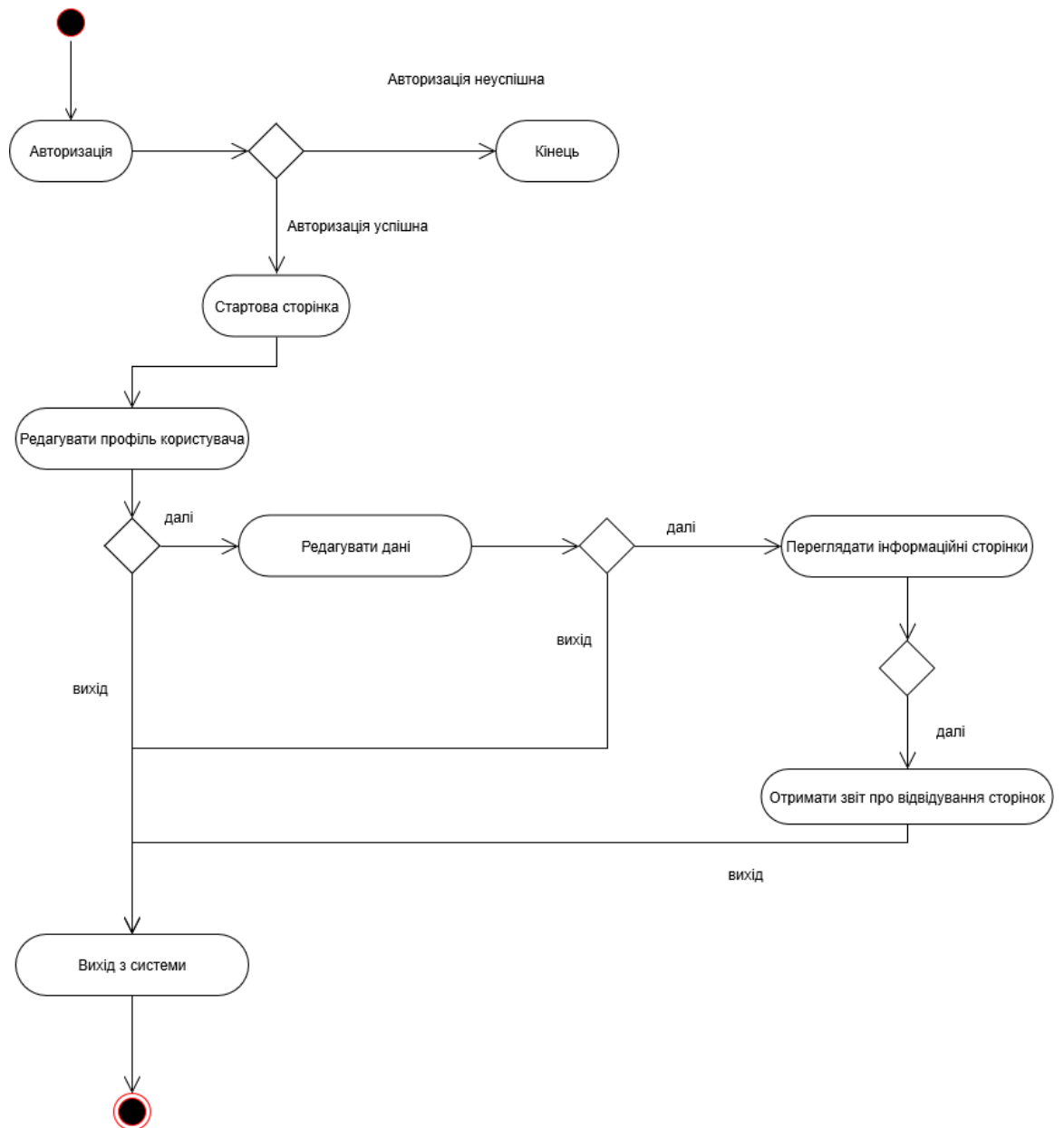


Рисунок 2.8 – Діграма діяльності

2.4 Рішення по технічному забезпеченню

2.4.1 Схема автоматизації

Раціональний вибір технічних засобів, необхідних для впровадження автоматизованої системи буде значною мірою визначати загальний успіх або невдачу проекту. На підставі прийнятих загальносистемних рішень були визначені склад і організація комплексу технічних засобів.

Вимоги до продуктивності:

Обчислювальний комплекс повинен забезпечувати достатню продуктивність для рішення передбачуваних задач при бажаній мінімізації фінансових витрат на його придбання. Крім того, потрібно забезпечити найменший час реакції системи при роботі в інтерактивних режимах, що складають значну частину від всіх експлуатаційних режимів. Це забезпечить комфортну роботу користувачам. Проектована система не буде мати підвищених вимог до обчислювальної потужності. Створювану систему можна віднести до класу облікових задач, у яких загальна продуктивність системи визначається швидкістю операцій введення-виведення інформації при неповному залученні потужності центрального процесора. У такий спосіб будь-яке сучасне апаратне рішення з пропонованих на ринку буде відповідати вимогам до продуктивності. Розподіл програмних компонентів майбутньої системи дозволить не збільшувати навантаження на існуючі клієнтські робочі місця і не висувати вимог до їх модернізації.

Вимоги до надійності:

Розв'язувана задача висуває стандартні вимоги до надійності апаратних засобів без вимоги забезпечення повної відмовостійкості. Основною вимогою для даної системи буде вимога до збереження даних. Передбачаються наступні заходи для її виконання:

- включення сервера в електромережу через пристрій безперебійного живлення (UPS);
- використання на сервері БД апаратного RAID контролера з можливістю віддзеркалення дисків (дане рішення має прийнятну вартість);

Вимоги до експлуатаційних характеристик:

З перерахованих вимог можна зробити висновок про застосовність для сервера БД і клієнтських робочих місць стандартного апаратного забезпечення на платформі x64. До сервера БД пред'являється традиційний набір вимог для такого рішення.

2.4.2 Структурна схема комплексу обчислювальних засобів

Проектована програмна система буде виконана за схемою “Клієнт-сервер”, тому апаратне рішення буде складатися з обчислювальної підсистеми і мережі передачі даних. Принципова схема виконана у виді схеми розміщення наведена на рисунку 2.9.

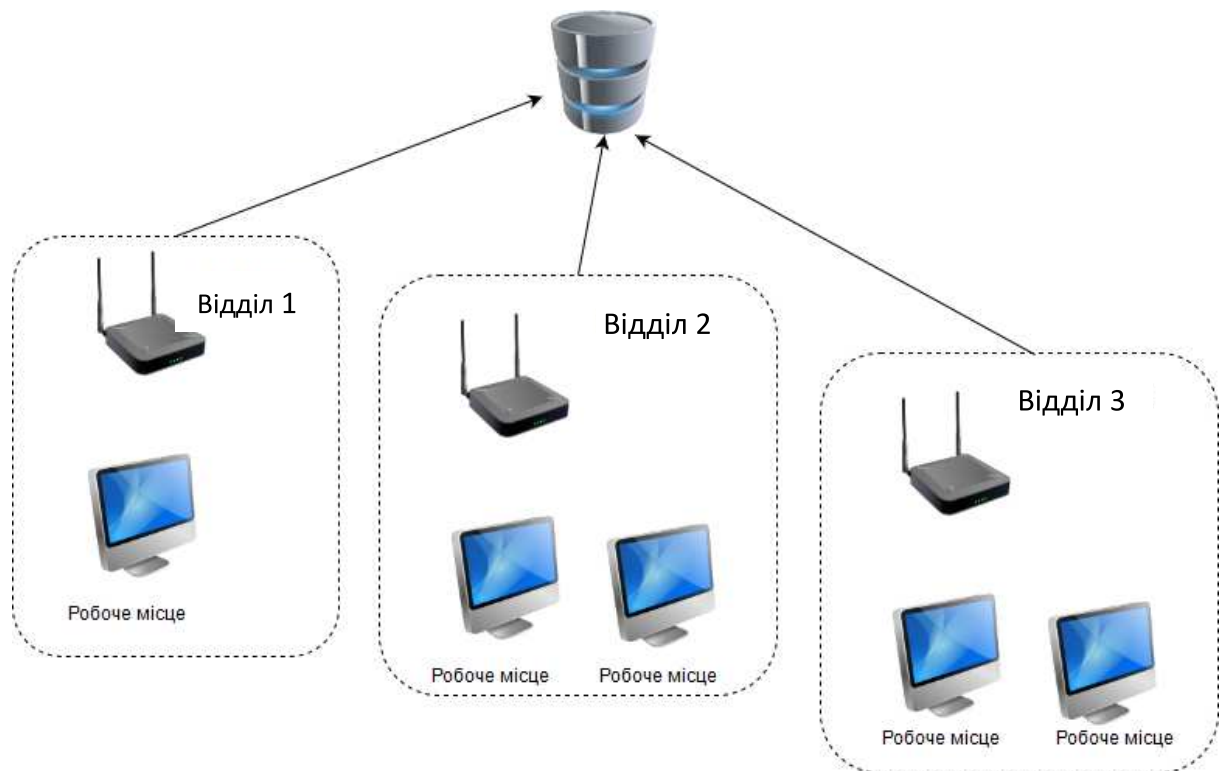


Рисунок 2.9 - Схема розміщення комплексу технічних засобів.

2.4.3 Опис комплексу технічних засобів

За результатами вибору концепції системи пропонуються наступні конфігурації описаних вище технічних засобів (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 серверу інструментів 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 мережного середовища системи

Компонент	Специфікація
Технологія створення мережі	FastEthernet, GigabitEthernet 100/1000/10000 Mbit/c
Мережеве середовище	оптоволокно, UTP5

2.5 Опис алгоритмів функціонування системи

Для уточнення алгоритмів функціонування розроблюваної системи використовуємо засоби об'єктно-орієнтованого аналізу й опису поведінки об'єктів. При цьому будемо розглядати як поведінку системи в цілому (для уточнення виду, характеру і порядку взаємодії між класами), так і реалізацію функціональності окремих класів системи.

Одним з основних алгоритмів функціонування системи є процес створення та публікації статей на web-порталі.

Діаграма класів підсистеми створення та публікації статей, побудована на підставі аналізу сутностей і операцій над ними, представлена на рисунку 2.10.

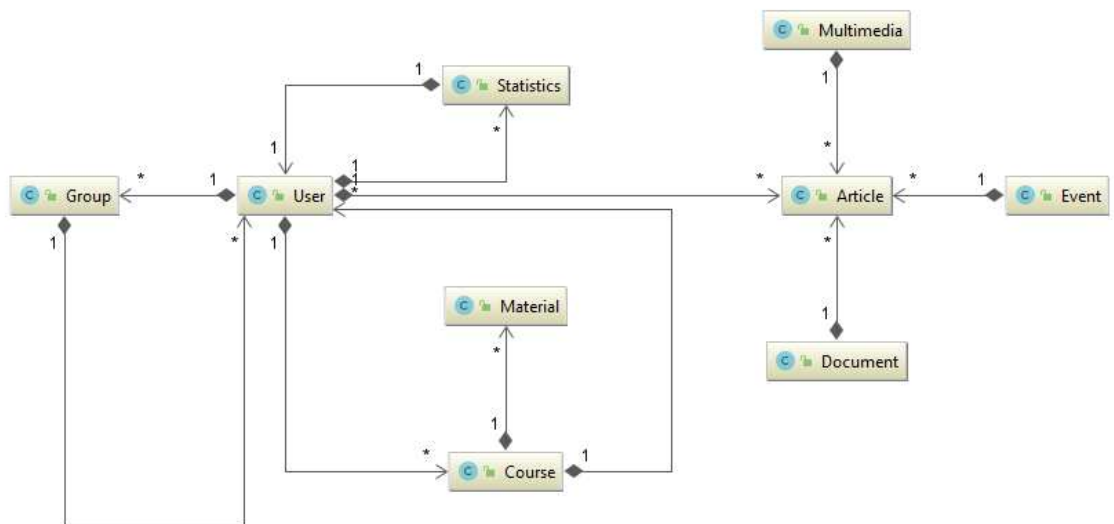


Рисунок 2.10 - Діаграма класів підсистеми створення та публікації статей

Класи відповідні до сутностей фізичної моделі даних створені для використання в ORM.

Іншим важливим алгоритмом роботи системи є робота з базою даних: авторизація користувачів, збереження результатів роботи користувачів. Діаграма класів підсистеми авторизації та збереження даних, побудована на підставі аналізу сутностей і операцій над ними, представлена на рисунку 2.11.

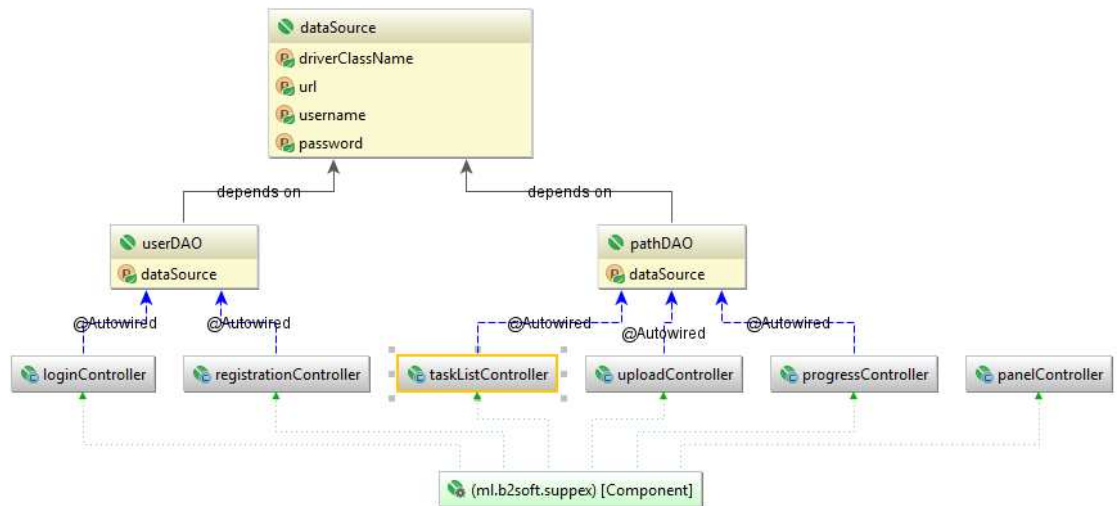


Рисунок 2.11 - Діаграма класів підсистеми авторизації та збереження даних

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 являють собою логічну модель статичного представлення системи, що моделюється. Мова йде про те, що на поданих діаграмах зображуються тільки взаємозв'язки структурного характеру, що не залежать від часу реакції системи на зовнішні події. На відміну від цього представлення, загальний алгоритм поведінки системи може бути описаний за допомогою діаграми станів (statechart diagram). Загальний алгоритм поведінки важливий для визначення взаємодій між

компонентами системи й уточнення причин, що викликає та чи інша відповідна дія. Алгоритм поведінки системи в нотації діаграми станів приведений на рисунку 2.12.

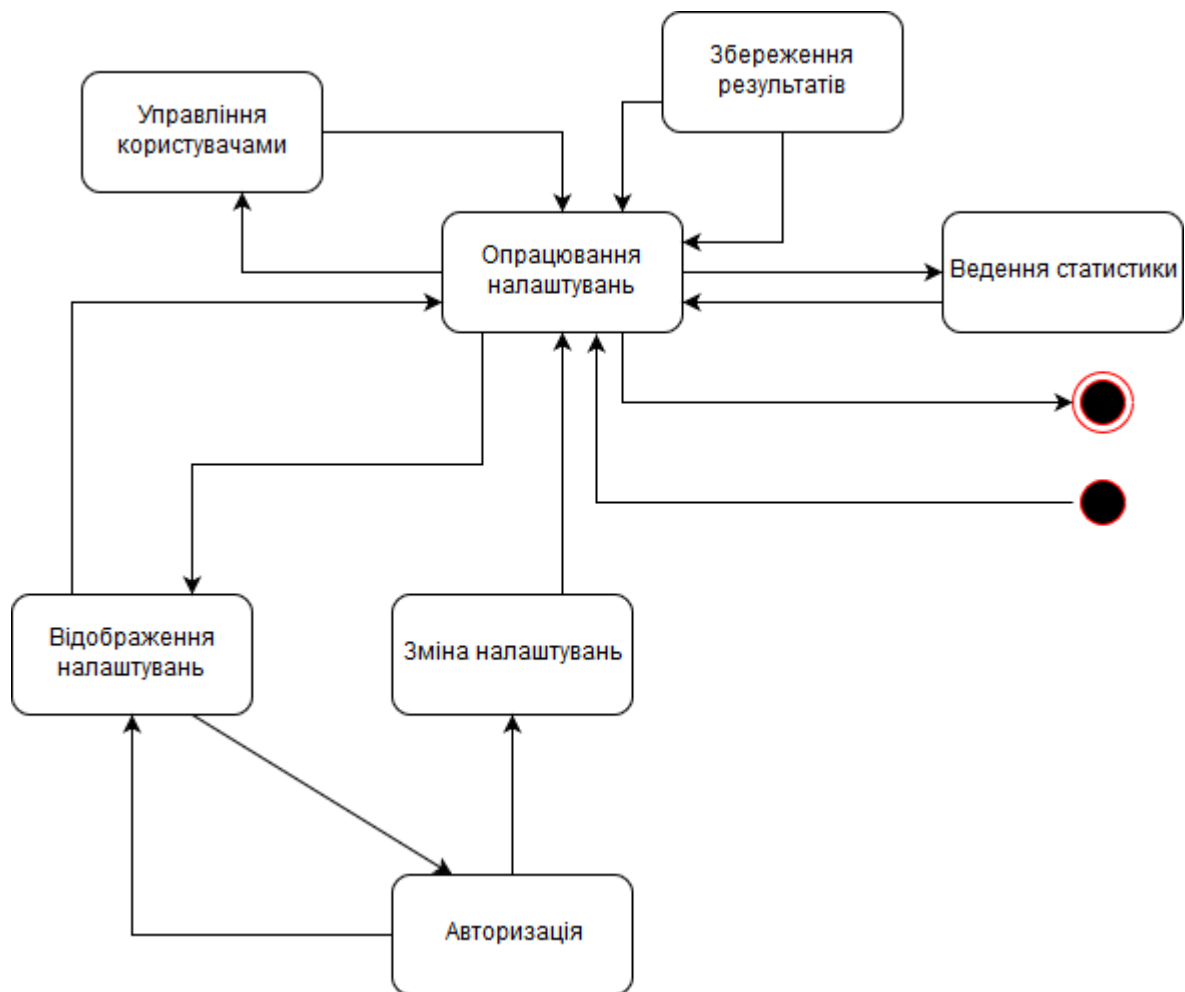


Рисунок 2.12 - Загальний алгоритм поведінки системи.

2.6 Рішення по програмному забезпеченню

2.6.1 Структура і функції частин програмного забезпечення

Функціональність і структуру проектованої автоматизованої системи поряд з моделлю поведінки визначить модель її станів. Будемо будувати модель станів шляхом уточнення моделі бізнесів-сутностей, побудованої нами на підставі вимог користувачів до функцій системи й оброблюваних даних. Вхідними даними для даного етапу проектування будуть моделі сутностей, а результатом стане модель специфікації проектних конструкцій розроблювального програмного забезпечення. У відповідності з прийнятим у

для даної роботи підходом об'єктно-орієнтованого аналізу і проектування виконаємо подальшу специфікацію проектних рішень у виді діаграм класів. Класи визначають стан пропонованої програмної системи (через значення своїх атрибутів) і, значною мірою, її поведінку (через операції чи методи класів).

Сформульовані користувальницькі вимоги тепер приймемо за обмеження. Поряд з моделлю сутностей для проектування класів будемо користатися побудованими раніше діаграмами прецедентів (рисунок 2.1), у відповідності до рекомендацій методики Rational Unified Process. При цьому модель класів буде відповідати словнику даних, що був отриманий на підставі аналізу прецедентів, і який є текстовим аналогом моделі сутностей.

Програмне забезпечення розроблюваної системи складатиметься з програмного комплексу, що розміщується на серверах системи. Для роботи користувача з системою у загальному випадку встановлення додаткового ПЗ не передбачається – взаємодія відбувається за допомогою веб-браузера.

Для входу в систему використовується *модуль авторизації*.

Модуль управління списком користувачів використовується для ведення даних про користувачів системи, дозволяє реєструватися новим користувачам, змінювати особисті дані, видаляти профіль користувача. Адміністратор за допомогою цього модулю може керувати доступом користувачів до системи.

Модуль управління контентом дозволяє модераторам завантажувати дані у систему. Адміністратор або модератор може додавати, редагувати або видаляти інформаційні сторінки та матеріали.

Модуль обліку відвідувань дозволяє вести статистику про перегляд інформаційних сторінок користувачами.

Модуль журналів дозволяє зберегти інформацію про усі дії користувачів у системі. Ця інформація є важливою при виникненні збоїв, адже здатна вказати на причину виникнення аварійної ситуації. Модуль звітів дозволяє користувачу та адміністратору переглянути статистику роботи системи за обраний період часу. Користувач може переглянути статистику використання

ним ресурсів системи, адміністратор може отримати звіти як по використанню ресурсів, так і на основі даних з модулю журналів про події у системі.

Модуль технічної підтримки забезпечує зв'язок між адміністратором та користувачем у разі, якщо користувач не повністю задоволений роботою системи.

2.6.2 Фізична модель бази даних

Для забезпечення можливості зберігати дані в реляційній базі даних, з моделі класів була отримана фізична модель даних. Для цього серед усіх класів були визначені постійні. Це такі класи, об'єкти яких повинні зберігати свої значення не тільки під час роботи програми. Далі було проведено відображення постійних класів на таблиці моделі даних. При цьому атрибути класів відобразилися в поля реляційних таблиць. Асоціації між класами змінилися в зв'язки між реляційними таблицями з використанням механізму зовнішніх ключів. Для забезпечення можливості підтримки цілісності даних по сутностям були використані первинні ключі. Обрана СКБД не підтримує зберігання ієрархії спадкування класів, тому при будівництві фізичної моделі даних ієрархії відображалися на таблиці за принципом одна таблиця на клас. Метод згортання типів в супертип при якому ієрархія змінюється в одну таблицю був відхилений із-за великої надмірності даних. Розроблена фізична модель наведена на рисунку 2.13.

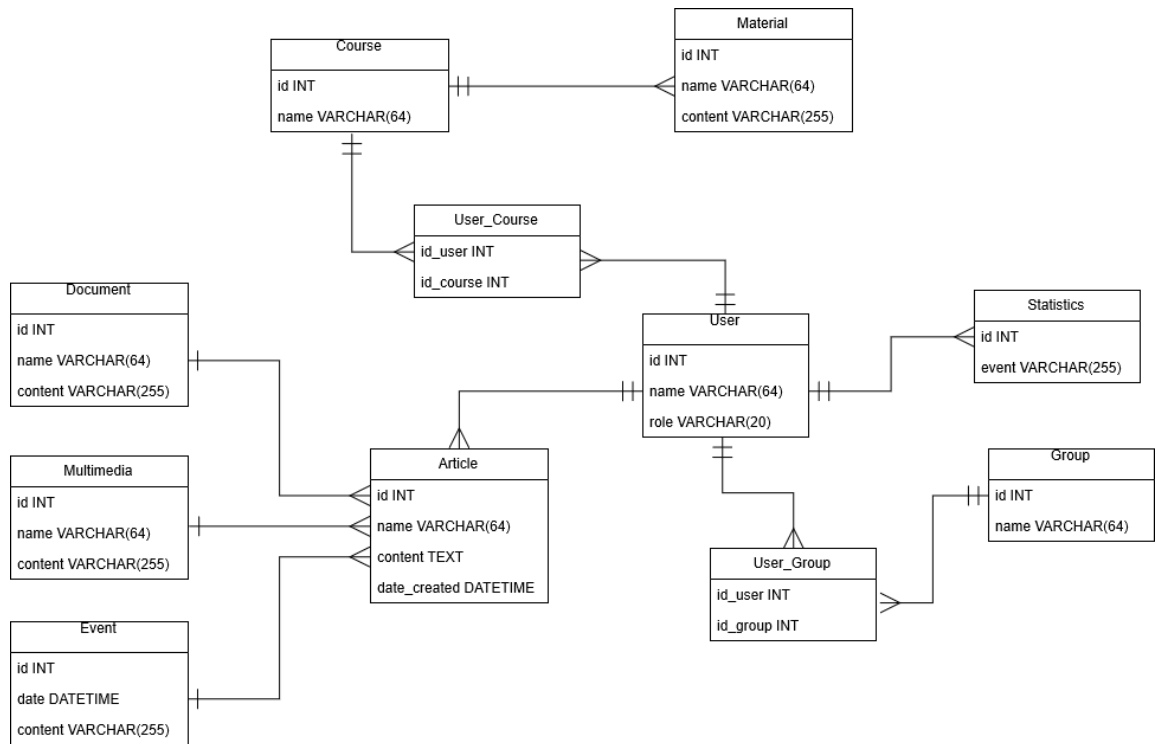


Рисунок 2.13 Фізична модель бази даних

2.6.3 Методи і засоби розробки програмного забезпечення

При аналізі і проектуванні Системи застосовувався об'єктно-орієнтований підхід (ООП), тому програмне забезпечення системи повинне бути розроблене з його використанням. Основними принципами ООП є спадкування, інкапсуляція і поліморфізм.

Використання ООП у розробці дозволяє:

- використовувати отримані в результаті аналізу і проектування логічні моделі для первинної генерації коду;
- скоротити час розробки за рахунок повторного використання коду програмних компонентів;
- проводити налаштування і тестування ґрунтуючись на поводженні об'єктів, без знання їхньої внутрішньої структури;
- цілком використовувати потенціал обраних засобів розробки ПО.

Для проектування й аналізу Системи використовувалася система проектування мовою UML з відкритим кодом - ArgoUML виробництва

спільноти Tigris Software – Tigris.org. На сьогоднішній день це один із потужних засобів проектування на ринку. Мова проектування UML - де-факто стандарт галузі. ArgoUML використовується на всіх етапах проектування від аналізу до автоматичної генерації коду програмної системи і має засоби для роботи з всіма інструментами, наданими мовою UML.

Основним засобом розробки програмного забезпечення системи є мова програмування високого рівня Java, використовуються такі технології: Servlets, EJB3, JSPx та JSF. У якості середовища розробки був використаний продукт IntelliJ IDEA 2017.

IntelliJ IDEA — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються в рамках ліцензії Apache 2.0. Двійкові збірки підготовлені для Linux, Mac OS X і Windows.

До складу IntelliJ IDEA включені напрацювання, створені в результаті спільної роботи з компанією Google, яка використовувала IntelliJ IDEA як базис для своєї нового відкритого середовища розробки Android Studio. Завдяки співпраці істотно розширені штатні можливості IntelliJ IDEA з розробки застосунків для платформи Android.

Community версія середовища IntelliJ IDEA підтримує інструменти (у вигляді плагінів) для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle, мови програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та

експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Комерційна версія «Ultimate Edition» відрізняється наявністю підтримки додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду, можливістю роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

Іншою бібліотекою, створеною компанією Apache, що використано у проекті, є бібліотека XMLBeans. XMLBeans є інструментом, який дозволяє отримати доступ до повної потужності XML в Java-вигляді. Ідея полягає в тому, щоб скористатися багатством і особливостями XML та XML-схеми. Ці функції відображаються як можна більш природно еквівалентним мові Java. XMLBeans використовує XML-схему для компіляції Java-інтерфейсів і класів, які потім можуть бути використані для доступу і зміни даних примірника XML.

Для підвищення швидкості розробки та надійності системи використовується архітектурний шаблон MVC. Модель–вигляд–контролер (або Модель–представлення–контролер, англ. Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення. Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних і

забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель.

Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.

Вигляд може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.

Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності. Однією з реалізацій цього архітектурного шаблону у мові Java є Spring MVC.

Spring MVC – це програмний каркас (фреймворк) з відкритим кодом та контейнер з підтримкою інверсії управління для платформи Java. Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Не зважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

Також потрібний Java WEB сервер встановлений на сервері будь який з:
- JBoss

- Payara
- GlassFish
- Tomcat

На сервері баз даних повинна бути встановлена серверна система керування базами даних (СКБД).

Необхідна сучасна серверна реляційна СКБД, що має високу продуктивність, здатна працювати з великими обсягами інформації і підтримує стандарт SQL-92. Крім того, СКБД повинна працювати під управлінням Windows. Серверними СКБД, що задовольняють зазначеним вимогам, на сьогодні є:

- MySQL;
- MS SQL;
- PostgreSQL.

На клієнтських машинах може використовуватися будь-яка операційна система з можливістю встановлення віддаленого мережного з'єднання і графічним інтерфейсом. Операційні системи, що рекомендуються – MS Windows, Linux, тощо.

Програмне забезпечення, з яким буде працювати користувач, являє собою Web-оглядач з підтримкою HTML 4.0, JavaScript 1.0, CSS 2 (наприклад, MS Edge, FireFox, Opera, Chrome, Safari і т. д.).

3 Розробка робочої документації системи управління контентом web-порталу МПТУ

Система управління контентом web-порталу МПТУ створена для підвищення ефективності роботи методистів навчального закладу. Функції, що автоматизуються системою, пов'язані з обробкою даних.

Основними функціями, що автоматизуються є:

- Надавати користувачам можливість переглядати інформацію (новини, матеріали);
- Надавати адміністраторам системи можливість керувати даними, що зберігаються у системі;
- Усі інші вимоги не є головними і можуть змінюватись під час розробки або експлуатації.

При впровадженні програмного забезпечення системи основними цілями є:

- Простий і зручний інтерфейс користувача.
- Доступ до системи через Internet з будь-якого комп'ютера.
- Багатокористувацька робота.
- Висока масштабуємість системи.
- Висока продуктивність.
- Надійність.
- Конфіденційність інформації.

Для ефективного використання програмного забезпечення важливо впровадити на підприємстві організаційні і технічні заходи, що передбачено в Розділі 2. Тому що підвищується рівень складності експлуатованого ПО, необхідно відповідне підвищення кваліфікації співробітників. При роботі з програмним забезпеченням системи варто керуватися документами «Технічне завдання» (Додаток А), «Програма та методика випробувань» (Додаток Б).

4 Охорона праці

Вступ

Охорона праці – це система законодавчих актів, соціально-економічних, організаційних, технічних і лікувально-профілактичних заходів і засобів, що забезпечують безпеку, збереження здоров'я і працездатність людини в процесі праці.

Охорона праці виявляє і вивчає можливі причини виробничих нещасливих випадків, професійних захворювань, аварій, вибухів, пожеж і розробляє систему заходів і вимог з метою усунення цих причин, і створення безпечних і сприятливих для людини умов праці.

З питаннями охорони праці нерозривно пов'язане і вирішення питань охорони природи.

Складність задач, які стоять перед охороною праці, вимагає використання досягнень і висновків багатьох наукових дисциплін, прямо або побічно пов'язаних із задачами створення здорових і безпечних умов праці.

Оскільки головним об'єктом охорони праці є людина в процесі праці, то при розробці вимог виробничої санітарії використовуються результати досліджень ряду медичних і біологічних дисциплін.

Особливо тісний зв'язок існує між охороною праці, науковою організацією праці, ергономікою, інженерною психологією і технічною естетикою.

Успіх у вирішенні проблем охорони праці здебільшого залежить від якості підготовки фахівців у цій області, від їхнього уміння приймати правильні рішення в складних і мінливих умовах сучасного виробництва.

4.1 Аналіз небезпечних і шкідливих факторів в університеті

При організації умов праці необхідно враховувати вплив на працюючих небезпечних і шкідливих виробничих факторів, які можуть привести до травми або іншого раптового різкого погіршення здоров'я та захворювання або зниження працездатності.

Небезпечні і шкідливі виробничі фактори (ДСТ 12.0.003-74) підрозділяються по природі дії на чотири групи: фізичні, хімічні, біологічні і психофізіологічні.

До небезпечних фізичних факторів відносяться: машини і механізми, що рухаються; різні підйомно-транспортні пристрої і переміщувані вантажі; незахищені рухливі елементи виробничого устаткування (приводні і передавальні механізми, різальні інструменти, пристосування, що обертаються і переміщуються й ін.); відлітаючі частки оброблюваного матеріалу та інструменту, електричний струм, підвищена температура поверхонь устаткування й оброблюваних матеріалів і т.д.

Шкідливими для здоров'я фізичними факторами є: підвищена чи знижена температура повітря робочої зони; висока вологість і швидкість руху повітря; підвищені рівні шуму, вібрації, ультразвуку і різних випромінювань – теплових, іонізуючих, електромагнітних, інфрачервоних і ін. До шкідливих фізичних факторів відносяться також запиленість і загазованість повітря робочої зони; недостатня освітленість робочих місць, проходів і проїздів; підвищена яскравість світла і пульсація світлового потоку.

Хімічні небезпечні і шкідливі виробничі фактори за характером дії на організм людини підрозділяються на наступні підгрупи: загальнотоксичні, подразнюючі, сенсibilізуючі (ті, що викликають алергійні захворювання), канцерогенні (ті, що викликають розвиток пухлин), мутогені (ті, що діють на статеві клітини організму). У цю групу входять численні пари і гази: пари бензолу і толуолу, окис вуглецю, сірчистий ангідрид, окисли азоту, аерозолі свинцю та ін., токсичні пили, що утворюються, наприклад, при обробці різанням берилію, свинцюватих бронз і латуней і деяких пластмас зі

шкідливими наповнювачами. До цієї групи відносяться агресивні рідини (кислоти, луги), що можуть заподіяти хімічні опіки шкіряного покриву при зіткненні з ними.

До біологічних небезпечних і шкідливих виробничих факторів відносяться мікроорганізми (бактерії, віруси й ін.) та макроорганізми (рослини і тварини), вплив яких на працюючих викликає травми або захворювання.

До психофізіологічних небезпечних і шкідливих виробничих факторів відносяться фізичні перевантаження (статичні та динамічні) і нервово-психічні перевантаження (розумові перенапруги, перенапруга аналізаторів слуху, зору та ін.).

Між шкідливими і небезпечними виробничими факторами спостерігається визначений взаємозв'язок. У багатьох випадках наявність шкідливих факторів сприяє прояву травмонебезпечних факторів. Наприклад, надмірна вологість у виробничому приміщенні і наявність струмопровідного пилу (шкідливі фактори) підвищують небезпеку ураження людини електричним струмом (небезпечний фактор).

Рівні впливу на працюючих шкідливих виробничих факторів нормовані гранично-допустимими рівнями, значення яких зазначені у відповідних стандартах системи стандартів безпеки праці і санітарно-гігієнічних правил.

Гранично припустиме значення шкідливого виробничого фактору (за ДСТ 12.0.002-80) – це граничне значення величини шкідливого виробничого фактору, вплив якого при щоденній регламентованій тривалості протягом усього виробничого стажу не приводить до зниження працездатності і захворювання як у період трудової діяльності, так і до захворювання в наступний період життя, а також несприятливо не впливає на здоров'я потомства.

Науково-технічний прогрес приніс серйозні зміни в умови виробничої діяльності працівників розумової праці. Їхня праця стала більш інтенсивною, напруженою, потребує значних витрат розумової,

емоційної і фізичної енергії. Це зажадало комплексного рішення проблем ергономіки, гігієни й організації праці, регламентації режимів праці і відпочинку.

На даний час комп'ютерна техніка широко застосовується у всіх галузях діяльності людини. При роботі з комп'ютером людина піддається впливу ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного та іонізуючого випромінювання, шуму і вібрації, статичної електрики та ін.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і досить великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має раціональна конструкція і розташування елементів робочого місця, що важливо для підтримки оптимальної робочої пози людини-оператора.

У процесі роботи з комп'ютером необхідно дотримуватися правильного режиму праці і відпочинку. В протилежному випадку в персоналу відзначається значна напруга зорового апарату з появою скарг на незадоволеність роботою, головні болі, дратівливість, порушення сну, утому і хворобливі відчуття в очах, у попереку, в області шиї і руках.

Розглянемо шкідливі виробничі фактори, які спостерігаються в приміщеннях, де здійснюється робота з електронно-обчислювальною технікою.

Правильно спроектоване і виконане виробниче освітлення поліпшує умови зорової роботи, знижує стомлюваність, сприяє підвищенню продуктивності праці, благотворно впливає на виробниче середовище, створюючи позитивний психологічний вплив на працюючого, підвищує безпеку праці і знижує травматизм.

Недостатність освітлення приводить до напруги зору, послабляє увагу, приводить до передчасної стомленості. Надмірно яскраве освітлення викликає осліплення, роздратування і різь в очах. Неправильний напрямок світла на

робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Усі ці причини можуть привести до нещасливого випадку або профзахворювання, тому настільки важливий правильний розрахунок освітленості.

Існує три види освітлення – природне, штучне і сполучене (природне і штучне разом).

Природне освітлення – освітлення приміщень денним світлом, яке проникає через світлові прорізи в зовнішніх конструкціях приміщень. Природне освітлення характеризується тим, що міняється в широких межах в залежності від часу дня, пори року, характеру області і ряду інших факторів.

Штучне освітлення застосовується при роботі в темний час доби і вдень, коли не вдається забезпечити нормовані значення коефіцієнта природного освітлення (похмура погода, короткий світловий день). Освітлення, при якому недостатнє по нормах природне освітлення доповнюється штучним, називається сполученим освітленням.

Штучне освітлення підрозділяється на робоче, аварійне, евакуаційне, охоронне. Робоче освітлення, в свою чергу, може бути загальним чи комбінованим. Загальне – освітлення, при якому світильники розміщуються у верхній зоні приміщення чи рівномірно стосовно до розташування устаткування. Комбіноване – освітлення, при якому до загального додається місцеве освітлення.

Згідно до Сніп II-4-79 у приміщеннях обчислювальних центрів необхідно застосувати систему комбінованого освітлення.

При виконанні робіт категорії високої зорової точності (найменший розмір об'єкта розрізнення 0,3...0,5 мм) величина коефіцієнта природного освітлення (КЕО) повинна бути не нижче 1,5%, а при зоровій роботі середньої точності (найменший розмір об'єкта розрізнення 0,5...1,0 мм) КЕО повинний бути не нижче 1,0%. В якості джерела штучного освітлення зазвичай використовуються люмінесцентні лампи типу ЛБ або ДРЛ, які попарно

поєднуються у світильники, та повинні розташовуватися над робочими поверхнями рівномірно.

Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні: при виконанні зорових робіт високої точності загальна освітленість повинна складати 300 лк, а комбінована – 750 лк; аналогічні вимоги при виконанні робіт середньої точності – 200 і 300 лк відповідно.

Крім того усе поле зору повинно бути освітлене досить рівномірно – це основна гігієнічна вимога. Іншими словами, ступінь освітлення приміщення і яскравість екрана комп'ютера повинні бути приблизно однаковими, тому що яскраве світло в районі периферійного зору значно збільшує напруженість очей і, як наслідок, приводить до їх швидкої стомлюваності.

Параметри мікроклімату можуть мінятися в широких межах, в той час як необхідною умовою життєдіяльності людини є підтримка сталості температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату – створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

Обчислювальна техніка є джерелом істотного тепловиділення, яке може привести до підвищення температури і зниження відносної вологості в приміщенні.

У приміщеннях, де встановлені комп'ютери, повинні дотримуватися визначені параметри мікроклімату. У санітарних нормах СН-245-71 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (таблиця 4.1).

Таблиця 4.1 – Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні	22...24 °С
	Відносна вологість	40...60 %
	Швидкість руху повітря	до 0,1 м/с
Теплий	Температура повітря в приміщенні	23...25 °С
	Відносна вологість	40...60 %
	Швидкість руху повітря	0,1...0,2 м/с

Шумом називають усякий несприятливо діючий на людину звук. З фізичної точки зору звук являє собою механічні коливання пружного середовища.

Слуховий орган людини сприймає у вигляді чутного звуку коливання пружного середовища, які мають частоту приблизно від 20 до 20000 Гц, але найбільш важливий для слухового сприйняття інтервал від 45 до 10000 Гц.

Сприйняття людиною звуку залежить не тільки від його частоти, але і від інтенсивності і звукового тиску.

Несприятлива дія шуму на людину залежить не тільки від рівня звукового тиску, але і від частотного діапазону шуму, а також від рівномірності впливу протягом робочого часу.

У результаті несприятливого впливу шуму на працюючу людину відбувається зниження продуктивності праці, збільшується брак у роботі, створюються передумови до виникнення нещасливих випадків. Усе це обумовлює велике оздоровче й економічне значення заходів щодо боротьби із шумом.

У таблиці 4.2 зазначені граничні рівні звуку в залежності від категорії ваги і напруженості праці, які є безпечними у відношенні збереження здоров'я і працездатності.

Таблиця 4.2 – Граничні рівні звуку, дБ, на робочих місцях.

Категорія напруженості праці	Категорія важкості праці			
	I Легка	II Середня	III Важка	IV Дуже важка
I Мало напружена	80	80	75	75
II Помірковано напружена	70	70	65	65
III Напружена	60	60	—	—
IV Дуже напружена	50	50	—	—

Електромагнітне й іонізуюче випромінювання

Більшість учених вважає, що як короткочасний, так і тривалий вплив усіх видів випромінювання від екрана монітора не небезпечний для здоров'я персоналу, що обслуговує комп'ютери. Однак вичерпних даних щодо небезпеки впливу випромінювання від моніторів на працюючих з комп'ютерами не існує і дослідження в цьому напрямку продовжуються.

Припустимі значення параметрів неіонізуючих електромагнітних випромінювань від монітора комп'ютера представлені в таблиці 4.3.

Максимальний рівень рентгенівського випромінювання на робочому місці оператора комп'ютера зазвичай не перевищує 10 мкбер/год, а інтенсивність ультрафіолетового й інфрачервоного випромінювань від екрана монітора знаходиться в межах 10...100 мВт/м².

Таблиця 4.3 – Припустимі значення параметрів неіонізуючих електромагнітних випромінювань (відповідно до СанПіН 2.2.2.542-96)

Найменування параметра	Припустимі
Напруженість електричної складової електромагнітного поля або на відстані 50 см від поверхні	10 В/м
Напруженість магнітної складової електромагнітного поля або на відстані 50 см від поверхні відеомонітора	0,3 А/м
Напруженість електростатичного поля не повинна перевищувати: для дорослих користувачів для дітей дошкільних установ і учнів середніх спеціальних і вищих навчальних закладів	20 кВ/м 15 кВ/м

4.2 Розрахунок системи штучного освітлення приміщення

Розрахунок освітленості робочого місця зводиться до вибору системи освітлення, визначення необхідного числа світильників, їхнього типу і розміщення. Виходячи з цього, розрахуємо параметри штучного освітлення.

Зазвичай штучне освітлення виконується за допомогою електричних джерел світла двох видів: ламп накаливання і люмінесцентних ламп. Будемо використовувати люмінесцентні лампи, які у порівнянні з лампами накаливання мають ряд істотних переваг:

- по спектральній сполучі світла вони близькі до денного, природного світла;
- володіють більш високим ККД (у 1,5-2 рази вище, ніж ККД ламп накаливання);
- мають підвищену світловіддачу (у 3-4 рази вище, ніж у ламп накаливання);
- більш тривалий термін служби.

Вхідні дані:

Розрахунок освітлення проводиться для кімнати площею 20 м², ширина якої 4 м, висота – 3 м. Висота робочої поверхні 1 м. Для освітлення приймається світильник типу ППД2-500. Мінімальна освітленість лампи накаливання по нормах $E_{\min} = 300$ лк. Коефіцієнт відображення стелі $S_{\text{п}}=70\%$, стін $S_{\text{с}}=50\%$ і робочої поверхні $S_{\text{р}}=30\%$. Напруга мережі 220В.

Скористаємося методом світлового потоку.

Для визначення кількості світильників визначимо світловий потік однієї лампи, який падає на поверхню по формулі:

$$F = \frac{E_{\min} \cdot K_z \cdot S \cdot z}{N \cdot \eta}$$

де

F – світловий потік, що розраховується, лм;

$E_{\min}$ – нормована мінімальна освітленість, лк;

S – площа освітлюваного приміщення (у нашому випадку $S = 20$ м²);

z – відношення середньої освітленості до мінімального (звичайно приймається рівним 1,1...1,2, нехай $z = 1,1$);

K_z – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників у процесі експлуатації (його значення залежить від типу приміщення і характеру проведених у ньому робіт і в нашому випадку $K = 1,5$);

N – необхідна кількість світильників, яка розраховується по формулі:

$$N = \frac{S}{L^2} = \frac{S}{(1,5 \cdot h)^2} = \frac{20}{(2,25)^2} \approx 4$$

η – коефіцієнт використання, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в частках одиниці; залежить від характеристик світильника, розмірів приміщення, фарбування стін, стелі і робочої поверхні, які характеризуються відповідними коефіцієнтами відображення). Значення коефіцієнтів приведені вище. Значення η визначимо по таблиці коефіцієнтів

використання різних світильників. Для цього обчислимо індекс приміщення по формулі:

$$i = \frac{S}{h \cdot (A + B)},$$

де

S – площа приміщення, $S = 20 \text{ м}^2$;

h – розрахункова висота підвісу, $h = 1,5 \text{ м}$;

A – ширина приміщення, $A = 4 \text{ м}$;

B – довжина приміщення, $B = 5 \text{ м}$.

Підставивши у формулу 5.3 значення одержимо: $i = \frac{20}{1,5 \cdot (4 + 5)} \approx 1,48$.

Знаючи індекс приміщення i , по таблиці знаходимо $\eta = 0,67$.

Підставимо всі значення у формулу 5.1 для визначення світлового потоку F :

$$F = \frac{300 \cdot 1,5 \cdot 20 \cdot 1,1}{4 \cdot 0,67} \approx 3347 \text{ (лк)};$$

Для освітлення вибираємо лампу типу Г125-135-200, потужністю 200 Вт, світловий потік якої $F = 3200 \text{ лк}$, найбільш близький до розрахункового.

При цьому фактична освітленість

$$F = E_{\min} \frac{F_n}{F_p} = 300 \frac{3200}{3347} \approx 287 \text{ (лк)}.$$

Загальна потужність $P_{\text{обц}} = P_{\text{л}} \cdot N = 200 \cdot 4 = 800 \text{ (Вт)}$.

4.3 Розробка заходів щодо зменшення впливу шкідливих факторів

Організація і поліпшення умов праці на робочому місці є одним з найважливіших резервів продуктивності праці й економічної ефективності виробництва, а також подальшого розвитку самої працюючої людини. У цьому головний прояв соціального й економічного значення організації і поліпшення умов праці.

Нижче приведені найбільш розповсюджені організаційні заходи і технічні засоби захисту людей, які працюють з комп'ютерами, від шкідливих виробничих факторів.

Колірне оформлення і світлове відображення

Фарбування приміщень і меблів повинні сприяти створенню сприятливих умов для здорового сприйняття, гарного настрою.

Джерела світла, такі як світильники і вікна, що дають відображення від поверхні екрана, значно погіршують точність знаків і спричиняють перешкоди фізіологічного характеру, які можуть виразитися в значній напрузі, особливо при тривалій роботі. Відображення, включаючи відображення від вторинних джерел світла, повинне бути зведене до мінімуму. Для захисту від надлишкової яскравості вікон можуть бути застосовані штори й екрани.

У залежності від орієнтації вікон рекомендується наступна фарбування стін і підлоги:

- *вікна орієнтовані на південь*: стіни – зеленувато-блакитний або світло-блакитний кольори; підлога – зелений;
- *вікна орієнтовані на північ*: стіни – світло-жовтогарячий або оранжево-жовтий кольори; підлога – червонясто-жовтогарячий;
- *вікна орієнтовані на схід*: стіни – жовто-зеленого кольору; підлога – зелений або червонясто-жовтогарячий;
- *вікна орієнтовані на захід*: стіни – жовто-зелений або блакитнувато-зелений кольори; підлога – зелений чи червонясто-жовтогарячий.

У приміщеннях, де знаходиться комп'ютер, необхідно забезпечити наступні величини коефіцієнта відображення: для стелі – 60...70%, для стін –

40...50%, для підлоги – близько 30%. Для інших поверхонь і робочих меблів – 30...40%.

Шум і вібрація

Зниження шуму, створюваного на робочих місцях внутрішніми джерелами, а також шуму проникаючого ззовні, є дуже важливою задачею. Зниження шуму в джерелі надходження можна забезпечити застосуванням пружних прокладок між підставою машини, приладу й опорною поверхнею. В якості прокладки використовуються гума, повсть, пробка, різної конструкції амортизатори. Під настільні шумливі апарати можна підкладати м'які коврики із синтетичних матеріалів, а під ніжки столів, на яких вони встановлені, – прокладки з м'якої гуми, повсті, товщиною 6-8 мм. Кріплення прокладок можливе шляхом приклейки їх до опорних частин.

Можливе також застосування звукоізолюючих кожухів, що не заважають технологічному процесу. Не менш важливим для зниження шуму в процесі експлуатації є питання правильного і своєчасного регулювання, змазування і заміни механічних вузлів шумливого устаткування.

Рациональне планування приміщення і розміщення устаткування є важливим чинником, що дозволяє знизити шум при існуючому устаткуванні ЕОМ. Приміщення необхідно розташовувати в далечині від шумливого і вібруючого устаткування.

Зниження рівня шуму, що проникає у виробниче приміщення ззовні, може бути досягнуте збільшенням звукоізоляції спеціальних конструкцій, ущільненням по периметру притворів вікон, дверей.

Таким чином для зниження шуму, створюваного на робочих місцях внутрішніми джерелами, а також шуму, що проникає з поза, слід:

- послабити шум самих джерел (застосування екранів, звукоізолюючих кожухів);
- знизити ефект сумарного впливу відбитих звукових хвиль (звуковбирні поверхні конструкцій);

- застосовувати раціональне розташування устаткування;
- використовувати архітектурно-планувальні і технологічні рішення ізоляцій джерел шуму.

Мікроклімат

Нормалізація мікроклімату приміщення може бути досягнута поліпшенням вентиляції приміщення за рахунок витяжних вентиляторів і нормального функціонування вбудованої вентиляції будинку.

Освітлення

У приміщеннях з обчислювальною технікою, як правило, застосовується бічне природне освітлення. Робочі кімнати і кабінети повинні мати природне освітлення. В інших приміщеннях допускається штучне освітлення.

Освітленість на поверхні столу в зоні розміщення робочого документа повинна бути 300-500 лк, також допускається установка світильників місцевого освітлення для підсвічування документів, але з такою умовою, щоб воно не створювало відблисків на поверхні екрана і не збільшувало освітленість екрана більш ніж на 300 лк.

В якості джерела світла при штучному висвітленні повинні застосовуватися переважно люмінесцентні лампи типу ЛБ. При порядкуванні відбитого освітлення в адміністративно-суспільних приміщеннях допускається застосування металогалогених ламп потужністю до 250 Вт. Допускається застосування ламп накаливання у світильниках місцевого освітлення.

Загальне освітлення варто виконувати у вигляді суцільних або переривчастих ліній світильників, розташованих збоку від робочих місць, паралельно лінії зору користувача при рядному розташуванні ПЕОМ. При периметральному розташуванні комп'ютерів лінії світильників повинні розташовуватися локалізовано над робочим столом, ближче до його переднього краю, зверненого до оператора.

Для забезпечення нормованих значень освітленості в приміщеннях використання ПЕОМ варто проводити чищення стекол віконних рам і світильників не рідше двох разів у рік і проводити своєчасну заміну перегорілих ламп.

Ергономіка робочого місця

Проектування робочих місць, оснащених відеотерміналами, відноситься до числа важливих проблем ергономічного проектування в області обчислювальної техніки.

Робоче місце і взаємне розташування всіх його елементів повинне відповідати антропометричним, фізичним і психологічним вимогам. Велике значення має також характер роботи. Зокрема, при організації робочого місця програміста повинні бути дотримані наступні основні умови: оптимальне розміщення устаткування, що входить до складу робочого місця і достатній робочий простір, що дозволяє здійснювати всі необхідні рухи і переміщення.

Ергономічними аспектами проектування відеотермінальних робочих місць, зокрема, є: висота робочої поверхні, розміри простору для ніг, вимоги до розташування документів на робочому місці (наявність і розміри підставки для документів, можливість різного розміщення документів, відстань від очей користувача до екрана, документа, клавіатури і т.д.), характеристики робочого крісла, вимоги до поверхні робочого столу, регулюємість елементів робочого місця.

Головними елементами робочого місця програміста є стіл і крісло. Основним робочим положенням є положення сидячи.

Робоча поза сидячи викликає мінімальне стомлення програміста. Рациональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташовується в зоні легкої досяжності робочого простору.

Для комфортної роботи стіл повинний задовольняти наступним умовам:

- висота столу повинна бути обрана з урахуванням можливості сидіти вільно, у зручній позі, при необхідності спираючись на підлокітники;
- нижня частина столу повинна бути сконструйована так, щоб програміст міг зручно сидіти, не був змушений підгортати ноги;
- поверхня столу повинна мати властивості, що виключають появу відблисків у полі зору програміста;
- конструкція столу повинна передбачати наявність висувних шухляд (не менше ніж 3 для збереження документації, лістингів, канцелярських предметів);
- висота робочої поверхні рекомендується в межах 680-760 мм. Висота поверхні, на яку встановлюється клавіатура, повинна бути близько 650 мм.

Велике значення надається характеристикам робочого крісла. Так, рекомендована висота сидіння над рівнем підлоги знаходиться в межах 420-550 мм. Поверхня сидіння м'яка, передній край закруглений, а кут нахилу спинки – регульований.

Необхідно передбачати при проектуванні можливість різного розміщення документів: збоку від відеотермінала, між монітором і клавіатурою і т.п. Крім того, у випадках, коли відеотермінал має низьку якість зображення, наприклад помітні мелькання, відстань від очей до екрана роблять більше (близько 700 мм), чим відстань від ока до документа (300-450 мм). Узагалі при високій якості зображення на відеотерміналі відстань від очей користувача до екрана, документа і клавіатури може бути рівною.

Велике значення також надається правильній робочій позі користувача. При незручній робочій позі можуть з'явитися болі в м'язах, суглобах і сухожиллях. Вимоги до робочої пози користувача відеотермінала наступні:

- голова не повинна бути нахилена більш ніж на 20°,
- плечі повинні бути розслаблені,
- лікті – під кутом 80°...100°,
- передпліччя і кисті рук – у горизонтальному положенні.

Причина неправильної пози користувачів обумовлена наступними факторами: немає гарної підставки для документів, клавіатура знаходиться занадто високо, а документи – низько, нікуди покласти руки і кисті, недостатній простір для ніг.

З метою подолання зазначених недоліків даються загальні рекомендації: краще пересувна клавіатура; повинні бути передбачені спеціальні пристосування для регулювання висоти столу, клавіатури й екрана, а також підставка для рук.

Істотне значення для продуктивної і якісної роботи на комп'ютері мають розміри знаків, щільність їхнього розміщення, контраст і співвідношення яркостей символів і фону екрана. Якщо відстань від очей оператора до екрана дисплея складає 60...80 см, то висота знака повинна бути не менше ніж 3 мм, оптимальне співвідношення ширини і висоти знака складає 3:4, а відстань між знаками – 15...20% їхньої висоти. Співвідношення яскравості фону екрана і символів – від 1:2 до 1:15.

Під час користування комп'ютером медики радять установлювати монітор на відстані 50-60 см від очей. Фахівці також вважають, що верхня частина відеодисплея повинна бути на рівні очей або трохи нижче. Коли людина дивиться прямо перед собою, її очі відкриваються ширше, ніж коли вона дивиться вниз. За рахунок цього, площа огляду значно збільшується, викликаючи зневоднювання очей. До того ж якщо екран установлений високо, а очі широко відкриті, порушується функція моргання. Це значить, що очі не закриваються цілком, не омиваються слізною рідиною, не одержують достатнього зволоження, що приводить до їх швидкої стомлюваності.

Режим праці

Для підтримки тривалої працездатності людини велике значення має режим праці і відпочинку. Під раціональним фізіологічно обґрунтованим режимом праці і відпочинком мається на увазі таке чергування періодів роботи з періодом відпочинку, при якому досягається висока ефективність суспільно-

корисної діяльності людини, гарний стан здоров'я, високий рівень працездатності і продуктивності праці.

У таблиці 4.4 представлені зведення про регламентовані перерви, які необхідно робити при роботі на комп'ютері, в залежності від тривалості робочої зміни, видів і категорій трудової діяльності з ВДТ (відеодисплейний термінал) і ПЕОМ (відповідно до СанПіН 2.2.2 542-96 «Гігієнічні вимоги до відеодисплейних терміналів, персональних електронно-обчислювальних машин і організації робіт»).

Таблиця 4.4 – Час регламентованих перерв при роботі на комп'ютері

Категорія роботи з ВДТ чи ПЕОМ	Рівень навантаження за робочу зміну при видах			Сумарний час регламентованих перерв, хв	
	Група А, кількість знаків	Група Б, кількість знаків	Група В, годин	При 8-годинній зміні	При 12-годинній зміні
I	до 20 000	до 15 000	до 2,0	30	70
II	до 40 000	до 30 000	до 4,0	50	90
III	до 60 000	до 40 000	до 6,0	70	120

Примітка. Час перерв даний при дотриманні зазначених Санітарних правил і норм. При невідповідності фактичних умов праці вимогам Санітарних правил і норм час регламентованих перерв варто збільшити на 30%.

У відповідності із СанПіН 2.2.2 546-96 усі види трудової діяльності, пов'язані з використанням комп'ютера, розділяються на три групи:

група А: робота зі зчитування інформації з екрана ВДТ чи ПЕОМ із попереднім запитом;

група Б: робота з введення інформації;

група В: творча робота в режимі діалогу з ЕОМ.

Ефективність перерв підвищується при сполученні з виробничою гімнастикою або організацією спеціального приміщення для відпочинку персоналу зі зручними м'якими меблями, акваріумом, зеленою зоною і т.п.

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, що позитивно впливає на продуктивність праці.

Висновки

В роботі була проведена розробка системи управління контентом web-порталу МПТУ.

В роботі були виконані:

- огляд систем управління контентом;
- огляд існуючих рішень для систем управління контентом;
- аналіз існуючих інтернет-технологій для створення систем управління контенту;
- аналіз основних інструментів для розробки web-порталів;
- розроблено web-портал з можливістю управління контентом, для чого було вивчено предметну область, сформовано вимоги до системи, розроблено концепцію системи, розроблено технічне завдання на створення системи, розроблено загальносистемні рішення, рішення з організаційного, інформаційного, технічного та програмного забезпечення системи та розроблено супроводжувальну документацію.

Під час прийняття загальносистемних рішень була визначена архітектура системи, її дерево функцій, а також вирішені питання її взаємодії з іншими системами.

Рішення з організаційного забезпечення торкнулися перерозподілу функцій фахівців підрозділів ЗВО, які беруть участь в роботі системи. В питанні інформаційного забезпечення була визначена модель даних системи та прийнято рішення про використання реляційної бази даних. Рішення з технічного забезпечення склалися зі схеми автоматизації системи, проробки необхідного комплексу технічних засобів та проектування схеми інтеграції системи у ЗВО.

На етапі прийняття рішень, що стосувалися програмного забезпечення, було обрано методологію проведення розробки, визначено структуру ПЗ та функції його частин.

Також було проведено економічну оцінку розробленого проекту системи. Розроблену систему планується впроваджувати у МПТУ. Також, вона може бути впроваджена в інших закладах освіти.

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. Полное руководство, 9-е изд., М.:Вильямс, 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. Внедрение SQL-кода. Википедия — свободная энциклопедии. [Електроний ресурс]. – Режим доступу URL: http://ru.wikipedia.org/wiki/SQL_injection.
10. Межсайтовый скриптинг. Википедия — свободная энциклопедии. [Електроний ресурс]. – Режим доступу URL: <http://ru.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. Безопасность жизнедеятельности. Под редакцией Белова Н.А. – М.: Знание, 2000.— 364с.

13. ГОСТ 34.603 – 92. Информационная технология. Виды испытаний автоматизированных систем.
14. Новиков Ю.В. Экология, окружающая среда и человек: Учебное пособие для вузов. – М.: Агентство —ФАИР®, 1998. – 295с.
15. Самгин Э.Б. Освещение рабочих мест. – М.: МИРЭА, 1989. – 186с.
16. Борьба с шумом на производстве: Справочник / Юдина Э.Я., Борисова Л.А. под редакцией Юдина Э.Я. – М.: Машиностроение, 1985. – 400с.
17. Савельева Н. Системи управління контентом (укр.)// Відкриті системи. - 2004. - № 4.
18. Madhusudhan Konda. Just Hibernate: A Lightweight Introduction to the Hibernate Framework – O'REILLY, 2014. – 137 p.

Додаток А – Технічне завдання на розробку

1 Вступ

1.1 Назва – Система управління контентом web-порталу МПТУ

1.2 Призначення програми: надати можливість зручно та швидко управляти контентом необхідним для виконання діяльності палацу творчості учнів.

1.3 Область застосування: МПТУ

2 Призначення розробки

Призначена для спрощення, уніфікації, прискорення роботи методистів навчального закладу позашкільної 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 МПТУ;
- Своєчасне доведення інформації до гуртківців, батьків 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 ПЗ, 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 ПЗ.

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: «МПТУ 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.mcms.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(jsonReques
            t, MCMSArticle.class);
                Map<String, String> errors =
            validator.validateCreate(article);

                if (errors.size()>0)
                {

                    httpServletToolbox.writeBodyResponseAsJson(response, "",
            errors);

                    return;
                }

                article.setLastModified(Calendar.getInstance(TimeZone.getTi
            meZone("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.DESENDING);
                } 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,
HttpServletRequest 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<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,
HttpServletRequest 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(jsonReques
t, 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.getTi
meZone("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.mcms.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 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 MCMSFile getDirectory(String externalKey,
        MCMSAdminDataStorage adminStorage) throws MCMSException
        {
            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 MCMSException
        {
            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 MCMSException
    {
        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
MCMSException, 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 (MCMSException 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 MCMSException
    {
        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() &&
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
    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
        {
            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
        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);
            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.getAdjustedConte
ntType());

        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,
HttpServletRequest 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.DESENDING);

        } 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.JSONFromObject(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 MCMSException("cannot get file
details", e);
    }
}

    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");
            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,
HttpServletRequest 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
MCMSEException

```



```

        {
            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);
    }
}

```