

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка веб-сайту для інформаційної підтримки
школи-інтернату: забезпечення доступу до ключових
ресурсів та взаємодія зі спільнотою»**

Виконала студентка групи ЗПЗк-22-1
СТРУК Вікторія Віталіївна

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СТРУК Вікторія Віталіївна**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Розробка веб-сайту для інформаційної підтримки школи-інтернату: забезпечення доступу до ключових ресурсів та взаємодія зі спільнотою»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №75/с**

Вихідні дані для роботи: **автоматизована інформаційна підтримка закладу освіти та веб-орієнтовані технології для розробки веб-сайту, що забезпечують доступ до ключових ресурсів та надають можливість взаємодії учасникам освітнього процесу**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Юрій ЄФРЕМОВ	02.03.2024	02.03.2024
2	Юрій ЄФРЕМОВ	04.04.2024	04.04.2024
3	Юрій ЄФРЕМОВ	02.05.2024	02.05.2024

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-сайту школи-інтернату та пояснювальної записки. Пояснювальна записка до випускної роботи містить 67 сторінок, 59 ілюстрацій, 11 таблиць, 1 додаток та 18 використаних джерел.

Метою роботи є розробка веб-сайту для інформаційної підтримки школи-інтернату. Сайт має забезпечувати доступ до ключових ресурсів закладу та надавати можливість взаємодіяти зі спільнотою.

В роботі було визначено основні завдання на розробку веб-сайту, виконано аналіз функціональних вимог, визначено технології та архітектуру для реалізації веб-сайту, спроектовано та реалізовано алгоритми роботи Web системи, виконано тестування програмного продукту.

Додаток має зручний інтерфейс, що надає користувачам легко знайти потрібну інформацію.

КЛЮЧОВІ СЛОВА: ВЕБ-САЙТ, ШКОЛА-ІНТЕРНАТ, СИСТЕМА УПРАВЛІННЯ КОНТЕНТОМ, ІНФОРМАЦІЙНА ПІДТРИМКА.

					ІПЗ.КР.Б – 121 – 24 - ПЗ					
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка веб-сайту для інформаційної підтримки школи-інтернату			Літ.	Арк.	Акрушів
Розроб.		Струк В.В.								
Перевір.		Єфремов Ю.М.							3	91
								Житомирська політехніка, група ЗІПЗк-22-1		
Н. Контр.		Тичина О.П.								
Затверд.		Вакалюк Т.А.								

ABSTRACT

The bachelor's thesis consists of a website for a boarding school and an explanatory note. The explanatory note for the thesis contains 67 pages, 59 illustrations, 11 tables, 1 appendix, and 18 references.

The purpose of the work is to develop a website for informational support of the boarding school. The site should provide access to key resources of the institution and enable interaction with the community.

The work identified the main tasks for developing the website, conducted an analysis of functional requirements, determined the technologies and architecture for implementing the website, designed and implemented algorithms for the operation of the web system, and performed testing of the software product.

The appendix features a user-friendly interface that allows users to easily find the necessary information.

KEYWORDS: WEBSITE, BOARDING SCHOOL, CONTENT MANAGEMENT SYSTEM, INFORMATIONAL SUPPORT.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-САЙТУ ШКОЛИ-ІНТЕРНАТ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	13
1.3 Вибір архітектури для веб-сайту школи-інтернат	16
1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення	20
Висновки до першого розділу	23
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-САЙТУ ШКОЛИ-ІНТЕРНАТ	24
2.1 Визначення варіантів використання.....	24
2.2 Розробка бази даних	31
2.3 Проектування та реалізація алгоритмів роботи системи	36
Висновки до другого розділу	44
РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ СИСТЕМИ	45
3.1 Розгортання веб-сайту	45
3.2 Структура інтерфейсу веб-сайту	47
3.3 Тестування роботи веб-сайту	62
Висновки до третього розділу	64
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	66
ДОДАТКИ	68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

CRUD – основні операції, що використовуються для управління даними (Create, Read, Update, Delete)

DOM – програмний інтерфейс для веб-документів, що представляє структуру документа (Document Object Model)

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах

ІСУО – Інформаційна система управління освітою (ІСУО, isuo.org) — вебсервіс, інтернет-портал, онлайн база даних, програмний комплекс, інформаційна система консолідації даних з дошкільних і загальноосвітніх навчальних закладів України

MVC – схема розподілення даних додатків і керуючої логіки на три окремі компоненти Model-View-Controller

MVVM – схема розробки з компонентів Model-View-ViewModel

MVP – схема розробки з компонентів Model-View-Presenter

NPM – це менеджер пакунків для мови програмування JavaScript (Node Package Manager)

ORM – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних» (Object-relational mapping)

ПЗ – програмне забезпечення

VPS – віртуальний виділений сервер

ВСТУП

Актуальність теми. Створення веб-сайту для освітнього закладу стає важливим кроком у впровадженні прозорості та відкритості. У сучасному цифровому світі веб-сайт є ключовим інструментом для забезпечення доступу до інформації для всіх зацікавлених осіб. Він дозволяє забезпечити широкий доступ до різноманітних даних, таких як розклад занять, академічні програми, фінансова звітність тощо. Такий інструмент допомагає у зміцненні взаємозв'язку між установою та її спільнотою, а також відповідає вимогам законодавства, визначеним у Законі України «Про освіту» [1].

Завдяки веб-сайту адміністрація школи-інтернату може краще керувати процесами, такими як розклад занять, відомості про учнів, збір документів тощо. Це може спростити рутинні процедури та звільнити час для більш якісної роботи з учнями.

Сучасні батьки все частіше обирають школи для своїх дітей на основі доступності інформації та технологічної розвиненості закладу. Наявність веб-сайту з можливістю ознайомлення з навчальним процесом, інфраструктурою школи-інтернату, відгуками інших батьків може зробити школу більш привабливою для нових учнів.

Отже, розробка веб-сайту для інформаційної підтримки школи-інтернату має значення як для покращення якості освіти та комфорту учнів, так і для оптимізації адміністративних процесів та підвищення привабливості закладу для нових учасників.

Метою даної роботи є розробка веб-сайту для інформаційної підтримки школи-інтернату. Сайт має забезпечувати доступ до ключових ресурсів закладу та надавати можливість взаємодіяти зі спільнотою.

Поставлена мета обумовлює необхідність виконання наступних **завдань**:

- виконати аналіз функціональних вимог веб-сайту школи-інтернат;
- визначити технології для реалізації веб-сайту;
- обрати архітектуру веб-орієнтованої системи закладу;
- виконати опис методів для взаємодії з сайтом;

- розробити базу даних для збереження інформації від освітнього закладу та користувачів;
- визначити об'єктну структуру веб-сайту школи-інтернат;
- розробити веб-сайт для школи-інтернату;
- протестувати веб-орієнтовану систему;

Об'єктом дослідження є автоматизована інформаційна підтримка закладу освіти.

Предметом дослідження є використання веб-орієнтованих технологій для розробки веб-сайту, що забезпечує доступ до ключових ресурсів та надає можливість взаємодії учасникам освітнього процесу.

В роботі було використано методи системного аналізу, веб-програмування та дизайну.

Практичне значення отриманих результатів: створений веб-додаток містить основні функції для керування адміністративними процесами навчального закладу.

Структура пояснювальної записки. Випускна робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел із 18 найменувань, 1 додатку, 11 таблиць та 59 ілюстрацій по тексту. Повний обсяг роботи становить 91 сторінку, з них 57 сторінок основного тексту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-САЙТУ ШКОЛИ-ІНТЕРНАТ

1.1 Постановка задачі

У важливому процесі гуманізації освіти актуальним є впровадження ресурсно-орієнтованого навчання з використанням інформаційно-комунікаційних технологій. Цей процес передбачає не тільки відповідність структури та інструментарію навчання сучасним вимогам суспільства, а і віковим, комунікативним, пізнавальним потребам учасників освітнього процесу. Вже декілька десятиліть тому у закладах освіти виникла необхідність створення власних сайтів, які, в першу чергу, сприймалися як їх візитівки. Поступово до інформаційної складової сайтів з'явилися нові вимоги, а в сучасних умовах вони все більше націлені стати сучасним інформаційно-освітнім ресурсом закладу освіти.

Головними завданнями для організації роботи сайту закладу освіти є:

- інформування про особливості закладу, освітні стандарти, навчальні програми та плани, запроваджені в освітньому процесі;
- забезпечення відкритості діяльності закладу освіти та висвітлення його діяльності в мережі Інтернет;
- інформаційно-освітнє забезпечення діяльності учасників та учасниць освітнього процесу, зокрема доступу до необхідних інформаційних та освітніх ресурсів, дистанційної форми здобуття освіти тощо;
- створення умов для мережевої взаємодії з іншими установами, спрямованої на розв'язання актуальних питань організації освітнього процесу, надання можливості відвідувачам сайту поставити питання й отримати на них відповіді тощо;
- обмін педагогічним досвідом;
- розвиток інтересу здобувачів освіти до дослідницької та інноваційної діяльності, зокрема із застосуванням інформаційних технологій.

Первісне сприймання сайту як звичайної візитівки закладу відійшло у минуле, адже зараз функціонування сайту як інформаційного ресурсу регламентується законодавством України й додатково – рішеннями педагогічної ради (ради закладу освіти) та/чи наказами директора закладу. Так відповідно до статті 30 Закону України «Про освіту» [1] заклади освіти формують відкриті та загальнодоступні ресурси з інформацією про їхню діяльність та оприлюднюють ліцензії на провадження освітньої діяльності, сертифікати про акредитацію освітніх програм, освітні програми, що реалізуються в закладі освіти, річний звіт про діяльність закладу тощо.

Визначимо основні функціональні можливості, які має мати будь-який веб-сайт школи-інтернату:

1. Доступ до ключових ресурсів
2. Система реєстрації користувачів
3. Взаємодія зі спільнотою
4. Зручність інтерфейсу

Ключовою функціональністю веб-сайту школи-інтернату є забезпечення легкого доступу до основних ресурсів, таких як документація, новини, оголошення, фото- та відеоматеріали та інші важливі ресурси. Важливо створити структуровані секції для різних вікових груп та предметних областей, щоб користувачі могли швидко знаходити необхідну інформацію.

Веб-сайт повинен мати ефективну систему реєстрації користувачів, щоб дозволити учням, батькам і вчителям зареєструватися та отримувати персоналізовану інформацію. Це допомагає в управлінні доступом до конфіденційних даних та створює можливість для ведення спільної комунікації.

Веб-сайт повинен підтримувати функції для взаємодії між учнями, вчителями і батьками. Це може включати форуми, чати, коментарі до навчальних матеріалів та інші засоби комунікації. Така взаємодія сприяє обміну досвідом та підтримці спільного духу.

Інтерфейс веб-сайту повинен бути зручним та інтуїтивно зрозумілим для всіх користувачів. Зрозумілі навігаційні елементи, послідовність дій та

привабливий дизайн допомагають створити позитивний досвід використання веб-сайту.

Основною метою розробленого проекту є забезпечення відкритості та доступності інформації про діяльність школи-інтернату для широкого кола зацікавлених осіб. Веб-сайт має відповідати сучасним вимогам до прозорості та інформаційної відкритості в освітній сфері згідно статті 30 Закону України «Про освіту», сприяти залученню громадськості та покращенню якості освіти.

Згідно мети було створено технічне завдання:

а) Функціональні вимоги:

1. Реєстрація та авторизація:

1.1. Створення системи реєстрації користувачів.

1.2. Авторизація та автентифікація за допомогою зареєстрованого імені та паролю.

2. Особистий кабінет:

2.1. Розробка особистого кабінету з доступом до персональної інформації.

2.2. Забезпечення можливості зміни особистих даних та паролю.

3. Форма зворотного зв'язку:

3.1. Розробка форми зворотного зв'язку для користувачів.

3.2. Забезпечення можливості автоматичного підтвердження отримання повідомлення та відображення інформації про час відповіді.

4. Мультимедійний контент:

4.1. Розробка системи для завантаження та відображення фотографій та відео.

4.2. Розробка системи для розміщення новин.

4.3. Розробка системи для завантаження документів.

5. Спілкування:

5.1. Розробка секції для коментування та обговорень відкритих матеріалів зареєстрованими користувачами.

6. Адміністративна панель:

6.1. Реалізація адміністративної панелі для здійснення операцій створення, читання, оновлення, видалення вмісту сайту.

б) Дизайн та інтерфейс:

1. Загальний дизайн:

1.1. Мінімалістичний та легкий дизайн.

1.2. Адаптивний дизайн для зручного використання на різних пристроях.

в) Технічні вимоги:

1. Мови програмування:

1.1. Використання HTML, CSS, JavaScript для клієнтської сторони.

1.2. Використання JavaScript для серверної частини.

2. Бази даних:

2.1. Використання реляційної бази даних PostgreSQL.

3. Хостинг та сервер:

3.1. Розгортання на віртуальному сервері з достатньою продуктивністю та ресурсами.

3.2. Встановлення та налаштування Nginx для обробки запитів до веб-сайту.

г) Розгортання та підтримка:

1. Процедура розгортання:

1.1. Докладний опис кроків для налаштування веб-сервера та встановлення необхідних залежностей на VPS.

1.2. Зазначення конфігураційних параметрів та налаштувань, необхідних для коректної роботи системи на VPS.

2. Тестування під час розгортання:

2.1. Сценарії та процедури тестування на VPS для переконання в коректності розгортання та оптимальності продуктивності.

1.2 Аналіз аналогів програмного продукту

Для того, щоб дізнатись актуальну інформацію про діючі заклади освіти спеціальних інтернатів Житомирської області використано портал ІСУО [2], та проаналізовано їх офіційні веб-сайти.

Веб-сайт Бердичівської спеціальної школи Житомирської обласної ради (рис. 1.1) є відмінним прикладом ефективного використання інформаційно-комунікаційних технологій у сфері освіти. Цей сайт відображає всю необхідну інформацію, яка має бути представлена згідно з законодавством України. Крім того, він пропонує користувачам широкий спектр можливостей, включаючи перегляд новин, фото та відео, розміщення оголошень, можливість реєстрації, залишення коментарів та звернень через форму зворотного зв'язку.

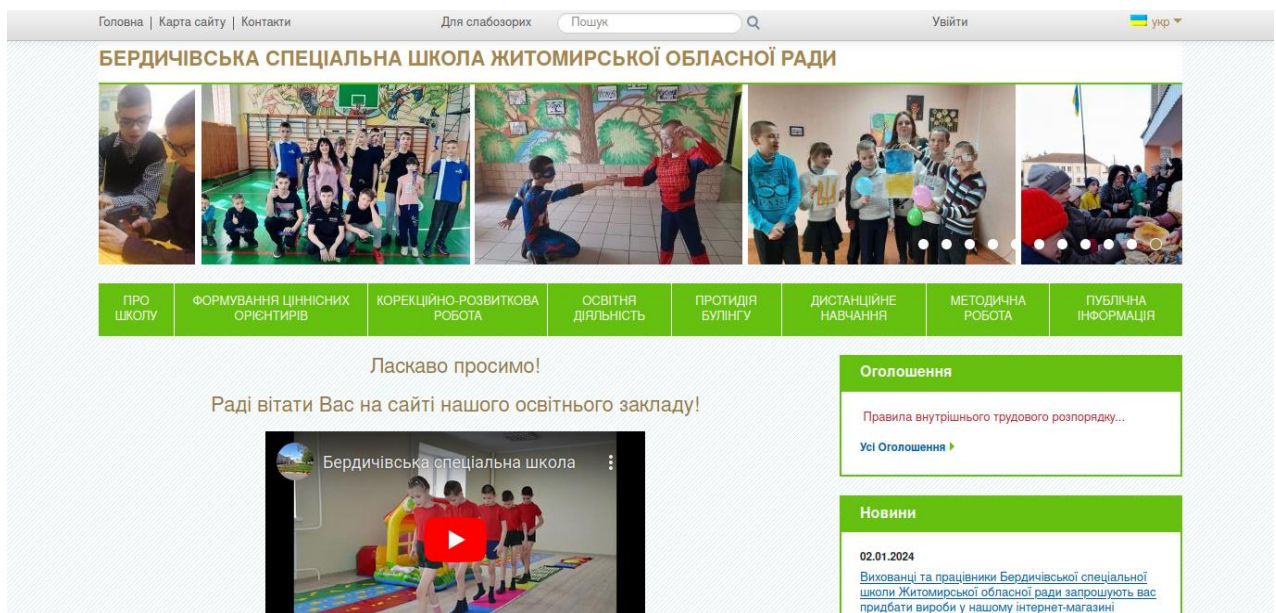


Рис. 1.1 - Веб-сайт «Бердичівська спецшкола» [3]

Особливу увагу слід звернути на зручний інтерфейс сайту, який доступний на пристроях різних розмірів. Наявність можливості перегляду сайту у чорно-білому варіанті підвищує його доступність для людей з порушенням зору. Крім того, наявність спеціального поля для пошуку полегшує навігацію та забезпечує швидкий доступ до необхідної інформації. Такий підхід відповідає

сучасним стандартам веб-дизайну і сприяє позитивному враженню від використання сайту.

Зазначений веб-сайт відповідає вимогам сучасного інформаційного середовища та служить важливим засобом комунікації для школи і її оточення. Його успішна реалізація демонструє важливість використання технологій у сфері освіти та підкреслює значення доступності та зручності взаємодії з користувачами.

При аналізі функціональних можливостей веб-сайту Денишівської спеціальної школи (рис. 1.2), можна відзначити, що він успішно відповідає деяким ключовим завданням, зокрема, забезпечує доступ до ключових ресурсів та має наявність мультимедійного контенту. Однак, слід відзначити, що відсутність можливості реєстрації користувачів та інструментів для взаємодії та обміну думками обмежують можливості співробітництва та комунікації на сайті.

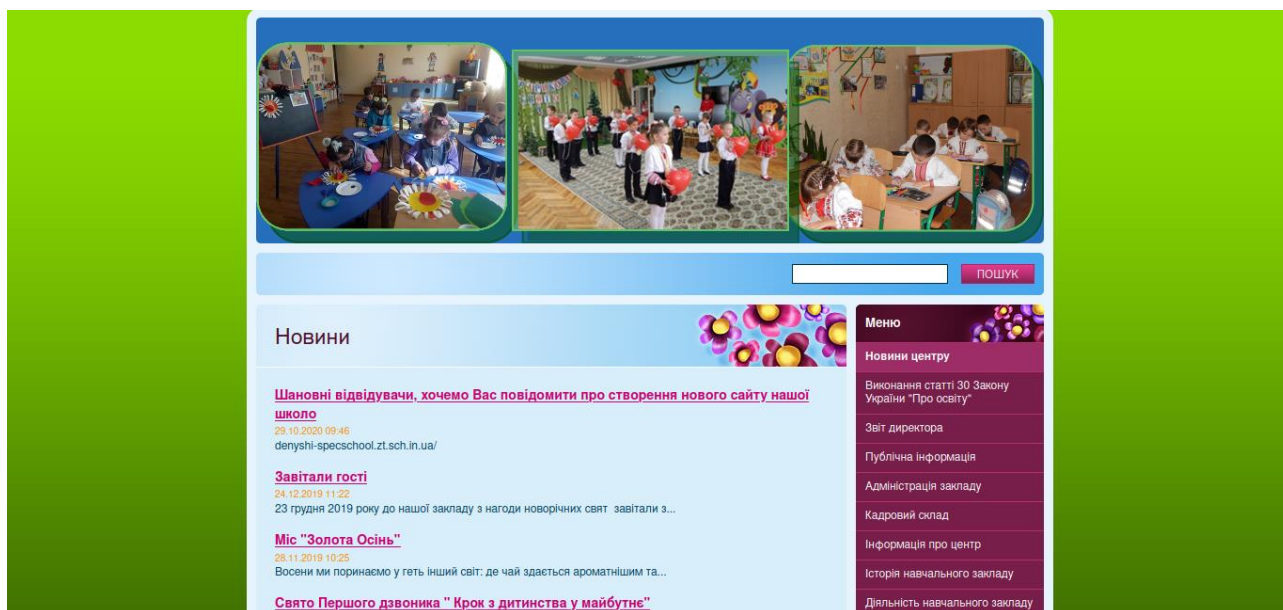


Рис. 1.2 - Веб-сайт «Денишівська спеціальна школа» [4]

Зазначена відмінність може перешкоджати створенню активної спільноти користувачів, а також обміну досвідом та ідеями. Крім того, важливою аспектом є те, що інтерфейс сайту орієнтований переважно на великі пристрої, що може ускладнювати використання сайту на мобільних пристроях чи планшетах.

Відзначаючи функціональність офіційного сайту КНЗ «Кмитівська спеціальна загальноосвітня школа-інтернат» Житомирської обласної ради (рис. 1.3), можна виявити кілька проблем, що потребують уваги. Відсутність адаптивного інтерфейсу ускладнює доступ до інформації для користувачів, що використовують пристрої з різними розмірами екранів. Проблеми з відображенням мультимедійного контенту можуть ускладнити сприйняття інформації та знизити користувацький досвід. На позитивній ноті, наявність можливості збільшення розміру тексту для людей із проблемами зору свідчить про деякий рівень доступності сайту. Хоча на сайті є форма зворотного зв'язку, відсутність можливості залишити коментар чи обговорити тему обмежує можливості спілкування та взаємодії користувачів. На сайті також присутня важлива документація, що може бути корисною для відвідувачів, така як регламенти, положення та інші документи. Наявність постів від психолога та логопеда сприяє розширенню функціональності сайту, забезпечуючи корисні поради та ресурси для психологічної та логопедичної підтримки відвідувачів. Додатково, наявність корисних посилань може допомогти відвідувачам знайти додаткові ресурси та інформацію на схожі теми.



Рис. 1.3 - Веб-сайт «Кмитівська спецшкола інтернат» [5]

Підсумуємо загальну характеристику щодо основних функціональних можливостей веб-сайтів вищезгаданих освітніх закладів (табл. 1.1).

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						15

Порівняльна таблиця

Назва	Доступ до ключових ресурсів	Система реєстрації користувача	Взаємодія зі спільнотою	Зручність інтерфейсу
Веб-сайт «Бердичівська спецшкола»	+	+	+	+
Веб-сайт «Денишівська спеціальна школа»	+	-	+	+
Веб-сайт «Кмитівська спецшкола інтернат»	-	+	-	+
Веб-сайт для інформаційної підтримки школи-інтернат	+	+	+	+

Після аналізу аналогів веб-сайтів для спеціальних шкіл-інтернатів стає очевидним, що деякі з них мають обмежені можливості, які є ключовими для ефективного поширення інформації про заклад та забезпечення зв'язку з представниками закладів у дистанційному режимі.

З урахуванням цього, було розроблено веб-сайт для КНЗ "Кмитівська спеціальна школа-інтернат", який включає всі основні функціональні можливості, що наразі відсутні на офіційному сайті, і також було проведено зміни в інтерфейсі користувача.

1.3 Вибір архітектури для веб-сайту школи-інтернат

У процесі розробки веб-сайту школи-інтернату, одним з ключових етапів є вибір архітектури, яка буде визначати структуру, організацію та взаємодію компонентів системи.

У додатку закладу реалізовано клієнт-серверну архітектуру. Її можна означити, як концепцію інформаційної мережі в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів (рис. 1.4). Клієнт-серверна архітектура визначає такі типи компонентів:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

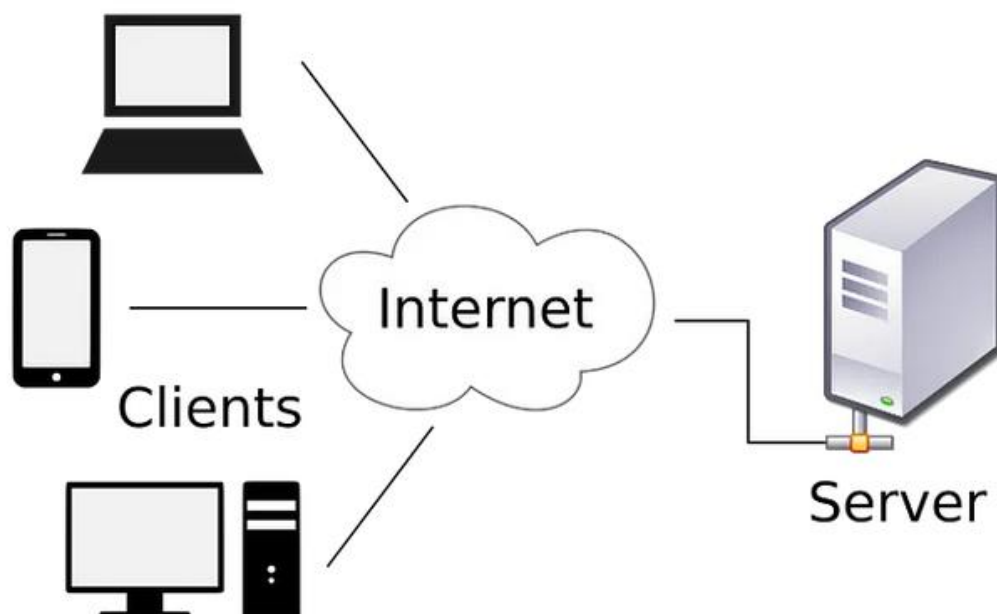


Рис. 1.4 - Клієнт-серверна архітектура веб-додатку

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Основна ідея архітектури «клієнт-сервер» полягає в поділі мережевого додатка на кілька компонентів, кожен з яких реалізує специфічний набір сервісів. Компоненти такого додатку можуть виконуватися на різних комп'ютерах, виконуючи серверні і/або клієнтські функції. Це дозволяє підвищити надійність, безпеку і продуктивність мережевих додатків і мережі в цілому.

При розробці клієнт-серверного додатку, який претендує на довготривале існування дуже важливо вибрати архітектурний патерн, який повністю задовільнить всі потреби команди та самого продукту.

Шаблони проектування ПЗ — ефектні способи вирішення задач проектування програмного забезпечення. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між класами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення.

При розробці клієнтської частини ПЗ найчастіше використовують такі патерни: MVP, MVC, MVVM. Кожен з патернів відрізняється роллю компонентів та способом взаємодії з іншими. З огляду на це, доцільним є дослідження всіх популярних рішень та способів реалізації.

Архітектурні патерни являються набором універсальних правил, які не залежать від жодної з платформ та мов.

Найпростіший патерн - це MVC. Компанія Apple рекомендує використовувати його для архітектурно простих додатків. MVC - це схема розробки програмного забезпечення, що складається з трьох основних об'єктів:

Model - це місце, де перебувають ваші дані. Там живуть такі речі, як об'єкти моделей, парсери, менеджери та мережевий код.

View - обличчя вашої програми. Її класи часто використовуються для багаторазового використання, оскільки не містять певної логіки.

Controller - опосередковує погляд і модель за допомогою шаблону делегування. В ідеальному сценарії суб'єкт контролера не знатиме конкретного

погляду, з яким він має справу. Натомість він буде спілкуватися з абстракцією за допомогою протоколу.

MVP - схема розробки, що складається з таких основних компонентів Model, View, Presenter. Model, View уже знайомі компоненти. Presenter відповідає за взаємодію між Model та View. При реалізації цього патерну View взагалі немає ніякої логіки і відповідає лише за відображення. Різниця Presenter буде відповідати за всю логіку, включаючи реагування на дії користувача та оновлення інтерфейсу користувача (через делегата) і найголовніше - наш ведучий не буде залежати від View. Важливо зазначити, що MVP використовує пасивний режим перегляду. Це означає, що всі дії будуть передані Presenter, що запустить оновлення інтерфейсу користувача за допомогою делегатів, тож перегляд буде лише проходити дії та слухати оновлення ведучого.

З розвитком реактивного програмування все більше набувають популярності патерни, в основу яких лежить патерн Спостерігач. Одним з таких є MVVM. Суть цього патерна полягає в тому, що компоненти не взаємодіють між собою на пряму, а реагують на зміну ViewModel.

Взаємодія між компонентами архітектурних патернів, описаних вище (рис. 1.5).

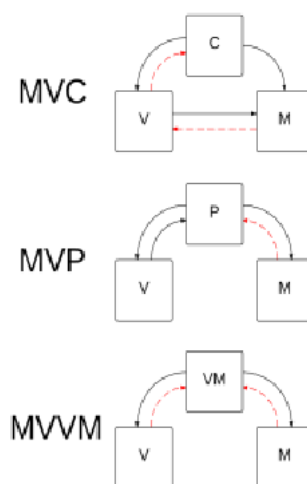


Рис. 1.5 - Зображення архітектурних патернів програмування [6]

Модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу автоматично йде зміна відображуваних даних в інших

компонентах. Всі компоненти незалежні між собою і обмінюються інформацією лиш через спостереження за ViewModel [6].

Для розробки веб-сайту було обрано MVC патерн, оскільки він забезпечує чіткий розподіл відповідальності між компонентами додатку, що робить його простим для підтримки та розширення в майбутньому, а також тому, що додаток потребує управління запитами HTTP та маршрутизацією.

1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

Для розробки проекту було обрано стек PERN.

Стек PERN є одним із багатьох корисних наборів технологій для створення веб-або мобільних додатків. Як випливає з аббревіатури, він складається з чотирьох ключових компонентів: PostgreSQL, Express, React і Node.js. Коли компоненти об'єднані, ми можемо створити програму з повним стеком із операціями CRUD [7].

Розглянемо кожен компонент детальніше.

ReactJS - це технологія з відкритим вихідним кодом, яка використовується для створення інтерфейсів користувача, спеціально призначених для односторінкових додатків. Розробники програмного забезпечення можуть створювати великі веб-додатки з використанням ReactJS, які можуть споживати дані та змінюватися з часом без перезавантаження сторінки. React використовує розумний алгоритм диференціації, щоб відновлювати в своєму DOM-вузлі лише ті дані, які потрібно відновити, залишаючи решту без змін. Наш додаток може бути легко побудований завдяки використанню багаторазових компонентів. Вражаючий дизайн React також робить розташування інтерфейсу користувача передбачуваним і знімає значну тягу з програмістів, дозволяючи їм зосередитися на більших цілях та бізнес-мисленні. React не нав'язує використання певного способу виконання конкретного завдання. Користувачі можуть вибирати з широкого спектру бібліотек для виконання певного завдання завдяки цій послугі. Порядок подій,

які викликаються протягом життєвого циклу сегмента, обробляється стратегіями життєвого циклу та реактивними хуками, двома додатковими важливими функціями [8].

PostgreSQL — це об'єктно-реляційна система керування базами даних (ORDBMS) з відкритим кодом, яка підтримує як SQL (реляційні), так і JSON (нереляційні) запити. Він сумісний з ACID і базується на таблиці, з повними обмеженнями, тригерами та ролями. Саме ці функції дозволяють створювати зв'язки між даними. Це, у свою чергу, призводить до таких переваг, як розширюваність, масштабованість, надійність, продуктивність і стійкість. Витоки PostgreSQL сягають 1986 року як частини проекту POSTGRES в Каліфорнійському університеті в Берклі та мають понад 35 років активного розвитку на основній платформі. PostgreSQL заслужив міцну репутацію завдяки своїй перевірній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти з відкритим кодом, яка стоїть за програмним забезпеченням, щоб постійно надавати ефективні та інноваційні рішення [9].

Express — це популярна веб-платформа JavaScript, розроблена для використання з Node.js. Її мета — швидко й легко створювати веб-додатки та, особливо, API [10]. Він надає набір функцій для маршрутизації, проміжного програмного забезпечення та обробки HTTP-запитів і відповідей. У стеку PERN цей компонент обробляє серверну частину програми.

Node.js — це безкоштовне, відкрите, крос-платформенне середовище виконання JavaScript, яке дозволяє розробникам створювати сервери, веб-додатки, інструменти командного рядка та скрипти [11].

Для розповсюдження сторонніх модулів для Node.js використовується менеджер пакетів NPM, що дозволяє будь-кому опублікувати власний модуль в публічний доступ, і відповідно завантажити його будь-кому у власний проєкт. NPM є популярним і має величезну спільноту, тому містить велику кількість різних пакетів бібліотек та фреймворків, що дає можливість з великою долею вірогідності знайти потрібне рішення під конкретну задачу [12].

За допомогою Node.js можна створювати різні додатки або сервіси, наприклад: месенджери, соціальні мережі, односторінкові веб-додатки, стрімінгові сервіси, платіжні сервіси тощо.

Також для роботи з базою даних без запитів SQL було використано ORM Sequelize v6. Це інструмент Node.js ORM на основі обіцянок для Postgres, MySQL, MariaDB, SQLite, Microsoft SQL Server, Oracle Database, Amazon Redshift і Snowflake's Data Cloud. Він має надійну підтримку транзакцій, відносини, активне та відкладене завантаження, реплікацію читання тощо [13].

Щоб клієнт та сервер взаємодіяли між собою було використано Axios. Це клієнт HTTP на основі Promise для Node.js та браузера. На стороні сервера він використовує рідний HTTP-модуль Node.js, тоді як на клієнті (браузер) він використовує XMLHttpRequests [14].

Веб-сайт було розгорнуто на VPS-сервері. Це фізичний сервер, на якому за допомогою набору спеціалізованих програм створюються виділені віртуальні середовища. Віртуальні середовища не взаємодіють один з одним, а їхня робота не може впливати на роботу інших віртуальних середовищ.

До переваг віртуальних серверів відносять:

- повний контроль власника системи;
- у разі перевищення максимального рівня навантажень на процесор, трафік чи операційну систему сайт не буде вимкнено;
- власник VPS не обмежений у виборі технологій; наявність власної IP-адреси.

Віртуальні сервери є високонадійними системами, які можуть масштабуватись практично під будь-які завдання та цілодобово готові до роботи.

Системні вимоги для відображення веб-сайту зазвичай є дуже низькими, оскільки більшість сучасних браузерів інтегрують технології, які працюють на різних конфігураціях комп'ютерів.

Проте, тут є деякі загальні вимоги:

1. Браузер: Сучасний та оновлений браузер, такий як Google Chrome, Mozilla Firefox, Safari або Microsoft Edge.
2. Операційна система: Windows, macOS або Linux.
3. Інтернет-з'єднання: Для завантаження веб-сторінок потрібне підключення до Інтернету.
4. Клавіатура та миша

Більшість користувачів зможуть відобразити веб-сайт на своєму комп'ютері, якщо вони мають сучасний браузер і базовий набір комп'ютерного обладнання.

Висновки до першого розділу

У першому розділі визначено перелік задач, що необхідно вирішити під час опрацювання роботи та реалізувати у вигляді програмного комплексу. Основним завданням є розробка веб-сайту для інформаційної підтримки школи-інтернату.

Описано аналогічні програмні продукти та визначено основні функціональні можливості, які полягають у забезпеченні користувачів доступом до ключових ресурсів та взаємодії між собою.

Здійснено порівняльний опис архітектурних патернів та опис архітектури, що реалізована в спроектованому додатку. Обґрунтовано вибір інструментальних засобів та визначено вимоги до апаратного забезпечення.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-САЙТУ ШКОЛИ-ІНТЕРНАТ

2.1 Визначення варіантів використання

Бізнес вимоги програмного продукту, що розробляється:

1. Основні цілі: проект створюється з метою надання інформації про школу-інтернат, включаючи контактні дані, графіки робіт, плани робіт, можливість зворотного зв'язку, посилання на соціальні мережі; звітності школою щодо проведення різних заходів у закладі у вигляді мультимедійного контенту; представлення документів.

2. Представлення проекту: проект буде реалізовано у вигляді сайту, що містить структуровану інформацію по даній тематичній області.

Вимоги користувачів. Веб-сайтом можуть користуватись зареєстровані та незареєстровані користувачі. Відмінність між ними полягає у розширеній функціональності для зареєстрованого користувача. Наприклад, додаткова можливість лишати коментарі під публікаціями. Також користувач, що виступає у ролі адміністратора слідкує за наповненням сайту актуальною інформацією.

Гість має доступ до наступних функцій (рис. 2.1):

1. Реєстрація або авторизація.
2. Перегляд головної сторінки.
3. Перегляд інформації про заклад.
4. Перегляд списку новин.
5. Перегляд певної новини.
6. Перегляд фотогалереї.
7. Перегляд фото з певного заходу.
8. Перегляд відеогалереї.
9. Перегляд певного відео.
10. Перегляд документації.
11. Перегляд контактів.
12. Зворотній зв'язок.

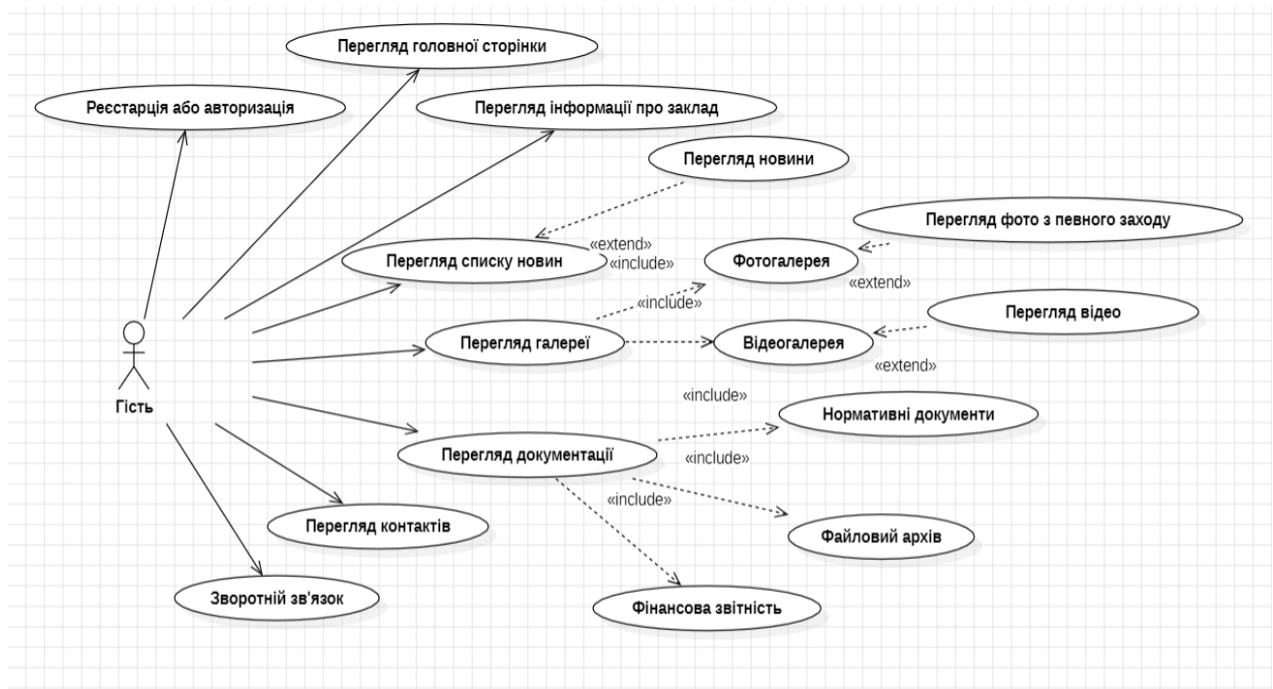


Рис. 2.1 - Діаграма використання гостя

Авторизований користувач має доступ до наступних функцій(рис. 2.2):

1. До можливості зайти в профіль користувача.
2. До редагування профілю.
3. До зміни паролю.
4. До оновлення профілю.
5. До можливості лишати коментарі.
6. До виходу з облікового запису.

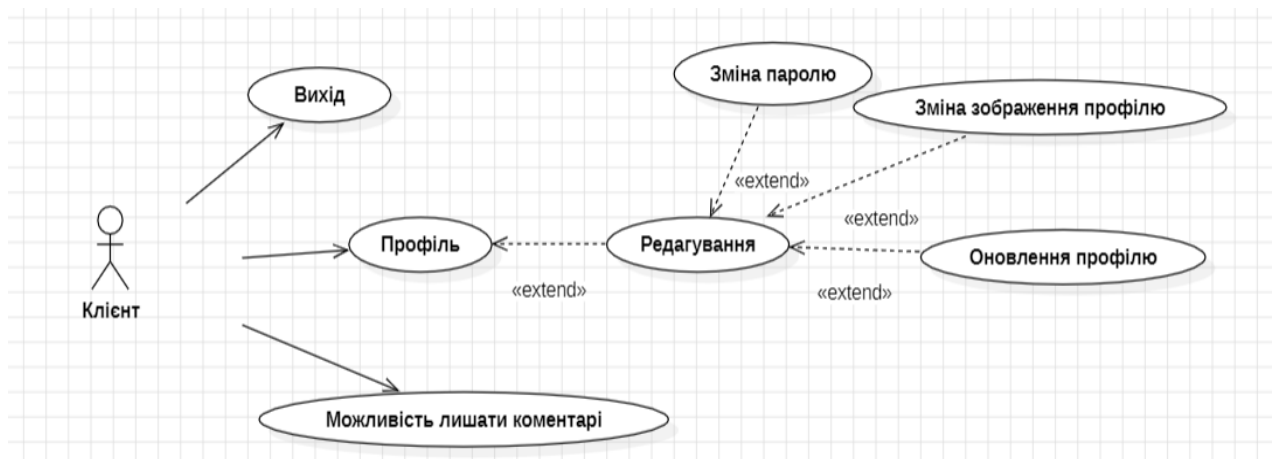


Рис. 2.2 - Діаграма використання авторизованого користувача

Адміністратор має доступ до наступних функцій(рис. 2.3):

- ## 1. До CRUD новин

2. До CRUD фотогалереї
3. До CRUD відеогалереї
4. До CRUD документації

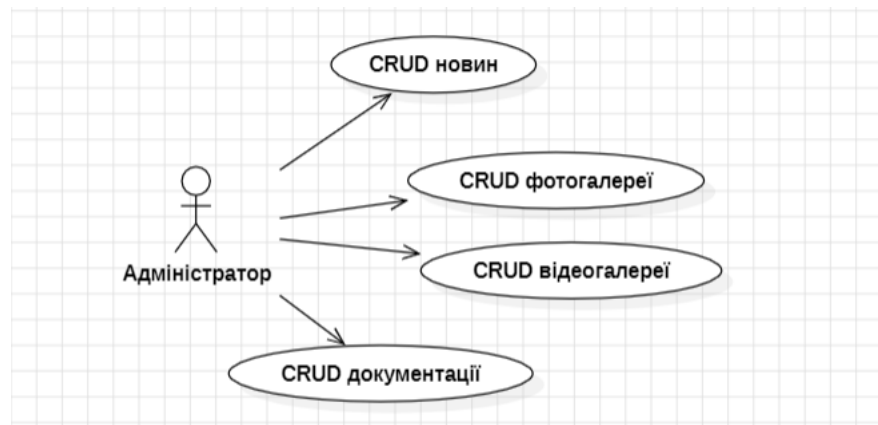


Рис. 2.3 - Діаграма використання адміністратора

Функціональні вимоги визначають конкретні функції та можливості, які має виконувати веб-сайт, серед них:

1. Реєстрація та авторизація.
- Налаштування профілю користувача.

3. Форма зворотного зв'язку.
4. Відображення новин.
5. Фотогалерея
6. Відеогалерея.
7. Документація.
8. Коментарі.
9. Адміністративна панель для CRUD операцій.

Нефункціональні вимоги до веб-сайту визначають параметри та характеристики, які не стосуються конкретних функцій продукту, але впливають на його загальну ефективність, надійність, безпеку, ефективність використання та інші аспекти.

Нефункціональні вимоги до сайту, що розробляється:

1. Продуктивність:
 - Час завантаження сторінок: Максимальний час завантаження сторінок повинен відповідати не більше 3 секунд.

- Продуктивність сервера: Сайт повинен витримувати певне навантаження користувачів, аби забезпечити швидке обслуговування запитів.

2. Безпека:

- Захист від атак: Веб-сайт повинен мати захист від різноманітних атак, таких як SQL-ін'єкції, перехоплення даних тощо.
- Шифрування: З'єднання повинно бути зашифрованим (SSL/TLS) для забезпечення конфіденційності даних.

3. Надійність:

- Доступність: Сайт повинен бути доступний протягом 99,9% відсотка часу.
- Резервне копіювання та відновлення: Забезпечення можливості регулярного резервного копіювання даних і можливості швидкого відновлення в разі втрати або пошкодження.

4. Сумісність:

- Браузерна сумісність: Сайт повинен коректно відображатися в різних веб-браузерах: Chrome, Firefox, Safari, Edge.

5. Ефективність використання:

- Мобільна сумісність: Сайт повинен бути оптимізований для перегляду на різних мобільних пристроях.

6. Управління інтерфейсом користувача:

- Інтуїтивний інтерфейс: Сайт повинен мати простий та зрозумілий інтерфейс користувача.

7. Скальованість:

- Масштабованість: Сайт повинен бути здатний ефективно працювати при збільшенні обсягу даних або навантаження користувачів.

Діаграма класів (рис. 2.4) описує структуру системи показуючи класи, їх атрибути, операції (або методи) і зв'язки між об'єктами:

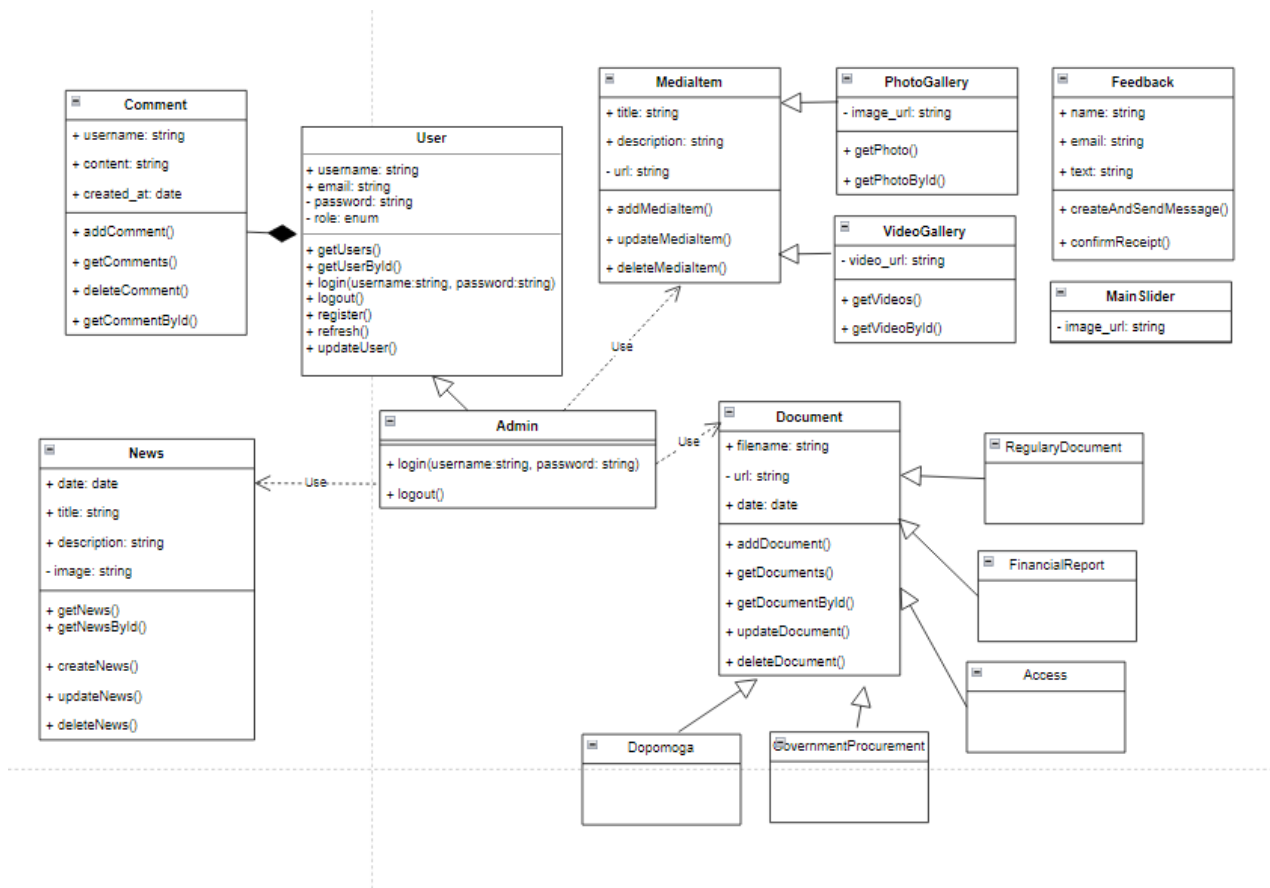


Рис. 2.4 - Діаграма класів

На діаграмі зображено наступні класи:

User – клас, який містить методи для роботи з користувачем.

Атрибути:

- username(тип string) – ім'я користувача;
- email(тип string) – електронна пошта користувача;
- password(тип string) – пароль до облікового запису користувача;
- role(тип enum) – роль користувача.

Методи:

- getUsers() – отримання списку користувачів з бази даних;
- getUserById() – отримання користувача за ідентифікатором;
- login(username(тип string), password(тип string)) – авторизація користувача;
- logout() – вихід з облікового запису;
- register() – реєстрація користувача;
- refresh() – оновлення унікального токена.

Admin – клас, який містить методи для роботи з адміністратором.

Методи:

- login(username(тип string), password(тип string)) – вхід адміністратора до адмінпанелі;
- logout() – вихід адміністратора з адмінпанелі.

News – клас, який містить методи для роботи з новинами закладу.

Атрибути:

- date(тип date) – дата створення новини;
- title(тип string) – заголовок новини;
- description(тип string) – опис новини;
- image(тип string) – зображення до новини.

Методи:

- getNews() – отримання списку всіх новин з бази даних;
- getNewsById() – отримання новини за ідентифікатором;
- createNews() – створити новину;
- updateNews() – оновити новину;
- deleteNews() – видалити новину.

Document – клас, який містить методи для роботи з документацією.

Атрибути:

- filename(тип string) – назва файлу;
- url(тип string) – посилання на файл;
- date(тип date) – дата додавання файлу.

Методи:

- getDocumentById() – отримати документ за ідентифікатором;
- getDocuments() – отримати список всіх документів з бази даних;
- createDocument() – додати документ;
- updateDocument() – оновити документ;
- deleteDocument() – видалити документ.

MediaItem – клас, який містить методи для роботи з мультимедійним контентом.

Атрибути:

- title(тип string) – заголовок вмісту;
- description(тип string) – опис вмісту;
- url(тип string) – посилання на вміст.

Методи:

- addItem() – створити вміст;
- updateItem() – оновити вміст;
- deleteItem() – видалити вміст;

PhotoGallery – клас, який містить методи для роботи з фотографіями.

Атрибути:

- image_url(тип string) – посилання на зображення.

Методи:

- getPhotos() – отримати список всіх зображень з бази даних;
- getPhotoById() – отримати зображення за ідентифікатором.

VideoGallery – клас, який містить методи для роботи з відео.

Атрибути:

- video_url(тип string) – посилання на відео.

Методи:

- getPhotos() – отримати список всіх відео з бази даних;
- getPhotoById() – отримати відео за ідентифікатором.

Feedback – клас, що містить методи для роботи з формою зворотного зв'язку.

Атрибути:

- name(тип string) – ім'я користувача;
- email(тип string) – електронна адреса користувача;
- text(тип string) – повідомлення користувача.

Методи:

- createAndSendMessage() – метод, для створення і відправки повідомлення;
- confirmReceipt() – підтвердження отримання.

Comment – клас, що містить методи для роботи з коментарями.

Атрибути:

- username(string) – ім'я користувача;
- content(string) – текст коментаря;
- created_at(date) – дата відправки коментаря.

Методи:

- addComment() – метод для додавання коментаря до бази даних;
- getComments() – отримати список всіх коментарів з бази даних.

MainSlider – зберігає зображення для слайдеру на головній сторінці.

2.2 Розробка бази даних

Для збереження даних на сайті використовується база даних PostgreSQL. Попередньо була розроблена схема бази даних зі зв'язками (рис. 2.5).

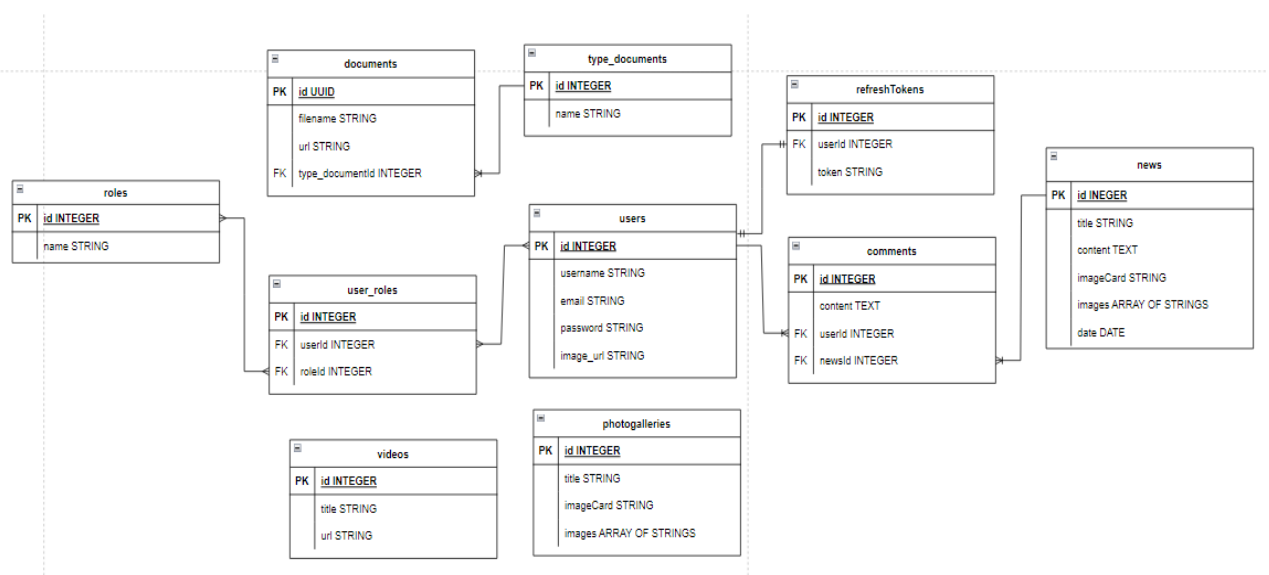


Рис. 2.5 – Схема бази даних

В результаті аналізу було розроблено 10 таблиць:

1. users – Користувачі системи (табл. 2.1).
2. roles - Ролі, які можуть бути призначені користувачам (табл. 2.2).
3. user_roles - Проміжна таблиця для зв'язку багато-до-багатьох між користувачами та ролями (табл. 2.3).
4. refreshToken - Токени для оновлення сесії користувача (табл. 2.4).
5. news - Таблиця для зберігання новин (табл. 2.5).

6. comments - Таблиця для зберігання коментарів (табл. 2.6).
7. documents - Таблиця для зберігання документів (табл. 2.7).
8. type_documents - Таблиця для зберігання типів документів (табл. 2.8).
9. photogalleries - Таблиця для зберігання фото (табл. 2.9).
10. videos - Таблиця для зберігання відео (табл. 2.10).

Таблиця 2.1

Опис таблиці users

users		
Поле	Тип	Призначення
id	integer	Ідентифікатор
username	string	Ім'я користувача
email	string	Ел. пошта
password	string	Пароль користувача

Таблиця 2.2

Опис таблиці roles

roles		
Поле	Тип	Призначення
id	integer	Ідентифікатор
name	string	Назва ролі

Таблиця 2.3

Опис таблиці user_roles

user_roles		
Поле	Тип	Призначення
userId	integer	Зовнішній ключ до таблиці користувачів
roleId	integer	Зовнішній ключ до таблиці ролей

Таблиця 2.4

Опис таблиці refreshTokens

refreshTokens		
Поле	Тип	Призначення
id	integer	Ідентифікатор
userId	integer	Зовнішній ключ до таблиці користувачів
token	string	Токен оновлення

Таблиця 2.5

Опис таблиці news

news		
Поле	Тип	Призначення
id	integer	Ідентифікатор
title	string	Заголовок новини
content	text	Вміст новини
imageCard	string	Шлях до основного зображення новини
images	array of strings	Масив зі шляхів до зображень
date	date	Дата створення новини

Таблиця 2.6

Опис таблиці comments

comments		
Поле	Тип	Призначення
id	integer	Ідентифікатор
content	text	Вміст коментаря
userId	integer	Зовнішній ключ до таблиці користувачів
newsId	integer	Зовнішній ключ до таблиці новин

Таблиця 2.7

Опис таблиці documents

documents		
Поле	Тип	Призначення
id	uuid	Ідентифікатор
filename	string	Назва файлу
url	string	URL-адреса документу
type_documentId	integer	Зовнішній ключ до таблиці типів документів

Таблиця 2.8

Опис таблиці type_documents

type_documents		
Поле	Тип	Призначення
id	integer	Ідентифікатор
name	string	Назва типу документу

Таблиця 2.9

Опис таблиці photogalleries

photogalleries		
Поле	Тип	Призначення
id	integer	Ідентифікатор
title	string	Заголовок фотогалереї
imageCard	string	Шлях до основного зображення фотогалереї
images	array of strings	Масив URL-адрес зображень у фотогалереї

Опис таблиці videos

videos		
Поле	Тип	Призначення
id	integer	Ідентифікатор
title	string	Заголовок для відео
videos	array of strings	Масив URL-адрес відео

Зв'язки між таблицями:

Користувачі та ролі. Таблиці users і roles мають багато-до-багатьох зв'язок через проміжну таблицю user_roles. Це означає, що один користувач може мати кілька ролей, а одна роль може бути призначена кільком користувачам.

Користувачі та токени оновлення. Таблиці users і refreshTokens мають один-до-одного зв'язок, де кожен користувач має один токен оновлення.

Користувачі та коментарі. Таблиці users і comments мають один-до-багатьох зв'язок, де один користувач може залишити багато коментарів, але кожен коментар належить одному користувачу.

Новини та коментарі. Таблиці news і comments мають один-до-багатьох зв'язок, де одна новина може мати багато коментарів, але кожен коментар належить одній новині.

Документи та типи документів. Таблиці documents і type_documents мають один-до-багатьох зв'язок, де один тип документу може мати багато документів, але кожен документ належить одному типу документу.

Розроблена структура бази даних дозволяє ефективно керувати даними користувачів, ролей, новин, коментарів, документів, фотогалереї та відеогалереї забезпечуючи при цьому гнучкість і розширюваність для майбутніх оновлень та доповнень системи.

2.3 Проектування та реалізація алгоритмів роботи системи

Одним з завдань у реалізації була розробка алгоритму роботи з документами на сайті. Документація школи міститься у таблиці documents. Щоб документи були доступними на потрібних сторінках, вони були розподілені за типами, назви яких зберігаються у таблиці type_documents. Тип документу відповідає сторінці, на якій повинен відображатись файл.

Фрагмент коду (рис. 2.6) завантажує документи з сервера при зміні typeDocumentId та зберігає їх у стан компоненту, обробляючи можливі помилки під час завантаження.

```
useEffect(() => {  
  const fetchDocuments = async () => {  
    try {  
      const response = await axios.get('http://localhost:8080/api/documents/type/${typeDocumentId}');  
      setAttachments(response.data);  
    } catch (error) {  
      console.error('Error fetching documents:', error);  
    }  
  };  
  fetchDocuments();  
}, [typeDocumentId]);
```

Рис. 2.6 - Завантаження документів з сервера

Тут використано React Hook useEffect для завантаження документів з API, коли змінюється значення typeDocumentId.

useEffect - це хук в React, який дозволяє виконувати побічні ефекти (наприклад, завантаження даних, підписка на потоки даних) у функціональних компонентах. Цей хук спрацьовує після кожного рендеру компонента [15].

Спочатку створюється асинхронна функція fetchDocuments, яка буде відповідальна за отримання документів з сервера. Використовуючи бібліотеку axios, функція надсилає GET-запит до сервера на адресу. Після успішного отримання відповіді від сервера, дані з відповіді зберігаються у стан компоненту за допомогою функції setAttachments. У разі виникнення помилки під час виконання запиту, вона обробляється у блоці catch і виводиться в консоль. Функція fetchDocuments викликається для завантаження документів одразу після визначення. useEffect приймає масив залежностей, в якому вказано

typeDocumentId. Це означає, що функція fetchDocuments буде викликана кожного разу, коли значення typeDocumentId змінюється.

Функція handleDownload (рис. 2.7) призначена для автоматичного завантаження файлу з заданої URL-адреси.

```
const handleDownload = ( url, filename ) => {  
  const link = document.createElement( 'a' );  
  link.href = url;  
  document.body.appendChild( link );  
  link.click();  
  document.body.removeChild( link );  
};
```

Рис. 2.7 - Автоматичне завантаження файлу

Функція створює новий HTML елемент <a> (посилання) за допомогою методу document.createElement. Властивість href створеного посилання встановлюється на значення параметра url. Це означає, що посилання вказуватиме на ресурс, який потрібно завантажити. Посилання додається до тіла документа (<body>). Це необхідно для того, щоб посилання стало частиною документа і на нього можна було клікнути. Викликається метод click на створеному посиланні, що програмно ініціює клік по цьому посиланню. Це запускає завантаження файлу з вказаної URL-адреси. Після того, як клік по посиланню відбувся і завантаження файлу розпочалось, посилання видаляється з документа. Це робиться для того, щоб не залишати в DOM непотрібні елементи, що можуть засмічувати документ. Отже, функція handleDownload дозволяє автоматично завантажити файл з заданої URL-адреси шляхом створення тимчасового посилання, клікання по ньому і видалення цього посилання після початку завантаження.

Фрагмент коду (рис. 2.8) створює список документів, де кожний документ відображається з відповідною іконкою та посиланням для завантаження. Класи використовуються для стилізації елементів списку, а функції getFileExtension, getIconForExtension і handleDownload забезпечують відповідну функціональність для відображення і завантаження файлів.

```

<ul className={styles.attachmentList}>
  {attachments.map((attachment, index) => (
    <li className={styles.attachmentItem} key={index}>
      <img
        src={getIconForExtension(getFileExtension(attachment.filename))}
        alt={`${getFileExtension(attachment.filename)} icon`}
        className={styles.fileIcon}
      />
      <a className={styles.attachmentLink} href={attachment.url}
        onClick={() => handleDownload(attachment.url, attachment.filename)}>
        {attachment.filename}
      </a>
    </li>
  ))}
</ul>

```

Рис. 2.8 - Відображення списку документів

Вищенаведений фрагмент коду рендерить список вкладень, використовуючи дані, отримані та збережені в стані `attachments`. Відкривається елемент списку `<ul>` з класом `attachmentList`. Клас використовується для стилізації списку. Використовується метод `map` для перебору масиву `attachments`. Для кожного документу створюється елемент списку `<li>` з класом `attachmentItem`. Атрибут `key` встановлюється на значення `index`, що дозволяє React оптимально оновлювати список. Для кожного документу відображається іконка, що відповідає типу файлу. Це досягається за допомогою функцій `getFileExtension` (отримує розширення файлу з його імені) і `getIconForExtension` (повертає URL іконки відповідно до розширення). Кожен документ відображається як посилання з класом `attachmentLink`. Посилання має атрибут `href`, встановлений на URL файлу. При кліку на посилання викликається функція `handleDownload`, яка автоматично завантажує файл з вказаною URL-адресою і ім'ям файлу. Кожен елемент списку `<li>` закривається, і після перебору всіх вкладень закривається список `<ul>`.

Опишемо алгоритм управління новинним контентом на сайті.

Коли користувач відправляє форму з даними новини, дані надсилаються на сервер у форматі `multipart/form-data`. Функція `handleSubmit` викликається при поданні форми новини (рис. 2.9). Вона перевіряє, чи заповнені основні поля, такі як заголовок і контент, і створює об'єкт `FormData`, який містить ці дані. Якщо вибране головне зображення, воно також додається до `FormData`. Потім всі додаткові зображення додаються в `FormData` у циклі. Після цього

відправляється POST-запит на сервер за допомогою Axios, який включає в себе дані форми. Якщо запит виконується успішно, встановлюються повідомлення для модального вікна, дані форми очищаються, і модальне вікно відображається. У разі помилки виводиться повідомлення про помилку.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append('title', title);
  formData.append('content', content);
  if (imageCard) {
    formData.append('imageCard', imageCard);
  }
  for (let i = 0; i < images.length; i++) {
    formData.append('images', images[i]);
  }
  try {
    await axios.post('http://localhost:8080/api/news', formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
        'x-access-token': auth.token,
      },
    });
    setModalMessage('Новину додано!');
    setShowModal(true);
    setTitle('');
    setContent('');
    setImageCard(null);
    setImages([]);
  } catch (err) {
    console.error("Помилка:", err.response || err);
    setModalMessage('Не вдалося додати новину. Спробуйте ще раз.');
```

Рис. 2.9 - Обробка форми для створення новини

Функція `exports.createNews` використовується для створення новини у базі даних на основі даних, що надсилаються через HTTP-запит (рис. 2.10).

```
exports.createNews = (req, res) => {
  const imageCardPath = req.files['imageCard'] ? req.files['imageCard'][0].path : null;
  const imagesPaths = req.files['images'] ? req.files['images'].map(file => file.path) : [];

  News.create({
    title: req.body.title,
    content: req.body.content,
    imageCard: imageCardPath,
    images: JSON.stringify(imagesPaths)
  })
  .then(news => {
    res.status(201).send(news);
  })
  .catch(err => {
    res.status(500).send({ message: err.message });
  });
};
```

Рис. 2.10 - Створення новини та збереження даних про неї у базі даних

Отримання шляхів до зображень. Функція перевіряє, чи надійшли файли з головним зображенням та іншими зображеннями через req.files. Якщо файл із головним зображенням існує, вона зберігає його шлях у змінну imageCardPath. Аналогічно, якщо надійшли файли із додатковими зображеннями, вона зберігає їх шляхи у масив imagesPaths.

Створення запису новини. Функція використовує модель News для створення нового запису в базі даних. Вона використовує дані, які надійшли через req.body, такі як заголовок та вміст новини, а також шляхи до головного зображення та додаткових зображень, які зберігалися в попередніх кроках.

Відправлення відповіді. Якщо створення новини в базі даних успішно, вона відправляє HTTP-відповідь з кодом статусу 201 (створено) та створеним записом новини. У разі виникнення помилки при створенні новини, вона відправляє HTTP-відповідь з кодом статусу 500 (внутрішня помилка сервера) та повідомленням про помилку.

Для забезпечення алгоритму пошуку за заголовком новини було використано код (рис. 2.11).

```
const filteredNews = news.filter(n =>
  n.title.toLowerCase().includes(searchTerm.toLowerCase()));
```

Рис. 2.11 - Реалізація пошуку новини за її заголовком

Цей рядок коду фільтрує масив новин news, застосовуючи фільтрацію за заголовком новини:

news.filter(...). Це метод масиву news, який створює новий масив, що містить елементи, які пройшли перевірку у функції зворотнього виклику.

(n => n.title.toLowerCase().includes(searchTerm.toLowerCase())). Це функція зворотнього виклику, яка викликається для кожного елемента n масиву news. У цьому випадку вона перевіряє, чи входить заголовок кожної новини n.title в рядок searchTerm, який вводить користувач.

n.title.toLowerCase(). Перетворює заголовок новини n.title у нижній регістр, щоб зробити порівняння нечутливим до регістру.

.includes(searchTerm.toLowerCase()). Перевіряє, чи містить перетворений у нижній регістр заголовок новини рядок searchTerm, який також перетворений

у нижній регістр. Якщо searchTerm зустрічається у заголовку новини (навіть частково), тоді ця умова буде істинною, і новина буде включена у фінальний масив filteredNews.

Отже, після виконання цього рядка коду filteredNews буде містити лише ті елементи news, чиї заголовки містять частину або весь рядок searchTerm.

Функція exports.deleteNews (рис. 2.12) призначена для видалення конкретної новини з бази даних, а також відповідних зображень, які пов'язані з цією новиною.

Порядок роботи:

Отримання ідентифікатора новини. Функція спочатку отримує ідентифікатор новини з параметрів запиту (req.params.id), який вказує на те, яку саме новину потрібно видалити.

Пошук новини у базі даних за ідентифікатором. Функція використовує метод findByPk моделі News, щоб знайти новину за її ідентифікатором. Якщо новина не знайдена, відправляється відповідь з кодом статусу 404 та відповідним повідомленням про помилку.

```
exports.deleteNews = (req, res) => {
  const id = req.params.id;
  News.findByPk(id)
    .then(news => {
      if (!news) {
        return res.status(404).send({ message: `Не можна видалити новину ${id}. Новину не знайдено.` });
      }
      const imageCardPath = news.imageCard;
      const imagesPaths = JSON.parse(news.images);
      if (imageCardPath) {
        fs.unlinkSync(imageCardPath);
      }
      imagesPaths.forEach(imagePath => {
        if (imagePath) {
          fs.unlinkSync(imagePath);
        }
      });
      News.destroy({ where: { id: id } })
        .then(num => {
          if (num == 1) {
            res.status(200).send({ message: "Новину видалено!" });
          } else {
            res.status(404).send({ message: `Не можна видалити новину ${id}.` });
          }
        })
        .catch(err => {
          res.status(500).send({ message: err.message });
        });
    })
    .catch(err => {
      res.status(500).send({ message: err.message });
    });
};
```

Рис. 2.12 - Видалення новини з відповідними файлами

Видалення зображень новини з файлової системи. Якщо новина знайдена, функція отримує шляхи до головного зображення та додаткових зображень новини. Вона використовує `fs.unlinkSync` для синхронного видалення кожного зображення з файлової системи.

Видалення новини з бази даних. Після видалення зображень з файлової системи, функція використовує метод `destroy` моделі `News`, щоб видалити саму новину з бази даних. Вона використовує `where: { id: id }` для вказівки, яку саме новину потрібно видалити. Після успішного видалення відправляється відповідь з кодом статусу 200 та відповідним повідомленням. Якщо новина не була знайдена для видалення, відправляється відповідь з кодом статусу 404 та повідомленням про помилку. У випадку будь-якої іншої помилки, пов'язаної з видаленням з бази даних, відправляється відповідь з кодом статусу 500 та відповідним повідомленням про помилку.

Налаштування збереження файлів. Об'єкт `storage` містить налаштування для збереження завантажених файлів (рис. 2.13). В даному випадку використовується `diskStorage`, що зберігає файли на диск. Властивість `destination` вказує шлях для збереження файлів. Властивість `filename` генерує нові імена для файлів, які мають бути унікальними. Вона також перекодує оригінальні імена файлів з латиниці в UTF-8, щоб уникнути проблем з кодуванням.

```
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const folderPath = path.join('public/images');
    fs.mkdirSync(folderPath, { recursive: true });
    cb(null, folderPath);
  },
  filename: (req, file, cb) => {
    file.originalname = Buffer.from(file.originalname, 'latin1').toString('utf8')
    cb(null, `${Date.now()}-${file.originalname}`);
  }
});
```

Рис. 2.13 - Налаштування збереження файлів

Створення об'єкту `upload`. Об'єкт `upload` містить налаштування для обробки завантаження файлів (рис. 2.14). Параметр `preservePath` зберігає шлях до файлу при збереженні його на сервері. Параметр `storage` вказує на об'єкт з налаштуваннями збереження файлів.

Метод `fileFilter` встановлює фільтр для прийняття або відхилення файлів на основі їхнього розширення та типу MIME. Цей фільтр дозволяє тільки зображення з розширеннями `.jpeg`, `.png`, `.jpg` або `.gif`.

```
const upload = multer({
  preservePath: true,
  storage: storage,

  fileFilter: (req, file, cb) => {
    console.log(file)
    const ext = path.extname(file.originalname).toLowerCase();
    if (ext !== '.jpeg' && ext !== '.png' && ext !== '.jpg' && ext !== '.gif') {
      return cb(new Error('Дозволено завантажувати тільки зображення'));
    }

    const allowedMimeTypes = [
      'image/jpeg',
      'image/png',
      'image/gif'
    ];

    if (!allowedMimeTypes.includes(file.mimetype)) {
      return cb(new Error('Дозволено завантажувати тільки зображення'));
    }
    cb(null, true);
  }
});
```

Рис. 2.14 - Реалізація обробки завантаження файлів

Функція `handleSubmitComment` (рис. 2.15) асинхронна і призначена для обробки події надсилання коментаря.

```
const handleSubmitComment = async (e) => {
  e.preventDefault();
  if (!quill) return;
  const newComment = {
    newsId: id,
    content: quill.root.innerHTML,
    userId: auth.user.id,
  };
  try {
    const response = await axios.post(`http://localhost:8080/api/comments/news`, newComment, {
      headers: {
        'x-access-token': auth.token,
      },
    });
    onCommentAdded(response.data);
    quill.setText('');
    setError(null);
  } catch (error) {
    console.error('There was an error submitting the comment!', error);
    setError('Помилка відправки коментаря. Будь ласка, спробуйте пізніше.');
```

Рис. 2.15 - Обробка події надсилання коментаря

Цей код забезпечує функціональність надсилання коментарів до новин на сервер і обробку можливих помилок під час цього процесу.

Висновки до другого розділу

У першому підрозділі визначено функціональні та нефункціональні вимоги від користувачів різних типів, сформовано діаграми варіантів використання для різних ролей, розроблено та описано діаграму класів.

У другому підрозділі детально описана структура збереження даних, встановлені зв'язки між таблицями.

У третьому підрозділі здійснено опис узагальнених алгоритмів роботи системи та опис взаємодії елементів системи під час роботи програмного комплексу.

РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розгортання веб-сайту

Розгортання програмного забезпечення – це процес встановлення і налаштування програмного забезпечення на комп'ютері або в мережі комп'ютерів, щоб зробити його готовим до використання [16].

На етапі розгортання веб-додаток запускається і стає загальнодоступним. Веб-додаток повинен бути розгорнутий на сервері, який є надійним і масштабованим, щоб впоратися зі зростаючою кількістю користувачів. Процес розгортання повинен бути плавним і ефективним, з мінімальним часом простою або перебоями в роботі кінцевих користувачів [17].

Розгортання веб-сайту на стеку PERN з ORM Sequelize на VPS включає налаштування і взаємодію між кількома компонентами: фронтом на React.js, бекендом на Node.js з Express.js, та базою даних PostgreSQL, керованою за допомогою Sequelize (рис. 3.1).

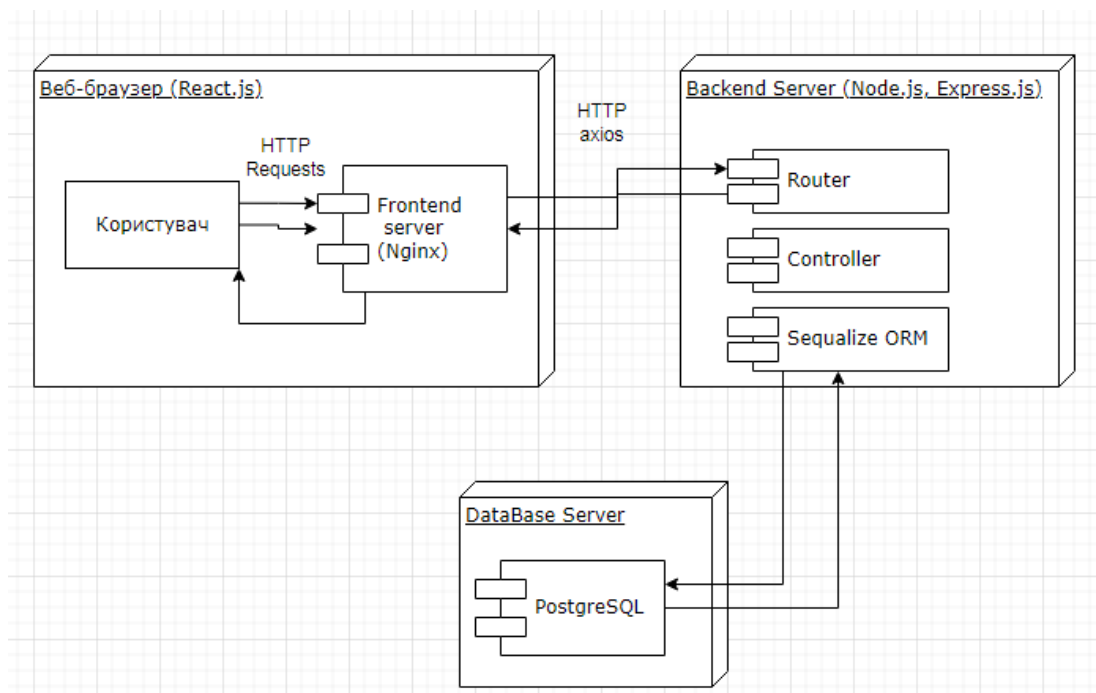


Рис. 3.1 - Діаграма розгортання веб-сайту

Компоненти системи.

Веб-браузер. Інтерфейс для користувача, через який він взаємодіє з веб-додатком. Веб-браузер завантажує та відображає фронтенд-додаток, розроблений на React.js.

Frontend Server. Сервер, що обслуговує статичні файли React.js додатку. Використовується веб-сервер Nginx, який також може виступати в ролі зворотного проксі до бекенд-сервера.

Backend Server. Сервер, який обслуговує Node.js додаток з використанням фреймворку Express.js. Цей сервер обробляє запити від фронтенд-сервера та взаємодіє з базою даних.

Database Server. Сервер баз даних, який використовує PostgreSQL для зберігання даних додатку. Sequelize ORM забезпечує зручний інтерфейс для взаємодії з базою даних через об'єктно-реляційне відображення.

Процес розгортання.

В середовищі розробки React.js додаток збирається за допомогою команди `npm run build`. Цей процес створює оптимізовані статичні файли для розгортання. Статичні файли завантажуються на сервер, де налаштовується Nginx для обслуговування цих файлів. Nginx налаштовано так (рис. 3.2):

```
server {  
    listen 80 default_server;  
    listen [::]:80 default_server;  
    root /var/www/dist;  
  
    server_name kmytiv-internat.com www.kmytiv-internat.com;  
    location / {  
        try_files $uri $uri/ /index.html;  
    }  
}
```

Рис. 3.2 - Налаштування Nginx

На сервері встановлюються необхідні залежності за допомогою `npm install`. Налаштовується файл `.env` для збереження конфіденційної інформації, такої як налаштування бази даних та секретні ключі. Бекенд-сервер запускається за допомогою команди `npm start`.

Виконуючи послідовні кроки розгортання, було налаштовано продуктивний і масштабований веб-додаток, що забезпечує швидку та надійну роботу для кінцевих користувачів.

3.2 Структура інтерфейсу веб-сайту

Головна сторінка сайту відображає навігаційне меню в хедері, привітальне повідомлення та слайдер на головному екрані (рис. 3.3).



Рис. 3.3 - Головна сторінка сайту

Нижче відображено стислу інформацію про заклад, щоб мати загальне уявлення про школу, а також посилання на новини та календар (рис. 3.4).

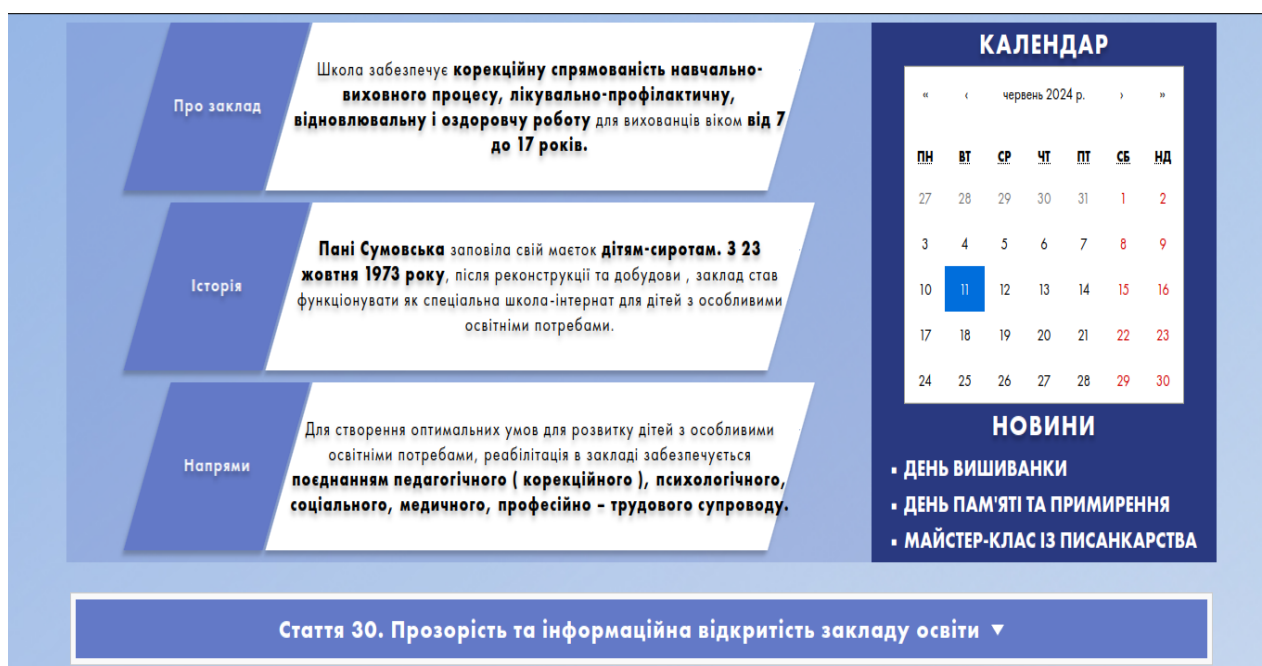


Рис. 3.4 - Другий екран головної сторінки

Внизу сайту міститься футер (рис. 3.5), що має всі необхідні дані для зв'язку з закладом освіти, а також головні посилання для швидкої навігації.



Рис. 3.5 - Футер сайту

За посиланням «Контакти» можна отримати інформацію для звернення до закладу (рис. 3.6).

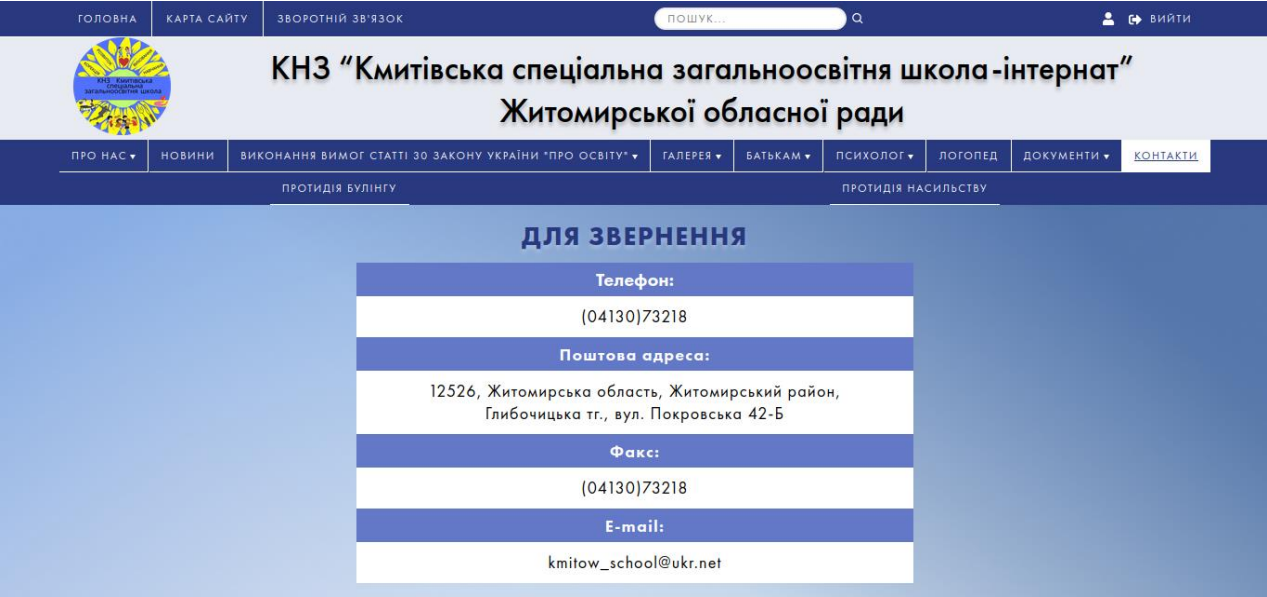


Рис. 3.6 - Контактні дані

Сторінка «Карта сайту» (рис. 3.7) містить всі маршрути додатку, за якими можна перейти та отримати необхідну інформацію. Це зроблено для того, що користувач мав змогу побачити загальну структуру сайту і швидко знайти потрібні сторінки, оскільки пункти навігаційного меню містять в собі підпункти, які одразу не видно користувачеві.



Рис. 3.7 - Карта сайту

Щоб користувач міг залишити повідомлення і отримати відповідь на нього, розроблено форму для зворотного зв'язку (рис. 3.8), використовуючи Email.js. Це сервіс, що допомагає підключити службу електронної пошти, створити шаблон повідомлень та надіслати електронний лист зі свого коду.

Рис. 3.8 - Форма зворотного зв'язку

Після успішного заповнення всіх полів, система залишає повідомлення про те, що згодом буде відправлена відповідь на електронну пошту, яку користувач залишив у полі E-mail (рис. 3.9).

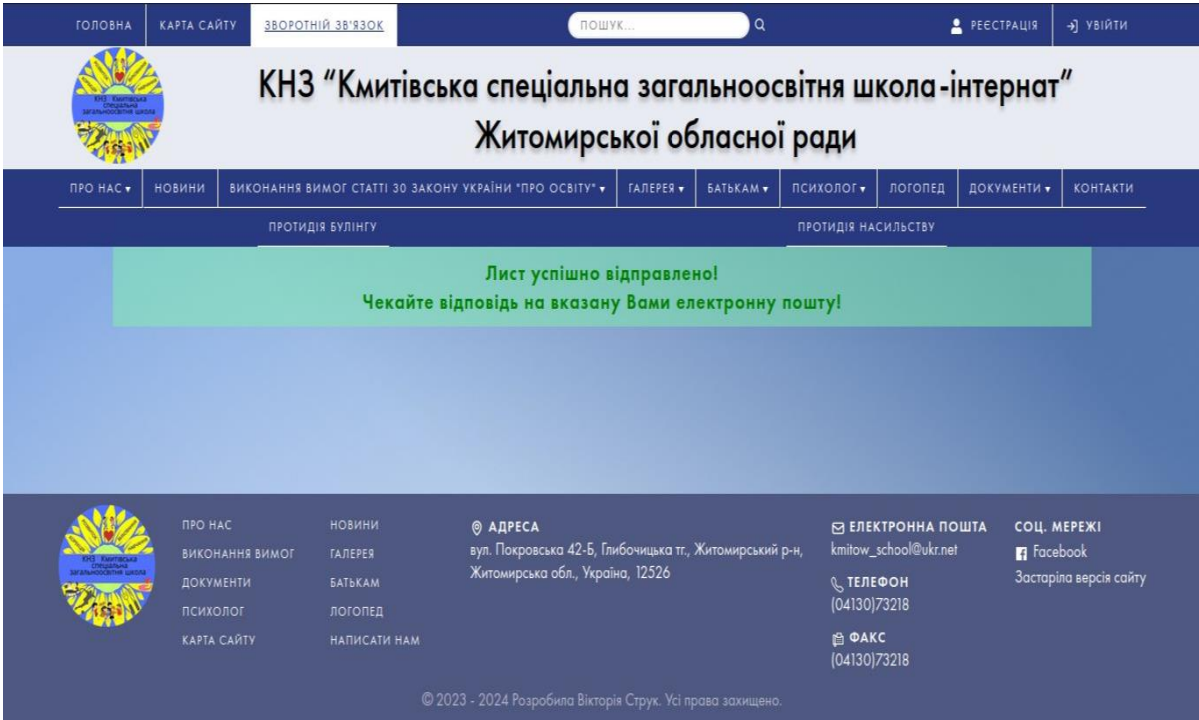


Рис. 3.9 - Повідомлення після відправки повідомлення

На сайті є можливість реєстрації (рис. 3.10) з валідованими полями (рис. 3.11) та авторизації користувача (рис. 3.12). Коли користувач введене некоректну інформацію або легкий пароль, то отримає відповідне повідомлення про помилку (рис. 3.13). Після успішного введення всіх даних при реєстрації, користувачу запропонують авторизуватись у системі.

Рис. 3.10 - Форма реєстрації

Рис. 3.11 - Валідація форми реєстрації

Рис. 3.12 - Форма авторизації

Рис. 3.13 - Перевірка форми авторизації на невірне значення вводу

Зареєстрований користувач може перейти на сторінку свого профілю та змінити там свої дані за бажанням (рис. 3.14).

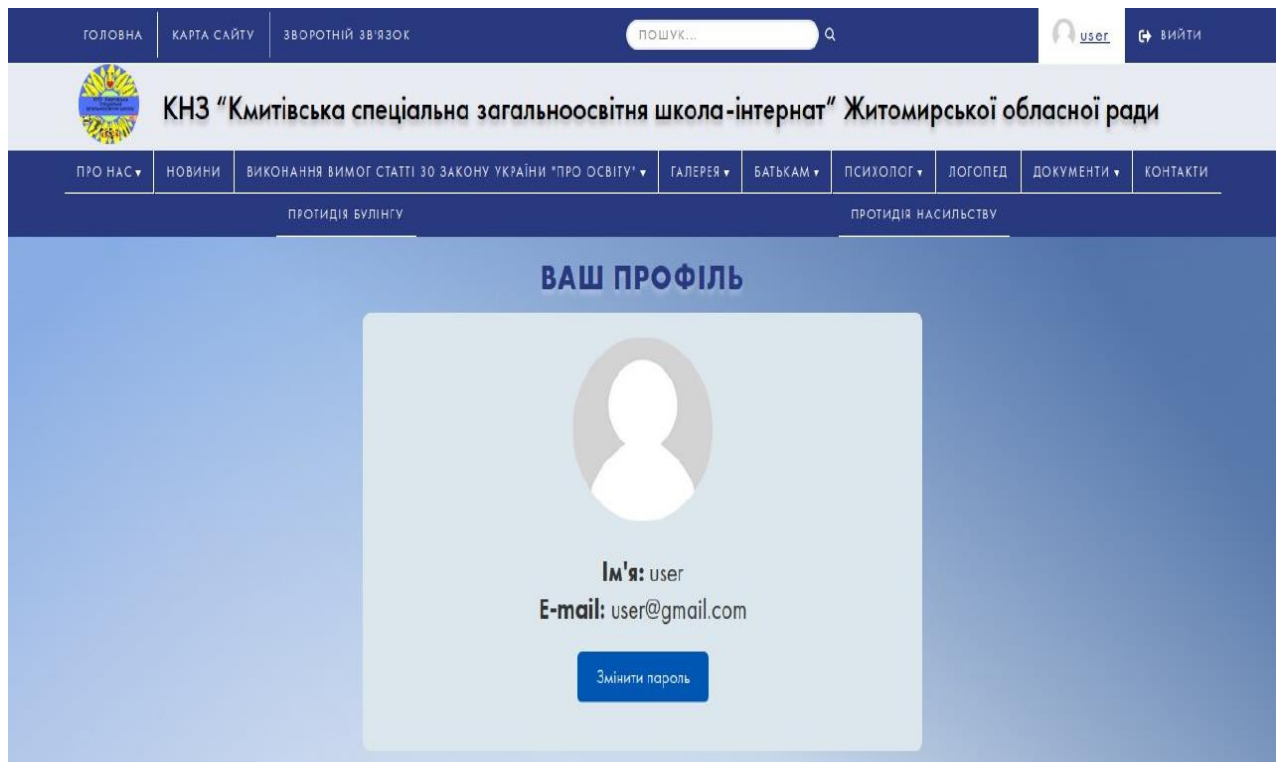


Рис. 3.14 – Профіль користувача

На сторінці «Допомога школі» документація сортується і виводиться за датою створення (рис. 3.15).

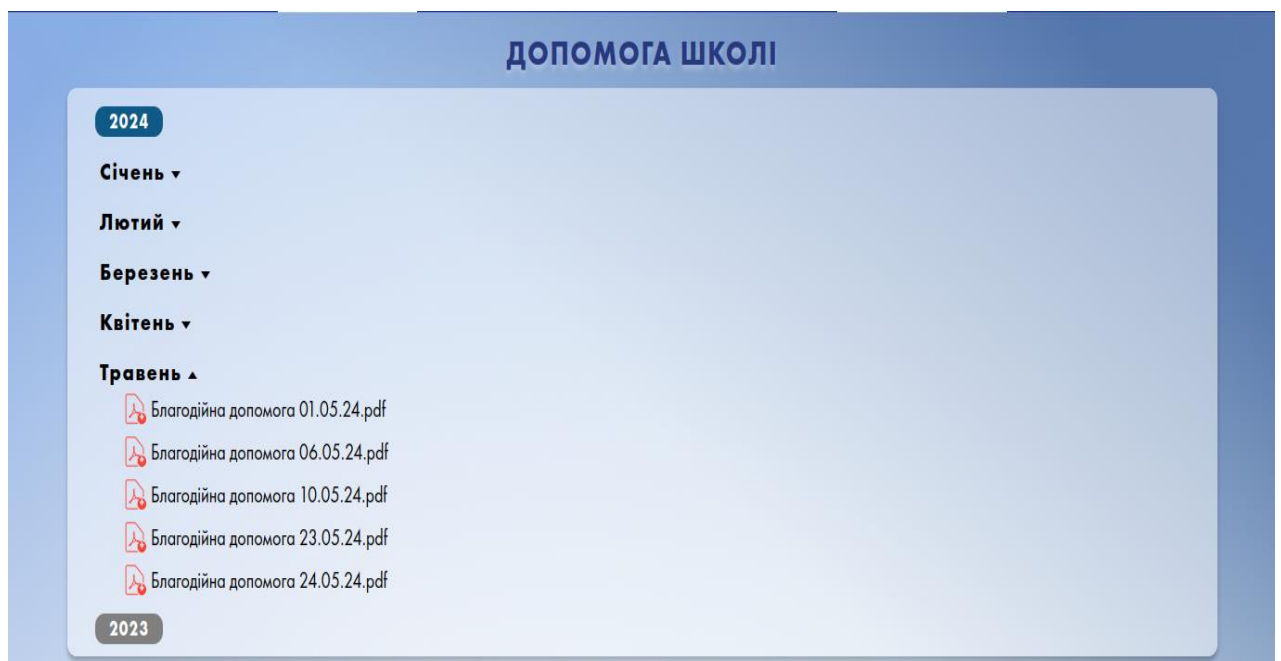


Рис. 3.15 - Розподілення документації

На інших сторінках документація розподіляється за датою додавання на сайт (рис. 3.16).



Рис. 3.16 - Розподілення документації

На сторінці «Фотогалерея» (рис. 3.17) є можливість побачити список карток з назвами заходу, які характеризують фото.

При натисненні на одну з карток відкривається слайдер на весь екран, який відображає всі фото відповідного заходу по одному (рис. 3.18).



Рис. 3.17 - Фотогалерея

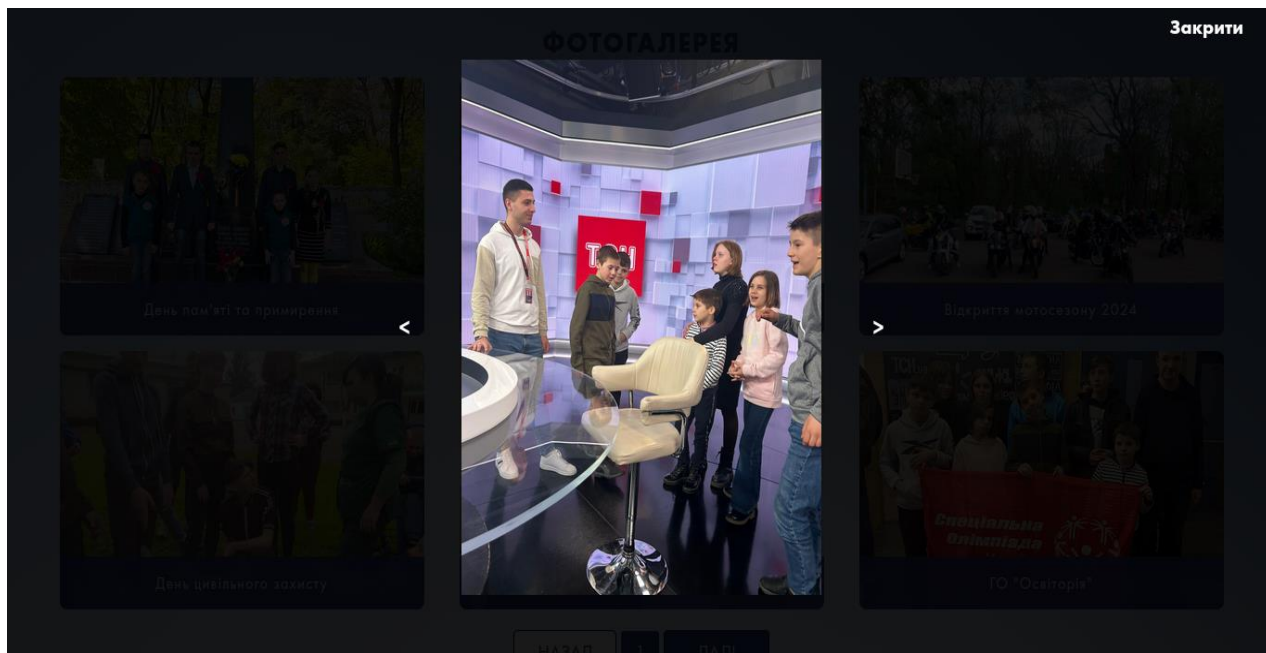


Рис. 3.18 – Фотогалерея

У відеогалереї зберігаються посилання на відео з заголовком (рис. 3.19).



Рис. 3.19 - Відеогалерея

На вкладці «Новини» (рис. 3.20) зберігається список новин у вигляді карток з головним фото, заголовком, коротким описом та датою їх створення.

При натисненні на картку відкривається повна інформація, яка може містити слайдери з фото, текст, коментарі. Для зручної навігації є кнопка повернення назад до списку новин.

Для незареєстрованого користувача є можливість перегляду коментарів без можливості залишити свій (рис. 3.21). Для користувача, що зареєстрований у системі така можливість надається (рис. 3.22).



Рис. 3.20 - Список новини

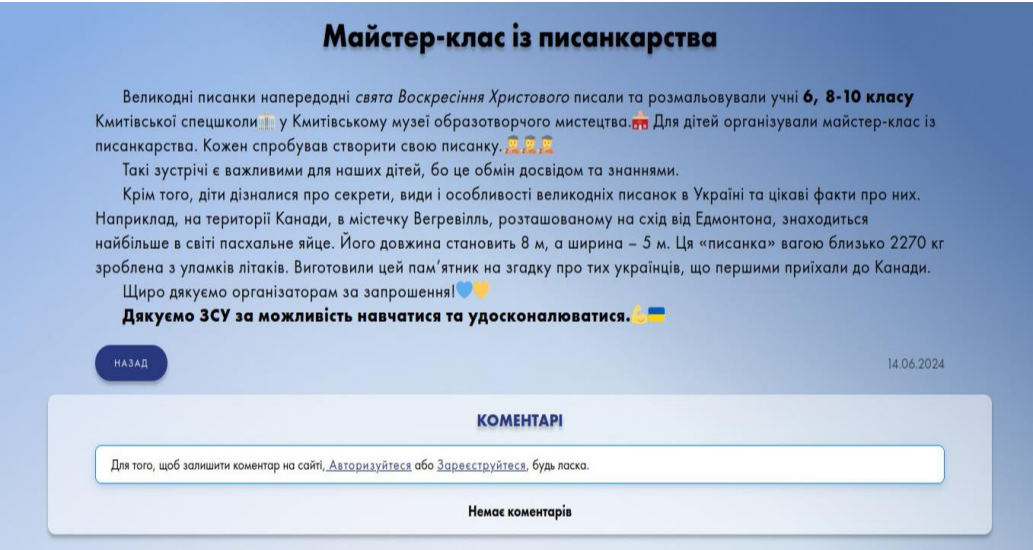


Рис. 3.21 - Відображення вибраної новини для гостя

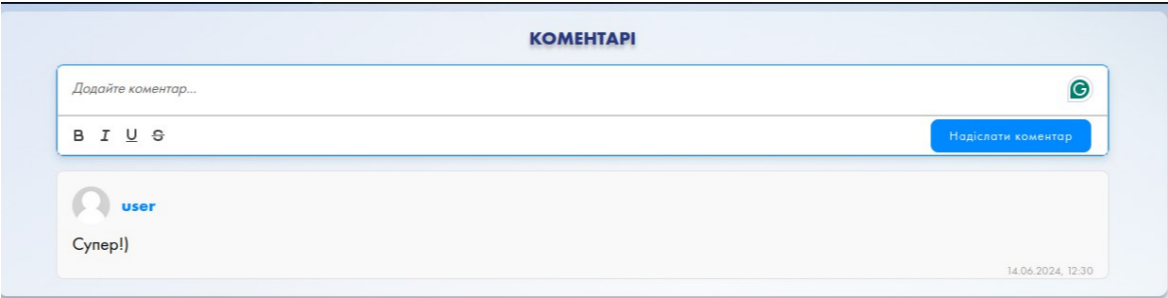


Рис. 3.22 - Відображення коментарів

Ввівши логін та пароль адміністратор може зайти до панелі адміністратора та мати можливість переглядати список наповнення (рис. 3.23), завантажувати дані на сайт через спеціальні форми, а також видаляти чи редагувати інформацію. Для зручної роботи з файлами передбачене поле для пошуку наявного контенту за заголовком статей та ім'ям файлів.

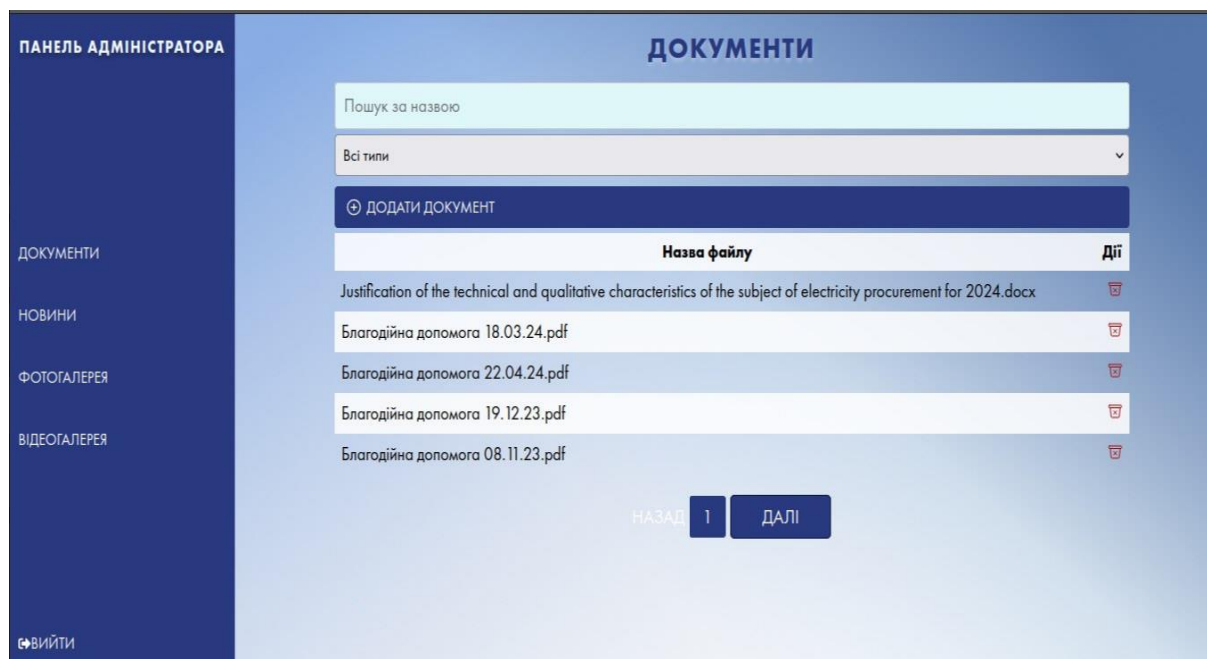


Рис. 3.23 - Список документації

Після успішного заповнення всіх полів на сторінці завантаження документів з'являється модульне вікно (рис. 3.24).

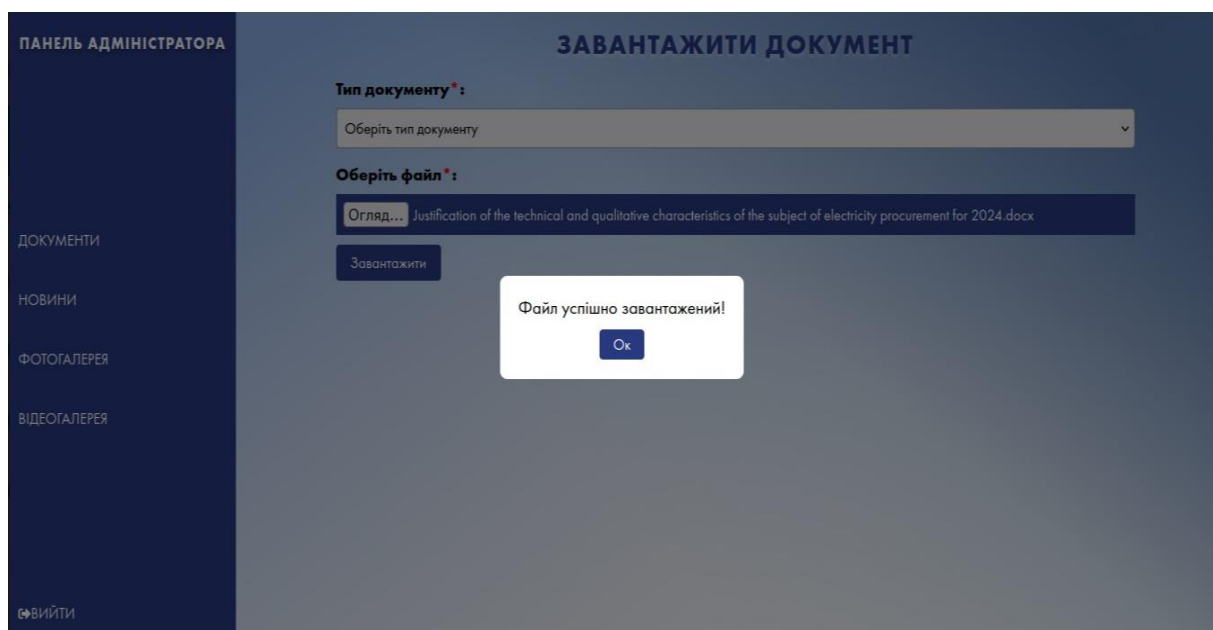


Рис. 3.24 - Додавання документу бо БД

Коли користувач натискає на козінку, що міститься напроти кожної назви файлу, він видаляється і з’являється відповідне повідомлення (рис. 3.25).

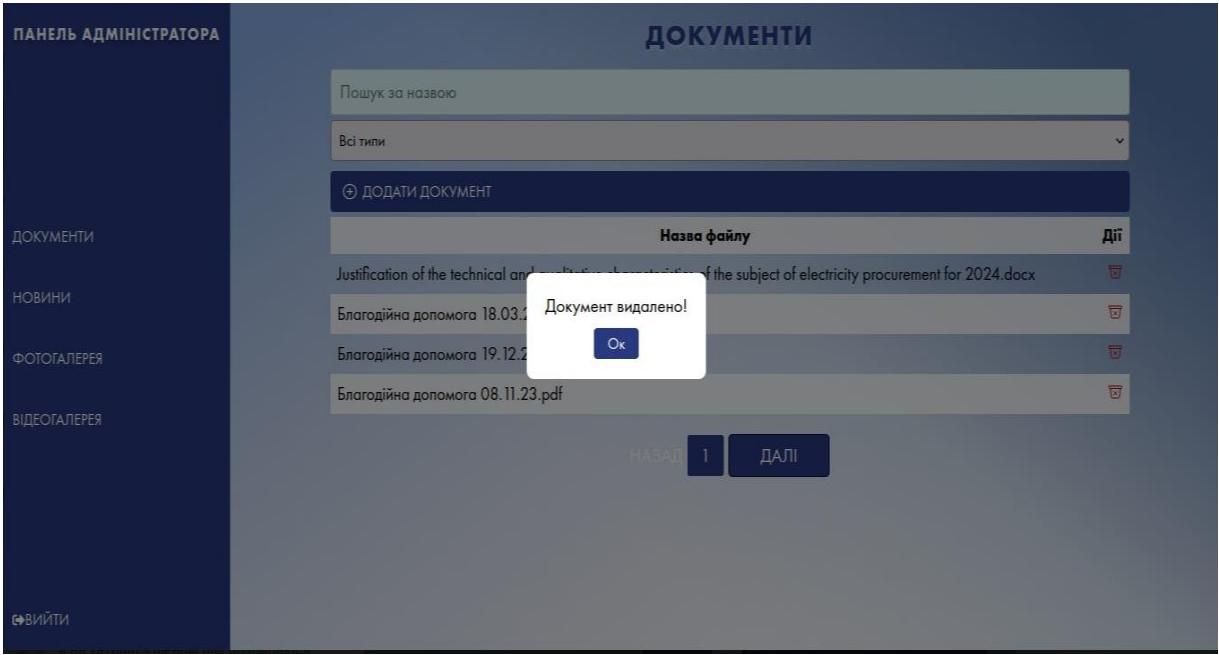


Рис. 3.25 - Видалення документу з БД

При вводиті у полі пошуку букв, система видає результат, що найбільше підходить запиту (рис. 3.26).

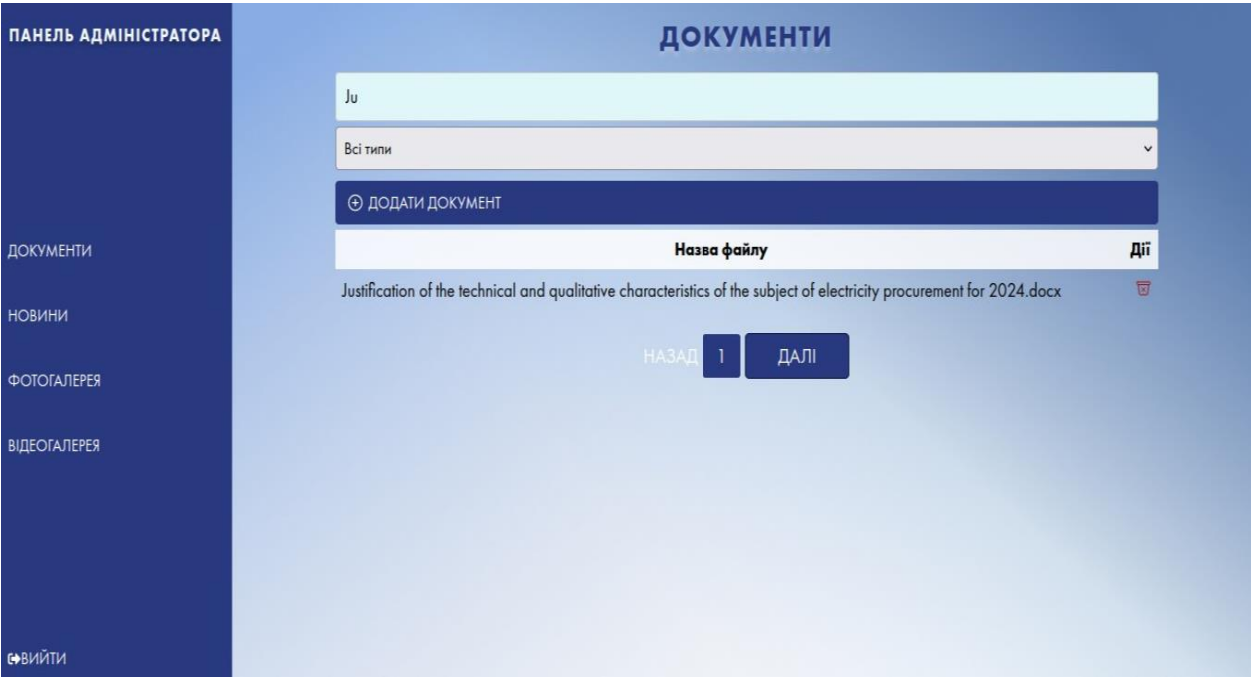


Рис. 3.26 - Пошук документу за назвою файлу

На вкладці новини зберігається список новин та кнопки, що дозволяють виконувати операції з ними (рис. 3.27).

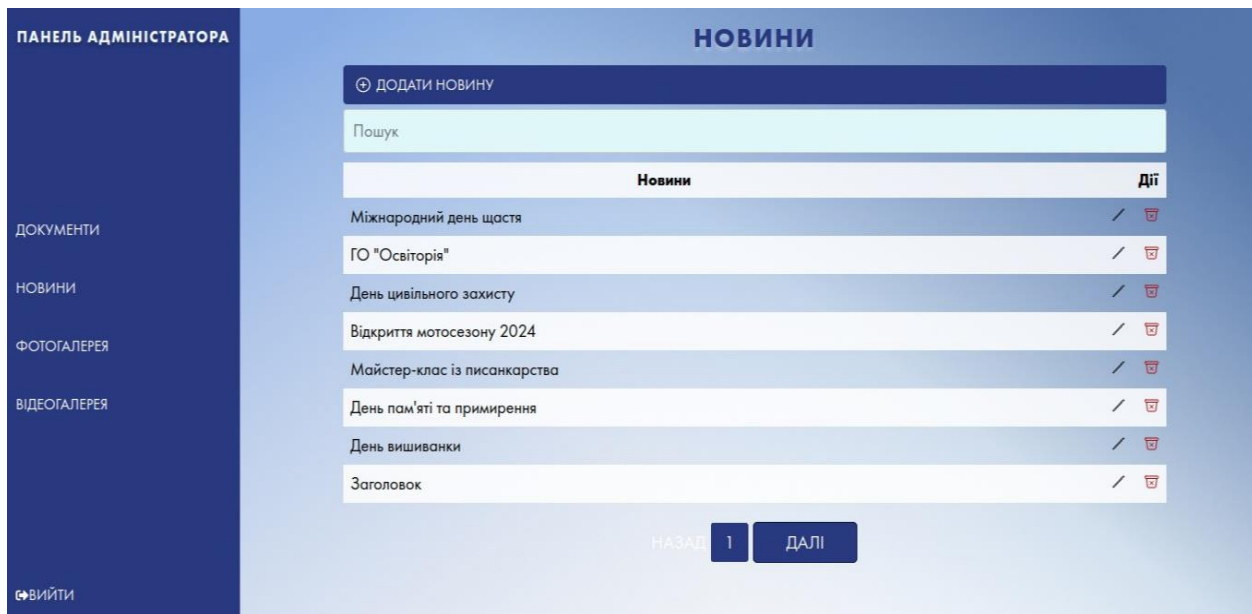


Рис. 3.27 - Список новин у системі

Після натиснення на кнопку «Додати новину», отримуємо форму для заповнення контентом (рис. 3.28). Завдяки багатофункціональному редактору React.Quill є можливість формувати текст безпосередньо у полі вводу, вставляти посилання, списки.

Є можливість додати більше зображень, які згодом можна побачити у вигляді слайдеру на сайті.

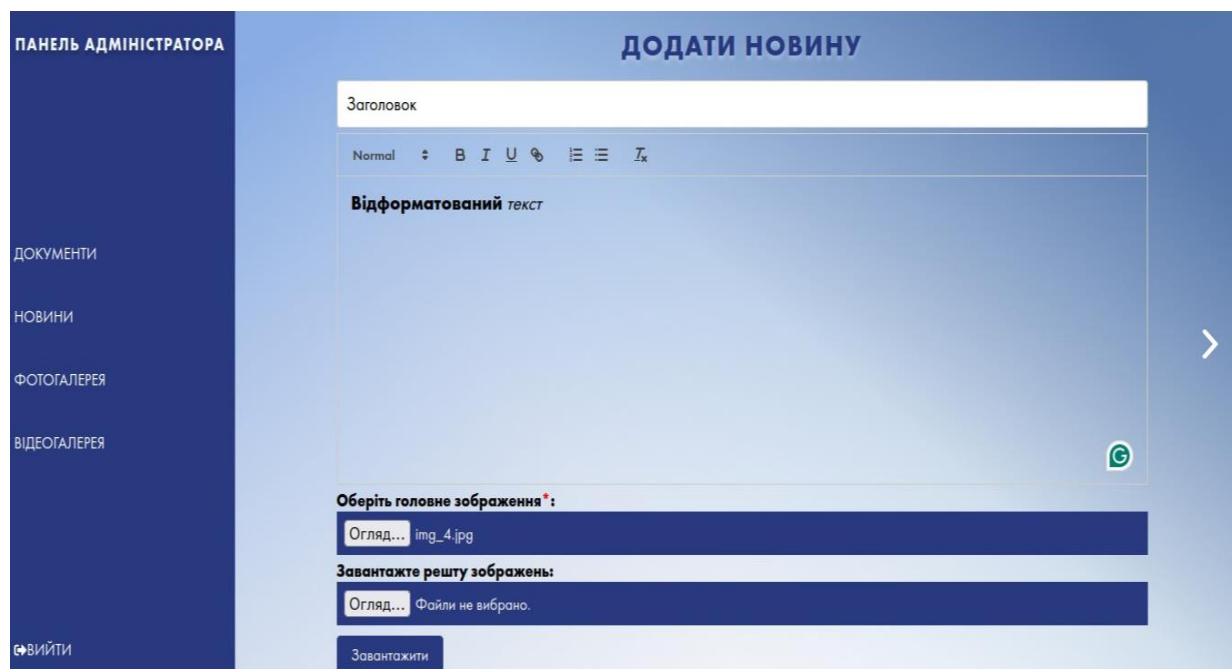


Рис. 3.28 - Сторінка для завантаження новини до БД

При додаванні новини система сповіщає про результат роботи (рис. 3.29).

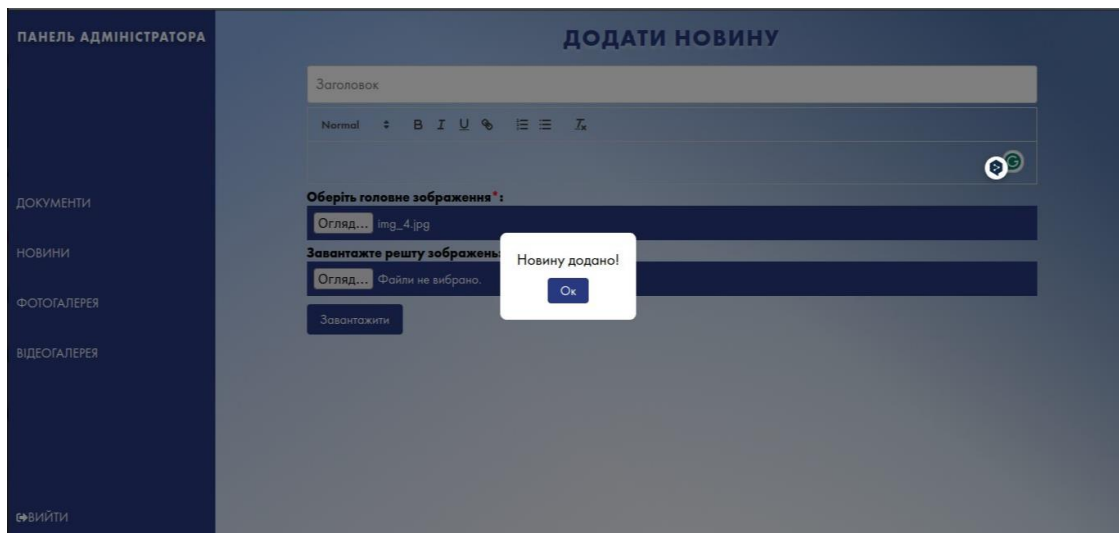


Рис. 3.29 - Завантаження новини до БД

З кліком на кошик новину можна видалити (рис. 3.30).

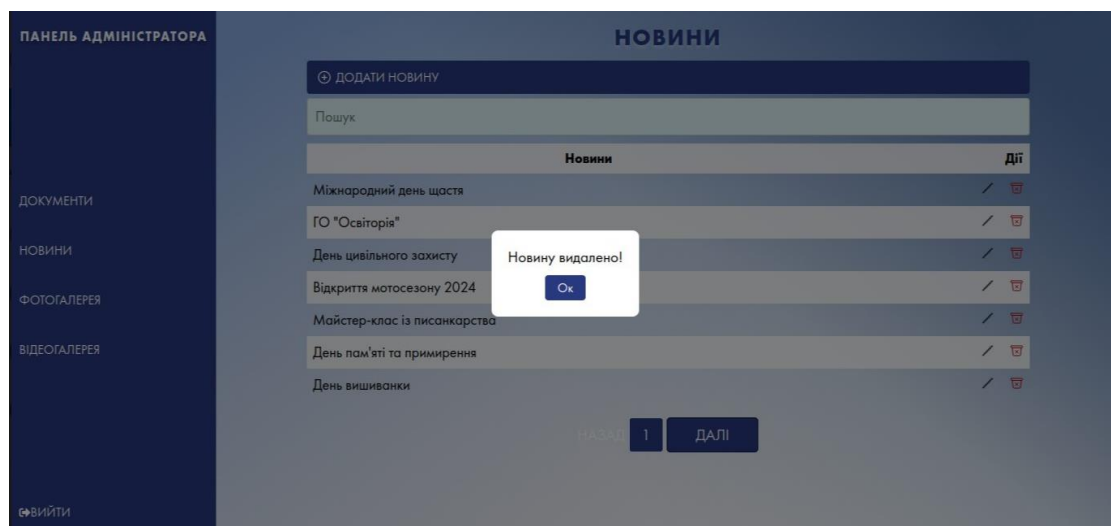


Рис. 3.30 - Видалення новини з БД

За заголовком новини можна здійснити пошук по БД (рис. 3.31).

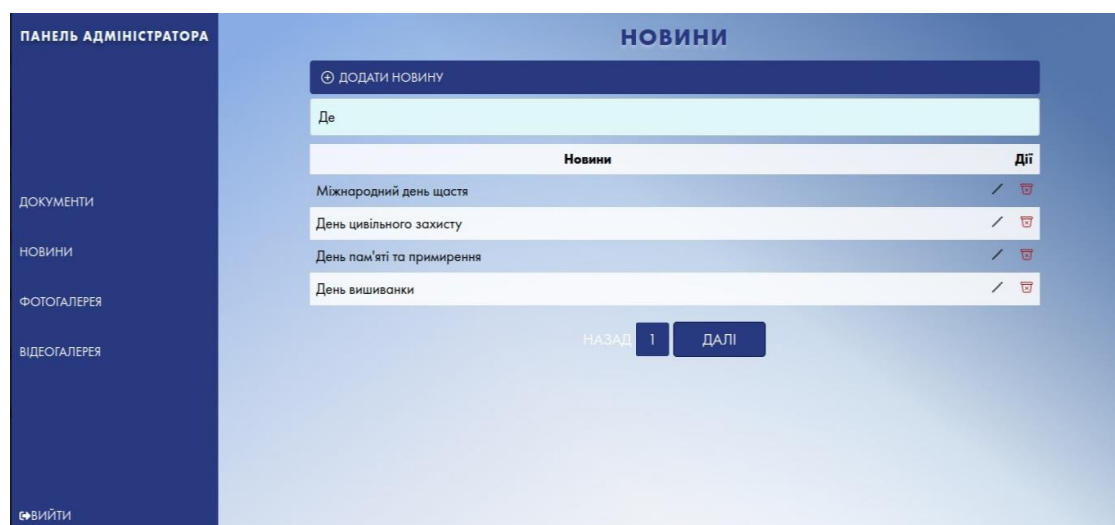


Рис. 3.31 - Пошук новини за заголовком

На вкладці «Фотогалерея» бачимо список записів за подіями (рис. 3.32).

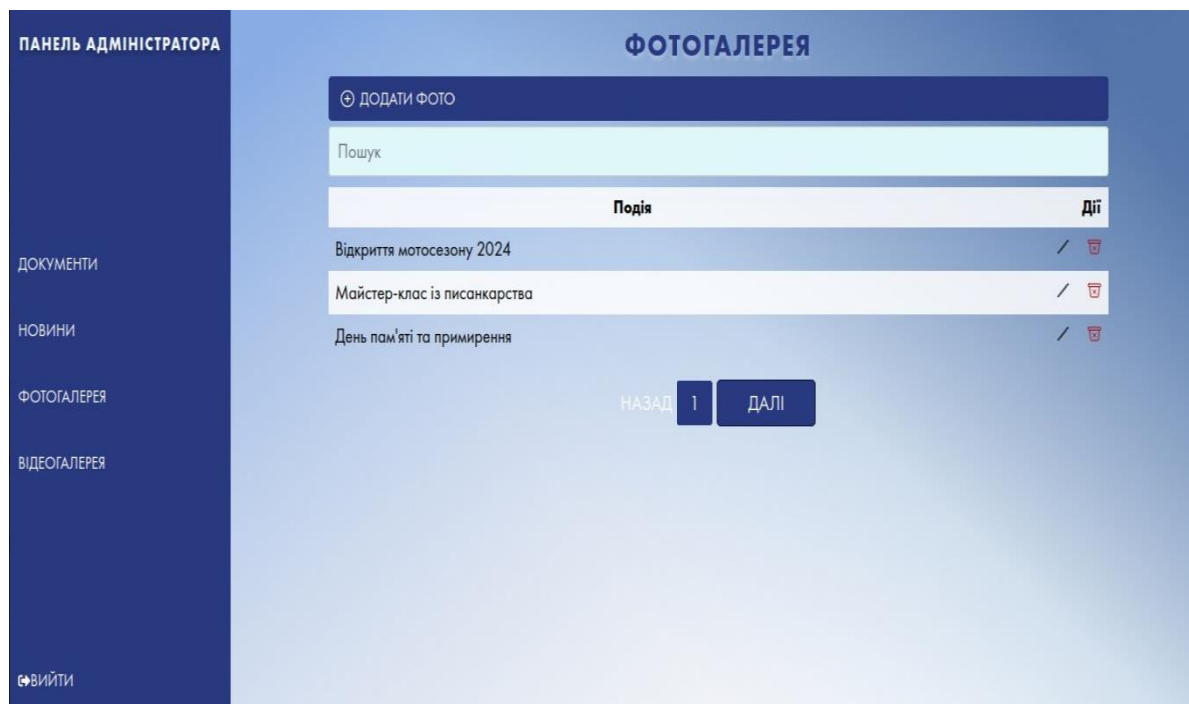


Рис. 3.32 - Список записів у фотогалереї

Після натиснення на кнопку «Додати фото» отримуємо форму для завантаження даних (рис. 3.33).

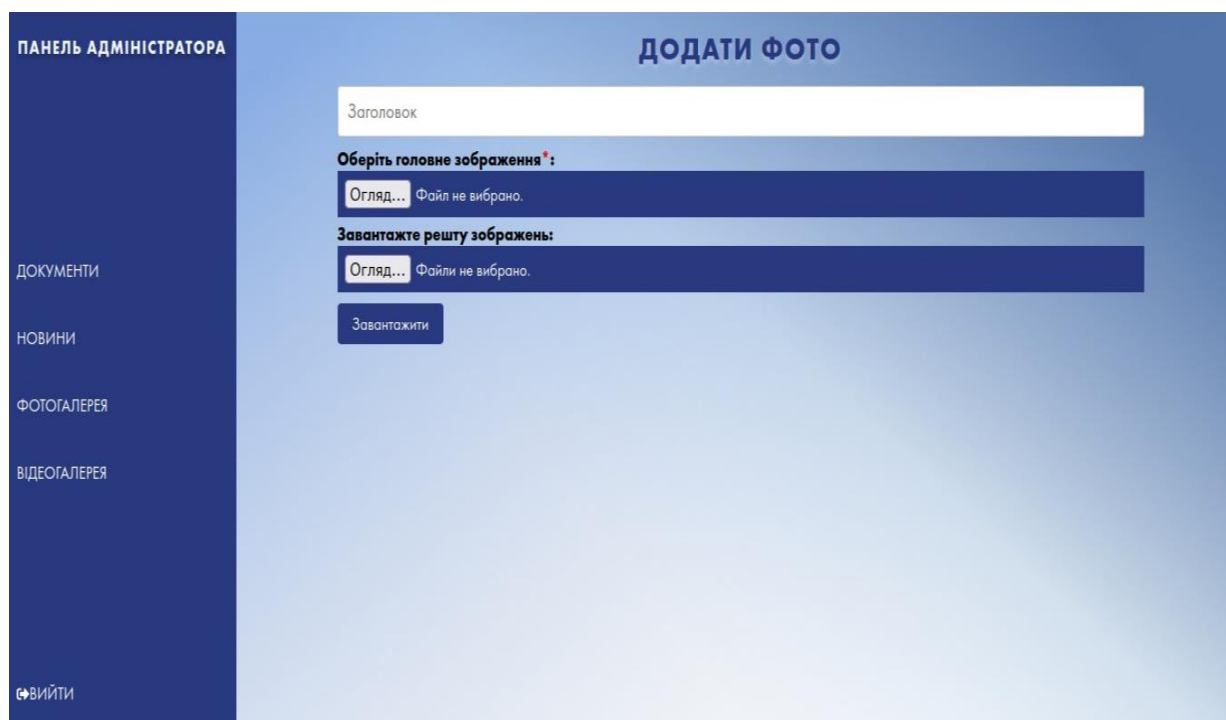


Рис. 3.33 - Сторінка для завантаження фото

Після натиснення на кнопку «Завантажити», отримуємо повідомлення про те, що фото успішно завантажені (рис. 3.34).

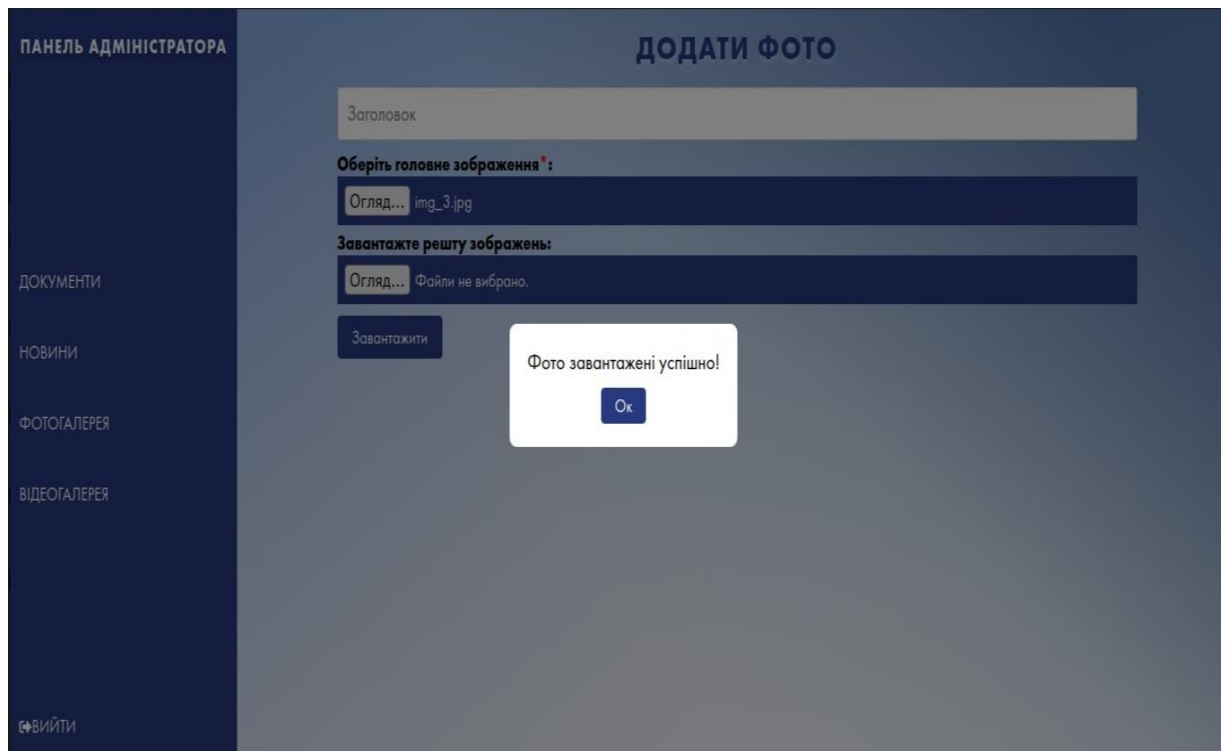


Рис. 3.34 - Повідомлення про успішне завантаження фото

Натиснення на кошик видаляє записи з бази даних та виводить повідомлення про успішне видалення (рис. 3.35).

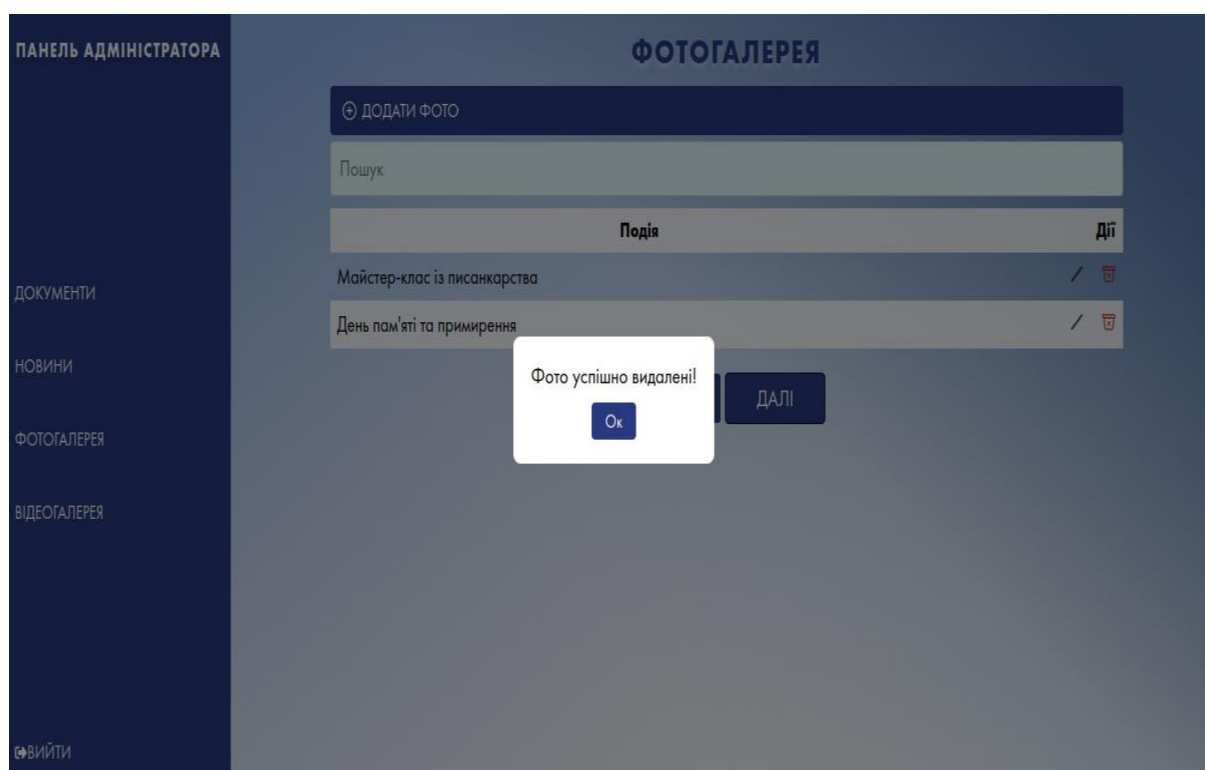


Рис. 3.35 - Видалення фотографій з БД

У фотогалереї також є можливість здійснити пошук за назвою події, яка відповідає завантаженим фото (рис. 3.36).

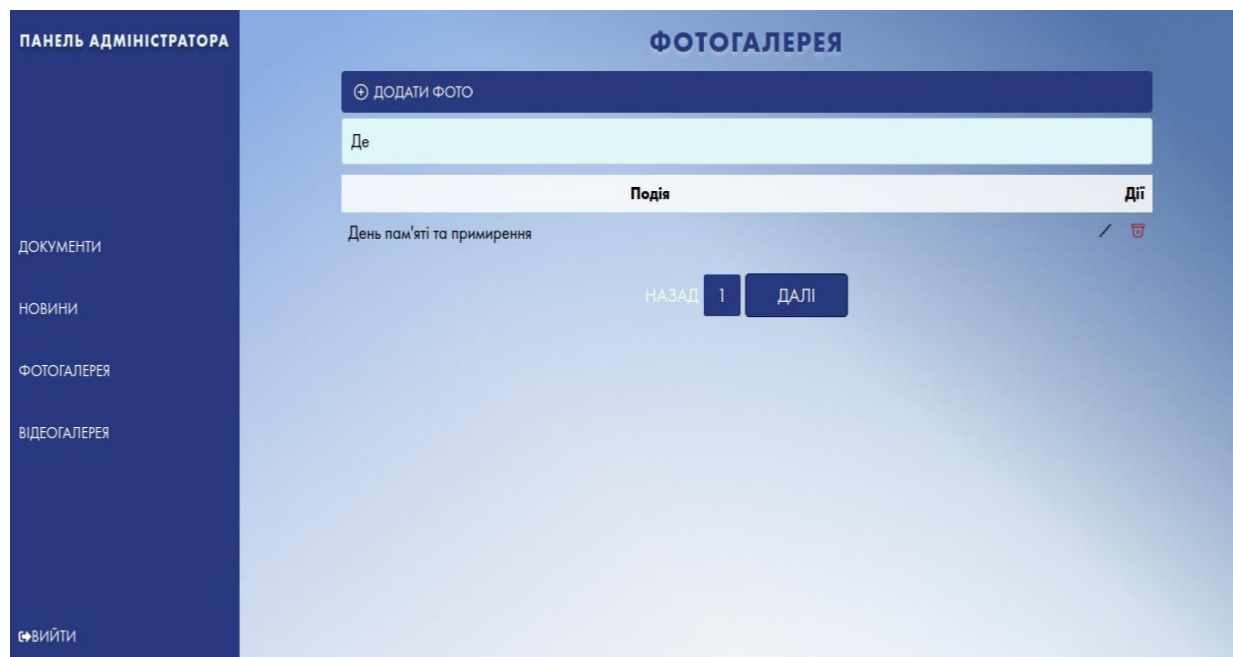


Рис. 3.36 - Пошук фото за назвою картки

3.3 Тестування роботи веб-сайту

Для тестування продуктивності та швидкості завантаження веб-сторінок було використано Pagespeed Insights.

PageSpeed Insights (PSI) звітує про взаємодію користувача зі сторінкою як на мобільних, так і на настільних пристроях, а також пропонує пропозиції щодо того, як цю сторінку можна покращити [18].

Продуктивність сайту (рис. 3.37) знаходиться в жовтій зоні, що вказує на те, що є місце для покращення.

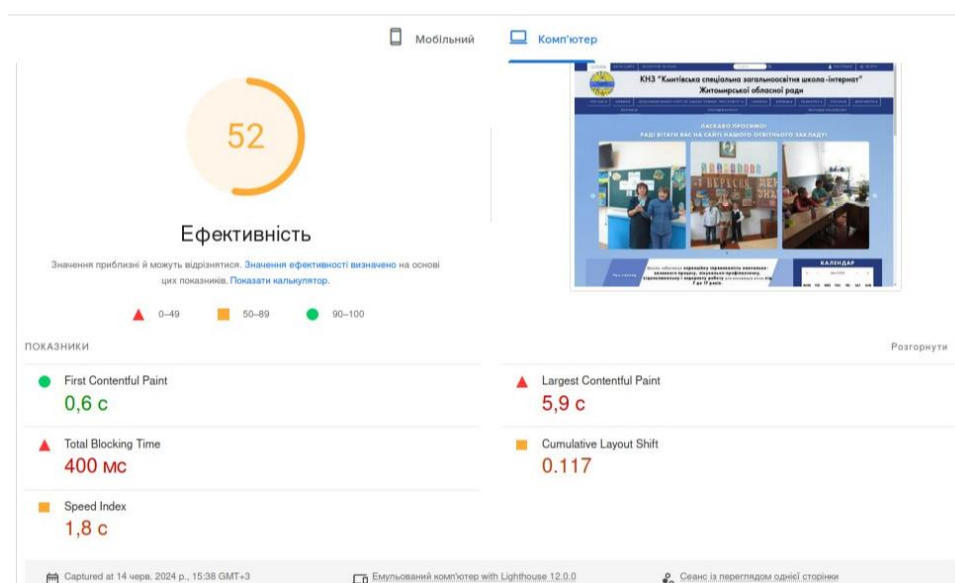


Рис. 3.37 - Загальна оцінка ефективності

Час, за який перший елемент контенту з'являється на екрані – 0.6 секунди. Показник знаходиться в зеленій зоні, що вказує на хороший результат.

Час, протягом якого сторінка заблокована і користувач не може взаємодіяти з нею – 400 мс. Показник знаходиться в червоній зоні, що вказує на проблеми з продуктивністю.

Показник, який вказує, наскільки швидко контент стає видимим під час завантаження сторінки – 1.8 секунди. Показник також у зеленій зоні, що добре.

Час, за який завантажуються найбільший елемент контенту на сторінці 5.9 секунд. Показник в червоній зоні, що вказує на необхідність оптимізації.

Показник, який вимірює, наскільки сторінка зсувається під час завантаження – 0.117, що вказує на незначні зміщення.

Щоб зменшити загальний час блокування (TVT) необхідно оптимізувати JavaScript і ресурси, що блокують рендеринг. Також потрібно використати методи оптимізації, такі як асинхронне завантаження ресурсів або затримка завантаження невидимих елементів.

Ці кроки можуть допомогти покращити ефективність сайту, підвищуючи його загальну продуктивність та користувацький досвід.

За результатами тестування доступності (рис. 3.38) виявили деякі проблеми з назвами та мітками, а також контрастом сайту. Загальний показник знаходиться в жовтій зоні.

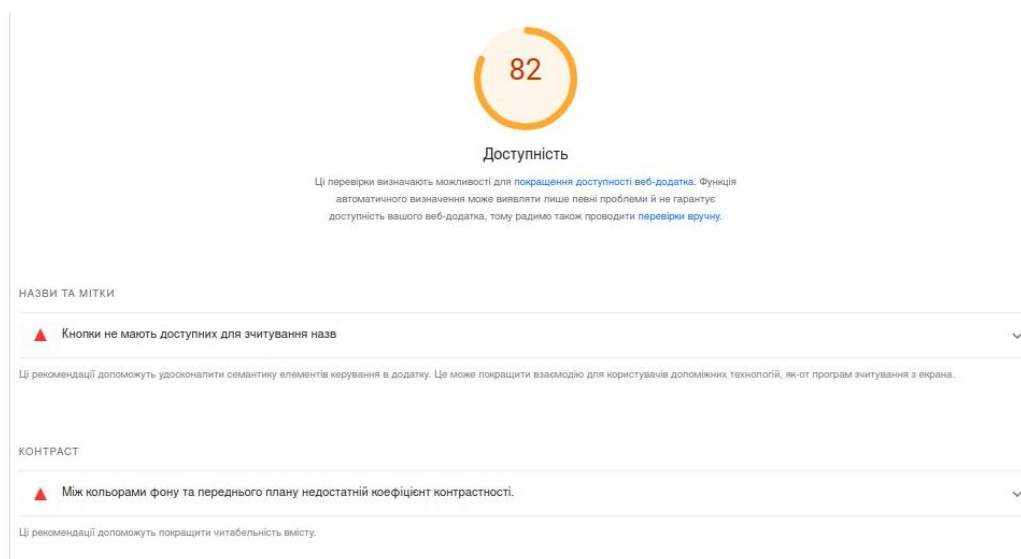


Рис. 3.38 - Доступність

При тестуванні оптимальних методів (рис. 3.39) було виявлено декілька помилок з API та картою джерела.

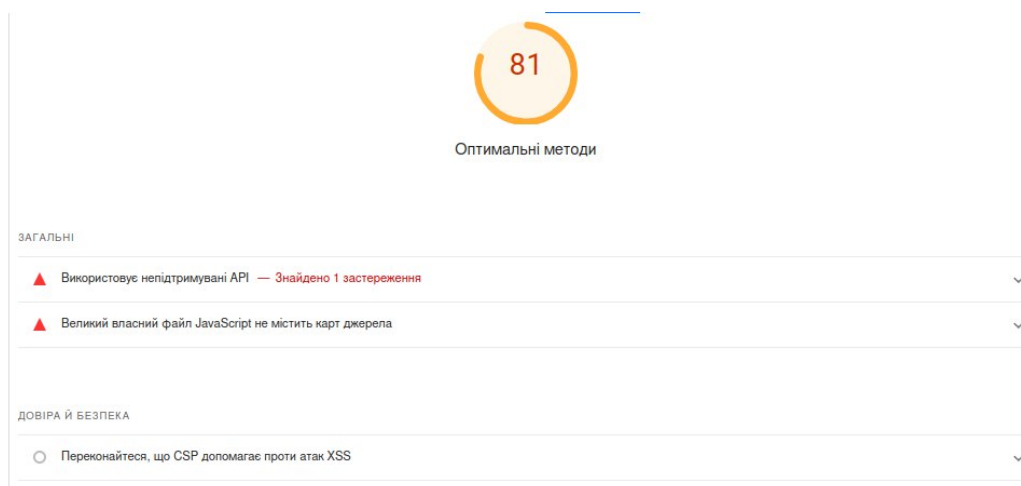


Рис. 3.39 - Оптимальні методи

Висновки до третього розділу

У першому підрозділі розглянуто процес розгортання веб-сайту школи-інтернат на віртуальному приватному сервері, описано компоненти системи та їх взаємодію, а також кроки для налаштування і запуску додатку.

У другому підрозділі зроблено опис інтерфейсу розробленого веб-сайту, який включає в себе функції з урахуванням ролі користувача.

Третій підрозділ присвячений тестуванню за допомогою сервісу Pagespeed Insights, під час якого було виявлено деякі проблеми з продуктивністю.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано веб-сайт для інформаційної підтримки школи-інтернат, проведено аналіз предметної області та порівняно аналоги програмного продукту. На основі аналізу було визначено функціональні вимоги до веб-додатку.

На етапі проектування визначено стек PERN для програмної реалізації та обрано клієнт-серверну архітектуру веб-орієнтованої системи закладу. Також було здійснено опис методів взаємодії з сайтом та розроблено базу даних для збереження інформації від освітнього закладу та користувачів, визначено об'єктну структуру веб-додатку.

На етапі реалізації програмного продукту розроблено алгоритми для виконання CRUD-операцій з контентом сайту для адміністратора та пошуку за введеними даними, а також забезпечення користувачів можливістю надати зворотній зв'язок, лишати коментарі, переглядати документацію та інший мультимедійний контент. Також розроблена система реєстрації, авторизації та автентифікації користувачів з допомогою JWT для надійності та розподілення доступу до сторінок за певними ролями.

Веб-сайт було розгорнуто на VPS-сервері та здійснено тестування програмного продукту на виявлення можливих помилок, перевірено швидкість завантаження сторінок.

Програмний продукт має простий та зрозумілий інтерфейс, що дозволяє користувачам легко знаходити необхідну інформацію.

Отже, розроблений веб-сайт повністю відповідає поставленим вимогам та досягнутим цілям. Він забезпечує необхідну інформаційну підтримку школи-інтернат, сприяє ефективному обміну інформацією між спільнотою, а також сприяє підвищенню прозорості та доступності освітнього процесу.

Результати тестування підтвердили надійність та ефективність функціонування веб-додатку. Програмний продукт готовий до подальшого використання та масштабування відповідно до потреб закладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України "Про освіту", № 2145-VIII, від 5 вересня 2017 р. Відомості Верховної Ради України (ВВР), 2017, № 38-39, ст. 380 [Електронний ресурс] – Режим доступу до ресурсу: <https://zakon.rada.gov.ua/laws/show/2145-19#Text>.
2. Україна. Інформаційна система управління освітою [Електронний ресурс] – Режим доступу до ресурсу: <https://isuo.org/>.
3. Бердичівська спеціальна школа Житомирської обласної ради [Електронний ресурс] - Режим доступу до ресурсу: <http://gryshkivtsi-internat.zt.sch.in.ua/>.
4. Денишівська спеціальна школа Житомирської обласної ради [Електронний ресурс] - Режим доступу до ресурсу: <https://kz-denishivskij-tsentr.webnode.com.ua/>.
5. КНЗ "Кмитівська спеціальна загальноосвітня школа-інтернат" Житомирської обласної ради [Електронний ресурс] - Режим доступу до ресурсу: <http://www.kmytiv-internat.edukit.zt.ua/>.
6. Резанова В. Г. Архітектурні патерни при розробці клієнт-серверних додатків / В. Г. Резанова, В. І. Марченко // Інформаційні технології в науці, виробництві та підприємстві : збірник наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. – Київ : Освіта України ; ФОП Маслаков, 2020. – С. 185.
7. Alves R. Get Started with the PERN Stack: An Introduction and Implementation Guide [Електронний ресурс] / Rita Alves // Medium. – 2023. – Режим доступу до ресурсу: <https://medium.com/@ritapalves/get-started-with-the-pern-stack-an-introduction-and-implementation-guide-e33c55d09994>.
8. JOURNAL I. ReactJS: A Comprehensive Analysis of its features, Performance, and Suitability for Modern Web Development. INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT. 2023. Vol. 07, no. 09. P. 1. URL: <https://doi.org/10.55041/ijsrem25667>.

9. What is PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/about/>.

10. API reference [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/en/4x/api.html>.

11. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en>.

12. РОЗРОБКА ВЕБ-ДОДАТКІВ ТА СЕРВІСІВ НА ПЛАТФОРМІ NODE.JS / О. П. Кошова та ін. *Таврійський науковий вісник. Серія: Технічні науки*. 2023. № 2. С. 82. URL: <https://doi.org/10.32782/tnv-tech.2023.2.9>.

13. Sequelize v6 [Електронний ресурс] – Режим доступу до ресурсу: <https://sequelize.org/docs/v6/>.

14. Що таке Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/uk/docs/intro>.

15. Synchronizing with Effects [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn/synchronizing-with-effects>.

16. Розгортання програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: <https://reporter.zp.ua/rozgortannya-programnogo-zabezpechennya-l-uk.html>.

17. РОЗРОБКА ВЕБ-ДОДАТКІВ: 8 ВАЖЛИВИХ ЕТАПІВ ДЛЯ СТВОРЕННЯ ЧУДОВИХ ДОДАТКІВ [Електронний ресурс] – Режим доступу до ресурсу: <https://whileweb.com/uk/blog/web-app-development-8-critical-stages-to-build-amazing-apps/>.

18. About PageSpeed Insights [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.google.com/speed/docs/insights/v5/about>.

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						68

server.js

```

const express = require("express");
const cors = require("cors");
const path = require('path');
const app = express();
let corsOptions = {
  origin: "http://localhost:3000"
};

app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use('/public', express.static(path.join(__dirname, 'public')));

const db = require('./models ');
const Role = db.role;
const TypeDocument = db.type_document;

db.sequelize.sync().then(() => {
  console.log('Drop and Resync DB');
});
require('./routes/auth.routes')(app);
require('./routes/user.routes')(app);
require('./routes/news.routes')(app);
require('./routes/document.routes')(app);
require('./routes/photogallery.routes')(app);
require('./routes/sliderImage.routes')(app);
require('./routes/typesDoc.routes')(app);
require('./routes/comment.routes')(app);

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
})
middleware/authJwt.js

const jwt = require('jsonwebtoken');
const config = require('../config/auth.config');
```


```

const db = require('../models ');
const User = db.user;

const { TokenExpiredError } = jwt;

const catchError = (err, res) => {
  if (err instanceof TokenExpiredError) {
    return res.status(401).send({ message: "Неавторизовано! Термін дії маркера доступу  
минув!" });
  }
  return res.sendStatus(401).send({ message: "Неавторизовано!" });
}

verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({ message: "Токен не надано!" });
  }
  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return catchError(err, res);
    }
    req.userId = decoded.id;
    next();
  });
};

isAdmin = (req, res, next) => {
  User.findById(req.userId).then(user => {
    if (!user) {
      return res.status(404).send({ message: "Користувача не знайдено!" });
    }
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "admin") {
          next();
          return;
        }
      }
    });
  });
};

```

```

    }
    res.status(403).send({ message: "Потрібна роль адміністратора!" });
  });
});
};

isModerator = (req, res, next) => {
  User.findById(req.userId).then(user => {
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "moderator") {
          next();
          return;
        }
      }
    });
    res.status(403).send({ message: "Потрібна роль модератора!" });
  });
});
};

isModeratorOrAdmin = (req, res, next) => {
  User.findById(req.userId).then(user => {
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "moderator" || roles[i].name === "admin") {
          next();
          return;
        }
      }
    });
    res.status(403).send({ message: "Потрібна роль адміністратора чи модератора!" });
  });
});
};

const authJwt = {
  verifyToken,
  isAdmin,
  isModerator,

```

```
isModeratorOrAdmin
```

```
};
```

```
module.exports = authJwt;
```

```
middleware/SignUp.js
```

```
const db = require('../models');
```

```
const ROLES = db.role;
```

```
const User = db.user;
```

```
checkDuplicateUsernameOrEmail = (req, res, next) => {
```

```
User.findOne({ where: { username: req.body.username } })
```

```
.then(user => {
```

```
if (user) {
```

```
res.status(400).send({ message: "Помилка! Ім'я вже використовується!" });
```

```
return;
```

```
}
```

```
User.findOne({ where: { email: req.body.email } })
```

```
.then(user => {
```

```
if (user) {
```

```
res.status(400).send({ message: "Помилка! Email вже використовується!" });
```

```
return;
```

```
}
```

```
next();
```

```
});
```

```
});
```

```
};
```

```
checkRolesExisted = (req, res, next) => {
```

```
if (req.body.roles) {
```

```
const validRoles = ['user', 'admin', 'moderator'];
```

```
for (let i = 0; i < req.body.roles.length; i++) {
```

```
if (!validRoles.includes(req.body.roles[i])) {
```

```
res.status(400).send({ message: "Помилка! Роль не існує: " + req.body.roles[i] });
```

```
return;
```

```
}
```

```
}
```

```
}
```

```
next();
```

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						72

```
};
```

```
const verifySignUp = {  
  checkDuplicateUsernameOrEmail,  
  checkRolesExisted  
};
```

```
module.exports = verifySignUp;
```

middleware/uplad.js

```
const multer = require('multer');  
const path = require('path');  
const fs = require('fs');
```

```
const storage = multer.diskStorage({  
  destination: (req, file, cb) => {  
    const folderPath = path.join('public/documents');  
    fs.mkdirSync(folderPath, { recursive: true });  
    cb(null, folderPath);  
  },  
  filename: (req, file, cb) => {  
    file.originalname = Buffer.from(file.originalname, 'latin1').toString('utf8')  
    cb(null, `${Date.now()}-${file.originalname}`);  
  }  
});
```

```
const upload = multer({  
  preservePath: true,  
  storage: storage,
```

```
  fileFilter: (req, file, cb) => {  
    console.log(file)  
    const ext = path.extname(file.originalname).toLowerCase();  
    if (ext !== '.pdf' && ext !== '.doc' && ext !== '.docx') {  
      return cb(new Error('Дозволено завантажувати тільки документи'));  
    }  
  }
```

```
const allowedMimeTypes = [
```

```

'application/pdf',
'application/msword',
'application/vnd.openxmlformats-officedocument.wordprocessingml.document'
];
if (!allowedMimeTypes.includes(file.mimetype)) {
return cb(new Error('Дозволено завантажувати тільки документи'));
}
cb(null, true);
}
});

```

```

module.exports = upload;
middleware/uploadPhotos.js

```

```

const multer = require('multer');
const path = require('path');
const fs = require('fs');

```

```

const storage = multer.diskStorage({
destination: (req, file, cb) => {
const folderPath = path.join('public/images');
fs.mkdirSync(folderPath, { recursive: true });
cb(null, folderPath);
},
filename: (req, file, cb) => {
file.originalname = Buffer.from(file.originalname, 'latin1').toString('utf8')
cb(null, `${Date.now()}-${file.originalname}`);
}
});

```

```

const upload = multer({
preservePath: true,
storage: storage,

```

```

fileFilter: (req, file, cb) => {
console.log(file)
const ext = path.extname(file.originalname).toLowerCase();
if (ext !== '.jpeg' && ext !== '.png' && ext !== '.jpg' && ext !== '.gif') {
return cb(new Error('Дозволено завантажувати тільки зображення'));
}
}
});

```

```

}

const allowedMimeTypes = [
  'image/jpeg',
  'image/png',
  'image/gif'
];
if (!allowedMimeTypes.includes(file.mimetype)) {
  return cb(new Error('Дозволено завантажувати тільки зображення'));
}
cb(null, true);
});

module.exports = upload;

router/auth.routes.js
const { verifySignUp, authJwt } = require("../middleware");
const controller = require("../controllers/auth.controller");

module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });

  app.post(
    "/api/auth/signup",
    [
      verifySignUp.checkDuplicateUsernameOrEmail,
      verifySignUp.checkRolesExisted
    ],
    controller.signup
  );
  app.post("/api/auth/signin", controller.signin);

```

```

app.post("/api/auth/refreshToken", controller.refreshToken);
app.get("/api/auth/user", [authJwt.verifyToken], controller.getUser);
};

routes/document.routes.js

const { authJwt } = require("../middleware");
const controller = require("../controllers/document.controller");
const upload = require("../middleware/upload");

module.exports = function (app) {
app.get("/api/documents/:id", controller.getDocumentById);
app.get("/api/documents/type/:typeId", controller.getDocumentsByType);
app.get("/api/documentsAll", controller.getAllDocument);

app.post('/api/documents',
[authJwt.verifyToken, authJwt.isAdmin],
(req, res) => {
upload.single('file')(req, res, (err) => {
if (err) {
return res.status(400).send({ message: err.message });
}
controller.uploadDocument(req, res);
});
},
);

app.put("/api/documents/:id", [authJwt.verifyToken, authJwt.isAdmin],
controller.updateDocument);
app.delete("/api/documents/:id", [authJwt.verifyToken, authJwt.isAdmin],
controller.deleteDocument);
};

routes/photogallery.routes.js

const { authJwt } = require("../middleware");
const controller = require("../controllers/photogallery.controller");
const upload = require("../middleware/uploadPhotos");
module.exports = function (app) {
app.get("/api/galleries", controller.getAllPhotoGallery);

```

```

app.get("/api/gallery/:id", controller.getPhotoGalleryById);
const cpUpload = upload.fields([ { name: 'imageCard', maxCount: 1 }, { name: 'images',
maxCount: 15 } ])
app.post('/api/gallery',
[authJwt.verifyToken, authJwt.isAdmin],
(req, res, next) => {
cpUpload(req, res, (err) => {
if (err) return res.status(400).send({ message: err.message });
controller.createPhotoGallery(req, res);
});
}
);

```

```

app.put("/api/gallery/:id", [authJwt.verifyToken, authJwt.isAdmin],
controller.updatePhotoGallery);
app.delete("/api/gallery/:id", [authJwt.verifyToken, authJwt.isAdmin],
controller.deletePhotoGallery);
};

```

controller/document.controller.js

```

const fs = require('fs');
const path = require('path');
const Sequelize = require('sequelize');
const db = require('../models');
const Document = db.documents;
const TypeDocument = db.type_document;

```

```

exports.uploadDocument = (req, res) => {
const file = req.file;

```

```

if (!file) {
return res.status(400).send({ message: 'Файл не завантажено' });
}
Document.create({
filename: file.originalname,
url: file.path,
type_documentId: req.body.type_documentId
})

```


```

.then(document => {
res.status(201).send(document);
})
.catch(err => {
res.status(500).send({ message: err.message });
});
};

exports.getAllDocument = (req, res) => {
const { name, typeId } = req.query;
const conditions = {};

if (name) {
conditions.filename = { [Sequelize.Op.like]: `%${name}%` };
}
if (typeId) {
conditions.type_documentId = typeId;
}

Document.findAll({
where: conditions,
order: [['createdAt', 'DESC']]
})
.then(documents => {
res.status(200).send(documents);
})
.catch(err => {
res.status(500).send({ message: err.message });
});
};

exports.getDocumentById = (req, res) => {
const id = req.params.id;

Document.findById(id)
.then(document => {
if (!document) {

```

```

res.status(404).send({ message: "Документ не найдено: " + id });
return;
}
res.status(200).send(document);
})
.catch(err => {
res.status(500).send({ message: err.message });
});
});

```

```

exports.getDocumentsByType = (req, res) => {
const typeId = req.params.typeId;

```

```

TypeDocument.findByPk(typeId, {
include: [{
model: Document,
as: 'documents'
}]
})
.then(type => {
if (!type) {
res.status(404).send({ message: "Тип документа не найдено" });
return;
}
res.status(200).send(type.documents);
})
.catch(err => {
res.status(500).send({ message: err.message });
});
});

```

```

exports.updateDocument = (req, res) => {
const id = req.params.id;

```

```

Document.update(req.body, {
where: { id: id }
})
.then(num => {

```


```

    if (num == 1) {
      res.status(200).send({ message: "Документ оновлено!" });
    } else {
      res.status(404).send({ message: `Не можна оновити документ з id=${id}. Document not found
or req.body is empty!` });
    }
  })
  .catch(err => {
    res.status(500).send({ message: err.message });
  });
};

exports.deleteDocument = (req, res) => {
  const id = req.params.id;

  Document.findByIdPk(id)
    .then(document => {
      if (!document) {
        res.status(404).send({ message: `Документ з id=${id} не знайдено.` });
        return;
      }
      const filePath = path.resolve(document.url);

      Document.destroy({
        where: { id: id }
      })
        .then(num => {
          if (num == 1) {
            fs.unlink(filePath, (err) => {
              if (err) {
                console.error(`Не вдалося видалити файл ${filePath}:`, err);
                res.status(500).send({ message: `Не вдалося видалити файл з файлової системи:
${err.message}` });
                return;
              }
              res.status(200).send({ message: "Документ видалено!" });
            });
          } else {

```

```

    res.status(404).send({ message: `Не можна видалити документ з id=${id}. Document not
found!` });
  }
})
.catch(err => {
  console.error(`Помилка видалення документа з БД:`, err);
  res.status(500).send({ message: err.message });
});
})
.catch(err => {
  console.error(`Помилка пошуку документа з id=${id}:`, err);
  res.status(500).send({ message: err.message });
});
});

```

controller/comment.controller.js

```

const db = require("../models");
const Comment = db.comment;

exports.create = (req, res) => {
  if (!req.body.content) {
    res.status(400).send({
      message: "Content cannot be empty!"
    });
    return;
  }

```

```

  const comment = {
    content: req.body.content,
    userId: req.userId,
    newsId: req.body.newsId
  };

```

```

  Comment.create(comment)
    .then(data => {
      res.send(data);
    })
    .catch(err => {

```


```

res.status(500).send({
  message: err.message || "Some error occurred while creating the comment."
});
});
};

```

```

exports.findAllForNews = (req, res) => {
  const id = req.params.id;

```

```

  Comment.findAll({
    where: { newsId: id },
    include: [{
      model: db.user,
      attributes: ['username']
    }]
  })
  .then(data => {
    res.send(data);
  })
  .catch(err => {
    res.status(500).send({
      message: err.message || "Some error occurred while retrieving comments."
    });
  });
};

```

App.js

```

import { HelmetProvider } from 'react-helmet-async';
import { Route, Routes } from 'react-router-dom';
import Layout from './components/Layout';
import Home from './pages/Home';
import SignUp from './pages/SignUp';
import SiteMap from './pages/SiteMap';
import Contacts from './pages/Contacts';
import ResetPassword from './pages/ResetPassword';
import SignIn from './pages/SignIn';
import Feedback from './pages/Feedback';

```


```
/*import about */
```

```
import About from './pages/About/About';  
import HistorySchool from './pages/About/HistorySchool';  
import Achievement from './pages/About/Achievement'  
import Symbols from './pages/About/Symbols'  
import Partners from './pages/About/Partners'  
import Cooperation from './pages/About/Cooperation'  
import Schedule from './pages/About/Schedule';  
import MassMedia from './pages/About/MassMedia';  
import ModeOfOperation from './pages/About/ModeOfOperation'  
import TeachingMethods from './pages/About/TeachingMethods'  
import ProjectActivity from './pages/About/ProjectActivity'  
import MedicalService from './pages/About/MedicalService'  
import Rules from './pages/About/Rules';  
import WorkPlan from './pages/About/WorkPlan';
```

```
/*about education */
```

```
import Zakon from './pages/AboutEducation/Zakon';  
import Vacancies from './pages/AboutEducation/Vacancies';  
import Collective from './pages/AboutEducation/Collective'  
import License from './pages/AboutEducation/License';  
import AboutViolence from './pages/AboutEducation/AboutViolence';  
import Admission from './pages/AboutEducation/Admission';  
import Conduct from './pages/AboutEducation/Conduct';  
import Monitoring from './pages/AboutEducation/Monitoring';  
import MaterialBaza from './pages/AboutEducation/MaterialBaza';  
import Language from './pages/AboutEducation/Language';  
import Statut from './pages/AboutEducation/Statut';  
import LicensedVolume from './pages/AboutEducation/LicensedVolume';  
import Territory from './pages/AboutEducation/Territory';  
import EducationalPrograms from './pages/AboutEducation/EducationalPrograms';  
import CertificateOfAttestation from './pages/AboutEducation/CertificateOfAttestation';  
import Vipiska from './pages/AboutEducation/Vipiska';  
import Structure from './pages/AboutEducation/Structure';  
import ReportActivity from './pages/AboutEducation/ReportActivity';
```

```
/*import parents */
```

```
import Parents from './pages/Parents/Parents';
```

```

import Advice from './pages/Parents/Advice'
import Committee from './pages/Parents/Committee'
import Conditions from './pages/Parents/Conditions'
import Meeting from './pages/Parents/Meeting'
import Preparation from './pages/Parents/Preparation'
/*import news */
import NewsListPage from './pages/News/NewsListPage'
import NewsSinglePage from './pages/News/NewsSinglePage'
/*import gallery */
import PhotoGallery from './pages/Gallery/PhotoGallery';
import VideoGallery from './pages/Gallery/VideoGallery'

/*import psychologist pages */
import PsychologistList from './pages/psychologist/PsychologistList'
import Psychologist from './pages/psychologist/Psychologist'
import WorkSchedule from './pages/psychologist/WorkSchedule';
import Seminars from './pages/psychologist/Seminars';
import ForParents from './pages/psychologist/ForParents';
import ForPedagogs from './pages/psychologist/ForPedagogs';
import ForSchoolchildren from './pages/psychologist/ForSchoolchildren';
import PsychologicalAssistance from './pages/psychologist/PsychologicalAssistance';
import PsychologistPost from './pages/psychologist/PsychologistPost';

/* import documents pages*/
import Documents from './pages/Documents/Documents';
import Dopomoga from './pages/Documents/Dopomoga';
import RegulatoryDocuments from './pages/Documents/RegulatoryDocuments';
import FinancialReport from './pages/Documents/FinancialReport';
import Access from './pages/Documents/Access';
import GovernmentProcurement from './pages/Documents/GovernmentProcurement';
/* import logoped pages*/
import LogopedPostsList from './pages/Logoped/LogopedPostsList';
import LogopedPost from './pages/Logoped/LogopedPost';
import AuthProvider from './hooks/auth';
import Gallery from './pages/Gallery/Gallery';
//buling
import BulingPage from './pages/BulingPage';
import AdminPanel from './pages/AdminPanel';

```

```

import ProfilePage from './pages/ProfilePage'
import PrivateRoute from './router/router';
import LayoutForAdmin from './components/LayoutForAdmin';
import DocumentsListPage from './pages/Admin/DocumentsListPage';
import DocumentUploadPage from './pages/Admin/DocumentUploadPage';
import NewsEditorPage from './pages/Admin/NewsEditorPage';
import NewsPage from './pages/Admin/NewsPage';
import PhotoListPage from './pages/Admin/PhotoListPage';
import PhotoAddPage from './pages/Admin/PhotoAddPage';
function App() {
return (
<AuthProvider>
<HelmetProvider>
<Routes>
<Route element={<PrivateRoute />}>
<Route path="/admin-panel" element={<LayoutForAdmin/>}>
<Route index element={<AdminPanel /> } />
<Route path="/admin-panel/doc-list" element={<DocumentsListPage/>}/>
<Route path="/admin-panel/upload" element={<DocumentUploadPage/>}/>
<Route path="/admin-panel/news-list" element={<NewsPage/> } />
<Route path="/admin-panel/upload-news" element={<NewsEditorPage/>}/>
<Route path="/admin-panel/photos-list" element={<PhotoListPage/> } />
<Route path="/admin-panel/upload-photo" element={<PhotoAddPage/>}/>
</Route>
</Route>
<Route path="/" element={<Layout />}>
<Route index element={<Home /> } />
<Route path="/auth" element={<SignIn /> } />
<Route element={<PrivateRoute />}>
<Route path="/profile" element={<ProfilePage /> } />
</Route>
<Route path="sitemap" element={<SiteMap /> } />
<Route path="signup" element={<SignUp /> } />
<Route path="auth" element={<SignIn /> } />
<Route path="reset-password" element={<ResetPassword /> } />
<Route path="history" element={<HistorySchool /> } />
{ /*About*/ }

```

```

<Route path="about" element={<About />} />
<Route path="achievement" element={<Achievement />} />
<Route path="symbols" element={<Symbols />} />
<Route path="partners" element={<Partners />} />
<Route path="cooperation" elenemt={<Cooperation />} />
<Route path="schedule" element={<Schedule />} />
<Route path="mass-media" element={<MassMedia />} />
<Route path="regime" element={<ModeOfOperation />} />
<Route path="rules" element={<Rules />} />
<Route path="work-plan" element={<WorkPlan />} />
<Route path="teaching_methods" element={<TeachingMethods />} />
<Route path="project_activity" element={<ProjectActivity />} />
<Route path="medical-service" element={<MedicalService />} />
/*about education */
<Route path="zakon" element={<Zakon />} />
<Route path="collective" element={<Collective />} />
<Route path="vacancies" element={<Vacancies />} />
<Route path="admission" element={<Admission />} />
<Route path="statut" element={<Statut />} />
<Route path="license" element={<License />} />
<Route path="structure" element={<Structure />} />
<Route path="vipiska" element={<Vipiska />} />
<Route path="certificate-of-attestation" element={<CertificateOfAttestation />} />
<Route path="educational-programs" element={<EducationalPrograms />} />
<Route path="territory" element={<Territory />} />
<Route path="licensed-volume" element={<LicensedVolume />} />
<Route path="language" element={<Language />} />
<Route path="material-baza" element={<MaterialBaza />} />
<Route path="monitoring" element={<Monitoring />} />
<Route path="conduct" element={<Conduct />} />
<Route path="protidiya_nasiljstvu" element={<AboutViolence />} />
<Route path="report-activity" element={<ReportActivity />} />
/*parents */
<Route path="parents" element={<Parents />} />
<Route path="preparation" element={<Preparation />} />
<Route path="conditions" element={<Conditions />} />
<Route path="advice" element={<Advice />} />
<Route path="meeting" element={<Meeting />} />

```

```

<Route path="committee" element={<Committee />} />
{ /*news */}
<Route path="news/">
<Route index element={<NewsListPage />} />
<Route path=":id" element={<NewsSinglePage />} />
</Route>
{ /*gallery*/}
<Route path="gallery" element={<Gallery />} />
<Route path="photos" element={<PhotoGallery />} />
<Route path="videos" element={<VideoGallery />} />
{ /*psychologist */}
<Route path="psychologist-list" element={<PsychologistList />} />
<Route path="psychologist" element={<Psychologist />} />
<Route path="work_schedule" element={<WorkSchedule />} />
<Route path="psychological-and-pedagogical-seminars" element={<Seminars />} />
<Route path="for-parents" element={<ForParents />} />
<Route path="for-pedagogs" element={<ForPedagogs />} />
<Route path="for-schoolchildren" element={<ForSchoolchildren />} />
<Route
                                path="psychological-assistance-during-martial-law"
element={<PsychologicalAssistance/>}/>
<Route path="psychologist-post" element={<PsychologistPost/>}/>
{ /* documents*/}
<Route path="documents" element={<Documents />} />
<Route path="regulary-documents" element={<RegulatoryDocuments />} />
<Route path="financial-report" element={<FinancialReport />} />
<Route path="access" element={<Access />} />
<Route path="dopomoga" element={<Dopomoga />} />
<Route path="government-procurement" element={<GovernmentProcurement />} />
<Route path="contacts" element={<Contacts />} />
<Route path="feedback" element={<Feedback />} />
{ /*Logopedist*/}
<Route path="logoped-posts/">
<Route index element={<LogopedPostsList />} />
<Route path=":id" element={<LogopedPost />} />
</Route>
<Route path="buling" element={<BulingPage />} />
</Route>
</Routes>

```

```

</HelmetProvider>
</AuthProvider>
);
}
export default App;
hook/auth.js
import { useContext, createContext, useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import axios from 'axios';

const AuthContext = createContext();
const AuthProvider = ({ children }) => {
const [user, setUser] = useState(null);
const [email, setEmail] = useState(null);
const [roles, setRoles] = useState(null);
const [token, setToken] = useState(localStorage.getItem("accessToken") || "");
const navigate = useNavigate();
useEffect(() => {
const token = localStorage.getItem('accessToken');
if (token) {
const fetchData = async () => {
try {
const response = await fetchUserByToken(token);
if (response.data) {
setUser(response.data.username);
setEmail(response.data.email);
setRoles(response.data.roles);
}
} catch (error) {
console.error("Помилка при отриманні даних користувача:", error);
localStorage.removeItem("accessToken");
setToken("");
}
};
fetchUserData();
}, [token]);
const loginAction = (data) => {

```

```

    setUser(data.username);
    setToken(data.accessToken);
    setEmail(data.email);
    setRoles(data.roles);
    localStorage.setItem("accessToken", data.accessToken);
  };

  const logout = () => {
    setUser(null);
    setToken("");
    setEmail(null);
    setRoles(null);
    localStorage.removeItem("accessToken");
    navigate("/login");
  };

  return (
    <AuthContext.Provider value={{ token, user, email, roles, loginAction, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

export default AuthProvider;
export const useAuth = () => {
  return useContext(AuthContext);
};

const fetchUserByToken = async (token) => {
  return axios.get('http://localhost:8080/api/auth/user', { headers: { 'x-access-token': token } });
};

components/Appachment/ Appachment.js
import { useState, useEffect } from 'react';
import axios from 'axios';
import TitleSmall from '../Titles/TitleSmall';
import styles from './Attachment.module.scss';

const Attachment = ({ typeDocumentId }) => {
  const [attachments, setAttachments] = useState([]);
  useEffect(() => {
    const fetchDocuments = async () => {

```

```

    try {
      const response = await
    axios.get(`http://localhost:8080/api/documents/type/${typeDocumentId}`);
      setAttachments(response.data);
    } catch (error) {
      console.error('Error fetching documents:', error);
    }
  };
  fetchDocuments();
}, [typeDocumentId]);
const getFileExtension = (filename) => {
  return filename.slice(((filename.lastIndexOf(".") - 1) >>> 0) + 2);
};
const getIconForExtension = (extension) => {
  switch (extension) {
    case 'docx':
      return '/images/documentIcons/docx-icon.png';
    case 'doc':
      return '/images/documentIcons/docx-icon.png';
    case 'xlsx':
      return '/images/documentIcons/xlsx-icon.png';
    case 'pptx':
      return '/images/documentIcons/pptx-icon.png';
    case 'pdf':
      return '/images/documentIcons/pdf-icon.png';
    default:
      return '/images/documentIcons/default.png';
  }
};
const handleDownload = (url, filename) => {
  const link = document.createElement('a');
  link.href = url;
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
};
return (
  <div className={styles.attachmentWrapper}>

```

```

<TitleSmall titleText="Вкладення" />
<ul className={styles.attachmentList}>
  {attachments.map((attachment, index) => (
    <li className={styles.attachmentItem} key={index}>
      <img
        src={getIconForExtension(getFileExtension(attachment.filename))}
        alt={`${getFileExtension(attachment.filename)} icon`}
        className={styles.fileIcon}
      />
      <a className={styles.attachmentLink} href={attachment.url}
        onClick={() => handleDownload(attachment.url, attachment.filename)}>
        {attachment.filename}
      </a>
    </li>
  ))}
</ul>
</div>
);
};
export default Attachment;

```